PRODIGY/ANALOGY: Analogical Reasoning in General Problem Solving*

Manuela M. Veloso

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3891
veloso@cs.cmu.edu

Abstract. This paper describes the integration of analogical reasoning into general problem solving as a method of learning at the strategy level to solve problems more effectively. The method based on derivational analogy has been fully implemented in PRODIGY/ANALOGY and proven empirically to be amenable to scaling up both in terms of domain and problem complexity. PRODIGY/ANALOGY addresses a set of challenging problems, namely: how to accumulate episodic problem solving experience, cases, how to define and decide when two problem solving situations are similar, how to organize a large library of planning cases so that it may be efficiently retrieved, and finally how to successfully transfer chains of problem solving decisions from past experience to new problem solving situations when only a partial match exists among corresponding problems. The paper discusses the generation and replay of the problem solving cases and we illustrate the algorithms with examples. We present briefly the library organization and the retrieval strategy. We relate this work with other alternative strategy learning methods, and also with plan reuse. PRODIGY/ANALOGY casts the strategy-level learning process for the first time as the automation of the complete cycle of constructing, storing, retrieving, and flexibly reusing problem solving experience. We demonstrate the effectiveness of the analogical replay strategy by providing empirical results on the performance of PRODIGY/ANALOGY, accumulating and reusing a large case library in a complex problem solving domain. The integrated learning system reduces the problem solving search effort incrementally as more episodic experience is compiled into the library of accumulated learned knowledge.

* Special thanks to Jaime Carbonell for his guidance, suggestions, and discussions on this work. A reduced version of this paper was published in the Proceedings of the Twelfth National Conference on Artificial Intelligence, 1994. 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 Introduction

The problem solving methods developed so far in Artificial Intelligence can be organized in a problem solving reasoning continuum ranging from *search-intensive* to *knowledge-intensive* methods. Pure search-intensive methods search exhaustively for a solution from first principles, i.e., individual steps that model atomic actions in the task domain and may be chained to form a solution to a problem. Pure knowledge-intensive methods for problem solving presuppose the existence of a collection of prototypical solutions from where the problem solver retrieves and instantiates an appropriate solution to a new problem. Variations from these two extreme approaches extend the search-intensive paradigm to search guided by local control knowledge while the knowledge-intensive extreme extends to case-based reasoning (CBR) approaches in which the retrieved solution may be adapted after being retrieved and instantiated.

Solving complex problems with multiple interacting goals and multiple alternative plan choices is a well-known difficult problem in either of the two extreme paradigms. The search-intensive methods face an exponential growth of the search space with the problem complexity. Similarly, to guarantee the success of the adaptation phase, CBR systems require accurate, hard to generate, similarity metrics and incur high retrieval costs.

Derivational analogy was proposed by Carbonell ([Carbonell, 1986]) as a method that would draw nearer the search- and the knowledge-intensive paradigms. Derivational analogy is a problem solving technique that replays and modifies past problem solving traces in new similar situations. Therefore the problem solver in addition to its domain principles is able to use past experience in the form of complete problem solving episodes.

This paper presents PRODIGY/ANALOGY, which draws upon the original derivational analogy strategy [Veloso and Carbonell, 1990]. Analogical reasoning in PRODIGY/ANALOGY integrates automatic case generation, case retrieval and storage, case replay, and general planning, exploiting and modifying past experience when available and resorting to general problem-solving methods when required. Learning occurs by accumulation and flexible reuse of cases. The planning search effort is reduced incrementally as more episodic experience is compiled into the case library.

The contributions of this work go well beyond the initial derivational idea proposed by Carbonell. They include: the refinement and full implementation of the derivational analogy replay method in the context of a nonlinear problem solver; development of efficient storage and retrieval techniques for the learned cases; demonstration of learning by analogy as a method to successfully transferring problem solving experience in partially matched new situations; and a flexible replay mechanism to merge (if needed) multiple similar episodes that jointly provide guidance for new problems. The method enables the problem solver and learner to solve complex problems after being trained in solving simple problems.

PRODIGY/ANALOGY is novel in the automation of the complete analogical cycle, namely the generation (annotation), storage, retrieval, and replay of episodic knowledge. It follows a domain-independent approach and it is demonstrated in particular in a case library of several orders of magnitude greater than most of the other case-based reasoning (or knowledge-intensive) systems, in terms of the size of the case library and the granularity of the individual cases.

The paper is organized in eight sections. Section 2 describes how a problem solving case is generated from a search episode. Section 3 discusses the indexing mechanism and Section 4 introduces the similarity metric and the retrieval procedure. Section 5 presents the replay algorithm which constructs a new solution to a problem by following and merging multiple guiding cases. Section 6 shows empirical results on the performance of PRODIGY/ANALOGY in a complex logistics transportation domain building a case library of more than 1000 cases. Finally, Section 7 discusses related work and Section 8 draws conclusions.

2 Generation of Problem Solving Cases

The purpose of solving problems by analogy is to reuse past experience to guide generation of solutions for new problems avoiding a completely new search effort. Transformational analogy and most CBR systems reuse past solutions by modifying (*tweaking*) the retrieved final solution as a function of the differences found between the source and the target problems. Instead, derivational analogy is a *reconstructive* method by which *lines of reasoning* are transferred and adapted to a new problem [Carbonell, 1986] as opposed to only the final solutions.

Automatic generation of the derivational episodes to be learned occurs by extending the base-level problem solver with the ability to examine its internal decision cycle, recording the justifications for each decision during its search process. We used NoLimit [Veloso, 1989], the first nonlinear and complete problem solver of the PRODIGY planning and learning system, as the base-level problem solver.[2] Throughout the paper, NoLimit refers to the base-level planner and PRODIGY/ANALOGY refers to the complete analogical reasoner with the capabilities to generate, store, retrieve, and replay problem solving episodes.

NoLimit is a domain-independent nonlinear planner with a rich action representation language. Each operator has a precondition expression that must be satisfied before the operator can be applied, and a list of effects that describe how the application of the operator changes the world. Preconditions are expressed in a typed first order predicate logic, encompassing negation, conjunction, disjunction, and existential and universal quantification. Variables in the operators may be constrained by arbitrary functions. The effects are atomic formulas that describe the conditions that are added or deleted from the current state when the operator is applied. Operators may also contain conditional effects, which represent changes to the world that are dependent on the state in which the operator is applied. A class (type) hierarchy organizes the objects of the world. These language constructs are important for representing complex and interesting domains.

[2] NoLimit was succeeded by the current planner, PRODIGY4.0 [Carbonell *et al.*, 1992, Fink and Veloso, 1994].

The nonlinear planner follows a means-ends analysis backward chaining search procedure reasoning about multiple goals and multiple alternative operators relevant to the goals. This choice of operators amounts to multiple ways of trying to achieve the same goal. Therefore, in addition to searching in the space of multiple goal orderings, as most of the standard nonlinear planners do, our planner searches equally in the space of multiple different approaches to solving a problem. The search in both of these spaces benefits from the analogical guidance provided by the similar planning episodes.

NoLimit's planning reasoning cycle involves several decision points, namely: the *goal* to select from the set of pending goals; the *operator* to choose to achieve a particular goal; the *bindings* to choose in order to instantiate the chosen operator; *apply* an operator whose preconditions are satisfied or continue *subgoaling* on a still unachieved goal. prodigy/analogy extends NoLimit with the capability of recording the context in which the decisions are made. Figure 1 shows the skeleton of the decision nodes. We created a language for the slot values to capture the reasons that support the choices [Veloso and Carbonell, 1993a].

Goal Node	Chosen Op Node	Applied Op Node
:step	:step	:step
:sibling-goals	:sibling-ops	:sibling-goals
:sibling-appl-ops	:why-this-op	:sibling-appl-ops
:why-subgoal	:relevant-to	:why-apply
:why-this-goal		:why-this-op
:precond-of		:chosen-at

Fig. 1. Justification record structure: Nodes correspond to search choice points. Each learned problem solving case is a sequence of justified nodes.

There are mainly three different kinds of justifications: links among choices capturing the subgoaling structure (slots `precond-of` and `relevant-to`), records of explored failed alternatives (the `sibling-` slots), and pointers to any applied guidance (the `why-` slots). A stored problem solving episode consists of the successful solution trace augmented with these annotations, i.e., the derivational trace.

In a nutshell, to automatically generate cases as planning episodes, we:

- Identify the decision points in the search procedure where guidance may prove useful to provide memory of the justifications for the choices made.
- Use a clear language to capture these justifications at planning time and associate a meaning so that they can be used at replay time.

Example

We use examples from a logistics transportation domain introduced in [Veloso, 1992]. In this domain packages are to be moved among different cities. Packages are carried within the same city in trucks and between cities in airplanes. At each city there are several locations, e.g., post offices and airports. The problems used in the examples are simple for the sake of a clear illustration of the learning process. Later in the paper we comment briefly on the complexity of this domain and show empirical results where PRODIGY/ANALOGY was tested with complex problems.

Consider the problem illustrated in Figure 2. In this problem there are two objects, `ob4` and `ob7`, one truck `tr9`, and one airplane `pl1`. There is one city `c3` with a post office `p3` and an airport `a3`. In the initial state, `ob4` is at `p3` and the goal is to have `ob4` inside of `tr9`.

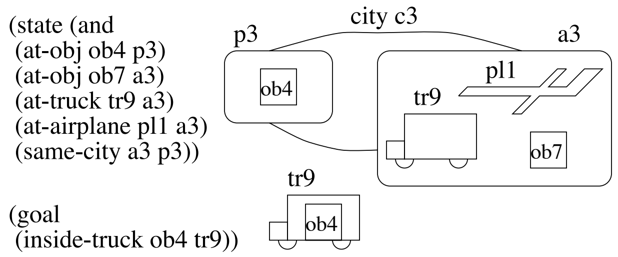


Fig. 2. Example: The goal is to load one object into the truck. Initially the truck is not at the object's location.

The solution to this problem is to drive the truck from the airport to the post office and then load the object.

There are two operators that are relevant for solving this problem.(The complete set of operators can be found in [Veloso, 1992].) The operator `LOAD-TRUCK` specifies that an object can be loaded into a truck if the object and the truck are at the same location, and the operator `DRIVE-TRUCK` states that a truck can move freely between locations within the same city.

Figure 3 (a) shows the decision tree during the search for the solution. Nodes are numbered in the order in which the search space is expanded. The search is a sequence of goal choices followed by operator choices followed occasionally by applying operators to the planner's internal state when their preconditions are true in that state and the decision for immediate application is made.

This trace illustrates PRODIGY handling multiple choices of how to instantiate operators. There are two instantiations of the operator `load-truck` that are relevant to the given goal, i.e., the instantiations `(load-truck ob4 tr9 p3)` and `(load-truck ob4 tr9 a3)` add the goal `(inside-truck ob4 tr9)`. An object can be loaded into a truck at both post office and airport locations. Node `n2` shows that the alternative of loading the truck at the airport `a3` is explored first. This leads to two failed paths. The solution is found after backtracking to the alternative child of node `n1`. Nodes `n8` through `n12` show the final sequence of successful decisions. `n8` shows the correct choice of loading the truck at the post office, where `ob4` is located. The solution corresponds to the two steps applied

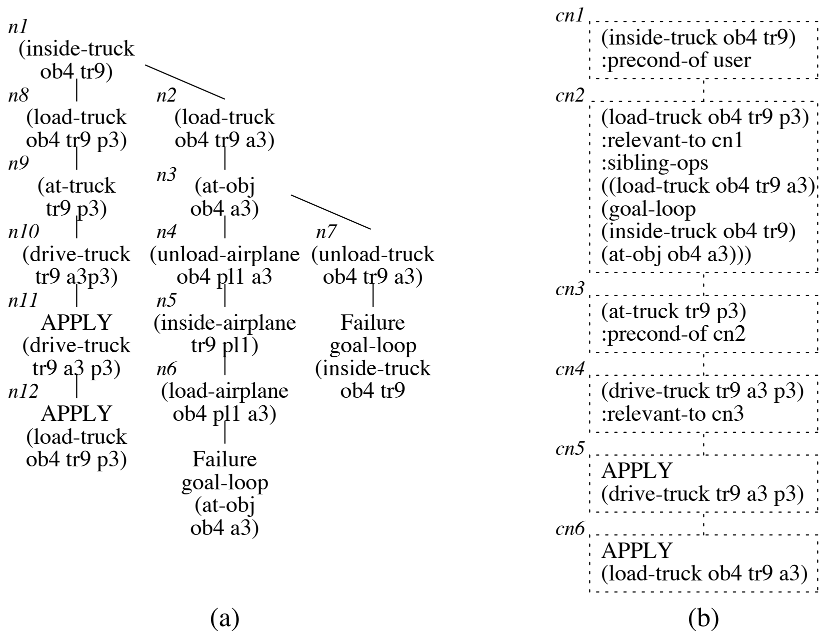


Fig. 3. (a) The search tree to solve the problem in Figure 2 – the numbering of the nodes shows the search order; (b) The corresponding learned problem solving episode to be stored (only a subset of the justifications is shown).

at nodes `n11` and `n12`: the truck `tr9` is driven from `a3` to `p3`, as chosen at node `n8` and then it is loaded with `ob4`.[3]

Figure 3 (b) shows the case generated from the problem solving episode shown in Figure 3 (a). The entire search tree is not stored in the case, but only the decision nodes of the final successful path. The subgoaling structure and the record of the failures are annotated at these nodes. Each goal is a precondition of some operator and each operator is chosen and applied because it is relevant to some goal that needs to be achieved. The failed alternatives are stored with an attached reason of failure.

As an example, node `cn2` corresponds to the search tree node `n8`. This search node has a sibling alternative `n2` which was explored and failed. The failed subtree rooted at `n2` has two failure leaves, namely at `n6` and `n7`. These failure reasons are annotated at the case node `cn2`. At replay time these justifications are tested and may lead to an early pruning of alternatives and constrain possible instantiations.

[3] Note that domain-independent methods to try to reduce the search effort [Stone *et al.*, 1994] in general do not capture domain specific control knowledge, which must be then acquired by learning.

3 Indexing the Problem Solving Cases

PRODIGY/ANALOGY constructs a case from the derivational trace of a problem solving episode. In PRODIGY, a problem is defined by the goal statement and the initial state of the problem situation. A simple indexing scheme may consider directly the goal statement and the complete initial state as indices to the case. This approach may be suited for simple one-goal problems where the initial state is specified with a reduced set of features, but for more complex problem solving situations with multiple goals and a very large number of literals in the initial state, we need to refine this indexing mechanism. This is done in two ways: the initial state is pruned to the set of features relevant to the particular solution, i.e., case, to be learned, and the goal statement is partitioned into conjunctive sets of interacting goals.

From the exploration of the search space and by following the subgoaling links in the derivational trace of the plan generated [Carbonell, 1986], the system identifies, for each goal, the set of *weakest preconditions* necessary to achieve that goal. We recursively create the *foot-print* of a user-given goal conjunct by doing goal regression, i.e. projecting back its weakest preconditions into the literals in the initial state [Mitchell *et al.*, 1986, Waldinger, 1981]. 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]. Foot-printing is similar to explanation-based indexing techniques [Barletta and Mark, 1988, Hickman and Larkin, 1990, Pazzani, 1990, Kambhampati and Kedar, 1991] and chunking [Laird *et al.*, 1986] in that it uses an explanation provided by the subgoaling chain supplied by the underlying domain theory.

The system automatically identifies the sets of interacting goals of a plan by partially ordering the totally ordered solution found [Veloso *et al.*, 1990]. The connected components of the partially ordered plan determine the independent fragments of the case each corresponding to a set of interacting goals. Each case is multiply indexed by these different sets of interacting goals.

When a new problem is presented to the system, the retrieval procedure must match the new initial state and goal statement against the indices of the cases in the case library.

Organizing the case library consists of designing appropriate data structures to store the set of indices such that the set of candidate analogs at retrieval time can be pruned as efficiently as possible. We use two levels of indexing — a hash table and a discrimination network — to store the features in the goal statement and in the initial state shared among the cases. There are many problem solving situations for which the parameterized goal statements are identical and the initial states are different. These different initial states are organized into a discrimination network to index efficiently these cases that completely share the goal statement, but differ in the relevant initial state. Figure 4 sketches the overall organization of the case library illustrated with goals from a logistics transportation domain.

The goals are used in a first level of indexing followed by the discrimination

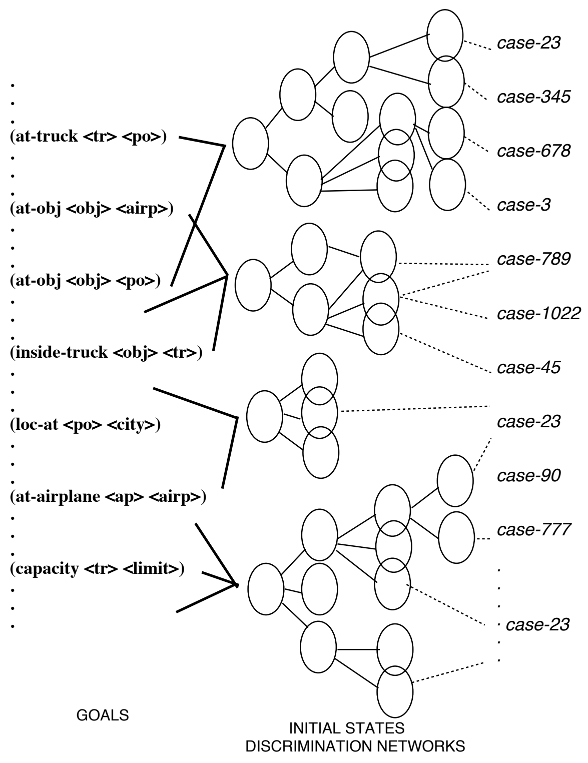


Fig. 4. PRODIGY/ANALOGY's case library organization. The goals at the left are indexed by a hash table (not shown).

network of the initial state. The leaves of this indexing structure point to the cases.

4 Similarity Metric and Retrieval Procedure

Mostly every research project in analogical or case-based reasoning studies the problem of assigning adequate similarity metrics to rank the similarity between a new situation and the past situations. The process is generally recognized to be complex and similarity metrics vary quite considerably, ranging from being more or less context dependent [Bareiss and King, 1989, Gentner, 1987, Kolodner, 1989, Porter *et al.*, 1989, Russell, 1986].

In PRODIGY/ANALOGY the retrieval procedure could simply search for cases to cover each goal conjunct individually. However the interactions among multiple goal conjuncts both in terms of operator choices and operator orderings are responsible for a major part of the problem solving search effort and the quality of the solutions encountered. Furthermore each case stored in the case library is indexed through the corresponding sets of interacting goals. The case library is a source of acquired knowledge of experienced goal interactions. Therefore when trying to retrieve cases to cover a set of goals, the retrieval procedure in PRODIGY/ANALOGY tries to find cases with similar interacting goals instead of choosing separate one-goal cases for each individual goal. We used a similarity metric that accounts for the combined match degree of the interacting goal conjuncts and the corresponding foot-printed initial state.

When assigning a match value to two problems, the interacting foot-printing similarity metric, as introduced in Definition 1, considers not only the *number* of goals and initial state literals that match, but also uses the matched goals themselves to determine the match degree of the initial state. The metric requires that if a case covers multiple goals then these were found in the past to be interacting goals.

Definition 1. Interacting foot-printed similarity metric:
Let P be a new problem and P' be a previously solved problem, respectively with initial states $\mathcal{S}^P$ and $\mathcal{S}^{P'}$, and goals $\mathcal{G}^P$ and $\mathcal{G}^{P'}$. Let $\delta^{\sigma}_{\mathcal{G}}$ be the match value of $\mathcal{G}^P$ and $\mathcal{G}^{P'}$, under substitution σ, such that **the matched goals** $G_1, \ldots, G_m$ **cover completely one or more sets of interacting goals.** Let $\mathcal{S}^{P'}_{fp}$ be the foot-printed initial state of problem P' for the set of matched goals $G_1, \ldots, G_m$. Let $\delta^{\sigma}_{\mathcal{S}}$ be the match value of $\mathcal{S}^P$ and $\mathcal{S}^{P'}_{fp}$, under substitution σ.
The two problems P and P' **interactively foot-print match** with **match value** $\boldsymbol{\delta^{\sigma} = \delta^{\sigma}_{\mathcal{G}} + \delta^{\sigma}_{\mathcal{S}}}$ for substitution σ.

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. The similarity metric captures 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. When the foot-printed literals are taken into account for the measure of the similarity among problems, the retrieved analogs provide expected adequate guidance at replay time, as the foot-printed initial state is in the subgoaling chain of the goal statement in the particular solution to be replayed. If the new situation shares some of these features, the problem solver encounters the same or parts of the past search space. The case may not be fully-sufficient due to the partial match, but, because of the shared foot-printed literals of the initial state, the case does not work against the goal, except for unexpected or uncovered goal interactions. These will be new learning opportunities to compile new cases to store and reuse.

Figure 5 shows the retrieval procedure where the underlying strategy is to get guidance from cases that cover the largest possible set of interacting goals. The algorithm focuses on retrieving past cases where the problem solver experienced equivalent goal interactions, as these are expectedly responsible for a large part of the problem solving search effort.

Initially step 1 sets the number of goals that the algorithm tries to cover simultaneously, i.e., *no_int_goals*, to the total number k of goal conjuncts. All the conjunctive goals are also declared uncovered. A goal remains uncovered

```
Input  : A new problem with goal statement $\mathcal{G} = G_1, G_2, \ldots, G_k$ and initial state $\mathcal{S}$.
Output  : A set of similar cases.

procedure **Retrieve_Similar_Cases** ($\mathcal{G}$, $\mathcal{S}$):
1.  covering_cases ← ∅; no_int_goals ← k; uncovered_goals ← $\mathcal{G}$;
2.  past_case ← nil; continue_retrieval_p ← true
3.  while uncovered_goals or continue_retrieval_p
4.    past_case ← Find_Another_Analog (no_int_goals, uncovered_goals, past_case)
5.    if past_case then
6.        (matched_goals, goal_substitution) ← Match_Goals (past_case, $\mathcal{G}$)
7.        (similarity_value, total_substitution) ←
            ← Match_Initial_States (past_case,matched_goals, goal_substitution, $\mathcal{S}$)
8.        if Satisfied_with_Match (similarity_value) then
9.            uncovered_goals ← uncovered_goals \ matched_goals
10.           covering_cases ← covering_cases ∪ {past_case}
11.   if (no_int_goals > 1) and (number of uncovered_goals <= no_int_goals) then
12.       no_int_goals ← Decrease_Interacting_Scope (no_int_goals, uncovered_goals)
13.   if Stop_Retrieval_p (past_case, uncovered_goals, no_int_goals)
14.       then continue_retrieval_p ← nil
15.Return covering_cases
```

Fig. 5. Retrieving similar past cases

throughout the procedure until a case is found that covers it. The procedure **Find_Another_Analog** at step 4 incrementally searches in the case library for a case that covers *no_int_goals* many of the still uncovered goals. This algorithm uses efficient data structures to perform an incremental generation of candidate analogs. Suppose that a case is returned. Then steps 6 and 7 evaluate the similarity value between the new and past situations. The goals are considered covered at step 9, if the procedure is satisfied with the match value as determined at step 8. Step 10 adds the *past_case* to the list of *covering_cases*. Step 11-14 establish the termination and continuation conditions. In particular, the procedure at step 12 decreases the scope of interactions to be considered according to the number of remaining *uncovered_goals* and no more interactions of size *no_int_goals* are found. The retrieval effort may be interrupted by the procedure **Stop_Retrieval_p** when a threat is recognized in its potential benefits with respect to problem solving search savings (see [Veloso and Carbonell, 1993b]).

5 Flexible Replay of Multiple Guiding Cases

When a new problem is proposed, PRODIGY/ANALOGY retrieves from the case library one or more problem solving episodes that may partially cover the new problem solving situation. The system uses a similarity metric that weighs goal-relevant features [Veloso and Carbonell, 1993b]. In a nutshell, it selects a set of past cases that solved subsets of the new goal statement. The initial state is partially matched in the features that were relevant to solving these goals in the past. Each retrieved case provides guidance to a set of interacting goals from the new goal statement. At replay time, a guiding case is always considered as a source of guidance, until all the goals it covers are achieved.

The general replay mechanism involves a complete interpretation of the justification structures annotated in the past cases in the context of the new problem to be solved. Equivalent choices are made when the transformed justifications hold. When that is not the situation, PRODIGY/ANALOGY plans for the new goals using its domain operators adding new steps to the solution or skipping unnecessary steps from the past cases. Table 1 shows the main flow of control of the replay algorithm.

```
1. Terminate if the goal is satisfied in the state.
2. Choose a step from the set of guiding cases or decide if there is need for additional problem solving work. If a failure is encountered, then backtrack and continue following the guiding cases at the appropriate steps.
3. If a goal from a past case is chosen, then
    3.1 Validate the goal justifications. If not validated, go to step 2.
    3.2 Create a new goal node; link it to the case node. Advance the case to its next decision step.
    3.3 Select the operator chosen in the case.
    3.4 Validate the operator and bindings choices. If not validated, base-level plan for the goal. Use justifications and record of failures to make a more informed new selection. Go to step 2.
    3.5 Link the new operator node to the case node. Advance the case to its next decision step.
    3.6 Go to step 2.
4. If an applicable operator from a past case is chosen, then
    4.1 Check if it can be applied also in the current state. If it cannot, go to step 2.
    4.2 Link the new applied operator node to the case node. Advance the case to its next decision step.
    4.3 Apply the operator.
    4.4 Go to step 1.
```

Table 1. The main flow of control of the replay procedure.

The replay functionality transforms the planner, from a module that costly generates possible operators to achieve the goals and searches through the space of alternatives generated, into a module that tests the validity of the choices proposed by past experience and follows equivalent search directions. The replay procedure provides the following benefits to the problem solving procedure as shown in the procedure of Table 1.

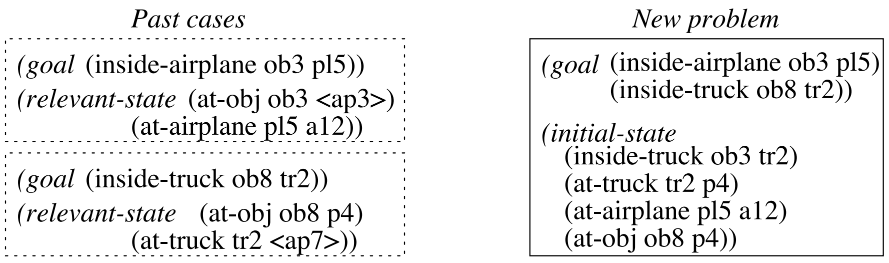
- Proposal and validation of choices versus generation and search of alternatives (steps 2, 3.1, 3.3, 3.4, and 4.1).

- Reduction of the branching factor – past failed alternatives are pruned by validating the failures recorded in the past cases (step 3.4); if backtracking is needed PRODIGY/ANALOGY backtracks also in the guiding cases – through the links established at steps 3.2, 3.5 and 4.2 – and uses information on failure to make more informed backtracking decisions.
- Subgoaling links identify the subparts of the case to replay – the steps that are not part of the active goals are skipped. The procedure to advance the cases, as called in steps 3.2, 3.5 and 4.2, ignores the goals that are not needed and their corresponding planning steps.

PRODIGY/ANALOGY constructs a new solution from a set of guiding cases as opposed to a single past case. Complex problems may be solved by resolving minor interactions among simpler past cases. However, following several cases poses an additional decision making step of choosing which case to pursue. We explored several strategies to merge the guidance from the set of similar cases. In the experiments from which we drew the empirical results presented below, we used an exploratory merging strategy. Choices are made arbitrarily when there is no other guidance available. This strategy allows an innovative exploration of the space of possible solutions leading to opportunities to learn from new goal interactions or operator choices.

5.1 Example

Figure 6 shows a new problem and two past cases selected for replay. The cases are partially instantiated to match the new situation. Further instantiations occur while replaying.


Past cases

(goal (inside-airplane ob3 pl5))
(relevant-state (at-obj ob3 <ap3>)
(at-airplane pl5 a12))

(goal (inside-truck ob8 tr2))
(relevant-state (at-obj ob8 p4)
(at-truck tr2 <ap7>))

New problem

(goal (inside-airplane ob3 pl5)
(inside-truck ob8 tr2))
(initial-state
(inside-truck ob3 tr2)
(at-truck tr2 p4)
(at-airplane pl5 a12)
(at-obj ob8 p4))

Fig. 6. Instantiated past cases cover the new goal and partially match the new initial state. Some of the case variables are not bound by the match of the goals and state.

Figure 7 shows the replay episode to generate a solution to the new problem. The new situation is shown at the right side of the figure and the two past guiding cases at the left.

The transfer occurs by interleaving the two guiding cases, performing any additional work needed to accomplish remaining subgoals, and skipping past work that does not need to be done. In particular, the case nodes `cn3'` through

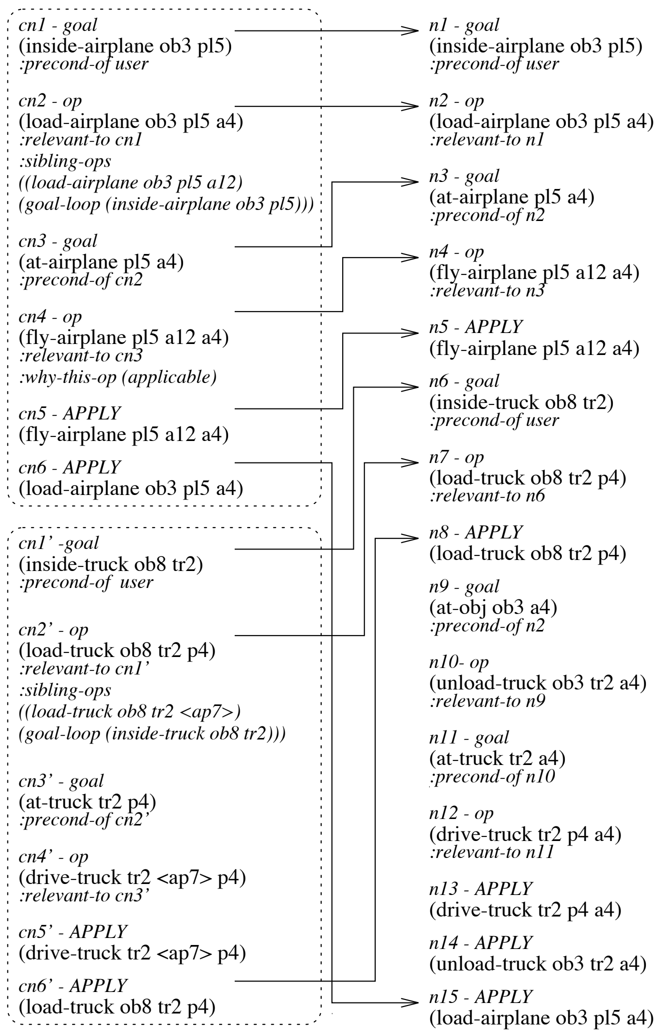


Fig. 7. Derivational replay of multiple cases.

`cn5'` are not reused, as there is a truck already at the post office in the new problem. The nodes `n9-14` correspond to unguided additional planning done in the new episode.[4] At node `n7`, PRODIGY/ANALOGY prunes out an alternative operator, namely to load the truck at any airport, because of the recorded past failure at the guiding node `cn2'`. The recorded reason for that failure, namely a goal-loop with the `(inside-truck ob8 tr2)`, is validated in the new situation, as that goal is in the current set of open goals, at node `n6`. Note that the two cases are merged using a bias to postpone additional planning needed. Different merges are possible.

[4] Note that extra steps may be inserted at any point, interrupting and interleaving the past cases, and not just at the end of the cases.

6 Empirical Results

We ran and accumulated in the case library a set of 1000 problems in the logistics transportation domain. In the experiments the problems are randomly generated with up to 20 goals and more than 100 literals in the initial state. The case library is accumulated incrementally while the system solves problems with an increasing number of goals. (Details on the exact set up of the experiments can be found in [Veloso, 1992].)

The logistics transportation is a complex domain. In particular, there are multiple operator and bindings choices for each particular problem, and those choices increase considerably with the size or complexity of the problem. For example, for the goal of moving an object to an airport, the problem solver does not have direct information from the domain operators on whether it should move the object inside of a truck or an airplane. Objects can be unloaded at an airport from both of these carriers, but trucks move within the same city and airplanes across cities. The specification of these constraints is embedded in the domain knowledge and not directly available. PRODIGY/ANALOGY provides guidance at these choices of operators and bindings through the successful and failed choices annotated in past similar problem solving episodes.

PRODIGY/ANALOGY increases the solvability horizon of the problem solving task: Many problems that NoLimit cannot solve within a reasonable time limit are solved by PRODIGY/ANALOGY within that limit. Figure 8 (a) plots the number of problems solved by NoLimit and PRODIGY/ANALOGY for different CPU time bounds. NoLimit solves 458 problems out of the 1000 problems even when the search time limit is increased up to 350s, while PRODIGY/ANALOGY solves the 1000 problems within the same CPU time limit.

This graph shows a significant improvement achieved by solving problems by analogy with previously solved problems. Although not shown in this figure, the percentage of problems solved without analogy decreases rapidly with the complexity of the problems. The gradient of the increase in the performance of PRODIGY/ANALOGY over the base-level NoLimit shows its large advantage when increasing the complexity of the problems to be solved. Figure 8 (b) shows the cumulative running time for the total set of problems. (For each unsolved problem, the running time bound is added.)

We also compiled results on the length of the solutions generated by PRODIGY/ANALOGY and on the impact of the size of the case library in the retrieval time [Veloso, 1992]. We concluded that PRODIGY/ANALOGY produces solutions of equal or shorter length in 92% of the problems. PRODIGY/ANALOGY includes an indexing mechanism for the case library of learned problem solving episodes [Veloso and Carbonell, 1993b]. We verified that with this memory organization, we reduced (or avoided) the potential utility problem [Doorenbos and Veloso, 1993]: The retrieval time suffers no significant increase with the size of the case library.

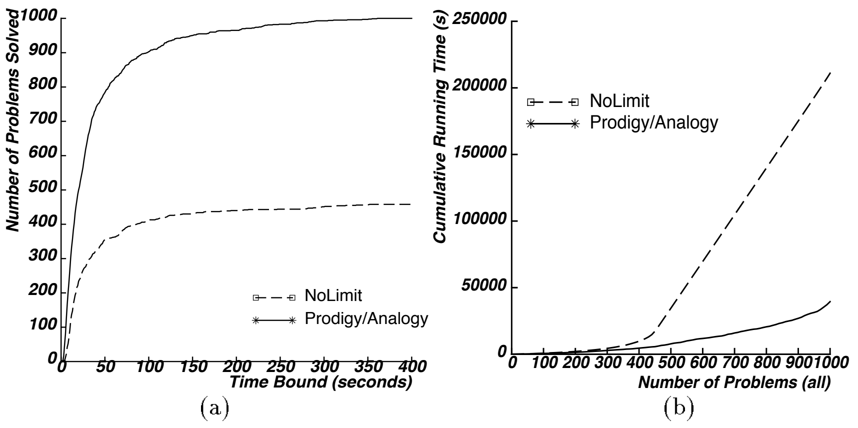


Fig. 8. (a) Number of problems solved from a set of 1000 problems versus different running time bounds. With a time limit of 350s NoLimit solves only 458 problems, while PRODIGY/ANALOGY solves the complete set of 1000 problems; (b) Cumulative running times for all the 1000 problems. The problems unsolved by NoLimit count as the maximum time limit given (350s).

7 Discussion and related work

PRODIGY's problem solving method is a combination of means-ends analysis, backward chaining, and state-space search. PRODIGY commits to particular choices of operators, bindings, and step orderings as its search process makes use of a uniquely specified state while planning [Fink and Veloso, 1994]. PRODIGY's learning opportunities are therefore directly related to the choices found by the problem solver in its state-space search. It is beyond the scope of this paper to discuss what are the potential advantages or disadvantages of our problem solving search method in particular compared with other planners that search a plan space. Any system that treats planning and problem solving as a search process will make a series of commitments during search. The pattern of commitments made will produce greater efficiency in some kinds of domains and less in others [Stone *et al.*, 1994]. The goal of strategy learning is precisely to *automate* the process of acquiring operational knowledge to improve the performance of a particular base-level problem solving reasoning strategy. Each particular problem solver may find different learning opportunities depending on its reasoning and searching strategies. However, the following aspects of this work may apply to other problem solvers: learning a chain of justified problem solving decisions as opposed to individual ones or final solutions; and flexibly replaying multiple complementary learned knowledge in similar situations as opposed to identical ones.

This work is related to other plan reuse work in the plan-space search paradigm, in particular [Kambhampati and Hendler, 1992]. In that framework, it proved beneficial to reuse the final plans annotated with a validation structure that links the goals to the operators that achieve each goal. In PRODIGY/ANALOGY we learn and replay the planner's decision making process directly. The justification structures in the derivational traces also encompass the record of past failures in addition to the subgoaling links as in [Mostow, 1989, Blumenthal, 1990, Kambhampati and Hendler, 1992, Bhansali and Harandi, 1993, Paulokat and Wess, 1994]. The derivational traces provide guidance for the choices that our problem solver faces while constructing solutions to similar problems. Adapted decisions can be interleaved and backtracked upon within the replay procedure.

Learning by analogy can also be related to other strategies to learn control knowledge. In particular analogical reasoning in PRODIGY can be seen as relaxing the restrictions to explanation-based approaches as developed in PRODIGY [Minton, 1988, Etzioni, 1993]. Instead of requiring complete axiomatic domain knowledge to derive general rules of behavior for individual decisions, PRODIGY/ANALOGY compiles annotated traces of solved problems with little post processing. The learning effort is done incrementally on an "if-needed" basis at storage, retrieval and adaptation time. The complete problem solving episode is interpreted as a global decision-making experience and independent subparts can be reused as a whole. PRODIGY/ANALOGY can replay partially matched learned experience increasing therefore the transfer of potentially over-specific learned knowledge. Chunking in SOAR [Laird *et al.*, 1986] also accumulates episodic global knowledge. However, the selection of applicable chunks is based on choosing the ones whose conditions match totally the active context. The chunking algorithm in SOAR can learn interactions among different problem spaces.

Analogical reasoning in PRODIGY/ANALOGY learns complete sequences of decisions as opposed to individual rules. Under this perspective analogical reasoning shares characteristics with learning macro-operators [Yang and Fisher, 1992]. Intermediate decisions corresponding to choices internal to each case can be bypassed or adapted when their justifications do not longer hold. Furthermore cases cover complete problem solving episodes and are not proposed at local decisions as search alternatives to one-step operators.

8 Conclusion

Reasoning by analogy in PRODIGY/ANALOGY consists of the flexible reuse of derivational traces of previously solved problems to guide the search for solutions to similar new problems. The issues addressed in the paper include: the generation of problem solving cases for reuse, and the flexible replay of possibly multiple learned episodes in situations that partially match new ones. The paper shows results that empirically validate the method and demonstrate that PRODIGY/ANALOGY is amenable to scaling up both in terms of domain and problem complexity.

References

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

[Barletta and Mark, 1988] Ralph Barletta and William Mark. Explanation-based indexing of cases. In *Proceedings of the First Workshop on Case-Based Reasoning*, pages 50–60, Tampa, FL, May 1988. Morgan Kaufmann.

[Bhansali and Harandi, 1993] Sanjay Bhansali and Mehdi T. Harandi. Synthesis of UNIX programs using derivational analogy. *Machine Learning*, 10, 1993.

[Blumenthal, 1990] Brad Blumenthal. *Replaying episodes of a metaphoric application interface designer.* PhD thesis, University of Texas, Artificial Intelligence Lab, Austin, December 1990.

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

[Carbonell *et al.*, 1992] Jaime G. Carbonell, Jim Blythe, Oren Etzioni, Yolanda Gil, Robert Joseph, Dan Kahn, Craig Knoblock, Steven Minton, Alicia Pérez, Scott Reilly, Manuela Veloso, and Xuemei Wang. PRODIGY4.0: The manual and tutorial. Technical Report CMU-CS-92-150, Carnegie Mellon University, June 1992.

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

[Doorenbos and Veloso, 1993] Robert B. Doorenbos and Manuela M. Veloso. Knowledge organization and the utility problem. In *Proceedings of the Third International Workshop on Knowledge Compilation and Speedup Learning*, pages 28–34, Amherst, MA, June 1993.

[Etzioni, 1993] Oren Etzioni. Acquiring search-control knowledge via static analysis. *Artificial Intelligence*, 62(2):255–301, 1993.

[Fink and Veloso, 1994] Eugene Fink and Manuela Veloso. PRODIGY planning algorithm. Technical Report CMU-CS-94-123, School of Computer Science, Carnegie Mellon University, 1994.

[Gentner, 1987] Dedre Gentner. The mechanisms of analogical learning. In S. Vosniadou and A. Ortony, editors, *Similarity and Analogical Reasoning.* Cambridge University Press, New York, NY, 1987.

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

[Kambhampati and Hendler, 1992] Subbarao Kambhampati and James A. Hendler. A validation based theory of plan modification and reuse. *Artificial Intelligence*, 55(2-3):193–258, 1992.

[Kambhampati and Kedar, 1991] Subbarao Kambhampati and Smadar Kedar. Explanation based generalization of partially ordered plans. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 679–685, 1991.

[Kolodner, 1989] Janet Kolodner. 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, May 1989.

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

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

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

[Mostow, 1989] Jack Mostow. Automated replay of design plans: Some issues in derivational analogy. *Artificial Intelligence*, 40(1-3), 1989.

[Paulokat and Wess, 1994] Juergen Paulokat and Stefan Wess. Planning for machining workpieces with a partial-order, nonlinear planner. In *Working notes of the AAAI Fall Symposium on Planning and Learning: On to Real Applications*, November 1994.

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

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

[Russell, 1986] Stuart J. Russell. *Analogical and Inductive Reasoning.* PhD thesis, Stanford University, 1986.

[Stone *et al.*, 1994] Peter Stone, Manuela Veloso, and Jim Blythe. The need for different domain-independent heuristics. In *Proceedings of the Second International Conference on AI Planning Systems*, pages 164–169, June 1994.

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

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

[Veloso and Carbonell, 1993b] Manuela M. Veloso and Jaime G. Carbonell. Towards scaling up machine learning: A case study with derivational analogy in PRODIGY. In S. Minton, editor, *Machine Learning Methods for Planning*, pages 233–272. Morgan Kaufmann, 1993.

[Veloso *et al.*, 1990] Manuela M. Veloso, M. Alicia Pérez, and Jaime G. Carbonell. Nonlinear planning with parallel resource allocation. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, pages 207–212, San Diego, CA, November 1990. Morgan Kaufmann.

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

[Veloso, 1992] Manuela M. Veloso. *Learning by Analogical Reasoning in General Problem Solving.* PhD thesis, School of Computer Science, Carnegie Mellon University, August 1992. Available as technical report CMU-CS-92-174. A revised version of this manuscript will be published by Springer Verlag, 1994.

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

[Yang and Fisher, 1992] Hua Yang and Douglas Fisher. Similarity-based retrieval and partial reuse of macro-operators. Technical Report CS-92-13, Department of Computer Science, Vanderbilt University, 1992.

This article was processed using the LaTeX macro package with LLNCS style