The Need for Different Domain-Independent Heuristics*

Peter Stone and **Manuela Veloso** and **Jim Blythe**
School of Computer Science
Carnegie Mellon University
Pittsburgh PA 15213-3890
{pstone, veloso, blythe}@cs.cmu.edu

Abstract

PRODIGY's planning algorithm uses domain-independent search heuristics. In this paper, we support our belief that there is no single search heuristic that performs more efficiently than others for all problems or in all domains. The paper presents three different domain-independent search heuristics of increasing complexity. We run PRODIGY with these heuristics in a series of artificial domains (introduced in (Barrett & Weld 1994)) where in fact one of the heuristics performs more efficiently than the others. However, we introduce an additional simple domain where the apparently worst heuristic outperforms the other two. The results we obtained in our empirical experiments lead to the main conclusion of this paper: planning algorithms need to use different search heuristics in different domains. We conclude the paper by advocating the need to learn the correspondence between particular domain characteristics and specific search heuristics for planning efficiently in complex domains.

Introduction

PRODIGY is an integrated architecture for research in planning and learning (Carbonell, Knoblock, & Minton 1990). The focus of the PRODIGY project has been on understanding how an AI planning system can acquire expertise by using different machine learning strategies. Like all planning systems, PRODIGY needs to be able to efficiently handle multiple interacting goals and different ways of achieving goals. In this sense, planning involves deciding among available choices.

Every planning algorithm uses some heuristic strategy to control its decision-making process. It could be argued that there exists a *best* planning heuristic that performs effectively for any general-purpose planning domain. Were that the case, there would be little or no need to analyze or learn domain and problem characteristics. However, this paper supports our belief that there is no single domain-independent search heuristic that performs more efficiently than others for all problems or in all domains.

In the next section we present PRODIGY's planning algorithm and three different domain-independent search heuristics of apparently increasing complexity. After this we show the empirical results we obtained by running PRODIGY with these different heuristics. We test them in a series of artificial domains (introduced in (Barrett & Weld 1994)) in which one of the heuristics performs more efficiently than the others. However, we introduce an additional simple domain, representative of a large class of domains, for which the apparently worst heuristic outperforms the other two. This example supports our claim that there is not a universally best heuristic. Our heuristics, which we compare with other planners in the following section, expend different levels of effort in analyzing interactions among operators. We conclude the paper by advocating the need to learn the correspondence between particular domain characteristics and specific search heuristics for planning efficiently in complex domains.

Domain-Independent Search Heuristics in PRODIGY

PRODIGY's planning algorithm can be described as a means-ends analysis planner in the sense that it uses its internal state of the world to identify what goals it needs to plan for. It changes the state when it finds a plan to achieve a goal. These changes in state are PRODIGY's way of simulating the execution of a plan during the planning process. Because it commits to a particular execution order while planning, PRODIGY is a totally ordered planner. Table 1 shows the skeleton of PRODIGY's planning algorithm.

This simplified version of PRODIGY's algorithm can be captured by the following regular expression: (Subgoal Apply*)*. In this context, "Subgoal," corresponding to steps 3 and 4 in table 1, means to choose a goal for which to plan. "Apply," as in step 5, means

*This research is 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 authors 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.

```
1. Terminate if the goal statement is satisfied in the
   current state.
2. Compute the set of pending goals $\mathcal{G}$, and the set
   of applicable operators $\mathcal{A}$. A goal is pending if it is
   a precondition, not satisfied in the current state,
   of an operator selected to be in the plan to achieve
   a particular goal. An operator is applicable when
   all its preconditions are satisfied in the state.
3. Choose a goal $G$ from $\mathcal{G}$ or select an operator $A$
   from $\mathcal{A}$.
4. If $G$ has been chosen, then
   • Expand goal $G$, i.e., get the set $\mathcal{O}$ of relevant
     instantiated operators that could achieve the
     goal $G$,
   • Choose an operator $O$ from $\mathcal{O}$,
   • Go to step 1.
5. If an operator $A$ has been selected as directly ap-
   plicable, then
   • Apply $A$,
   • Go to step 1.
```

Table 1: A skeleton of PRODIGY's planning algorithm and choice points.

to change the internal state. Since PRODIGY uses the state to decide which goals still need to be achieved (step 2), it makes a difference whether or not PRODIGY applies an operator at a given time.

In this paper we analyze three different domain-independent search heuristics that PRODIGY uses to control the decision of when to subgoal and when to apply an operator, i.e, how to simulate the execution of a plan. The heuristics represent different levels of effort in analyzing interactions among operators when selecting an ordering.

Eager state changes

The first heuristic corresponds to the situation in which PRODIGY eagerly applies an operator as soon as it becomes applicable. We name this heuristic SAVTA, standing for "*S*ubgoal *A*fter e*V*ery *T*ry to *A*pply." SAVTA corresponds to the following behavior: PRODIGY expands goals and changes the state as soon as it finds a plan that achieves a particular goal. Here, PRODIGY applies operators as soon as they become applicable; it only subgoals after it has applied every possible operator. Table 2 shows PRODIGY's planning algorithm when using SAVTA as its heuristic to control the decision of when to change its state. Notice that PRODIGY loops in step 4 until no more operators are applicable.

The assumption underlying SAVTA's behavior is that an immediate change in state will generally lead to a more informed planning situation. SAVTA expects that the changes in state achieve the goals for which the operators were selected. Furthermore, SAVTA expects that these changes will not prevent other pending goals from being achieved.

```
1. Terminate if the goal statement is satisfied in the
   current state.
2. Compute the set of pending goals $\mathcal{G}$.
3. Choose a goal $G$ from $\mathcal{G}$.
   • Expand goal $G$, i.e., get the set $\mathcal{O}$ of relevant
     instantiated operators that could achieve the
     goal $G$,
   • Choose an operator $O$ from $\mathcal{O}$.
   • Let $A$=$O$ be the current operator.
4. If $A$ is directly applicable in the current state,
   then
   • Apply $A$,
   • Identify a new possible applicable operator $A$.
   • Go to step 4.
5. Go to step 1.
```

Table 2: PRODIGY's planning algorithm using SAVTA. PRODIGY subgoals after it has applied every possible applicable operator.

Delayed state changes

The other two heuristics maximally delay the possible changes in state, i.e., they only apply operators when there are no pending goals. We name both of these heuristics SABA, standing for "*S*ubgoal *A*lways *B*efore *A*pplying." The SABA heuristics share the following behavior: PRODIGY changes the state only after there are no more goals to expand. Here PRODIGY builds up a set of applicable operators from which to choose. The two SABA heuristics differ in the way they deal with this set of applicable operators. The first, SABA-reactive, selects an operator to apply based solely on the current set, while the second, SABA-memory, takes past sets of applicable operators into account. Table 3 shows PRODIGY's planning algorithm following the SABA behavior. Notice that PRODIGY may have to enter the loop in step 3 more than once, since some goals may need to be reachieved.

Were PRODIGY to just apply the applicable operators in the order in which they were discovered, the SABA heuristics would be quite similar to SAVTA. However, by spending a little bit of effort examining the interactions among applicable operators, PRODIGY can often apply them in a better order.

With the SABA-reactive heuristic, PRODIGY only examines the currently applicable operators when deciding which one to apply. In effect, PRODIGY checks for an operator that satisfies two conditions. First, an operator should not delete any preconditions of any of the other applicable operators. Second, no other operators should delete any effects of the operator. Further, if none of the applicable operators satisfy both of these conditions, SABA gives preference to the first condition.

By using the two conditions mentioned above, PRODIGY may increase its chances of successfully applying operators without having to backtrack. These conditions guide PRODIGY through the search space,

```
1. Terminate if the goal statement is satisfied in the
   current state.
2. Compute the set of pending goals $\mathcal{G}$.
3. If $\mathcal{G}$ is not empty then
   • Choose a goal $G$ from $\mathcal{G}$ and remove $G$ from
     $\mathcal{G}$.
   • Expand goal $G$, i.e., get the set $\mathcal{O}$ of relevant
     instantiated operators that could achieve the
     goal $G$,
   • Choose an operator $O$ from $\mathcal{O}$.
   • Go to step 3.
4. Compute the set of applicable operators $\mathcal{A}$.
5. Select an operator $A$ from $\mathcal{A}$.
   • Apply $A$.
   • Go to step 1.
```

Table 3: PRODIGY's planning algorithm using SABA. PRODIGY always subgoals before applying any operators.

possibly helping it find a more direct path to the solution. However, they do not prune PRODIGY's search space.[1]

Like SABA-reactive, SABA-memory first checks to see if any operator satisfies the same two conditions: the operator doesn't delete any preconditions, and its effects are never deleted. However, when no operator satisfies these conditions, it examines the interactions among operators more deeply. It uses a bias to maintain its focus of attention by preferring operators that work towards the same goal as the most recently applied operator did. The most interesting difference between the two heuristics is that, as indicated by its name, SABA-memory saves the information it extracts from this examination and uses it for future choices. Using SABA-memory, PRODIGY can thus choose an operator based on past interactions between operators, even if they are no longer apparent in the current list of operators.

One can view the heuristics we have examined in this section as representing different patterns of commitment strategies in search. SAVTA commits eagerly to the order of applying operators, and delays commitment to the choice of operators. In contrast, the SABA heuristics delay commitment to step ordering, but commit eagerly to the operators used to achieve goals.

Domain-Dependence of the Heuristics

We empirically compare the performance of our three heuristics in a set of artificial domains created by Barrett and Weld.[2] These domains were devised to investigate the relative performance of planners that reason about a partial order of plan steps and those that use a total order. The domains were used to test the hypothesis that a set of serializability properties play a key role in predicting performance. Barrett and Weld found that their two total-order planners performed poorly in these domains. Since Prodigy uses a total order we were interested in its performance on this set of domains. We have also been testing our heuristics in a set of classical domains.

We ran all of our tests on a SPARC station. In each domain, we generated random problems having one to fifteen goals: ten problems with each number of goals. We used these same 150 problems to test each of the three heuristics. To get our data points, we averaged the results for the ten problems with the same number of goals. We graph the average time that PRODIGY took to solve the problems versus the number of goals.

D^0S^1: all heuristics work

D^0S^1 is a domain in which all goals are independent. Each of the fifteen operators looks like:[3]

Operator	preconds	adds	deletes
A_i	$\{I_i\}$	$\{G_i\}$	$\{\}$

Since none of the operators delete anything, it does not matter in what order the goals are solved, or in what order the operators are applied. In this case, all three heuristics perform roughly linearly[4] (Figure 1). Note

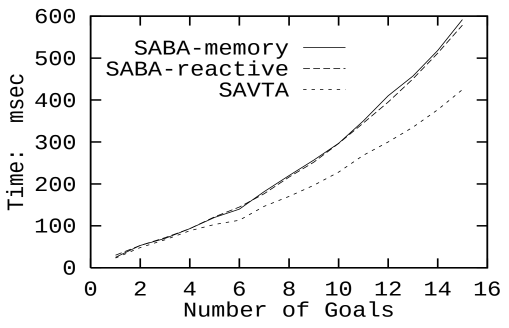


Figure 1: PRODIGY's performance with different heuristics in domain D^0S^1

that the SABA heuristics have some overhead involved in choosing from among several applicable operators, so that SAVTA slightly outperforms the SABAs. However, since we are mostly interested in the orders of the

[1]We have also created a version of SABA that prunes the search space when it is provably impossible to be deleting a solution from the space.

[2]We use D^0S^1, D^mS^1, D^1S^{1*}, and D^mS^{2*} as presented in detail in (Barrett & Weld 1994) and in (Barrett & Weld 1993).

[3]In all domains described in this paper, I_i are true in the initial state, and G_i are goals to be achieved.

[4]Although the time is not linear, it is significantly subquadratic. The number of nodes searched, however, is perfectly linear in the number of goals since PRODIGY does not have to backtrack. This type of linearity holds in all other cases where we mention sub-quadratic time behavior.

graphs, the three heuristics essentially perform equally well.

D^mS^1 and D^1S^{1*}: SAVTA fails

D^mS^1 and D^1S^{1*} are two domains in which the order of application matters.[5] The operators in D^mS^1 look like:

Operator	preconds	adds	deletes
A_i	$\{I_i\}$	$\{G_i\}$	$\{I_j \mid j < i\}$

Since each operator deletes the preconditions of all operators numerically before it, these operators can only be applied in increasing numerical order. Similarly, the operators must be applied in a particular order in D^1S^{1*}. This domain has the following operators:

Operator	preconds	adds	deletes
A_i	$\{I_i\}$	$\{G_i\}$	$\{I_{i+1}, I_*\}$
A_*^2	$\{P_*\}$	$\{G_*\}$	$\{I_1\}$
A_*^1	$\{I_*\}$	$\{P_*\}$	$\{\}$

In D^1S^{1*}, G_* is an additional goal in every problem.

Since SAVTA applies operators as soon as they become applicable, PRODIGY can end up backtracking heavily when using this heuristic. For instance in D^mS^1, suppose we give PRODIGY the problem to attain G_7 and G_4 with I_7 and I_4 true in the initial state. PRODIGY will begin by finding that operator A_7 achieves G_7, and then will immediately apply A_7. However, A_7 deletes I_4. Thus, when PRODIGY next tries to solve G_4, it will fail and need to backtrack. With more than two goals, this problem is compounded exponentially (Figure 2).

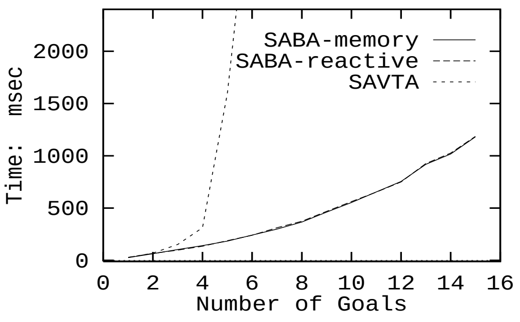
On the other hand, the two SABA heuristics choose the operators to apply more carefully. Using the same example, PRODIGY still finds that A_7 achieves G_7, but now before applying A_7, it finds that A_4 achieves G_4. Then, noticing that A_7 deletes I_4, PRODIGY realizes that A_4 must be applied before A_7. In this way, PRODIGY avoids having to backtrack. In fact, in both of these domains, PRODIGY avoids having to backtrack at all, resulting in sub-quadratic performance in the number of goals (Figure 2). Thus, for D^mS^1 and D^1S^{1*} (both laboriously serializable according to Barrett and Weld), PRODIGY with either of the SABA heuristics performs well.

D^mS^{2*}: SABA-reactive fails

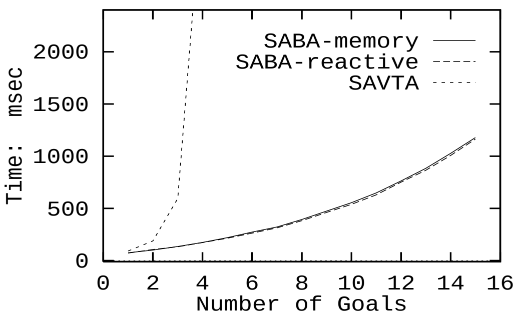
D^mS^{2*} is a domain in which SABA-memory clearly outperforms both SAVTA *and* SABA-reactive. As in D^1S^{1*}, G_* is always one goal of a problem. Here the operators look like:

Operator	preconds	adds	deletes
A_i^1	$\{I_i\}$	$\{P_i\}$	$\{P_j \mid j < i\}$
A_i^2	$\{P_i\}$	$\{G_i\}$	$\{P_j \mid j < i\}$
A_*	$\{I_*\}$	$\{G_*\}$	$\{I_i \mid \forall i\} \cup \{G_i \mid \forall i\}$

[5] D^mS^2 from (Barrett & Weld 1994) is another such domain. We ran the same experiments on this domain and achieved results similar to those for D^mS^1 and D^1S^{1*}.



(a)



(b)

Figure 2: PRODIGY's performance with different heuristics in domains D^mS^1 (a) and D^1S^{1*} (b)

Note that with two goals plus G_*, the optimal ordering of operators is A_2^1, A_1^1, A_*, A_1^2, A_2^2.

In this domain, SABA-memory is the best of our three heuristics. SAVTA performs poorly for the same reason as in the previous three domains. However here, SABA-reactive also performs poorly (Figure 3). The essential reason for SABA-reactive's failure is that after the first operator is applied, every applicable operator either deletes a precondition of another operator, or has an effect deleted by another operator. Furthermore, several of the applicable operators only have an effect deleted (thus giving them priority according to SABA-reactive's two conditions). Since several operators end up with equal priority, SABA-reactive ends up making arbitrary choices. SABA-memory rectifies this situation by analyzing more deeply the interactions among operators, and thus achieving the optimal order without backtracking. In particular, SABA-memory's use of past applicable-operator lists allows it to exhibit sub-quadratic behavior in the number of goals (Figure 3).

D^0-side-effect: The SABA's fail

In every domain to this point, SABA-reactive has performed at least as well as SAVTA, and SABA-memory has performed at least as well as SABA-reactive. How-

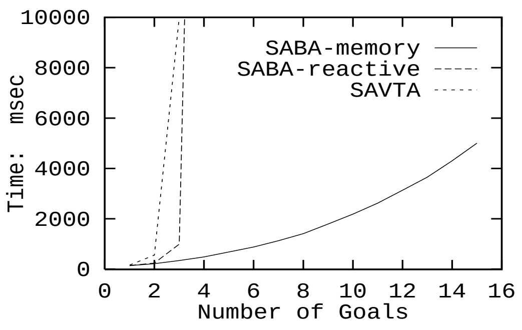


Figure 3: PRODIGY's performance with different heuristics in domain D^mS^{2*}

ever, we are not trying to argue that SABA-memory is a universally ideal heuristic for PRODIGY. On the contrary, we argue that there is no such thing as a universally ideal search heuristic for any planner: the best type of search depends at least on the domain, and probably on the problem as well. To this end, we now present a simple domain in which SAVTA outperforms both of the SABA heuristics. This domain represents the class of domains where one operator opportunistically adds several different goals.

D^0-side-effect is a domain in which it is advantageous to apply an operator as soon as possible. There is only one operator in this domain:

Operator	preconds	adds	deletes
A	{}	$\{G_i \mid 0 < i \leq 15\}$	{}

Here all the goals are achieved by this one operator.

Using the SABA heuristics in this domain, PRODIGY subgoals on each of its goals before it ever applies the operator. Then, it applies the operator once and is done. In this case, PRODIGY behaves linearly in the number of goals it needs to solve (Figure 4).

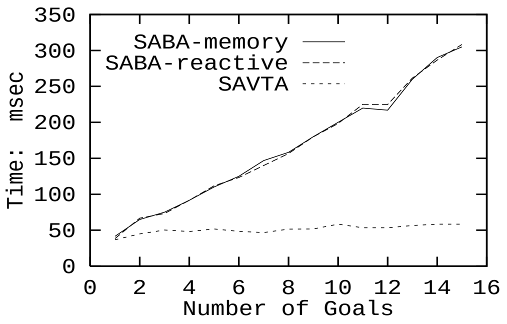


Figure 4: PRODIGY's performance with different heuristics in domain D^0-side-effect

Using the SAVTA heuristic, however, PRODIGY solves problems in this domain in constant time, i.e. independent of the number of goals (Figure 4). Since the same operator achieves all goals, PRODIGY now discovers that this operator achieves its first goal and immediately applies the operator, thus solving all its other goals as well.

Another class of domains where SAVTA outperforms the SABA heuristics is those in which there are several different operators that can achieve a goal. Here the SABA heuristics run into problems because PRODIGY chooses operators that are appropriate in the current state; however, the state may be different when PRODIGY tries to apply them. We have also created and tested such a domain, achieving similar results, where SAVTA outperforms the SABA heuristics.

Discussion

We have shown that different domain-independent search heuristics work best in different domains. In particular, SABA-memory works best in D^0S^1, D^mS^1, D^1S^{1*}, and D^mS^{2*}, while SAVTA is more appropriate in D^0-side-effect. The cyclical dominance relationships among the different heuristics supports our belief that no domain-independent search heuristic dominates others across all domains (Figure 5).

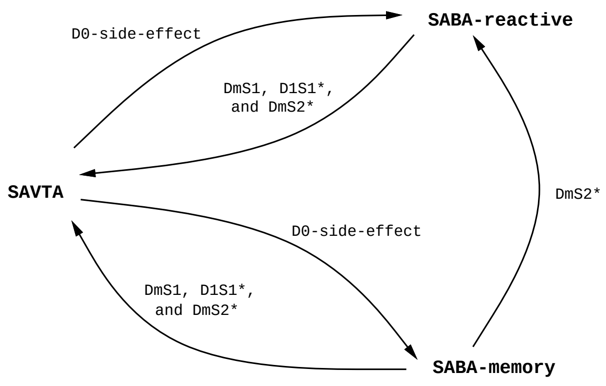


Figure 5: The cyclical dominance relationships of our heuristics. An arrow from x to y means that heuristic x performs better than heuristic y in the indicated domain(s).

Again, we are not trying to suggest that SABA-memory is a "better" heuristic than SAVTA. Instead, we believe we have evidence that no heuristic is appropriate for all problems and in all domains. Thus we are interested in trying to discover which domain-independent heuristics are best for which domains (for related work, see (Ginsberg & Geddis 1991; Minton 1993)). We would like to extend PRODIGY's learning strategies to include the ability to decide for itself which domain-independent search heuristic to use in which situation.

Our results suggest that the SABA heuristics work best for problems in which operators for different goals interfere with one another. However SAVTA works best for problems in which operators fortuitously achieve goals other than those for which they were chosen.

SAVTA is also more appropriate than the SABA heuristics for problems in which the correct choice of operator to achieve a goal depends on the state at the time it is going to be applied. Perhaps there is a better heuristic than SAVTA to handle cases such as these. And there are certainly more heuristics that work well in domains that we have not yet explored. In any case, we feel it is now clear that planners should have a library of different domain-independent search heuristics that they can use for different problems and for different domains.

PRODIGY and Other Planners

We briefly consider the relationship between PRODIGY and the three simple planners tested by Barrett and Weld on the same domains that we use in this paper (Barrett & Weld 1993; 1994). These are POCL, which maintains a partial order of applicable operators and uses "causal links" to maintain goal protections, TOCL, which also uses causal links but maintains applicable operators in a total order, and TOPI, which maintains applicable operators in a total order and restricts applicable operators to be added to the front of the list (the "PI" stands for "prior insertion"). Barrett and Weld do not investigate the effect of heuristics on any of these planners, but extend a relationship between planners and domains—serializability—to one that captures the behavior of the planners on these domains without heuristics.

In order to compare PRODIGY with these planners, we must point out an assumption made by Barrett and Weld that does not hold for PRODIGY: that partially ordering operators is synonymous with delaying commitment to operator ordering. These two capabilities occur together in their planners, since POCL has both and TOCL and TOPI have neither. PRODIGY, however, is able to delay commitment to ordering without using a partial order for operators. When it expands subgoals it makes no commitment to the order of the operators used to achieve them, making decisions about the total ordering of these operators when more information is available.

Like TOPI, PRODIGY adds operators to the beginning of the total order when it commits to the ordering of an operator. However it also uses causal links similar to those of POCL and TOCL in a backtracking strategy that significantly reduces the search space.

Barrett and Weld show experimentally that POCL can perform exponentially better than TOCL and TOPI in some domains and attribute this success to POCL's use of a partial order. Our results show that it may, in fact, have more to do with the ability to delay ordering commitment.

Conclusion

We have examined the performance of three search heuristics on a set of artificial domains. The results show that none of the heuristics leads to better performance in all the domains: no heuristic dominates the others. It is our belief that no simple search heuristic will provide good performance in every domain for a domain-independent planner. Rather than being a negative view, this leads us to continue looking for a better understanding of the mapping between domains and heuristics and for learning methods that can determine and make use of this mapping.

For example, features such as the ones discussed to explain the different performance of the heuristics could be measured relatively cheaply. Future domain-independent planners should therefore be able to make intelligent choices about which commitment strategy to use based on the domain. Learning will be necessary in part because the correct search technique may depend as much on the distribution of problems in the domain as on its structure. In any case, this paper establishes the clear need for different domain-independent search heuristics in general purpose planners.

References

Barrett, A., and Weld, D. S. 1993. Characterizing subgoal interactions for planning. In *Proceedings of IJCAI-93*, 1388–1393.

Barrett, A., and Weld, D. S. 1994. Partial-order planning:Evaluating possible efficiency gains. *Artificial Intelligence* 67(1).

Carbonell, J. G.; Knoblock, C. A.; and Minton, S. 1990. Prodigy: An integrated architecture for planning and learning. In VanLehn, K., ed., *Architectures for Intelligence*. Hillsdale, NJ: Erlbaum. Also Technical Report CMU-CS-89-189.

Ginsberg, M. L., and Geddis, D. F. 1991. Is there any need for domain-dependent control information? In *Proceedings of the Ninth National Conference on Artificial Intelligence.*

Minton, S. 1993. Integrating heuristics for constraint satisfaction problems: A case study. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 120–126.