An Analysis of Search Techniques for a Totally-Ordered Nonlinear Planner *

Jim Blythe
Computer Science Dept.
Carnegie Mellon University
Pittsburgh, PA 15213
blythe@cs.cmu.edu

Manuela Veloso
Computer Science Dept.
Carnegie Mellon University
Pittsburgh, PA 15213
veloso@cs.cmu.edu

Abstract

In this paper we present several domain-independent search optimizations and heuristics that have been developed in a totally-ordered nonlinear planner in PRODIGY. We also describe the extension of the system into a full hierarchical planner with the ability to search among the different levels of abstraction. We analyze and illustrate the performance of the system with its different search capabilities in a few domains.

1 Introduction

The PRODIGY architecture is designed to be both a unified testbed for different learning methods and a general purpose planner to solve interesting problems in complex task domains. Since the beginning of the PRODIGY research project [Minton et al., 1989], we have been focusing our attention on both the aspects of machine learning and planning, as we believe that this integration is necessary in the presence of complex domains. In this paper we focus on PRODIGY's planning algorithm. We present some recent developments in PRODIGY's planner, as an efficient totally-ordered nonlinear hierarchical planner. We first briefly situate our approach with respect to the related research and also historically within the project.

Nonlinear planning was developed to deal with problems like Sussman's anomaly, which could not be solved optimally by linear planners such as STRIPS [Fikes and Nilsson, 1971, Sussman, 1975]. Least-commitment planners handle this anomaly by deferring decisions while building the plan [Sacerdoti, 1975, Tate, 1977a, Wilkins, 1989]. These planners typically reason about a partially-ordered plan as opposed to a totally-ordered one. Typically, such plans have been referred to as *nonlinear* plans. However, the essence of the *nonlinearity* is not in the fact that the plan is partially ordered, but in the fact that a plan need not be a linear concatenation of complete subplans, each for a goal presumed independent of all others [Rich and Knight, 1991, Veloso, 1989, Regnier and Fade, 1992]. It is in this sense that we refer to our planner as totally-ordered and nonlinear.

The initial planner of PRODIGY [Minton et al., 1989] is linear and cannot therefore interleave subplans for different interacting goals. Its search procedure follows a *casual-commitment* approach, in which the planner, at choice points, commits to a particular alternative, generating the planning steps and testing their consequences, while searching for a solution. Background knowledge, whether hand-coded expertise, learned control rules or heuristic evaluation functions, guides the efficient exploration of the most promising parts of the search space. Provably incorrect alternatives are eliminated and heuristically preferred ones are explored first.

Later we developed a nonlinear planner, NOLIMIT [Veloso, 1989], that follows the same causal committment approach as the linear one does, but can also fully interleave goals at the different search levels. The plans generated are nonlinear, as they cannot be decomposed into a linear sequence of complete subplans for interacting conjunctive goals. The search space in a nonlinear planner is, by nature, much larger than in a linear planner. Efficiency becomes a major concern.

There have been a few recent attempts to analyze the advantages and disadvantages of totally-ordered versus partially-ordered nonlinear planning [Minton et al., 1991, Barrett et al., 1991]. In this paper we are not presenting a comparison between the

*This research was sponsored by the Avionics Laboratory, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U. S. Air Force, Wright-Patterson AFB, OH 45433-6543 under Contract F33615-90-C-1465, Arpa Order No. 7597. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

two paradigms. It is well known that both algorithms in the general case are either exponential or NP-complete [Chapman, 1987].

We recently reimplemented and extended this nonlinear planner, focusing on developing efficient matching techniques [Wang, 1992], and an efficient hierarchical search algorithm [Blythe, 1992]. In this paper we analyze some efficient search techniques and present empirical results for the complete nonlinear planner in our latest version of the system, PRODIGY4.0.

We have been increasingly aware [Veloso et al., 1990] that scaling up a planner to deal with complex domains requires at least the following capabilities:

- An adequate representation for the specification of the domain.
- A baseline planner with an efficient default search strategy.
- The ability to analyze the domain and automatically extract different levels of abstraction among its predicates.
- An efficient and complete planner that can organize its search hierarchically using different abstraction levels.
- Learning methods, capable of analyzing and introspecting the problem solving performance, compiling experience to guide the planner in subsequent problems.

We want to preserve, and where needed extend, PRODIGY's ability to cope with these requirements. In fact PRODIGY provides a rich action representation language coupled with an expressive control language [?]. The operators are represented by preconditions and effects. The preconditions are expressed in a typed first order predicate logic. They can contain conjunctions, disjunctions, negations, and both existential and universal quantifiers with typed variables. In addition, the operators can contain conditional effects, which depend on the state in which the operator is applied. (Chapman showed that extending a least-commitment planner — but *not* a casual-commitment planner — to reason about conditional effects is NP-hard [Chapman, 1987].) These language constructs are important for representing complex and interesting domains. The control language allows the problem solver to represent and learn control information about the various problem-solving decisions, such as selecting which goal or subgoal to address next, which operator to apply, what bindings to select for the operator or where to backtrack in case of failure. In PRODIGY, there is a clear division between the declarative domain knowledge (operators and inferences rules) and the more procedural control knowledge. This simplifies both the initial specification of a domain and the automatic learning of the control knowledge.

On the other hand, PRODIGY's planner produces a complete search tree, encapsulating all decisions – right ones and wrong ones – as well as the final solution. This information is used by each learning component in different ways: to extract control rules via EBL [Minton, 1988], to build derivational traces (cases) by the derivational analogy engine [Veloso and Carbonell, 1993], to analyze key decisions by a knowledge acquisition interface [Joseph, 1989], or to formulate focused experiments [Carbonell and Gil, 1990]. The axiomatized domain knowledge is also used to learn abstraction layers [Knoblock, 1991], and statically generate control rules [Etzioni, 1990].

Although we do not present an analysis supporting our planning algorithm, we provide in this paper empirical evidence that PRODIGY's planner can handle complicated problems efficiently. In fact the paper only illustrates a small sample of the large variety of tasks that PRODIGY4.0 can effectively solve.

This paper is organized as follows. In the next section we describe the search algorithm used in PRODIGY4.0, explaining how the search space is elaborated as well as some of the heuristics and search reduction techniques used to make the system more efficient. The following section describes the integration of abstraction in PRODIGY4.0. This is done in a flexible way, so as to allow the system complete freedom to move between the different abstraction levels in the search. Finally, we present and discuss preliminary results with PRODIGY4.0 in a number of domains.

2 Search in PRODIGY4.0

The philosophy of the PRODIGY project is that a simple domain-independent planner can be combined with learning methods for acquiring control knowledge within a domain to provide sophisticated domain-independent planning [Minton, 1988, Etzioni, 1990, Veloso, 1992]. While PRODIGY4.0 still subscribes to this approach, a number of domain-independent search optimizations and heuristics have been added in order to improve the base-line performance of the system. It is intended that these will be used in conjunction with domain-dependent knowledge expressed as control rules [Minton, 1988, Etzioni, 1990] or other forms of learned control guidance, such as past problem-solving episodes stored as derivational cases [Veloso, 1992].

In this section we outline the search space explored by PRODIGY4.0 and present the search reduction techniques and heuristics that are used. For the purposes of this discussion, we view search reduction techniques, for example dependency-directed backtracking, as removing parts of the search space, while heuristics direct the attention of the searcher. Neither class can guarantee to reduce the amount of work needed to

reach a solution.

Both dependency-directed backtracking and the heuristics used in PRODIGY4.0 are customized to its search algorithm, which we now briefly describe. By default, PRODIGY performs a depth-first, means-ends analysis in the state space to construct a plan. However, control rules can be used to switch attention to different points of the search tree as well as to determine local choices. Thus control rules can produce best-first or breadth-first search, for example.

PRODIGY4.0 uses four different types of nodes in its search tree, corresponding to the different decisions that are made.

- *Goal nodes* record the choice to work on a specific goal from the set of currently active goals in the system.
- *Operator nodes* represent the choice to use a specific operator to achieve the goal currently being worked on.
- *Binding nodes* represent a set of bindings chosen to instantiate the variables of the current operator.
- *application nodes* represent the decision to apply an instantiated operator whose preconditions are true. The application nodes in a path through the search tree from the root to a leaf correspond to the partial plan that the path represents — a complete plan if the leaf node satisfies the goal description.

A simplified version of the basic search algorithm used by PRODIGY4.0 is shown in figure 1. It can be found in more detail in [**?**]. $G(n)$, the goal set at node n, is by default calculated as the goal set of n's parent in the search tree, but augmented in step 4.

1. Initialize search tree T and the list O of open nodes to consist of one node r, an applied-op node recording the applications of inference rules on the initial state. Set $G(r)$ to the set of top-level goals.
2. Choose a node n from O to expand.
3. If n is a goal node, choose an operator to achieve n and make a child operator node reflecting this decision point.
4. If n is an operator node, choose a set of bindings for the operator and make a child binding node b. Add the preconditions of the instantiated operator to $G(b)$.
5. If n is a binding or applied-op node, choose either one goal from the set $G(n)$ of current goals or the nearest ancestor binding node that has not been applied on this branch of the search tree.
6. If n has no possible expansions that have not been explored, remove it from O. This is referred to as *closing* the node n.
7. Goto step 2.

Figure 1: A simplified version of the search algorithm.

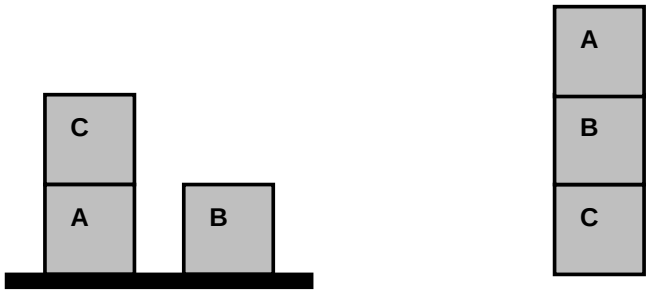

Figure 2: The initial and goal state descriptions for Sussman's Anomaly

2.1 Search Reduction Techniques

PRODIGY4.0 uses three different kinds of search reduction techniques: checking for infinite loops, dependency-directed backtracking and a variant of look-ahead [Haralick and Elliot, 1980].

Two kinds of infinite loops are detected during search: *state loops*, where a sequence of operators has resulted in the same state, and *goal loops*, where through a chain of subgoals the system repeats the goal it is trying to solve. These two loops lie in different networks, which are maintained simultaneously. State loops are found by looking at the order in which operators are applied, recorded in PRODIGY's search tree, while goal loops are found by looking at the subgoal tree. These are different because PRODIGY4.0 interleave subgoals from different higher-level goals.

The main effect of *dependency-directed backtracking* in PRODIGY4.0 is to remove binding nodes from consideration when a necessary subgoal introduced by the node is later found to be unachievable. To illustrate this and some of the techniques described later, we will use the blocksworld domain, and in particular Sussman's anomaly [Sussman, 1975]. The empirical results presented in section 4 include more realistic planning domains. The representation of the blocksworld for this paper uses the predicates `on`, `ontable`, `clear`, `holding` and `armempty`, and the four operators `unstack`, `stack`, `pickup` and `putdown`. Figure 2 shows a diagram of the starting and goal states for Sussman's anomaly. Figure 3 shows part of the goal tree that PRODIGY4.0 expands for this problem. Note that the goal tree is not the same as PRODIGY's search tree, in which a plan is represented by a linear path.

When PRODIGY4.0 solves this problem, it begins with the goal `(on A B)`, subgoaling as shown in the fig-

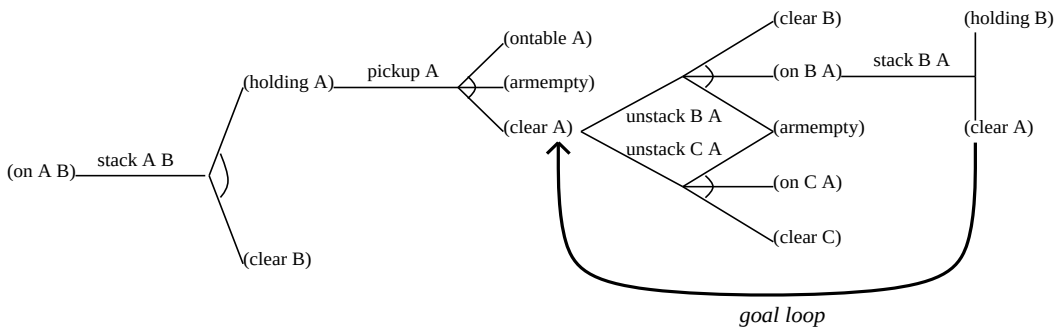

Figure 3: Part of the goal tree for Sussman's Anomaly

ure on `(holding A)` and then on `(clear A)`. Among the operators considered for achieving the last goal is `<unstack B A>`, leading to a subgoal `(on B A)`, which is unachievable because the only operator for it, `stack`, causes a goal loop with `(clear A)`. Dependency-directed backtracking at this point leads the system to close the binding node for `<unstack B A>` from the search tree, because it introduced this unachievable subgoal. Brute-force search would result in keeping this node in PRODIGY's search space in order to expand the path where the system switches attention to the goal `(on B C)` at this point, having now committed to using the `unstack` operator to achieve `(clear A)`, although this can never work.

In this example the subgoal was unachievable because of a goal loop. Subgoals are also unachievable if no satisfactory bindings can be found for any relevant operator. We note in passing that it is not necessary at run time in PRODIGY to consider the case where there are no relevant operators for a subgoal. A pre-processing stage automatically detects such goals, and includes the necessary tests to avoid them in the matching stage as bindings are computed [Wang, 1992]. These predicates are called *static predicates* in PRODIGY, and are analogous in function to filters in Nonlin [Tate, 1977b].

One instance of *look-ahead* in PRODIGY4.0 is to eliminate fruitless operator nodes. Once an operator is chosen, a set of bindings is chosen to instantiate the operator. However, an analysis of the possible bindings made by regressing the goal through the operator and examining the weakest preconditions [Waldinger, 1977] may show that no good set of bindings can be found. When this happens, PRODIGY4.0 will remove the operator node from the search space without generating any of the binding nodes. Binding nodes represent are the most expensive portion of the search algorithm, the matching stage [Wang, 1992].

PRODIGY4.0 also uses lookahead to check for goal loops. Consider when the system is first trying to solve `(clear A)` in our example. Although it is not shown in the trace, the system actually first considers the operators `put-down` and `stack` to achieve `(clear A)`, since it has no control knowledge to lead it to `<unstack C A>` at this point. Both of these operators are rejected before considering bindings, however. The weakest preconditions for `stack` to achieve `(clear A)` include `(holding A)`, which would create a goal loop with the goal node above `(clear A)` in figure 3.

The savings in terms of the reduction in the search space produced by look-ahead as currently used in PRODIGY4.0 depend on the number of binding nodes that would otherwise be created, which is roughly exponential in the number of variables in the operator.

2.2 Search Heuristics

The search heuristics currently in use in PRODIGY4.0 take the form of orderings on lists of choices at a decision point, analogous to preference control rules. One heuristic is to prefer bindings with a minimum number of unsolved preconditions derived from considerations of conspiracy numbers for and/or trees [Elkan, 1989], [Blythe, 1992]. Another heuristic is to prefer goals with a maximum "abstraction level", which makes use of automatically generated information expressed as an abstraction hierarchy (see the next section) without explicitly using different abstraction spaces in the planning process.

2.3 Limits to the scope of search optimizations and heuristics

We now note some of the potential interactions between the search techniques described in this section and the control rules that also direct the search in PRODIGY4.0.

The search techniques listed here only use knowledge about PRODIGY's search algorithm, not domain-dependent knowledge. While such techniques will be very generally applicable, care has to be taken when using them in the presence of domain-dependent control knowledge to ensure that they do not combine to produce poor decisions.

Figure 4 illustrates this point. Suppose that the plan-

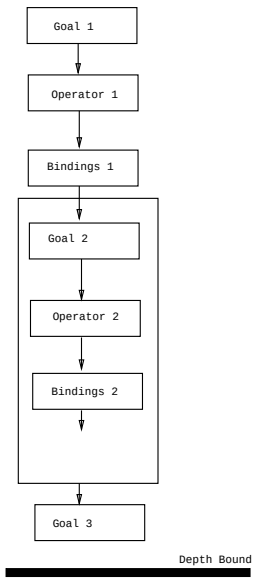

Figure 4: Possible incompleteness of dependency-directed backtracking in the presence of other control parameters.

ner is attempting to achieve "goal 1", represented by the uppermost box in the figure, and that in doing so it chooses the expand the bindings node "bindings 1", which requires solving the conjunction of goals 2 and 3. Suppose that the planner works on goal 2 for a while, indicated by the surrounding box, and then turns attention to goal 3. At this point the planner hits a depth bound that has been imposed on the search, and fails. If dependency-directed backtracking was used in this case, the system would close the node for "bindings 1", since it requires a goal that failed. However, if an alternative plan could have achieved goal 2 in fewer nodes, then goal 3 may be solvable in some search tree rooted under "bindings 1". Thus in this case naive application of these techniques can destroy the completeness of the planner. Similar interactions can take place with control rules using information about the state of the search tree.

Of course, the use of a depth bound may destroy completeness single-handedly if it set too low, and in general the use of control rules that select or reject among the choices can have this effect. While this paper does not address the issue of maintaining completeness in the presence of control knowledge, it is worth mentioning that the issue is there.

3 Integration of abstraction in prodigy4.0

PRODIGY4.0 includes a mechanism for generating abstraction hierarchies automatically, and can incorporate different abstraction levels within the same search tree. The method for generating and plannig with abstraction hierarchies was developed by Knoblock [Knoblock, 1991] for PRODIGY's linear planner, although we have extended this. In particular, subgoals of goals at one abstraction level can be interleaved at the lower level, a feature which is necessary to preserve the completeness of PRODIGY4.0.

Hierarchical planning is a relatively widely-used technique for dealing with the complexity of fine detail in planning ([Sacerdoti, 1974], [Tenenberg, 1990], [Knoblock, 1991]), as well as having uses for limiting the effects of uncertainty in planning. In its most common form, a successive set of simplifications of the problem is generated. Then plans are made in the simplest version of the problem and progressively refined until the full problem is solved in the most detailed or *ground space*. If the hierarchy of versions of the planning problem is well chosen, this technique can lead to significant improvements in planning efficiency.

In work by Knoblock and others ([Knoblock et al., 1991],[Yang and Tenenberg, 1990]), some properties of abstraction hierarchies were identified that would allow hierarchical planners to be complete and correct, and enable efficient backtracking. The *monotonicity property*, which holds for all abstraction spaces, says that the existence of a solution in the ground space implies that there exists a solution in the more abstract space that can be refined to the ground level solution without undoing any of the domain features established in the abstract plan. This property can be used to constrain the refinement of an abstract plan.

The hierarchical planner implemented in the earlier version of PRODIGY makes use of a ground-space planner as a subroutine, which is called recursively on each subproblem at the different abstraction levels. Knoblock shows that this hierarchical planner is correct, but that it is only complete for the class of planning problems that is *linearizable* — in other words where every conjunctive set of goals that arises during problem solving can be solved in the order the goals appear in the abstraction hierarchy. The mechanism for planning with abstraction in PRODIGY4.0 maintains completeness by allowing steps needed to refine a plan to be inserted anywhere in the plan. This extension to the refinement method retains the *monotonicity property* and thus allows for efficient backtracking between abstraction levels.

In PRODIGY4.0, each node in the search tree has an abstraction level, and the single search tree represents plans at all levels once planning is completed. This allows for more flexible control of planning with abstraction than the previous approach, where the planner is called as a subroutine by the abstraction module. Control rules can be written to decide at a given node in the search whether to continue searching at the current abstraction level or move to a different one.

Min Unsolved Preconditions	Dep-directed backtracking			
	On		Off	
On	107	4.7	107	4.7
Off	128	5.6	128	5.6

Table 1: Different combinations of search techniques with abstraction.

Min Unsolved Preconditions	Dep-directed backtracking			
	On		Off	
On	65	1.3	65	1.4
Off	69	1.3	797	12.2

Table 2: Sussman's anomaly with no use of abstraction.

4 Empirical Results

The results in this section show preliminary results for the effects of the search optimizations and speedups and of the abstraction techniques described in the previous sections. PRODIGY4.0 has been applied to a number of domains, including a transportation domain which models moving packages by airplane and truck across the country, and a reduced version of a process planning domain [Gil, 1991].

In each of these domains, we have written a set of control rules that allow most problems to be solved efficiently. For the results in this section, however, they were not used. We found that the hierarchical planner was sometimes able to run comparably to the domain with control rules, but was not usually within an order of magnitude of this speed. Various improvements are possible, however, as there are a number of places in the algorithm for generating abstractions where domain knowledge and domain-independent heuristics can be brought to bear [Knoblock, 1991].

Perhaps as could have been expected, abstraction showed typically a larger margin of improvement over planning in the ground space for more complex scenarios than for simpler ones. In various versions of the transportation domain without control rules, we only compared different combinations of the search reduction techniques and heuristics while using abstraction, because times for the ground space planner were so much greater. Table 1 is such an example, where we have used abstraction and various combinations of dependency-directed backtracking (along the row, labelled "D") and the heuristic to prefer fewer unsolved preconditions. In each cell we have the total number of nodes in the search tree, generated in all levels of the hierarchy, and the CPU time in seconds. This problem involved the routing of three packages in the transportation domain. Here, dependency directed backtracking has no bearing, and minimizing preconditions very little.

By contrast, table 2 shows the same combinations of techniques for the Sussman's anomaly problem mentioned in section 2.1, without the use of abstraction. In fact, the literals of the blocksworld do not exhibit sufficient structure to create an abstraction hierarchy using the reduced model method. The use of at least one of the two search techniques reduces the search by an order of magnitude in this case.

5 Conclusions and some future directions

In this paper we briefly described the nonlinear planning algorithm we use in PRODIGY. We discussed its philosophy in the context of an architecture for learning and planning.

We presented different search optimization techniques that improved the efficiency of the base planner as well as the hierarchical planning algorithm we developed. All of them were general in nature, and have been applied to a number of domains. We showed some results that illustrated the trade-offs among the different techniques used.

One advantage of this more flexible control of abstract planning is to allow a time-critical planner to flesh out the complete plan only at the most abstract levels and to progressively refine only the first step from each more abstract level. This would allow the planner to quickly arrive at a plausible first action to take, although it may not be guaranteed to succeed, and then to continue to expand the search space more thoroughly as time allowed. In this way abstraction hierarchies can provide a simple form of "anytime algorithm" for computing at least the first step of a plan ([Dean and Boddy, 1988], [Elkan, 1990]). We see this aspect as well as the potential for reasoning about plan quality in a simplified, abstract domain as very interesting lines of future work.

The current default version of the algorithm returns with the first successful plan found, although it is possible to allow the system to produce more than one plan. Eventually we would like PRODIGY to consider the quality of the plan it produces, and to search for the best plan that can be found within given resource bounds. The search for a better plan, as opposed to simply one that satisfies the goals, will increase the size of the search space, however. One promising way to focus the search for better plans might be to first look for the best plans at more abstract levels of the system, and then to refine the better plans in the search for a best plan at the less abstract levels.

The PRODIGY system is freely available to research groups. We encourage interested parties to contact

the Prodigy Project at the authors' address, or to send electronic mail to prodigy@cs.cmu.edu.

Acknowledgements

The design of PRODIGY4.0 was, and still is, a joint effort, for which the authors acknowledge the whole PRODIGY research team, but very much in particular Craig Knoblock, Xuemei Wang, Alicia Pérez, Dan Kahn, Yolanda Gil, Robert Joseph, Oren Etzioni, and William Reilly. The initial design was developed by Jim Blythe, Xuemei Wang, and Dan Kahn. We specially thank Craig Knoblock for his work on abstraction and for the many helpful discussions on the development of PRODIGY4.0's hierarchical planning strategy. Yolanda Gil provided some very helpful comments on an earlier draft of this paper. Last, but not least, the authors acknowledge Steven Minton, Jaime Carbonell, Craig Knoblock, and Dan Kuokka for starting the PRODIGYproject, in particular Steven Minton for PRODIGY2.0, and Daniel Borrajo for a major part of NOLIMIT's implementation. It was our past experience with PRODIGY2.0 and NOLIMIT that definitely drove our design and implementation of PRODIGY4.0.

References

[Barrett et al., 1991] Barrett, A., Soderland, S., and Weld, D. (1991). The effect of step-order representations on planning. Technical Report 91-05-06, Department of Computer Science and Engineering, University of Washington.

[Blythe, 1992] Blythe, J. (1992). Search techniques for PRODIGY4.0. Technical report, School of Computer Science, Carnegie Mellon University. forthcoming.

[Carbonell and Gil, 1990] Carbonell, J. G. and Gil, Y. (1990). Learning by experimentation: The operator refinement method. In Michalski, R. S. and Kodratoff, Y., editors, *Machine Learning: An Artificial Intelligence Approach, Volume III*, pages 191–213. Morgan Kaufmann, Palo Alto, CA.

[Chapman, 1987] Chapman, D. (1987). Planning for conjunctive goals. *Artificial Intelligence*, 32:333–378.

[Dean and Boddy, 1988] Dean, T. and Boddy, M. (1988). An analysis of time-dependent planning problems. In *National Conference on Artificial Intelligence*.

[Elkan, 1989] Elkan, C. (1989). Conspiracy numbers and caching for searching and/or trees and theorem proving. In *International Joint Conference on Artificial Intelligence*.

[Elkan, 1990] Elkan, C. (1990). Incremental, approximate planning. In *National Conference on Artificial Intelligence*.

[Etzioni, 1990] Etzioni, O. (1990). *A Structural Theory of Explanation-Based Learning*. PhD thesis, School of Computer Science, Carnegie Mellon University. Available as technical report CMU-CS-90-185.

[Fikes and Nilsson, 1971] Fikes, R. E. and Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208.

[Gil, 1991] Gil, Y. (1991). A domain-independent framework for effective experimentation in planning. In *Proceedings of the Eight International Workshop on Machine Leaning*, Evanston, IL. Morgan Kaufmann.

[Haralick and Elliot, 1980] Haralick, R. M. and Elliot, G. L. (1980). Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313.

[Joseph, 1989] Joseph, R. L. (1989). Graphical knowledge acquisition. In *Proceedings of the 4^{th} Knowledge Acquisition For Knowledge-Based Systems Workshop*, Banff, Canada.

[Knoblock, 1991] Knoblock, C. A. (1991). *Automatically Generating Abstractions for Problem Solving*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. Available as technical report CMU-CS-91-120.

[Knoblock et al., 1991] Knoblock, C. A., Tenenberg, J. D., and Yang, Q. (1991). Characterizing abstraction hierarchies for planning. In *National Conference on Artificial Intelligence*.

[Minton, 1988] Minton, S. (1988). *Learning Effective Search Control Knowledge: An Explanation-Based Approach*. PhD thesis, Computer Science Department, Carnegie Mellon University. Available as technical report CMU-CS-88-133.

[Minton et al., 1991] Minton, S., Bresina, J., and Drummond, M. (1991). Commitment strategies in planning: A comparative analysis. In *Proceedings of IJCAI-91*, pages 259–265.

[Minton et al., 1989] Minton, S., Knoblock, C. A., Kuokka, D. R., Gil, Y., Joseph, R. L., and Carbonell, J. G. (1989). PRODIGY 2.0: The manual and tutorial. Technical Report CMU-CS-89-146, School of Computer Science, Carnegie Mellon University.

[Regnier and Fade, 1992] Regnier, P. and Fade, B. (1992). Integrating linear planning and parallel execution of actions. In *IEEE, Robotics and Automation*.

[Rich and Knight, 1991] Rich, E. and Knight, K. (1991). *Artificial Intelligence*. McGraw-Hill, inc.

[Sacerdoti, 1974] Sacerdoti, E. D. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135.

[Sacerdoti, 1975] Sacerdoti, E. D. (1975). The nonlinear nature of plans. In *Proceedings of IJCAI-75*, pages 206–213.

[Sussman, 1975] Sussman, G. J. (1975). *A Computer Model of Skill Acquisition*. American Elsevier, New York. Also available as technical report AI-TR-297, Artificial Intelligence Laboratory, MIT, 1975.

[Tate, 1977a] Tate, A. (1977a). Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 888–900.

[Tate, 1977b] Tate, A. (1977b). Generating project networks. In *International Joint Conference on Artificial Intelligence*.

[Tenenberg, 1990] Tenenberg, J. (1990). Some representative article. *Artificial Intelligence*.

[Veloso, 1989] Veloso, M. M. (1989). Nonlinear problem solving using intelligent casual-commitment. Technical Report CMU-CS-89-210, School of Computer Science, Carnegie Mellon University.

[Veloso, 1992] Veloso, M. M. (1992). *Learning by Analogical Reasoning in General Problem Solving*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. Available as technical report CMU-CS-92-174.

[Veloso and Carbonell, 1993] Veloso, M. M. and Carbonell, J. G. (1993). Derivational analogy in PRODIGY: Automating case acquisition, storage, and utilization. *Machine Learning*, 10:249–278.

[Veloso et al., 1990] Veloso, M. M., Carbonell, J. G., and Knoblock, C. A. (1990). Nonlinear planning in complex domains using a casual-commitment approach. In *Preprints of the AAAI-90 Workshop on Planning in Complex Domains*, MIT, MA.

[Waldinger, 1977] Waldinger, R. (1977). Achieving several goals simultaneously. *Machine Intelligence 8*.

[Wang, 1992] Wang, X. (1992). Constraint-based efficient matching in prodigy. Technical Report CMU-CS-92-128, School of Computer Science, Carnegie Mellon University.

[Wilkins, 1989] Wilkins, D. E. (1989). Can AI planners solve practical problems? Technical Note 468R, SRI International.

[Yang and Tenenberg, 1990] Yang, Q. and Tenenberg, J. D. (1990). Abtweak, abstracting a nonlinear, least commitment planner. In *National Conference on Artificial Intelligence*.