In *Machine Learning Methods for Planning*, Morgan Kaufmann, 1993, ed. S. Minton, pages 233-272

CHAPTER 8

Towards Scaling Up Machine Learning: A Case Study with Derivational Analogy in PRODIGY

MANUELA M. VELOSO
JAIME G. CARBONELL

1. Motivation: Why, What and How

Machine learning has proven itself in the small, although theoretical, algorithmic and implementational advances at the foundational level will continue to improve the basic building blocks in the field. Empirical induction methods have been developed [Michalski et al., 1983, Michalski et al., 1986, Michalski and Kodratoff, 1990, Carbonell, 1990] at the symbolic level and tested on standard (albeit small) test suites, [1] and occasionally they have been used externally, as in the case of decision tree induction [Quinlan, 1983, Quinlan, 1986, Núñez, 1991]. Subsymbolic induction methods, including genetic algorithms [Holland, 1986, DeJong, 1989] and connectionist approaches [Hinton, 1989, Minsky and Selfridge, 1961, Touretzky, 1989] are also well developed. Analytical generalization methods are proving successful in improving performance in a variety of planning and other reasoning tasks. These methods include macro-operator formation [Korf, 1985, Iba, 1989, Minton, 1985, Cheng and Carbonell, 1986, Shell and Carbonell, 1989], explanation-based learning [Mitchell et al., 1986, DeJong and Mooney, 1986, Minton et al., 1989a], abstraction [Sacerdoti, 1977, Korf, 1987, Knoblock, 1991] and within-domain analogy [Carbonell, 1983, Carbonell, 1986, Veloso and Carbonell, 1989, Hickman et al., 1990].

The time has come to address machine learning in the large, including both for inductive concept acquisition [Catlett, 1991, Quinlan, 1987] and analytic performance improvements. In this chapter,

1. The University of California at Irvine maintains informally a varied set of training and test data for inductive generalization deposited and accessible by researchers in machine learning.

we address the latter in the context of PRODIGY [Carbonell et al., 1990, Minton et al., 1989b, Veloso, 1989], a general-purpose complete planner that incorporates various learning techniques: explanation-based learning (EBL) [Minton, 1988], acquisition of control knowledge through static analysis [Etzioni, 1990], learning by analogy [Veloso, 1991], learning by experimentation [Carbonell and Gil, 1990], learning abstraction hierarchies for effective planning [Knoblock, 1991], and semiautomated knowledge acquisition interfaces [Joseph, 1989]. These techniques have been developed and tested in a variety of small and medium domains, and all exhibit varying degrees of improved performance. Addressing large-scale problems, however, requires several types of analyses and extensions:

- Measuring the performance improvements produced by these techniques as functions of domain size and complexity. Ideally, if the learning system undergoes adequate training in increasingly complex problems, performance should improve with domain size, rather than be just a constant factor.
- Measuring the learning-time and run-time overhead of acquiring and using the new knowledge with increasing complexity. At worst, the overhead should remain a constant fraction of overall problem solving, and at best it should be a diminishing fraction with increased domain size; otherwise the utility problem [Minton, 1988] will prove a serious hindrance.
- The synergistic combination of multiple learning techniques producing far more performance improvements than individual learning techniques, without paying a correspondingly large overhead cost [Knoblock et al., 1991]. Measurements of performance with different subsets of the learning techniques employed should shed light on their synergistic utility as domain size grows.

This chapter focuses primarily on a single learning technique, derivational analogy [Carbonell, 1986, Carbonell and Veloso, 1988], in PRODIGY. However, some discussion at the end of the chapter addresses the first steps towards synergistic multitechnique integration. Derivational analogy is a generalized form of case-based reasoning [Hammond, 1986, Kolodner, 1984, Riesbeck and Schank, 1989, Schank, 1982, Simpson, 1985, Sycara, 1987] that incorporates case generation from problem-solving experience, case organization into an indexed long-term case

library, case retrieval for problem solving, case replay during problem solving, and feedback to memory on the utility of the retrieved cases. These stages of derivational analogy are reviewed with examples from a version of the STRIPS domain [Fikes and Nilsson, 1971]. We show results illustrating different degrees of performance improvement. We address the scaling up characteristics of the derivational analogy framework, and we discuss how the framework is applied to a complex logistics/transportation domain for which we are building a 1000-case library. The chapter concludes with a discussion of multi-technique synergy in PRODIGY to address progressively larger domains.

2. Derivational Analogy: The Basic Method

PRODIGY's derivational analogy reasoner is a learning method for incorporating and reusing past experience. Analogical reasoning can be seen as an alternative to the explanation-based learning (EBL) paradigm. EBL generalizes control rules from an example trace of a solved problem and domain axioms. It proves the correctness of decisions at choice points and synthesizes the control rules from these proofs. Hence, EBL invests substantial effort in deriving general rules for behavior from each example. The analogical reasoner automatically generates and stores annotated traces of solved problems (cases) that are elaborated further when and if needed to guide reasoning in similar future problems. Compiled experience is therefore stored with little additional processing. The explanation effort is done incrementally on an "if needed" basis at storage, retrieval and adaptation time when new similar situations occur. Thus, EBL and analogy are at opposite points in the eager-versus-lazy evaluation spectrum. In principle, their combination may prove optimal, acquiring simpler control knowledge for repetitive situations by means of EBL, and reserving analogy for more complex but less frequent decisions.

Analogical reasoning in PRODIGY integrates automatic case generation, case storage, case retrieval, case replay, and general problem solving – exploiting and modifying experience when available and resorting to general problem-solving methods when it is required. Automatic case generation occurs by extending the general problem solver with limited ability to examine its internal decision cycle, recording the justifications for each decision during its extensive search process. Examples of these justifications are links between choices that capture the subgoal-

ing structure, records of explored failed alternatives, and pointers to applied control knowledge. A stored problem-solving episode consists of the solution trace augmented with these annotations.

Past problems that match partially the new problem solving situation are its candidate analogs. To rank these partially similar candidate analogs, a similarity metric is used to explore the relevance of past features identified using that were identified using goal regression in the successful derivational trace. The retrieval module returns to the analogical reasoner a set of past cases that should adequately cover significant portions of the new problem-solving situation.

The analogical reasoner's main functionality consists of a sophisticated replay mechanism that is able to reconstruct solutions from the retrieved past cases when only a partial match exists between the new and past situations. The replay mechanism coordinates the set of multiple retrieved cases and uses the annotated justifications to guide reconstruction of the solution for use in problem-solving situations in which equivalent justifications hold true.

Learning, therefore, occurs by accumulation and reuse of cases, especially in situations that required extensive problem solving, and by tuning the memory model's indexing structure to retrieve progressively more appropriate cases. On one hand, we reduce search at the problem-solving level by replaying past similar cases (derivational traces of problem-solving episodes) [Veloso, 1991]. On the other hand, we learn incrementally better similarity metrics by interpreting the behavior of the problem solver replaying retrieved cases. Furthermore we explore an efficient way of balancing the costs of retrieval and search [Veloso and Carbonell, 1991], as discussed in section 7.

We present empirical results that illustrate the performance of the replay mechanism in two domains using two different similarity metrics. Recent tests with up to 1000 cases in the library demonstrated the scaling properties of the memory organization, of the match/retrieval process, and of the reconstruction mechanism replaying multiple cases. We also show preliminary empirical results in this large domain.

3. The Derivational Trace: Case Generation

Derivational analogy is a *reconstructive* method by which *lines of reasoning* are transferred and adapted to a new problem [Carbonell, 1986].

The ability to replay previous solutions using the derivational analogy method requires that the problem solver be able to examine its internal decision-making cycle, recording the justifications for each decision during its extensive search process. These justifications augment the solution trace and are used to guide future reconstruction of the solution for use in subsequent problem-solving situations in which equivalent justifications hold true.

In PRODIGY [Minton et al., 1989b] a domain is specified as a set of operators, inference rules, and control rules. Additionally, in a more recent version of the system [Veloso, 1989, Carbonell et al., 1992], the entities of the domain are organized in a class hierarchy. Each operator (or inference rule) has a precondition expression that must be satisfied before the operator can be applied, and an effects-list that describes how the application of the operator changes the world. Search control in PRODIGY allows the problem solver to represent and use control information about the various problem-solving decisions. A problem consists of an initial state and a goal expression. To solve a problem, PRODIGY must find a sequence of operators that, if applied to the initial state, produces a final state that satisfies the goal statement. The problem solver 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, 1991], 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].

The derivational analogy work in PRODIGY takes place in the context of PRODIGY's *nonlinear* problem solver [Veloso, 1989]. The planning system is called NOLIMIT, standing for *No*n*li*near problem solver using casual com*mit*ment. NOLIMIT can fully interleave goals at the different search levels. The plans generated are nonlinear because they cannot be decomposed into a linear sequence of complete subplans for interacting conjunctive goals.

The basic search procedure is, as with the linear planner [Minton et al., 1989b], means-ends analysis (MEA) in backward-chaining mode. Basically, given a set of goal literals not true in the current world, the

planner selects one goal and an operator that adds that goal to the state in the case of a positive goal, or deletes it in the case of a negative goal. We say that this operator is *relevant* to the given goal. If the preconditions of the chosen operator are true, the operator can be *applied*. If the preconditions are not true in the state, then they become *subgoals*, that is new goals to be achieved. The cycle repeats until all the conjuncts from the goal expression are true in the world. NOLIMIT's nonlinear character stems from working with a *set* of goals in this cycle, as opposed to the top goal in a goal stack. Dynamic goal selection enables NOLIMIT to interleave plans, exploiting common subgoals and addressing plan interactions, such as issues of resource contention.

To generate a derivational trace from a problem-solving episode, the problem solver must identify and capture the reasons for the decisions taken at the different choice points encountered during the search for a solution. In NOLIMIT's search procedure, we identify the following types of choice points [Veloso, 1989]:

- What *goal* is to become a subgoal, choosing it from the set of pending goals
- What *operator* to choose to pursue the particular goal selected
- What *bindings* to choose to instantiate the selected operator
- Whether to *apply* an applicable operator or defer application and continue *subgoaling* on a pending goal
- Whether the search path being explored should be *suspended*, continued, or abandoned
- On failure, which *past choice point* to backtrack to, or which *suspended path* to reconsider for further search.

Justifications at these choice points may point to user-given guidance, preprogrammed control knowledge, automatically learned control rules responsible for decisions that have been made, or past cases used for guidance (more than one case can be used to solve a complete problem). They also represent links within the different choices and their related generators, in particular capturing the subgoaling structure. At choice points, we also record failed alternatives (explored earlier) and the causes of their failures. Note that the term *cause of failure* here

refers to the reason why the search path starting at that alternative failed. It does not necessarily mean that the failed alternative is directly responsible for the failure of the global search path. There may be an indirect relationship, but this is the best attribution so far. We now present an example to illustrate the automatic generation of an annotated case.

The extended-STRIPS domain [Minton, 1988] consists of a set of rooms connected by doors. A robot can move among the rooms carrying or pushing objects. The doors can be locked or unlocked. The keys to the doors are in the rooms and can be picked up by the robot. The set of operators include moving to be next to objects, going through doors, pushing objects to rooms, picking up keys, and opening, closing, locking, and unlocking doors. (Our version of the domain includes 13 operators and 4 inference rules.) Variables and instances are organized in the class hierarchy shown in Figure 1 where CLASS is the top of the hierarchy.

```
(is-a ROOM CLASS)
(is-a DOOR CLASS)
(is-a OBJECT CLASS)
(is-a BOX OBJECT)
(is-a KEY OBJECT)
(is-a AGENT CLASS)
```

Figure 1. The class hierarchy in the extended-STRIPS domain.

For example, the operator to push an object through a door is shown in Figure 2; [2] variables are in brackets and types are written in uppercase letters.

Consider Figure 3 in which we show the initial state in (a) and in (b) the goal statement of an example problem from the extended-STRIPS domain, say problem *strips2-5*. The rooms are numbered at their corners and the doors are named accordingly to the rooms they connect. Doors may be open, closed, or locked. In particular, door24 connects the rooms 2 and 4 and is locked. Door34 is closed and, for example, door12 is open. The number of the boxes can be inferred by the attached description of the initial state. Note that box3 is in room4. The problem solver must find a plan to reach a state in which door34, connecting room3 and

2. The complete set of operators for the nonlinear planner is obtained directly from the set of operators for the linear planner of PRODIGY in [Minton, 1988] by applying some fixed syntax modifications [Veloso, 1989].

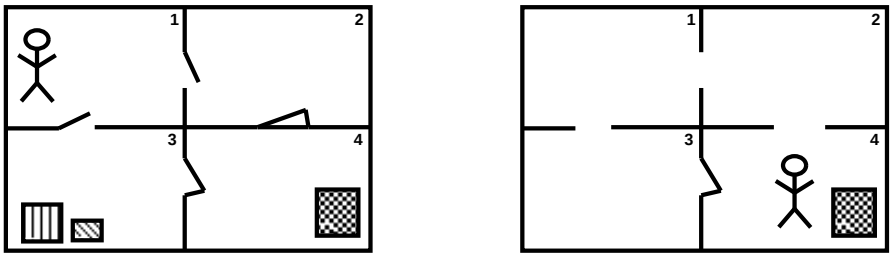
```
(OPERATOR PUSH-THRU-DOOR
 (params
   ((<box>
     (and BOX (pushable <box>)))
    (<roomx>
     (and ROOM (adjacent-room-p <roomx> <roomy>)))
    (<roomy>
     (and ROOM (adjacent-room-p <roomx> <roomy>)))
    (<door>
     (and DOOR (connects <door> <roomx> <roomy>)))))
 (preconds
   (and (dr-open <door>)
        (inroom robot <roomx>)
        (next-to robot <box>)
        (next-to <box> <door>)))
 (effects
  ((del (inroom robot <roomx>))
   (del (inroom <box> <roomx>))
   (add (inroom robot <roomy>))
   (add (inroom <box> <roomy>))
   (if ((<obj> OBJECT))
       (holding <obj>)
       ((del (holding <obj>))
        (add (inroom <obj> <roomx>))
        (add (next-to <obj> <door>)))))))
```

Figure 2. The operator to push an object through a door in the extended-STRIPS domain.

room4, is closed, and the agent "hero" is next to box3. The problem is simple, so we can show a full case corresponding to a problem-solving search episode.

Without any analogical guidance (or other form of control knowledge), the problem solver searches for a solution by applying its primitive means-ends analysis procedure. In Figure 4 we show a complete successful solution path. Goals are in parenthesis, chosen operators are in angle brackets ($<,>$), and applied operators are in uppercase letters. For simplicity, we represent the sequence of decision nodes annotated only with their subgoaling structure.

NOLIMIT starts working on the goal `(next-to robot box3)` at node `cn1`, since door34 is closed in the initial state. The relevant operator for this goal is `<goto-box box3>` as shown in node `cn2`. This operator is not applicable, since one of its preconditions, namely that the agent be in the same room as the box, is not true in the state. Therefore, at node `cn3`, it subgoals on getting the robot into room4. Now, note that both room2 and room3 are adjacent to room4. By backward chaining, NOLIMIT finds these two alternatives as relevant operators




```
(inroom robot room1)   (inroom box1 room1)    (and (door-closed door34)
(inroom box2 room3)    (inroom box3 room4)         (next-to robot box3))
(door-open door12)     (door-open door13)
(door-closed door34)   (door-locked door24)
```

(a) Initial State *(b) Goal Statement*

Figure 3. problem situation in the extended-STRIPS domain. The goal statement is a partial specification of the final desired state: The location of other objects and the statuses of other doors remain unspecified.

Node type and number	*:choice*	*:precond of*	*:relevant to*
goal cn1	(next-to robot box3)	user	
chosen-op cn2	<goto-box box3>		cn1
goal cn3	(inroom robot room4)	cn2	
chosen-op cn4	<go-thru dr34>		cn3
goal cn5	(inroom robot room3)	cn4	
chosen-op cn6	<go-thru dr13>		cn5
applied-op cn7	<GO-THRU dr13>		
goal cn8	(door-open dr34)	cn4	
chosen-op cn9	<open-door dr34>		cn8
applied-op cn10	<OPEN-DOOR dr34>		
applied-op cn11	<GO-THRU dr34>		
goal cn12	(door-closed dr34)	user	
chosen-op cn13	<close-door dr34>		cn12
applied-op cn14	<CLOSE-DOOR dr34>		
applied-op cn15	<GOTO-BOX box3>		

Figure 4. A simplified case corresponding to a solution to the problem in Figure 3. A case is an annotated successful problem-solving episode.

to the goal `(inroom robot room4)`, namely the operators `<go-thru door34>`, shown as node `cn4`, or `<go-thru door24>`. In Figure 5 we see the complete annotated decision node `cn4` for the situation where NOLIMIT searched the alternative `<go-thru door24>` before pursuing the successful operator `<go-thru door34>`. Note that door24 is locked,

and there is no key for it in the initial state. In the search episode, this failure corresponds to a subtree off the finally successful node `cn4`. However, the analogical reasoner creates a case by annotating the successful path with its sibling failed alternatives (and other justifications as fully introduced in [Veloso and Carbonell, 1993].) It simply attributes the reason of a failure to the last failed leaf of the searched subtree, but it also records the other failed leaves.

```
Frame of class chosen-op decision node cn4
 :choice (go-thru dr34)
 :sibling-relevant-ops
   (((go-thru dr24)
      (:no-relevant-ops (is-key dr24 <key>))))
 :why-this-operator nil
 :relevant-to (inroom robot room4)
```

Figure 5. A chosen operator decision node with its justifications: Zoom of the decision node cn4 in Figure 4.

NOLIMIT pursues its search as shown in Figure 4. It alternates choosing the relevant operator for each goal and applying it if all its preconditions are true in the state, or it continues subgoaling on a goal of the new goal set.

Node `cn12` is also worth remarking about, and we show its expansion in Figure 6. At that search point, NOLIMIT has the alternative of immediately applying the operator `(goto-box box3)`, since it becomes applicable as soon as robot enters room4 at node `cn11`, or subgoaling in the goal `(door-closed dr34)` that became a goal when door34 was open at node `cn10`. Because NOLIMIT is a nonlinear planner with the ability to fully interleave all the decisions at any search depth [Veloso, 1989, Rosenbloom et al., 1990], it finds the optimal plan to solve this problem. Note that in Figure 6 we show the problem-solving search situation in which NOLIMIT explores first the eager choice of applying any applicable operator, namely the sibling-applicable-op `<GOTO-BOX box3>`. This ordering, however, leads to a failure: When returning back to close door34, after achieving `(next-to robot box3)`, NOLIMIT encounters a state loop. It recognizes that it was in the same state before and backtracks to the correct ordering, postponing application of the operator `<GOTO-BOX box3>`, at node `cn15`, until the goal `(door-closed dr34)` is accomplished.

Without guidance NOLIMIT explores the space of all possible attention foci and orderings of alternatives, and only after backtracking does

```
Frame of class goal decision node cn12
 :choice (door-closed dr34)
 :sibling-goals nil
 :sibling-applicable-ops
  (((GOTO-BOX box3) (:state-loop)))
 :why-subgoal nil
 :why-this-goal nil
 :precond-of (*finish*)
```

Figure 6. A goal decision node with its justifications: Zoom of the decision node cn12 in Figure 4.

it find the correct goal interleaving. The idea of compiling problem-solving episodes is to learn from its earlier exploration and reduce search significantly by replaying the same reasoning process in similar situations.

The generated case corresponds to the search tree compacted into the successful path annotated with the justifications that resulted in the sequence of correct decisions that led to a solution to the problem. In the next section we introduce a global view of the organization of the accumulated library of cases.

4. Organization of the Case Library - Indexing

The goal statement and the initial state define a problem and act as its immediate indexes. In order to efficiently prune the case library, we index the cases in a three-level access structure.

At the top level of access, a hash table, named CASE-LIBRARY, associates each generalized goal with the list of problems that have corresponding instantiated literal as conjuncts in their goal statements.

As an example consider the problem *strips2-5* shown in Figure 3 with goal statement `(and (door-closed door34) (next-to robot box3))`. Consider, too, the problem *strips2-17* shown in Figure 12 with goal statement `(and (inroom box1 room2) (door-open door34))`. For the purpose of better illustration, consider an additional problem *strips3-9* with goal statement `(and (next-to robot box4) (inroom box4 room1) (holding key13))`. In Figure 7 (a) we show the relevant entries of the CASE-LIBRARY after *strips2-5*, *strips2-17*, and *strips3-9* are stored into memory. For example, the value of the hash key (inroom BOX ROOM) is the list (*strips2-17 strips3-9*), because these two problems have corresponding instantiated literals in their goal statements, (in-

room box1 room2) and (inroom box4 room1), respectively.

Top *CASE-LIBRARY* hash table
****

Goal-generalized-to-class	Problems
(door-closed DOOR)	(strips2-5)
(next-to AGENT BOX)	(strips2-5 strips3-9)
(inroom BOX ROOM)	(strips2-17 strips3-9)
(door-open DOOR)	(strips2-17)
(holding KEY)	(strips3-9)

(a)

Top *STATE-NET-NAMES* hash table
****

Sorted-variable-goal	State-net-names
(and (door-closed <door0>) (next-to <agent0> <box0>))	"state-net-1"
(and (door-open <door0>) (inroom <box0> <room0>))	"state-net-2"
(and (holding <key0>) (inroom <box0> <room1>) (next-to <agent0> <box0>))	"state-net-3"

(b))

Figure 7. (a) The hash table *CASE-LIBRARY*: The hash key is the generalized literal and the value is the list of problems that have a corresponding instantiated literal in their goal statement. (b) The hash table *STATE-NET-NAMES*: It associates the sorted generalized goal conjunct with the pointer to the discrimination network where the initial state is stored.

At the second level of indexing, a hash table STATE-NET-NAMES stores each conjunctive variable goal with the name of the discrimination network where the initial state is stored. Each instance in an argument of a goal conjunct is translated into a variable and the goal conjunct is sorted alphabetically. In Figure 7 (b) we show the relevant entries of

the STATE-NET-NAMES after *strips2-5* and *strips2-17* are stored into memory.

The goal statement filters the candidate past problems through the first and second levels of access.

For exactly the same goal conjunct there may be several different problems with different initial states. The initial state is itself stored in a discrimination net that corresponds to the third level of access. Each network has a root frame of class "state-root" as shown in Figure 8 that summarizes the contents of the network. The nodes of the network are frames of class "state" also shown in Figure 8. Their content is a set of literals in the initial state.

```
(def-frame state-root (:is-a tofu)
  :prob-names nil    ;list of problem names stored
                     ;in the state discrimination net
  :goal nil          ;goal conjunct of state-root
  :case-names nil    ;list of all the cases in net
  :children nil      ;points to a state frame,
    ;though the root of state discrimination net has
    ;only one child node, the slot is named children,
    ;instead of child, to be uniform with the state frame.
  )

(def-frame state (:is-a tofu)
  :content nil       ;list of state-goal literals
  :parent nil        ;state-root frame or state frame
  :children nil      ;list of state frames or nil
  :cases nil         ;list of leaf cases
  )
```

Figure 8. The frame structure of the nodes of the discrimination net for the initial state.

The purpose of the dynamic organization of the discrimination state network is to learn the degree of relevance of the literals in the initial state as a function of the common goal they address. The structure of the network should be such that the literals closer to the root are more relevant than the ones at the leaves.

Each leaf of the network points to one or more cases (derivational analogy traces) whose relevant initial state is the set of literals in the path from the corresponding leaf up to the root of the network [Kolodner, 1983]. We show a simple example though the discrimination network is maintained dynamically. For the sake of illustration in Figure 9 we show the contents of "state-net-2" where two different solutions for problem *strips2-17* (see Figures 12, 13 (a1), and 13 (b1)) are stored.

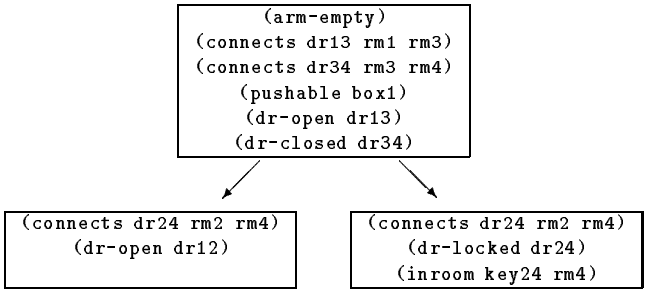


Figure 9. A simple discrimination tree for the initial state.

Later on we explain the meaning of the contents of the state nodes in the context of *foot-printing* the initial state. This example should only be considered from the organizational point of view, and after reading section 5, the reader can return to this example and comprehensively follow its contents.

In the next section we describe the retrieval algorithm and mention how it makes use of the memory organization. The complete algorithm for the dynamic memory update is still under development as we are still analyzing results of its performance in scaled-up domains.

5. Retrieving Similar Past Cases for Guidance

Several research projects have studied the problem of assigning adequate similarity metrics (recent work includes [Bareiss and King, 1989, Kolodner, 1989, Porter et al., 1989]). Our approach relies on an incremental understanding of an increasingly more appropriate similarity metric. In [Veloso and Carbonell, 1990] we presented our proposed memory model, SMART (standing for *S*torage in *M*emory and *A*daptive *R*etrieval over *T*ime). NOLIMIT, the nonlinear analogical problem solver, provides to SMART information about the utility of the candidate cases suggested as similar in reaching a solution. This information is used to refine the case library organization and in particular the similarity metric. In this section we analyze two similarity metrics with different degrees of problem-context sensitivity. We first introduce a simple direct similarity metric and proceed to refine it by analyzing the derivational trace produced by the analogical problem solver.

5.1 A direct similarity metric

Let $\mathcal{S}$ be the initial state and $\mathcal{G}$ be the goal statement, both given as conjunctions of literals. A *literal* is an instantiated predicate, i.e. literal = (predicate argument-value*). As an example, (inroom key12 room1) is a literal where inroom is the predicate and key12 and room1 are its instantiated arguments.

Each past case P in memory is indexed by the corresponding initial state and goal statement, respectively $\mathcal{S}^P$ and $\mathcal{G}^P$. When a new problem P' is given to the system in terms of its $\mathcal{G}^{P'}$ and $\mathcal{S}^{P'}$, retrieving one (or more) analog consists in finding a *similar* past case by comparing these two inputs $\mathcal{G}^{P'}$ and $\mathcal{S}^{P'}$ to the indexes of past cases.

Definition 1 *We say that a conjunction of literals $L = l_1, \ldots, l_n$ directly matches a conjunction of literals $L' = l'_1, \ldots, l'_m$ under a substitution σ with match value $\boldsymbol{\delta}$, if there are $\boldsymbol{\delta}$ many literals in L that directly match some literals in L' under σ. A literal l directly matches a literal l', iff*

- *The predicate of l is the same as the predicate of l'.*
- *Each argument of l is of the same class as its corresponding argument of l'.*

In this case, there is a substitution σ, such that $l = \sigma(l')$.

As an example, the literal (inroom box1 room1) *directly matches* the literal (inroom boxA roomX), where box1 and boxA are both of class BOX and room1 and roomX are of class ROOM. Under the substitution σ ={box1/boxA, room1/roomX}, (inroom box1 room1) = σ ((inroom boxA roomX)).

We first compute a simple partial match value between problems as the sum of the match value of their corresponding initial states and goal statements calculated independently, as presented in definition 2.

Definition 2 *Let P and P' be two particular problems, respectively with initial states $\mathcal{S}^P$ and $\mathcal{S}^{P'}$, and goal $\mathcal{G}^P$ and $\mathcal{G}^{P'}$. Let $\delta_{\mathcal{G}}^{\sigma(P),P'}$ be the match value of $\mathcal{G}^P$ and $\mathcal{G}^{P'}$, under substitution σ. Let $\delta_{\mathcal{S}}^{\sigma(P),P'}$ be the match value of $\mathcal{S}^P$ and $\mathcal{S}^{P'}$, under the substitution σ. Then we say that the two problems P and P' directly match with match value $\boldsymbol{\delta}^{\sigma(P),P'} = \boldsymbol{\delta}_{\mathcal{G}}^{\sigma(P),P'} + \boldsymbol{\delta}_{\mathcal{S}}^{\sigma(P),P'}$ under substitution σ.*

The partial match value of two problems as expected is substitution dependent. As an example, consider the goal $\mathcal{G}$ ={(inroom key12 room1), (inroom box1 room1)}, and the goal $\mathcal{G}'$ ={(inroom key13 room4), (inroom key14 room2), (inroom box53 room4)}. Then $\mathcal{G}$ directly matches $\mathcal{G}'$ *with match value* $\delta = 2$ under the substitution σ ={key12/key13, room1/room4, box1/box53}, and *match value* $\delta = 1$ under the substitution σ' ={key12/key14, room1/room2}.

In a first experiment we used this direct similarity metric to evaluate the partial match between problems, not considering therefore any relevant correlations between the initial states and the goal statements. The procedure in Figure 10 retrieves the set of *most similar* past cases. Each goal conjunct is generalized to its class arguments. By hashing each goal conjunct in the CASE-LIBRARY (see section 4) we retrieve directly all and only the past problems that share at least one goal conjunct with the current new problem. This set of problems is the input for the procedure shown in Figure 10.

Input: A case library $\mathcal{L} = \mathcal{P}_1, \ldots, \mathcal{P}_p$, and a new problem $\mathcal{P}'$.
Output: The set of problems from $\mathcal{L}$ and corresponding substitutions that make them the most similar ones to $\mathcal{P}'$.

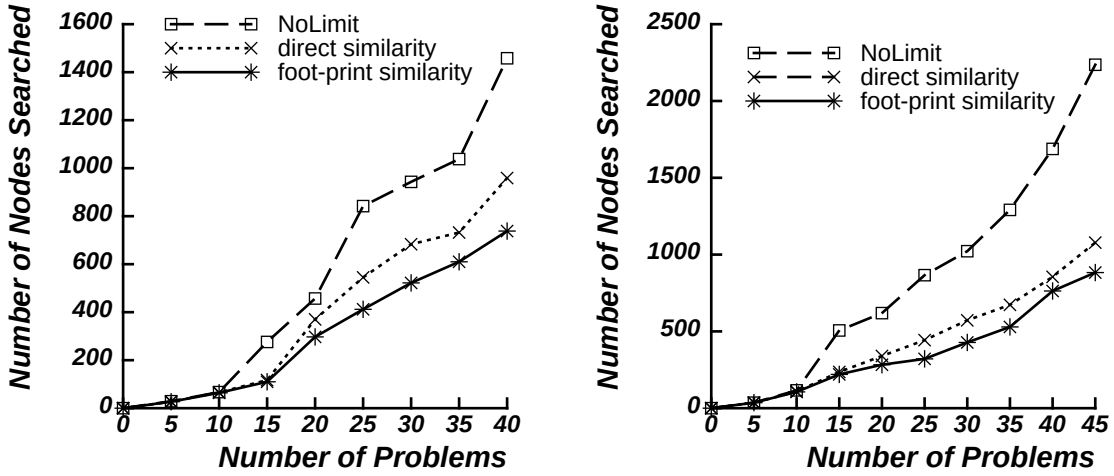
procedure Retrieve_Most_Similar_Past_Cases($\mathcal{L}$, $\mathcal{P}'$):
1. **current_max_match** $\leftarrow 0$
2. **Most_Similar** $\leftarrow \{\}$
3. **for** $i \leftarrow 1$ to p **do**
4. Compute $\delta_{\mathcal{G}}^{P_i,P'}$ for all the possible goal substitutions.
5. **for** each substitution σ such that $\delta_{\mathcal{G}}^{\sigma(P_i),P'} \neq 0$ **do**
6. Apply substitution to the initial state $\mathcal{S}_i$.
7. Compute $\delta_{\mathcal{S}}^{\sigma(P_i),P'}$.
8. $\delta^{\sigma(P_i),P'} \leftarrow \delta_{\mathcal{G}}^{\sigma(P_i),P'} + \delta_{\mathcal{S}}^{\sigma(P_i),P'}$
9. **if** $\delta^{\sigma(P_i),P'} >$ **current_max_match**
10. **current_max_match** $\leftarrow \delta^{\sigma(P_i),P'}$
11. **Most_Similar** $\leftarrow \{(P_i, \sigma)\}$
12. **if** $\delta^{\sigma(P_i),P'} =$ **current_max_match**
13. Add (P_i, σ) to **Most_Similar**.
14. Return **Most_Similar**

Figure 10. Retrieving the most similar past cases.

5.1.1 EXAMPLES IN PROCESS-JOB PLANNING AND EXTENDED-STRIPS DOMAINS

We ran NOLIMIT without analogy over a set of problems in the process-job planning and in the extended-STRIPS domains. [3] We accumulated a library of cases. In order to factor away other issues in memory organization, the case library was simply organized as a linear list of cases. We then ran again a new set of problems using the case library.

The dotted curves in Figures 11 (a) and (b) show the results for these two domains. We plotted the average cumulative number of nodes searched. We note from the results that analogy showed an improvement over basic blind search (dashed curves): a factor of 1.5 speed up for the process-job planning and scheduling domain and 2.0 speed for the extended-STRIPS domain. (We will see later the meaning of the filled curves.) In general the direct similarity metric lead to acceptable results. Allen and Langley [Allen and Langley, 1990] obtained similar results in simple domains by replaying one-step past cases as opposed to a complete sequence of problem-solving decisions.



(a) Process-Job Planning Domain (b) Extended-STRIPS Domain

Figure 11. Comparison in the process-job planning and extended-STRIPS domains.

However, analyzing the results, we notice that the straightforward similarity metric does not always provide the best guidance when there

3. This set is a sampled subset of the original set used by [Minton, 1988].

are several conjuncts in the goal statement.

The problem of matching conjunctive goals turns out to be rather complex. As conjunctive goals may interact, it is not at all clear to decide that problems are more similar based simply on the *number* of literals that match the initial state and the goal statements. Noticing therefore that matching conjunctive goals involves reasoning over a large lattice of situations, we developed a new similarity metric by refining the indexing based on the derivational trace of a past solution.

5.2 The foot-print similarity metric

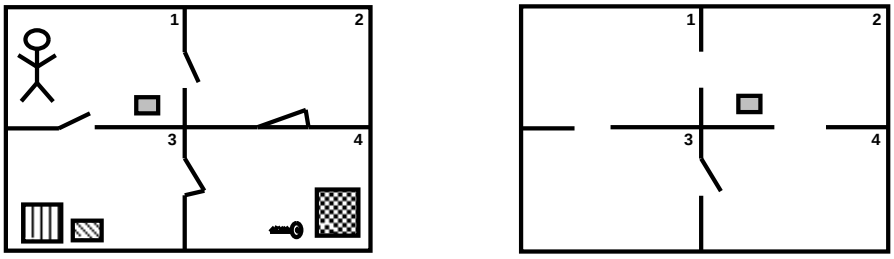
The derivational trace identifies for each goal the set of *weakest preconditions* necessary to achieve that goal. Then recursively we create the *foot-print* of a user-given goal conjunct by doing a goal regression, i.e. projecting back its weakest preconditions into the literals in the initial state [Waldinger, 1981, Mitchell et al., 1986]. The literals in the initial state are therefore *categorized* according to the goal conjunct that employed them in its solution. Goal regression acts as an explanation of the successful path [Cain et al., 1991].

Definition 3 *For a given problem P and corresponding solution, a literal s in the initial state is in the* **foot-print** *of a goal conjunct g if it is in the set of weakest preconditions of g according to the derivational trace of the solution.*

The purpose of retrieving a similar past case is to provide a problem solving episode to be replayed for the construction of the solution to a new problem. We capture into the similarity metric the role of the initial state in terms of the different goal conjuncts for a particular solution found. Situation details are not similar per se. They are similar as a function of their relevance in the solution encountered.

Consider Figure 12 where in (a) we show the initial state and in (b) the goal statement of an example problem from the extended-STRIPS domain, say problem *strips2-17*.

Assume we solved the problem in Figure 12 by pushing box1 from room1 into room2, and then going to room3 back through room1 to open the door dr34. The actual solution searched and found would be the plan shown in Figure 13 (a1).




```
(inroom robot room1)   (inroom box1 room1)    (and (inroom box1 room2)
(inroom box2 room3)    (inroom box3 room3)          (door-open door34))
(inroom box4 room4)    (inroom key24 room4)
(door-open door12)     (door-open door13)
(door-closed door34)   (door-locked door24)
```

(a) Initial State *(b) Goal Statement*

Figure 12. Problem situation in the extended-STRIPS domain (strips2-17). The goal statement is a partial specification of the final desired state: the location of other objects and the status of other doors remains unspecified.

In this way of solving the problem, for example, key24 for the locked door dr24 did not play any role and is therefore not a *relevant* literal in the initial state of this problem if this problem-solving episode is to be replayed. In Figure 13 (a2) we show the actual foot-print of the initial state corresponding to this first solution to the problem. Each literal in the initial state is associated with the list of goals that it contributed to achieve.

However NOLIMIT could have encountered a different solution to this problem, namely to push box1 along on its way to door dr34, open it, and push box1 through door dr24 into room2, after unlocking this door. The actual solution searched and found would be the plan shown in Figure 13 (b1). In this way of solving the problem, for example, the key24 for the locked door dr24 is a *relevant* literal in the initial state of this problem if this problem-solving episode is to be replayed. In Figure 13 (b2) we show the actual foot-print of the initial state for this solution.

We formally define the new similarity metric that evaluates the degree (or value) of match of the initial state as a function of the goal conjuncts that directly matched. This similarity emphasizes even more the goal

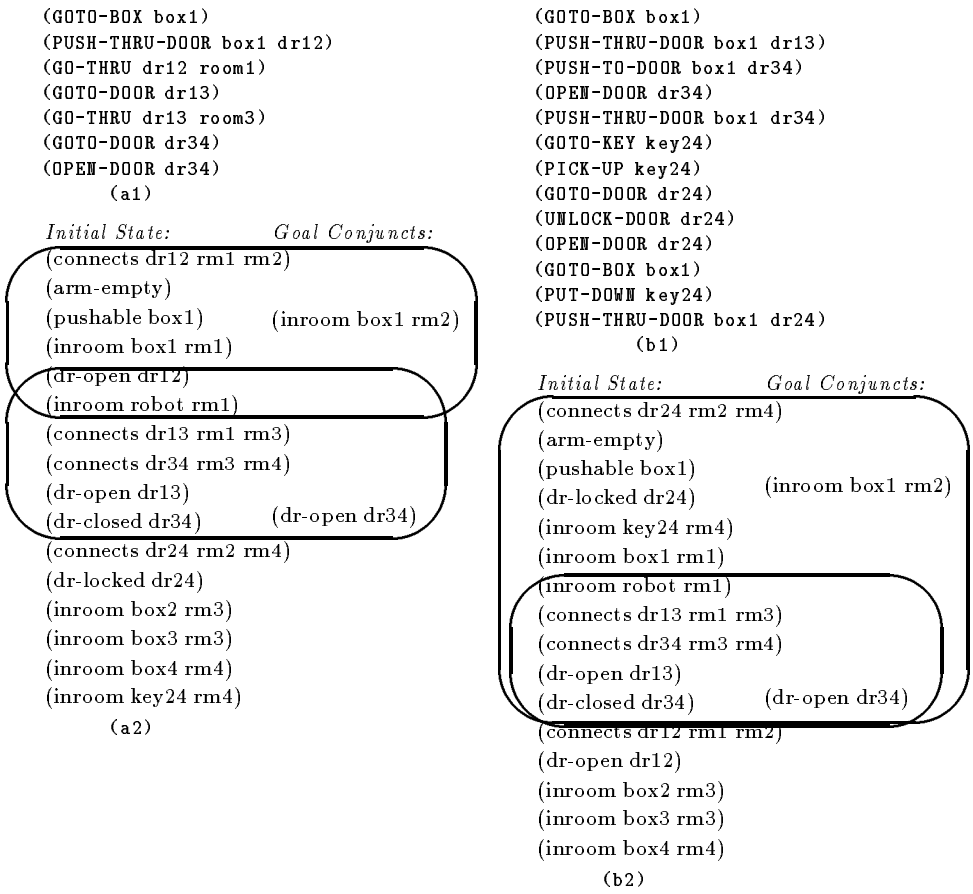


Figure 13. Two different solutions for the problem in Figure 3: Plans (a1), (b1), and their corresponding foot-printed initial states (a2) and (b2)

oriented behavior [Kedar-Cabelli, 1985, Hammond, 1989] than the one introduced earlier, by focusing only on the goal-relevant portions of the initial state [Hickman and Larkin, 1990, Pazzani, 1990] as determined by the problem solver for each case in the library.

Definition 4 *We say that the initial state $\mathcal{S}$ foot-print matches an initial state $\mathcal{S}'$ under a substitution σ and given matched goals $g_1^m, \ldots, g_k^m$ with match value δ if there are $\boldsymbol{\delta}$ many literals l in $\mathcal{S}$, such that (i) l directly matches some literal l' in $\mathcal{S}'$ under σ, and (ii) l is in the foot-print of some goal g_i^m, for $i = 1, \ldots, m$.*

When assigning a match value to two problems we now consider not only the *number* of goals that match, but also use the matched goals themselves to determine the match degree of the initial state.

From the definition 4 we change steps 4 and 8 of the procedure presented in Figure 10 accordingly. Namely in step 4 we compute the match value for the goal statements but further return which goals matched. In step 8 we use these goals to compute the match value for the initial states. The rest of the algorithm is invariant to selection of similarity metric.

5.2.1 Further search reduction examples

We ran new experiments with this foot-print similarity metric in the extended-Strips and process-job planning domains. The filled curves in Figures 11 (a) and (b) show the results for these two domains. We note that the results with the foot-print similarity metric show an improvement over base search of a factor of 2.0 speed up for the process-job planning and scheduling domain and 2.6 speed up for the extended-Strips domain. The curves obtained do not represent the best improvement expected as the set of forty problems used does not completely cover the full range of problems in either domain. We expect further improvements with a denser coverage (more cases). One of the directions of our current research is to develop techniques for learning similarity metrics by further automatically analyzing the analogical replay mechanism. [4]

6. The Logistics Transportation Domain

To scale up the system in both the size and diversity of domains, we have currently a 1000-case library in a complex logistics transportation domain. In this domain, packages are to be moved among different cities. Packages are carried within the same city in trucks and across cities in airplanes. Trucks and airplanes may have limited capacity. At each city there are several locations, e.g. post offices and airports. This transportation domain represents scale up in length of the solution and size of the search space over the other domains we have been using like the extended-Strips or the process-job planning domains.

4. In fact we currently have generated a more sophisticated similarity metric also derived from the derivational trace where better improvements are noticed [Veloso, 1992].

To generate such a large collection of problems, we implemented tools to automatically create random problems of different complexity in this domain. The user specifies the number of cities, trucks, airplanes, and packages desired, and the system randomly assigns the locations for all these entities. The complexity of the problem is directly related to the size of the configuration entered as well as the number of goals specified.

We organize the 1000-case library in the three-level indexing structure presented above. The top level of access associates each generalized goal literal with the problems that solved a corresponding instantiated version of that literal. A second-level of indexing links the full generalized conjunctive goal to the generalized initial state configuration of the problem. The final indexing level is a discrimination network that organizes the initial state according to its relevance in solving the problem. We performed several retrieval experiments that show that this three-level access structure acts as a very useful filtering mechanism to effectively prune the number of candidate retrieved analogs.

Finally we developed a more powerful replay mechanism that is able to reconstruct a solution to a new problem by analogy with multiple guiding similar past situations. This method evolved from our previously developed one-case replay mechanism [Veloso, 1991] and consists of a sophisticated algorithm to merge the multiple analog past cases. It has been shown to produce very significant results in complex problems that require guidance from several individual simpler past situations.

Recent tests with up to 1000 cases in the library have demonstrated the scaling properties of the memory organization, of the match/retrieval process, and of the reconstruction mechanism replaying multiple cases. Details on the multiple-case replay mechanism and results of these scaling up tests are being compiled and forthcoming in [Veloso, 1992].

We now show two examples from the logistics domain that illustrate the replay mechanism when one or more cases are used for guidance.

6.1 Following one case – Subgoaling structure and failures

In Figure 14 we illustrate how the subgoaling structure and the failure records at a stored case can help guiding the reconstruction process for a similar new problem. First let us focus on the basic representation in Figure 14. The detailed explanation of the picture follows later. On the left side of Figure 14 we show a sequence of nodes named cn1 through

cn21 that correspond to a past stored case. We describe below the meaning of these nodes as well as the problem-solving situation. The sequence of nodes, n1,... n21, on the right side of Figure 14 represent the new problem-solving episode. Both sequences represent successful search paths and follow NOLIMIT's search cycle [Veloso, 1989]. The sequence of decisions taken is captured therefore by the regular expression (goal chosen-(relevant)-operator applied-operator *) *. The final plan itself can be read by the sequence of applied operator nodes. The arrows across the nodes show the transfer occurred.

The past case was stored with instances generalized to variables of the same class as illustrated in section 4. When a case is retrieved as similar to a new situation, the partial match found between the old and new situations defines partial bindings to the variablized past case.[5] The past generalized problem involved moving an object ?ob9 from the post office at some city ?po35 to an airport at a different city ?ap17. In the initial state there was a truck ?tr35 at the post office ?po35 and an airplane ?pl3 at the airport ?ap17. Formally the relevant past initial state was: `(at-obj ?ob9 ?po35) (at-truck ?tr35 ?po35) (at-airplane ?pl3 ?ap17) (same-city ?po35 ?ap35)`. The goal statement was the simple goal `(at-obj ?ob9 ?ap17)`.

Assume that this case was retrieved as the most similar to a new problem where an object ob0 is also to be moved from a post office p0 to an airport a2 at a different city. In this new initial state however a truck tr0 is at the airport a0 and there is an airplane pl0 also at a0. Formally the initial state is: `(at-obj ob0 p0) (at-truck tr0 a0) (at-airplane pl0 a0) (same-city p0 a0)`, and the goal statement is: `(at-obj ob0 a2)`. The retrieval procedure returns the substitution (?ob9/ob0, ?po35/p0, ?ap35/a0, ?ap17/a2) as a partial match between that past case and the new situation. Note that neither the truck ?tr35 nor the airplane ?pl3 get bindings from this partial match. However the replay mechanism further assigns bindings as the match between the two situations becomes clearer along the reconstruction. In the figure we show the case further instantiated with the substitutions ?tr35/tr0 and ?pl3/pl0 that occur dynamically at transfer time as we now also explain.

5. The complete storage, matching, and retrieval procedures are described in [Veloso, 1992].

Figure 14. Following one case – Subgoaling structure and failures.

The same goal is chosen at the node n1 as it was at node cn1. At node cn2 the past case records that the operator (unload-airplane ob0 ?pl3 a2) was successfully chosen and that the operator (unload-truck ob0 ?tr77 a2) failed. This information guides the decision at node n2 of choosing (also successfully as realized in the sequence) the relevant operator (unload-airplane ob0 pl0 a2) instead of (unload-truck ob0 tr2 a2). The substitution ?pl3/pl0 is set and applied to the case. The transfer continues interleaving the choices of goals and relevant operators in the same subgoaling chains. At node cn6 the alternative choice of (unload-airplane ob0 pl0 a0) is pruned from the new case, because the justification for failure in the past, namely the goal-loop of the goal (inside-airplane ob0 pl0), also holds. This goal is chosen in this search path, namely at node n3, and was not achieved yet at the current node n6.

As we noticed, the problems diverge in the location of the truck and airplane. In fact at node cn9, the past decision of loading the truck at the post office cannot now be immediately transferred as the operator (load-truck ob0 tr0 p0) is not applicable in the new problem. The new solution diverges then from the past case at nodes n9, n10, and n11, where the conditions for applying that operator are set, namely by driving the truck tr0 from the airport a0 to the post office p0. The past case is stopped at the node cn9. The past decision is tested at each new step to see whether it is justified. This happens at node n12 where the transfer continues. A somehow symmetric situation occurs when, at the node cn14 the goal (at-airplane pl0 a0) is not a pending goal in the new problem, as the airplane was initially already at the airport a0. In this case, the past case is advanced and the steps in the subgoaling structure of that goal are skipped. The transfer is pursued at node cn17 and the reconstruction process terminates successfully.

6.2 Following multiple cases

In the Figure 15 we show a reconstruction process guided by two past cases. The new situation is shown at the center of the figure and the two past guiding cases on its left and right sides.

The new problem to be solved consists of a two-goal conjunct, namely to load an object o4 into a truck tr9 and to load another object o2 into an airplane pl7. The goal conjunct is `(and (inside-truck o4 tr9) (inside-airplane o2 pl17))`. The literals `(at-obj o4 p5)` `(inside`

Figure 15. Following multiple cases – Merging during derivational replay.

`-truck o2 tr9) (at-airplane pl7 a11)` are in the new initial state.

The retrieval procedure returns two past cases each partially matching one of the goal conjuncts. [6] The case represented on the left corresponds to a situation where an object was also to be loaded into a truck. However this truck was at the airport of the city and not at the post office. The case represented on the right corresponds to a past solved problem where an object is to be loaded into an airplane and the object is already at the airport.

The transfer occurs by interleaving the two guiding cases and performing any additional work needed to accomplish remaining subgoals. The subgoaling structure stored at the past cases defines which case should be followed. When there is nothing specifying which case to follow, the replay mechanism randomly decides on the case to pursue. This randomness occurs in a small percentage of the decisions as most of them are guided by the justifications stored in particular by the subgoaling chaining. We have been noticing interestingly that the random behavior allows innovative merging of past cases leading to solutions of a better quality in several situations.

6.3 Empirical Results

Although our analysis in this large-scaled domain is not complete yet, the results so far show high positive transfer reducing significantly the total memory retrieval and problem-solving times. We have experienced reductions of up to 90% in the size of the search space and in the total running time. In Figure 16 we show results from running 200 complex problems in this domain. We note that the system without analogy performs considerably worse than the analogical reasoner. The case library was incrementally built as the problems were being solved.

We are still in the process of compiling and interpreting empirical and analytical results on the complexity of this logistics domain in order to quantify more precisely the scaling-up capabilities of our methods [Veloso, 1992].

6. In general the retrieval method tries to find cases that solved the maximum possible number of goals with good match of the initial state. The complete retrieval procedure is presented in [Veloso, 1992].

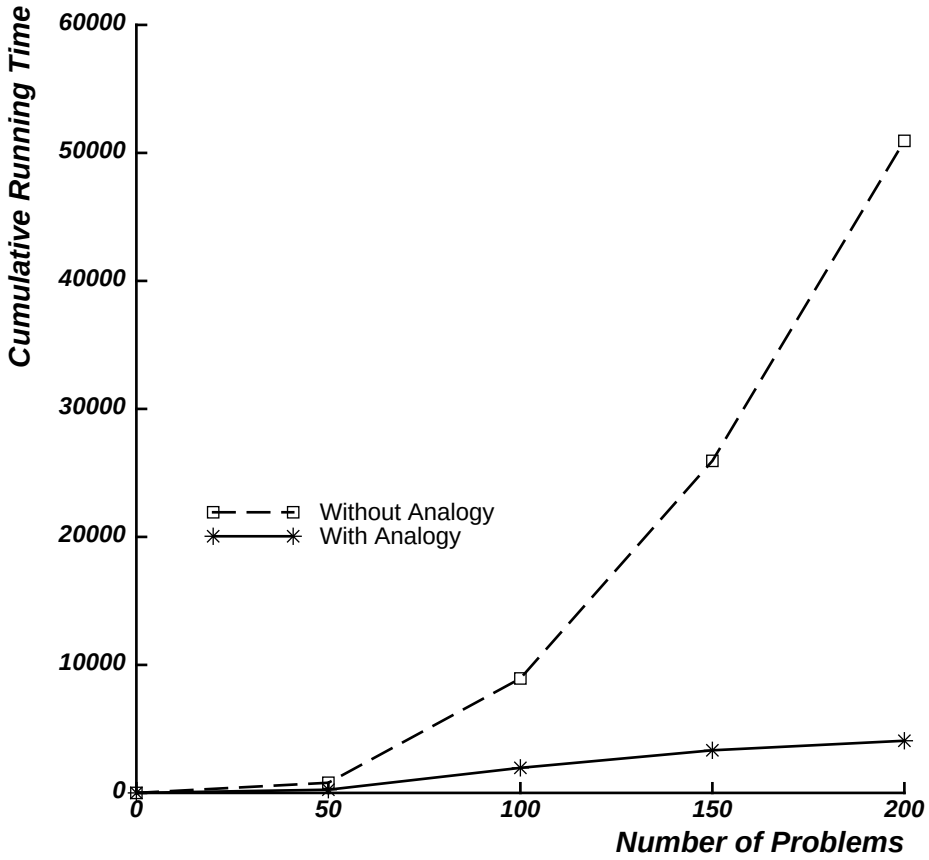


Figure 16. Cumulative running time for 200 problems in the logistics domain with and without analogy.

7. Trading Off Retrieval and Search Costs

In pure general-purpose problem solvers the cost of search is exponential in the length of the solution. (By pure PS systems, we refer to systems that search without any control knowledge to prune the search space of possible operators). In pure CBR systems the cost of retrieval is very high as the system fully relies on retrieving the *best* case in memory to maximize its chance of successful adaptation.

In the analogical version of PRODIGY, where we integrate a search-based problem solver with an analogical reasoner, we balance the cost of retrieving and the residual problem-solving cost. We show how we balance the cost of retrieval as a function of the degree of partial match.

In the retrieval procedure of Figure 10, suppose that the memory is organized in a discrimination network. The organization of the memory is such that the indexes for the cases are less relevant as we move away from the root of the discrimination network. On the other hand, given a new problem P' with initial state $\mathcal{S}^{P'}$ and goal $\mathcal{G}^{P'}$, we can compute the absolute maximum possible match value. In fact it is simply $absolute\_max\_match = \text{length } (\mathcal{G}^{P'}) + \text{length } (\mathcal{S}^{P'})$.

In general, we integrate analogy and search to reduce the size of the search space in terms of the number of the nodes searched and consequently achieve an improvement in running time. Harandi and Bhansali [Harandi and Bhansali, 1989] confirmed that analogy would be useful if the time to find analogues is small and the degree of similarity is high. Hickman, Shell, and Carbonell [Hickman et al., 1990] also showed that internal analogy can reduce the search cost. We show now that there is an optimal range of retrieval time to spend searching for candidate analogs. Intuitively the deeper that memory is searched, the better the analog and the less search required by the problem solver. However searching memory also takes time. Is there, hence, an optimal amount of effort to spend searching memory?

We assume that the memory is organized in such a way that the match degree increases monotonically with retrieval time [Kolodner, 1984, Schank, 1982] though not necessarily in a linear manner. This also means that there is always one (or more) case available to return when retrieval is halted. However if the retrieval time increases, the match value between the case returned and the new problem also increases. We now formalize this model. Let

- t_r be the time spent to retrieve a similar past case,
- δ_{t_r} be the match value between the case retrieved and the new problem,
- m be the absolute_max_match as introduced above, and
- d be the percentage of deviation from the absolute_max_match of the match value of the case retrieved if the retrieval time is null (or close to null).

Then we say that

$$\delta_{t_r} = m(1 - dC^{-\alpha t_r}), \tag{1}$$

where C and α are constants.

In Figure 17 we sketch three possible curves for the match value as

a function of the retrieval time. Curves 1 and 2 show situations where the initial match is poor, i.e. with low match degree. However for curve 1 the rate of match-degree improvement is very low (low α) while for curve 2 the match degree increases rapidly with the retrieval time. Situations 1 and 2 depict two different rates of improvement for the match result while traversing down the discrimination net. Curve 3 plots a situation where the initial match is immediately high and continues to improve gradually towards the maximum.

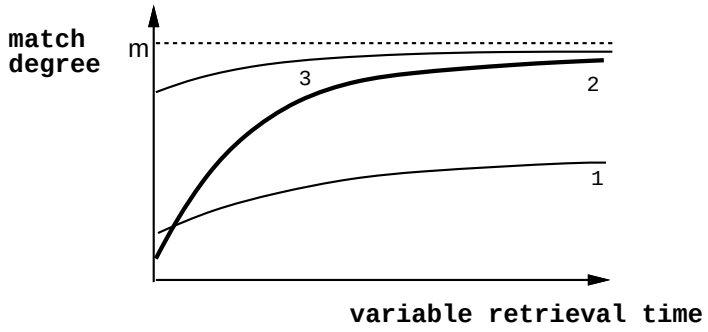


Figure 17. Three different curves for the match value as a function of the retrieval time.

In the situations captured by the curves 1 and 3, the system should not invest a long time in retrieving a *better, or best* similar past case. In both cases termination occurs because the rate of improvement, α, is low. In case 1, the system should solve the problem by general MEA search because there are no good cases, and in case 3 it should immediately start derivational replay on the retrieved high-match case, rather than waste time seeking a marginally better one. Situation 2 illustrates the case where retrieval time is more wisely invested. Given the fact that the match degree is on average directly related to search savings in PS, we now show analytically that there is an optimal amount of effort to spend searching memory.

PRODIGY's search tree can be viewed as an OR-tree, branching alternatively among possible goal orderings and possible operators to achieve a goal. Let b be the average branching factor of the search tree, Let l be the solution length for a given problem, and S be the search effort without analogy. Then the complexity of S is [Hickman et al., 1990], $S = \mathcal{O}(b^{\mathcal{O}(l)})$. (From now on we skip the order of, $\mathcal{O}$, notation for simplicity.) Assume that the effect of analogical reasoning is captured in

a decrease of the average branching factor [Hickman et al., 1990]. This reduction of the search effort is in direct relationship with the match degree of the guiding case(s). Let $S_{analogy}$ be the search spent with analogy. We can then say that, for some linear function f,

$$S_{analogy} = ((1 - f(\delta_{t_r}))b)^l. \quad (2)$$

The goal of the integrated analogical reasoner is to improve the effort to reach a solution: PS search time plus memory search time [Harandi and Bhansali, 1989]. The objective is to find the situation when this sum is much smaller than brute-force PS search without any analogical guidance. We capture this goal in the inequality below, where we do not represent, for simplicity, the function f introduced in eq. 2:

$$t_r + ((1 - \delta_{t_r})b)^l \ll b^l. \quad (3)$$

Substituting eq. 1 into the eq. 3, we get the final equation as a function of the retrieval time t_r:

$$t_r + (1 - m(1 - dC^{-\alpha t_r}))^l b^l \ll b^l \quad (4)$$

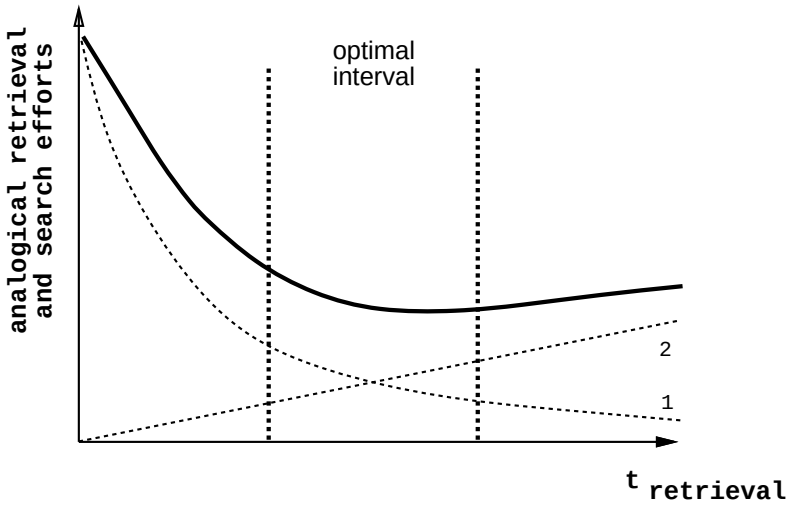


Figure 18. Retrieval time (curve 2) plus analogical search effort (curve 1).

Figure 18 sketches the left hand side of inequality 4. [7] Analyzing this qualitative curve, we conclude that there is an optimal retrieval time interval, which is a function of the dynamic match rate α. Retrieval should then stop when a given threshold is reached, namely when the derivative of the expected search savings approaches the incremental memory search cost.

8. Discussion: EBL and Analogy, Abstraction and Analogy

Previous work in the linear planner of PRODIGY uses explanation-based learning (EBL) techniques [Minton, 1988] to extract from a problem-solving trace the explanation chain responsible for a success or failure and, together with domain axioms, compile search control rules therefrom. The axiomatized domain knowledge is also used to learn abstraction layers [Knoblock, 1991], and statically generate control rules [Etzioni, 1990]. In this section we discuss the benefits that we foresee in a deeper integration of EBL and analogy, and abstraction and analogy.

8.1 EBL and analogy

Although we do not directly yet integrate full EBL with analogy we expect scaling-up benefits from EBL-compiled control rules for high-utility common situations plus a case library for lower-frequency more complex situations. We discuss first the simple explanation mechanism we use in the foot-print similarity metric. We then extend our discussion to the deeper integration that we envision.

While performing the goal regression on the derivational trace to determine the relevant foot-print of the initial state, the analogical reasoner performs a lazy explanation of the solution encountered. The evaluation is termed "lazy" because it goes up the successful path following the subgoaling chain without trying to prove any generalization of the immediate success or recorded failures. However it turns out quite useful to take into account this explanation though simple, as we discussed in section 5 to isolate the relevant part of the initial state.

7. This smooth curve does not correspond to data from any particular domain. It captures solely the qualitative behavior of the search effort according to our analytical analysis.

We plan to pursue a deeper integration with the EBL module in PRODIGY [Minton, 1988, Etzioni, 1990]. On one hand, this integration would allow the joint EBL-analogical reasoner to decide whether to invest effort in statically analyzing a domain theory or a trace of a solved problem to generalize control rules therefrom, or to store the problem-solving episode as a case for eventual future retrieval and replay. On the other hand, the dynamic memory organization in analogy converges by clustering problems that are similar to each other. An explanation-based strategy would be capable of generalizing the situations captured by the more densely populated clusters and generate control rules therefrom.

Analogy can also complement EBL in incomplete domain theories where the proofs cannot be pursued. In such situations, EBL could apply to completely defined subtheories (for parts of the domain) and use analogy as a backup for explanations that cannot be proved correct. The joint EBL-analogical reasoner could switch from an eager costly attempt to explain a difficult trace to a lazy attitude of storing it as a case, as a function of a cost function between the two approaches.

8.2 Abstraction and analogy

A key issue in the process of solving problems by analogy is the identification of what are the *details* and what are the *relevant* features of a particular situation. As new and past situations are not expected to fully match, knowing the relevance of the information available increases the ability for successful partial matching of different problems. ALPINE [Knoblock, 1991] provides a mechanism that analyzes a particular domain, and generates abstraction levels that group together features (literals) in a hierarchical structure, the most crucial, interrelated ones at the top. We plan to explore the use of the abstraction levels generated by ALPINE as a measure of relevance to rank partially matched candidate analogs. In other words, the partial match is weighted towards features and relations at the top of the abstraction hierarchy.

The way the case memory is organized directly relates to the eventual success of the retrieval phase. As we have seen before, a new situation is presented in terms of an initial world state and a goal statement. The initial similarity metric we use requires a total match at the goal predicate level, i.e. situations that refer to different uninstantiated goals are not proposed as candidate analogs. This supports a goal-oriented

matching method [Kedar-Cabelli, 1985] that we advocate as the base strategy.

The dynamic organization of memory currently does a simple abstraction by generalizing instances into their classes. However we plan to use multiple abstraction levels to help the dynamic organization of the discrimination network. Namely, we would like to have the most relevant features of the problem being unified with the new problem before the less relevant ones, where relevant is both a function of past experience and the level in the abstraction hierarchy.

A second major benefit of the integration of analogy and abstraction is the *generality* of stored plans for later indexing. That is, a solution at an abstract level is much more likely to be an applicable candidate analog then one at the ground level – although it will require refinement by adding in details of the current problem. In general, we propose to use abstract analogs when specific grounded ones are not present to guide search in derivational analogy.

9. Conclusion

In this chapter we presented the automatic case generation, storage, and retrieval of cases in the derivational analogy module in the PRODIGY architecture, and showed how the methods contribute to scaling up.

We showed how the problem solver introspects into its decision cycle recording the justifications of the decisions taken and automatically generating cases from its problem-solving experience. These cases are organized in a case library indexed by their goal and relevant initial state. We illustrated the case generation procedure and the organization of memory with simple examples from the extended-STRIPS domain. We presented results of the performance of the replay mechanism when the retrieval procedure used two different similarity metrics, the direct and foot-print ones. The direct metric does not consider any correlations between the initial state and the goal conjuncts. The foot-print metric performs a goal regression in the solution path and is able to identify the relevant features of the initial state for each goal conjunct. The system shows better performance when the foot-print similarity metric is used as more similar cases are replayed. Given our memory indexing method, similarity metric, and multiple-case replay mechanism, the analogical reasoner of PRODIGY can scale up to complex domains such as

logistics/transportation with a 1000-case library. Finally we discussed the benefits we foresee of a deeper integration of the EBL and abstraction modules with the analogical reasoner within PRODIGY.

Acknowledgements

The authors thank Daniel Borrajo for a major part of NOLIMIT's implementation, and Oren Etzioni, Yolanda Gil, Robert Joseph, Craig Knoblock, Steve Minton and Alicia Pérez for many useful discussions and suggestions on both the analogy work and the synergy of the learning modules. The authors thank the whole PRODIGY research group for helpful comments on this work.

This research was sponsored by the Defense Advanced Research Projects Agency (DOD) and monitored by the Avionics Laboratory, Air Force Wright Aeronautical Laboratories, Aeronautical Systems Division (AFSC), Wright-Patterson AFB, OH 45433-6543 under Contract F33615-87-C-1499, ARPA Order No. 4976, Amendment 20. 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 Defense Advanced Research Projects Agency or of the U.S. Government.

References

[Allen and Langley, 1990] Allen, J. and Langley, P. (1990). Integrating memory and search in planning. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, pages 301–312, San Diego, CA. Morgan Kaufmann.

[Bareiss and King, 1989] Bareiss, R. and King, J. A. (1989). Similarity assessment in case-based reasoning. In *Proceedings of the Second Workshop on Case-Based Reasoning*, pages 67–71, Pensacola, FL. Morgan Kaufmann.

[Cain et al., 1991] Cain, T., Pazzani, M., and Silverstein, G. (1991). Using domain knowledge to influence similarity judgments. In *Proceedings of the 1991 DARPA Workshop on Case-Based Reasoning*, pages 191–199. Morgan Kaufmann.

[Carbonell et al., 1992] Carbonell, J. G., and the PRODIGY Research Group (1992). PRODIGY4.0: The manual and tutorial. Technical Report CMU-CS-92-150, School of Computer Science, Carnegie Mellon University.

[Carbonell, 1983] Carbonell, J. G. (1983). Learning by analogy: Formulating and generalizing plans from past experience. In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors, *Machine Learning, An Artificial Intelligence Approach*, pages 137–162, Palo Alto, CA. Tioga Press.

[Carbonell, 1986] Carbonell, J. G. (1986). Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors, *Machine Learning, An Artificial Intelligence Approach, Volume II*, pages 371–392. Morgan Kaufman.

[Carbonell, 1990] Carbonell, J. G., editor (1990). *Machine Learning: Paradigms and Methods*. MIT Press, Boston, MA.

[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.

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

[Carbonell and Veloso, 1988] Carbonell, J. G. and Veloso, M. M. (1988). Integrating derivational analogy into a general problem solving architecture. In *Proceedings of the First Workshop on Case-Based Reasoning*, pages 104–124, Tampa, FL. Morgan Kaufmann.

[Catlett, 1991] Catlett, J. (1991). Overpruning large decision trees. In *Proceedings of IJCAI-91*, pages 764–769.

[Cheng and Carbonell, 1986] Cheng, P. W. and Carbonell, J. G. (1986). The FERMI system: Inducing iterative rules from experience. In *Proceedings of AAAI-86*, pages 490–495, Philadelphia, PA.

[DeJong and Mooney, 1986] DeJong, G. F. and Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine Learning*, 1(2):145–176.

[DeJong, 1989] DeJong, K., editor (1989). *Machine Learning. Special Issue on Genetic Algorithms*. Kluwer Academic Publishers.

[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.

[Hammond, 1986] Hammond, K. J. (1986). *Case-based Planning: An Integrated Theory of Planning, Learning and Memory*. PhD thesis, Yale University.

[Hammond, 1989] Hammond, K. J. (1989). Opportunistic memory. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 504–510, San Mateo, CA. Morgan Kaufmann.

[Harandi and Bhansali, 1989] Harandi, M. T. and Bhansali, S. (1989). Program derivation using analogy. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 389–394, Detroit, MI.

[Hickman and Larkin, 1990] Hickman, A. K. and Larkin, J. H. (1990). Internal analogy: A model of transfer within problems. In *The 12th Annual Conference of The Cognitive Science Society*, pages 53–60, Hillsdale, NJ. Lawrence Erlbaum Associates.

[Hickman et al., 1990] Hickman, A. K., Shell, P., and Carbonell, J. G. (1990). Internal analogy: Reducing search during problem solving. In Copetas, C., editor, *The Computer Science Research Review 1990*. The School of Computer Science, Carnegie Mellon University.

[Hinton, 1989] Hinton, G. E. (1989). Connectionist learning procedures. *Artificial Intelligence*, 40:185–234.

[Holland, 1986] Holland, J. H. (1986). Escaping brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-based systems. In Michalski, R. S., Carbonell, J. G., and

Mitchell, T. M., editors, *Machine Learning, An Artificial Intelligence Approach, Volume II.* Morgan Kaufman.

[Iba, 1989] Iba, G. A. (1989). A heuristic approach to the discovery of macro-operators. *Machine Learning*, 3(4):285–317.

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

[Kedar-Cabelli, 1985] Kedar-Cabelli, S. (1985). Purpose-directed analogy. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, pages 150–159.

[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., Minton, S., and Etzioni, O. (1991). Integrating abstraction and explanation based learning in PRODIGY. In *Proceedings of AAAI-91*, pages 541–546.

[Kolodner, 1983] Kolodner, J. (1983). Maintaining organization in a dynamic long-term memory. In Charniak, E., Norman, D., and Smith, E., editors, *Cognitive Science*, volume 7, pages 243–280, Norwood, NJ. Ablex Publishing Company.

[Kolodner, 1989] Kolodner, J. (1989). Judging which is the "best" case for a case-based reasoner. In *Proceedings of the Second Workshop on Case-Based Reasoning*, pages 77–81. Morgan Kaufmann.

[Kolodner, 1984] Kolodner, J. L. (1984). *Retrieval and Organization Strategies in Conceptual Memory.* Lawrence Erlbaum Associates, Inc., Hillsdale, New Jersey.

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

[Korf, 1987] Korf, R. E. (1987). Planning as search: A quantitative approach. *Artificial Intelligence*, 33:65–88.

[Michalski et al., 1983] Michalski, R. S., Carbonell, J. G., and Mitchell, T., editors (1983). *Machine Learning: An Artificial Intelligence Approach*, volume I. Morgan Kaufmann Publishers, Inc., Los Altos, CA.

[Michalski et al., 1986] Michalski, R. S., Carbonell, J. G., and Mitchell, T., editors (1986). *Machine Learning: An Artificial Intelligence Approach*, volume II. Morgan Kaufmann Publishers, Inc., Los Altos, CA.

[Michalski and Kodratoff, 1990] Michalski, R. S. and Kodratoff, Y., editors (1990). *Machine Learning, An Artificial Intelligence Approach, Volume III*. Morgan Kaufmann, Palo Alto, CA.

[Minsky and Selfridge, 1961] Minsky, M. and Selfridge, O. G. (1961). Learning in neural nets. In *Proceedings of Fourth London Symposium on Information Theory*, New York, NY. Academic Press.

[Minton, 1985] Minton, S. (1985). Selectively generalizing plans for problem solving. In *Proceedings of AAAI-85*, pages 596–599.

[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., 1989a] Minton, S., Carbonell, J. G., Knoblock, C. A., Kuokka, D. R., Etzioni, O., and Gil, Y. (1989a). Explanation-based learning: Optimizing problem solving performance through experience. *Artificial Intelligence*.

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

[Mitchell et al., 1986] Mitchell, T. M., Keller, R. M., and Kedar-Cabelli, S. T. (1986). Explanation-based generalization: A unifying view. *Machine Learning*, 1:47–80.

[Núñez, 1991] Núñez, M. (1991). The use of background knowledge in decision tree induction. *Machine Learning*, 6(3):231–250.

[Pazzani, 1990] Pazzani, M. (1990). *Creating a Memory of Causal Relationships: An integration of empirical and explanation-based learning methods.* Lawrence Erlbaum Associates, Hillsdale, NJ.

[Porter et al., 1989] Porter, B., Bareiss, R., and Holte, R. (1989). Knowledge acquisition and heuristic classification in weak-theory domains. Technical Report AI-TR-88-96, Department of Computer Science, University of Texas at Austin.

[Quinlan, 1983] Quinlan, J. R. (1983). Learning efficient classifiation procedures and their application to chess end games. In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors, *Machine Learning, An Artificial Intelligence Approach, Volume I.* Morgan Kaufman.

[Quinlan, 1986] Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1:81–106.

[Quinlan, 1987] Quinlan, J. R. (1987). Simplifying decison trees. *International Journal of Man-machine Studies*, 27:221–234.

[Riesbeck and Schank, 1989] Riesbeck, C. K. and Schank, R. C. (1989). *Inside Case-Based Reasoning.* Lawrence Erlbaum Associates, Inc., Hillsdale, New Jersey.

[Rosenbloom et al., 1990] Rosenbloom, P. S., Lee, S., and Unruh, A. (1990). Responding to impasses in memory-driven behavior: A framework for planning. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control.* Morgan Kaufmann.

[Sacerdoti, 1977] Sacerdoti, E. D. (1977). *A Structure for Plans and Behavior.* American Elsevier, New York.

[Schank, 1982] Schank, R. C. (1982). *Dynamic Memory.* Cambridge University Press.

[Shell and Carbonell, 1989] Shell, P. and Carbonell, J. G. (1989). Towards a general framework for composing disjunctive and iterative macro-operators. In *Proceedings of IJCAI-89.*

[Simpson, 1985] Simpson, R. L. (1985). *A computer model of case-based reasoning in problem solving: An investigation in the domain of dispute mediation.* PhD thesis, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA.

[Sycara, 1987] Sycara, E. P. (1987). *Resolving adversarial conflicts: An approach to integrating case-based and analytic methods.* PhD thesis, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA.

[Touretzky, 1989] Touretzky, D. (1989). Connectionism and compositional semantics. In Barnden, J. A. and Pollack, J. B., editors, *Advances in Connectionist and Neural Computational Theory.* Ablex, Norwood, NJ.

[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, 1991] Veloso, M. M. (1991). Efficient nonlinear planning using casual commitment and analogical reasoning. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, pages 938–943, University of Chicago, IL. Lawrence Erlbaum.

[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, 1989] Veloso, M. M. and Carbonell, J. G. (1989). Learning analogies by analogy - The closed loop of memory organization and problem solving. In *Proceedings of the Second Workshop on Case-Based Reasoning*, pages 153–158, Pensacola, FL. Morgan Kaufmann.

[Veloso and Carbonell, 1990] Veloso, M. M. and Carbonell, J. G. (1990). Integrating analogy into a general problem-solving architecture. In Zemankova, M. and Ras, Z., editors, *Intelligent Systems*, pages 29–51. Ellis Horwood, Chichester, England.

[Veloso and Carbonell, 1991] Veloso, M. M. and Carbonell, J. G. (1991). Variable-precision case retrieval in analogical problem solving. In *Proceedings of the 1991 DARPA Workshop on Case-Based Reasoning*, pages 93–106, Washington, DC. Morgan Kaufmann.

[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.

[Waldinger, 1981] Waldinger, R. (1981). Achieving several goals simultaneously. In Nilsson, N. J. and Webber, B., editors, *Readings in Artificial Intelligence*, pages 250–271. Morgan Kaufman, Los Altos, CA.