In Proceedings of AAAI-94, pgs. 595-600

Flexible Strategy Learning: Analogical Replay of Problem Solving Episodes*

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. Learning occurs by the generation and replay of annotated derivational traces of problem solving episodes. The problem solver is extended with the ability to examine its decision cycle and accumulate knowledge from the chains of successes and failures encountered during its search experience. Instead of investing substantial effort deriving general rules of behavior to apply to individual decisions, the analogical reasoner compiles complete problem solving cases that are used to guide future similar situations. Learned knowledge is flexibly applied to new problem solving situations even if only a partial match exists among problems. We relate this work with other alternative strategy learning methods, and also with plan reuse. We demonstrate the effectiveness of the analogical replay strategy by providing empirical results on the performance of a fully implemented system, PRODIGY/ANALOGY, accumulating and reusing a large case library in a complex problem solving domain.

Introduction

The machine learning approaches to acquiring strategic knowledge typically start with a general problem solving engine and accumulate experience by analyzing its search episodes. The strategic or control knowledge acquired can take many forms, including macro-operators (Fikes & Nilsson 1971; Korf 1985), refined operational operators (DeJong & Mooney 1986; Mitchell, Keller, & Kedar-Cabelli 1986), generalized chunks of all decisions taken by the problem solver (Laird, Rosenbloom, & Newell 1986), explicit control rules that guide the selection of domain-level subgoals and operators (Minton 1988), annotated or validated final solutions (Hammond 1986; Mostow 1989; Kambhampati & Hendler 1992), or justified derivational traces of the decision making process during search, as presented in this paper.

We integrated learning by analogy into general problem solving. The learned knowledge is acquired and used flexibly: its construction results from a direct and simple explanation of the episodic situation, and it is proposed to be used also in situations where there is a partial match for the relevant parts of its applicability conditions.

The method is based on derivational analogy (Carbonell 1986) and it has been fully implemented within the PRODIGY planning and learning architecture, in PRODIGY/ANALOGY. It casts the strategy-level learning process as the automation of the complete cycle of constructing, storing, retrieving, and reusing problem solving experience (Veloso 1992).

In this paper we focus on presenting the learning techniques for the acquisition and reuse of problem solving episodes by analogical reasoning. We illustrate the method with examples. We also provide empirical results showing that PRODIGY/ANALOGY is amenable to scaling up both in terms of domain and problem complexity.

The contributions of this work include: the 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 learner to solve complex problems after being trained in solving simple problems.

Generation of Problem Solving Episodes

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. Derivational analogy instead 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 cy-

*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.

cle, 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.[1] 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'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 & 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

Figure 1: Justification record structure. Nodes are instantiated at decision points during problem solving. Each learned episode is a sequence of such 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.

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`.

[1] NOLIMIT was succeeded by the current planner, PRODIGY4.0 (Carbonell *et al.* 1992; Fink & Veloso 1994).

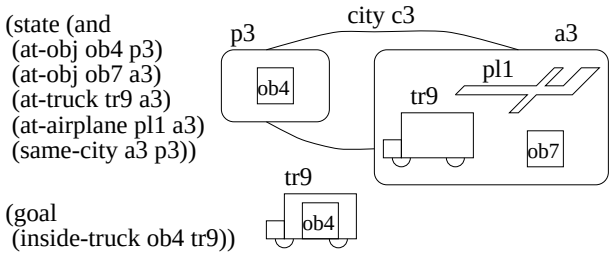


Figure 2: Example: The goal is to load one object into the truck. Initially the truck is not at the object's location.

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.

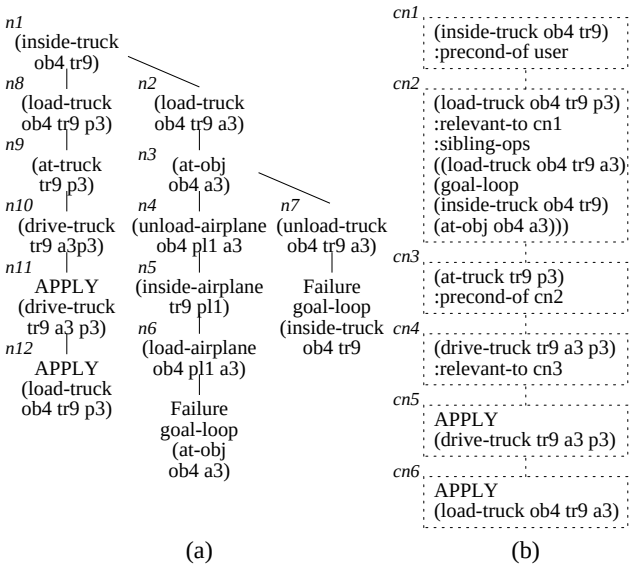


Figure 3: (a) The search tree to solve the problem in Figure 2 – the numbering of the nodes shows the search order; (b) The corresponding learned problem solving episode to be stored (only a subset of the justifications is shown).

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 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.[2]

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.

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 & 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.

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.

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.

- 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

[2] Note that domain-independent methods to try to reduce the search effort (Stone, Veloso, & Blythe 1994) in general do not capture domain specific control knowledge, which must be then acquired by learning.

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.

Example

Figure 4 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.

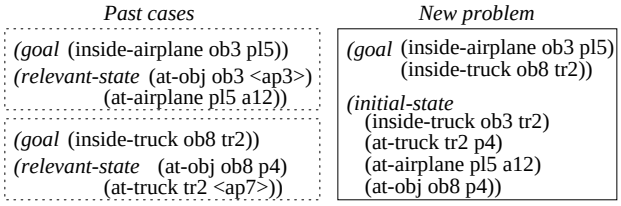


Figure 4: Instantiated past cases cover the new goal and partially match the new initial state. Some of the case variables are not bound by the match of the goals and state.

Figure 5 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 `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.[3] 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.

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.

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,

[3] 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.

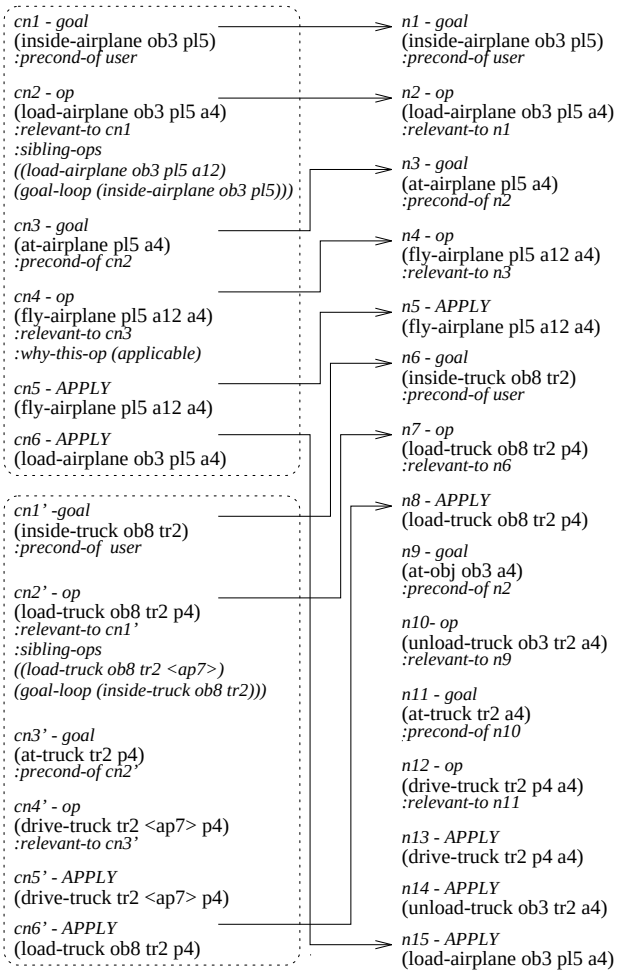
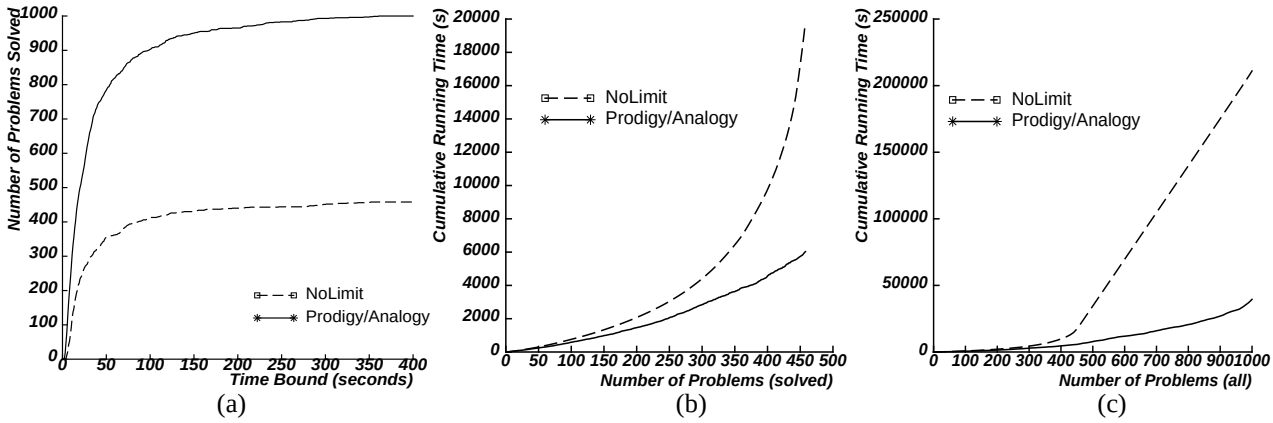


Figure 5: Derivational replay of multiple cases.

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. So if the object must go to an airport within the same city where it is, it should be moved in a truck, otherwise it should be moved in an airplane. The specification of these constraints is embedded in the domain knowledge and not directly available. The city at which the object is located is not immediately known, as when the object is inside of a carrier or a building, its city location is specified indirectly. 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 6 (a) plots the number of problems solved by NOLIMIT and PRODIGY/ANALOGY for different CPU time bounds. NOLIMIT solves only 458 problems out of the 1000 problems even when the search time limit is increased up to 350s.



(a) (b) (c)

Figure 6: (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 the 458 problems solved by both configurations; (c) Cumulative running times for all the 1000 problems. The problems unsolved by NOLIMIT count as the maximum time limit given (350s).

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.

In other previous work, comparisons between the performance of a problem solver before and after learning control knowledge were done by graphing the cumulative running times of the two systems over a set of problems. Figure 6 (b) shows the cumulative running time for the set of problems (458) that were both solved by both configurations. The graph shows a final factor of 3.6 cumulative speed up of PRODIGY/ANALOGY over the base NOLIMIT. (The maximum individual speed up was of a factor of approximately 40.) In Figure 6 (c) we extend this comparison to account also for the unsolved problems (similarly to what was done in previous comparisons (Minton 1988)). For each unsolved problem, we add the running time bound.

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 & Carbonell 1993b). We verified that with this memory organization, we reduced (or avoided) the potential utility problem (Doorenbos & Veloso 1993): The retrieval time suffers no significant increase with the size of the case library.

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 & 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, Veloso, & Blythe 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 & 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 & Hendler 1992; Bhansali & Harandi 1993). 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, Rosenbloom, & Newell 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 & 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.

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.

Acknowledgements Special thanks to Jaime Carbonell for his guidance, suggestions, and discussions on this work. Thanks also to Alicia Pérez and the anonymous reviewers for their helpful comments on this paper.

References

Bhansali, S., and Harandi, M. T. 1993. Synthesis of UNIX programs using derivational analogy. *Machine Learning* 10.

Blumenthal, B. 1990. *Replaying episodes of a metaphoric application interface designer*. Ph.D. Dissertation, University of Texas, Artificial Intelligence Lab, Austin.

Carbonell, J. G.; Blythe, J.; Etzioni, O.; Gil, Y.; Joseph, R.; Kahn, D.; Knoblock, C.; Minton, S.; Pérez, A.; Reilly, S.; Veloso, M.; and Wang, X. 1992. PRODIGY4.0: The manual and tutorial. Technical Report CMU-CS-92-150, Carnegie Mellon University.

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., eds., *Machine Learning, An Artificial Intelligence Approach, Volume II*, 371–392. Morgan Kaufman.

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

Doorenbos, R. B., and Veloso, M. M. 1993. Knowledge organization and the utility problem. In *Proceedings of the Third International Workshop on Knowledge Compilation and Speedup Learning*, 28–34.

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

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.

Fink, E., and Veloso, M. 1994. PRODIGY planning algorithm. Technical Report CMU-CS-94-123, School of Computer Science, Carnegie Mellon University.

Hammond, K. J. 1986. *Case-based Planning: An Integrated Theory of Planning, Learning and Memory*. Ph.D. Dissertation, Yale University.

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

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

Laird, J. E.; Rosenbloom, P. S.; and Newell, A. 1986. Chunking in SOAR: The anatomy of a general learning mechanism. *Machine Learning* 1:11–46.

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

Mitchell, T. M.; Keller, R. M.; and Kedar-Cabelli, S. T. 1986. Explanation-based generalization: A unifying view. *Machine Learning* 1:47–80.

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

Stone, P.; Veloso, M.; and Blythe, J. 1994. The need for different domain-independent heuristics. In *Proceedings of the Second International Conference on AI Planning Systems*, 164–169.

Veloso, M. M., and Carbonell, J. G. 1993a. Derivational analogy in P<RODIGY: Automating case acquisition, storage, and utilization. *Machine Learning* 10:249–278.

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

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

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