Machine Learning 1: 249–286, 1986

Experimental Goal Regression: A Method for Learning Problem-Solving Heuristics

BRUCE W. PORTER (PORTER@UTEXAS)
Computer Sciences Department, University of Texas at Austin, Austin, TX 78712, U.S.A.

DENNIS F. KIBLER (KIBLER@CIP.UCI.EDU)
Irvine Computational Intelligence Project, Department of Information and Computer Science, University of California, Irvine, CA 92717, U.S.A.

(Received December 30, 1985)

Key words: goal regression, problem solving heuristics, example generation

Abstract. This research examines the process of learning problem solving with minimal requirements for *a priori* knowledge and teacher involvement. Experience indicates that knowledge about the problem solving task can be used to improve problem solving performance. This research addresses the issues of what knowledge is useful, how it is applied during problem solving, and how it can be acquired. For each operator used in the problem solving domain, knowledge is incrementally learned concerning why it is useful, when it is applicable, and what transformation it performs. The method of *experimental goal regression* is introduced for improving the learning rate by approximating the results of analytic learning. The ideas are formalized in an algorithm for learning and problem solving and demonstrated with examples from the domains of simultaneous linear equations and symbolic integration.

1. Introduction

Although the earliest successful research in AI was on problem solving (Newell and Simon, 1961) and learning (Samuel, 1959) the symbiotic relationship between the two areas was not exploited until recently (Anzai, 1978; Brazdil, 1978; Neves, 1978). This paper presents a learning method for automating the acquisition of knowledge for problem solving which requires few instances, minimal teacher involvement, and only a weak theory of the domain. A major issue in this research is the integration of learning and problem solving. This integration permits the learner to use the problem solver as an evaluator and reduces both the role of the teacher and the requirement for *a priori* knowledge.

Assumption of a highly cooperative teacher or extensive *a priori* knowledge limits the applicability of machine learning algorithms. In particular, within the context of improving problem solving performance by learning rules or heuristics for guiding the choice of an appropriate operator or sequence of operators, we have found that many common assumptions are unnecessary. More specifically, our learning algorithm does not rely on the following:

- a benevolent teacher to carefully order and select training examples;
- extensive searches of the problem state-space in order to assign credit or blame to particular operations;
- a large number of training examples to guide an induction algorithm;
- a static evaluation function to evaluate the quality of states in the problem state-space;
- a STRIPS-like "transparent" representation of operators.
- definitions of operator inverses.

To a large extent, our learning method reduces reliance on the teacher and *a priori* knowledge by a technique that we call *experimental goal regression*. Experimental goal regression (EGR) approximates the results of analytic goal regression, which is an important technique for explanation-based learning (Mitchell, Keller and Kedar-Cabelli, 1986). The technique enables back-propagation of goal conditions to guide the learning of search heuristics. This improves the learning rate while reducing reliance on the teacher to provide training.

Analytic and experimental goal regression are both knowledge intensive learning techniques. One essential difference between the two is that the analytic approach relies on a strong *a priori* domain theory, while experimental goal regression requires little *a priori* knowledge and is guided by the knowledge acquired during the normal process of learning. EGR requires a few training examples from the teacher and an *a priori* concept hierarchy which decomposes the examples into features and relates features to generalized categories.

PET is a Prolog implementation of this learning method, and has been applied to learning to solve simultaneous linear equations and problems in symbolic integration. PET, named for the idealized student who learns conscientiously and forgets nothing, has allowed us to test our learning method and to compare it with other learning methods applied to similar tasks.

2. Overview of the learning system

PET can be viewed as a *learning apprentice system*, which Mitchell, Mahadevan, and Steinberg (1985) have defined as:

"... a knowledge-based system that provides *interactive* aid in solving some problem, and that acquires new domain knowledge by generalizing from training examples acquired through the *normal course* of its use."

PET consists of a user-interface, a simple problem solver, and a learner. A problem domain for PET is represented as a state-space. For each problem domain, there is a fixed set of operators and a fixed goal. During the solution of a problem, the user can suggest operators to apply. The problem solver is able to apply operators to problems and to detect goal states. The learner forms heuristic rules from the user's suggestions, and these are used to guide the problem solver on subsequent problems. A high-level description of the flow of control in PET is:

```
LOOP UNTIL teacher satisfied
  Get problem from teacher
  LOOP UNTIL problem solved
    IF learned knowledge applies to the current problem
    THEN apply it (no learning)
    ELSE
      Ask teacher for advice
      Learn from advice (if possible)
      Integrate new knowledge
      Apply operator
  REPEAT
REPEAT
```

Notice that PET learns only when it cannot decide what to do next. At such an indecision point, PET asks for advice from the teacher. If PET is unable to understand why the advice is useful, then the advice is followed but nothing is learned or recorded. As with other inductive learning algorithms, PET generalizes the specific advice given by the teacher to extend its applicability.

The remainder of the paper describes the knowledge that PET learns, the mechanisms for acquiring the knowledge, and the use of the knowledge to guide further learning. For clarity, we introduce the learning components incrementally, although the PET system is an integration of the components. For each component, we present algorithms which expand upon the high-level description given above.

Section 3 describes *episodic learning*, which segments a sequence of operator applications into useful subsequences. These subsequences, or episodes, are used to guide subsequent problem solving and learning. Learned episodes guide problem solving by (effectively) introducing macro operators. Episodes guide learning by providing explicit solution paths to goals.

Section 4 describes *perturbation*, which is used to generalize the rules formed by episodic learning. Perturbation is an automatic example generation technique. Given a single training example from the teacher, perturbation generates near-examples by

making small changes to the original "seed." The problem solver reapplies the teacher's suggested operator to the new examples and classifies each of the generated examples as positive or negative. An example is positive if the operator suggested for the original problem begins a successful solution of the generated problem. The set of positive examples is generalized to form a heuristic for suggesting the operator. The main contribution of perturbation for learning search heuristics is the substantial reduction in the role of the teacher.

Section 5 describes *relational modeling*, which is a technique for learning a STRIPS-like operator description given a procedural operator representation. STRIPS-like, relational descriptions are useful because they make explicit the transformation performed by the operator. In fact, a relational model is a more detailed representation of an operator than provided by a STRIPS-like definition in that it hypothesizes linkages between the PRE and POST conditions of the operator. While relational descriptions are extremely useful for learning and problem solving, PET learns these representations rather than assume that they are *a priori* knowledge.

Section 6 describes *experimental goal regression*, which is the integration of these techniques into an efficient learning and problem solving system. The role of experimental goal regression is to associate constraints from goal states with partially learned subgoal descriptions and to use these constraints to guide perturbation in generalizing search heuristics. Experimental goal regression relies on the problem solving paths which are made explicit in learned episodic segments and relational models.

Section 7 is a brief comparison of this learning method with both analytic and empirical learning techniques. Section 8 summarizes our findings and identifies three limitations of this (and other) research on learning problem solving.

3. Episodic learning

Episodic learning (Kibler and Porter, 1983b) is a technique for acquiring and ordering heuristic rules so that they recommend operator sequences for problem solving. Episodes learned from the solution of a problem are used to improve subsequent problem solving and learning. This section describes the episodic learning method and its implementation in PET.

Many researchers have recognized the need for macro operators. MACROPS (Fikes and Nilsson, 1971) remembers and generalizes robot plans generated by the STRIPS planning system so that they can be reused. Unfortunately, the operators in a MACROPS plan are not segmented into meaningful sequences. Any sequence of operators can be extracted from the plan and used as a macro. Since the most useful sequences are not identified, MACROPS suffers from a combinatorial explosion of (macro) operators to apply.

Recent research has addressed the issue of selective learning of macros. In the con-

text of a problem solver guided by a heuristic evaluation function, Iba's (1985) program automates the search for useful macros by choosing only those sequences that move from one peak in the evaluation function to another peak. Similarly, Minton's (1985) MORRIS program acquires two types of useful macros. The first type are s-macros, which are generalizations of operator sequences that occur frequently during problem solving. The second type are t-macros, which record useful sequences of operators that violate the advice of the heuristic evaluation function. The emphasis in this research by Iba and Minton is the use of heuristic guidance to selectively learn useful macros.

Episodic learning in the PET system also constructs useful operator sequences, which we call episodes. Unlike macros, which are rigid sequences of operators, episodes do not enforce a strict sequencing of operators. Instead, episodes are independent heuristic rules whose sequencing in only suggested. Since each heuristic rule recommends an operator to apply, an episode of rules recommends an operator sequence.

A "loose packaging" of heuristic rules in an episode permits incremental learning of new rules. As the result of incremental training, new rules are constantly being integrated into existing episodes. This integration modifies the operator sequence recommended by the rules in an episode and permits the learning of shorter solution paths.

A macro is not as easily modified as an episode because it is a sequence of operators rather than a sequence of heuristic rules. Each of the steps in a macro cannot "stand on its own" as an independent piece of problem solving advice. Instead, macros must be viewed as non-divisible operators. Since it is difficult to interlace a new macro into an existing macro, incremental learning of search heuristics complicates macro formation. We now examine the process for learning and applying operator sequences represented as episodes.

3.1 The episodic learning method

This section describes the method for learning and applying episodes in the PET system. Episodic learning constructs sequences of heuristic rules, where each rule is of the form *state-description* → *operator*. These rules are learned incrementally and integrated into episodes. Learned episodes are then applied by the problem solver to guide the search for problem solutions.

PET builds episodes by learning heuristic rules and assigning scores to each rule. The score for a rule is the distance from a problem state in which the rule is applied to a known goal. The first rule learned by PET is one which recommends an operator for immediately achieving a goal. This rule is assigned a score of one since it applies to a problem state which is a distance of one from the goal. PET extends this episode of length one by learning a rule which recommends an operator for reaching this

problem state. This recursive process constructs episodes of arbitrary length.

The left hand side of a heuristic rule formed by episodic learning can be viewed as a description of a subgoal in the state space. Because of the incremental growth of episodes, if the subgoal is reached during problem solving then it is guaranteed that an episode of heuristic rules has been learned to guide the problem solver to the goal. With this viewpoint, the score of a heuristic rule corresponds to the evaluation of the subgoal described with the left hand side of the rule. This evaluation is the length of the shortest episode, or known solution path, from the subgoal.

Learned episodes are used by the problem solver to suggest solution paths. When presented with a problem to solve, PET searches for an applicable rule. Conflicts are resolved in favor of the rule with lowest score. Application of the operator recommended by the rule results in a new problem state, and the process is repeated until a goal is reached. Notice that the problem solver is not constrained to follow a fixed sequence of rules. The learned sequences are loosely packaged so that multiple tracks through the state-space are possible. This internal flexibilty in operator sequences greatly reduces the combinatorial growth of macros while simplifying the incremental learning of operator sequences.

The PET algorithm for learning and applying episodic segments is formally described below. We will incrementally extend this description as we introduce the additonal components of the PET learning system in the following sections.

```
GIVEN an initially empty rulebase of heuristics
LOOP UNTIL teacher satisfied
  get problem from the teacher
  LOOP UNTIL problem solved
    IF some rule ∈ rulebase matches problem
    THEN apply.episode (rulebase, rule, problem) with no learning
    ELSE
      ask teacher for recommended operator
      form.rule (rulebase, problem, operator, newrule)
      IF newrule ≠ ∅ THEN integrate.rule (rulebase, newrule)
  REPEAT
REPEAT
```

Apply.episode (rulebase, rule, problem-state)

```
S ← score of rule
problem-state ← APPLY (RHS(rule), problem-state)
  (apply operator at head of episode)
LOOP UNTIL problem-state matches GOAL (apply rules in remainder
  of episode to achieve goal)
SELECT rule ∈ rulebase with score ≤ S which matches problem-state
  (resolve conflicts in favor of rule with smallest score)
```

problem-state ← APPLY (*RHS(rule), problem-state*)
S ← score of rule
REPEAT

Form.rule (rulebase, problem-state, operator, newrule)
IF APPLY (*operator, problem-state*) yields a state which matches GOAL
THEN *newrule* ← (*problem-state* → *operator*) with score 1
ELSEIF APPLY (*operator, problem-state*) yields a state *S* which enables *R* ∈ *rulebase* AND *apply.episode (rulebase, R, S)* matches GOAL
THEN *newrule* ← (*problem-state* → *operator*) with score of (score of *R*) + 1
ELSE *newrule* ← ∅ (no rule formed since reason for operator application is not understood)

Integrate.rule(rulebase, newrule)
ADD *newrule* to *rulebase*
(Note: This function is the main topic of section 4, which discusses how rules are generalized and integrated into the rulebase.)

Table 1. Operators in the domain of simultaneous linear equations.

Operator	Semantics
combinex (Eq)	Combine x-terms in equation Eq.
combiney (Eq)	Combine y-terms in equation Eq.
combinec (Eq)	Combine constant terms in equation Eq.
deletezero (Eq)	Delete term with 0 coefficient or 0 constant from equation Eq.
sub (Eq1, Eq2)	Replace Eq2 by the result of subtracting Eq1 from Eq2
add (Eq1, Eq2)	Replace Eq2 by the result of adding Eq1 to Eq2
mult (Eq, N)	Replace Eq by the result of multiplying Eq by N

3.2 Examples of episodic learning in simultaneous equations

This section presents examples of episodic learning by PET in the domain of simultaneous linear equations. These examples will also be used in later sections to demonstrate the other components of the PET system.

PET starts with a problem to solve, a goal to be achieved and a set of operators which can be applied to the problem. In the domain of simultaneous linear equations

the goal is to simplify the problem-state by reducing the number of terms in the equations. This "improvement in state evaluation function" form of goal allows the problem solver to recognize progress without completely solving the problem. The problem solving operators are listed in Table 1. Initially, there is an empty rule base of knowledge concerning *when* to apply each operator.

Problems are presented to PET by the teacher in the familiar form of simultaneous linear equations. For example:

$$a : 2x - 5y = -1$$
$$b : 3x + 4y = 10$$

where a and b are labels for the equations in the problem state. However, PET represents these equations internally as:

$$\{term(a,2 * x),term(a, -5 * y),term(a,1),$$
$$term(b,3 * x),term(b,4 * y),term(b, -10)\}$$

For descriptive purposes, we will use the more familiar form whenever possible.

In our first example, PET is presented the following training example by the teacher:

(State 1)
$$a : 6x + 3y = 12$$
$$b : 6x + 4y = 14$$

with the advice to apply operator sub (a,b). PET applies the operator, which yields the state:

(State 2)
$$a : 6x + 3y = 12$$
$$b : 6x - 6x + 4y - 3y = 14 - 12$$

The operator did not simplify the problem state since the number of terms increased from six to nine. Furthermore, the operator did not enable any rules in the (currently empty) rule base. PET is unable to understand why the operator is useful. No learning takes place and PET must "bear with" the teacher.

Now the teacher suggests that operator combinex (b) be applied to the current state. PET applies the operator, yielding:

(State 3)
$$a : 6x + 3y = 12$$
$$b : 0x + 4y - 3y = 14 - 12$$

This state is a simplification since the number of terms is reduced to eight. PET can now add knowledge to the rule base.

From this training, PET learns a solution path from state 2 to the goal of reducing the number of terms. The evaluation of the learned subgoal (state 2) is based on the distance to the goal, which in this case is one. The solution path is the single operator combinex (*b*). Consequently, the following heuristic (with a score of one) is added to the knowledge base:

$$\overbrace{\left\{\begin{array}{l} term(a,\ 6 * x), term(a,\ 3 * y), term(a,\ -\ 12), \\ term(b,\ 6 * x), term(b,\ -\ 6 * x), term(b,\ 4 * y), \\ term(b,\ -\ 3 * y), term(b,\ -\ 14), term(b,\ 12) \end{array}\right\}}^{State\ 2} \rightarrow combinex\ (b)$$

Now the teacher suggests the operator sequence combiney (*b*), combinec (*b*), and deletezero (*b*). The operators are applied sequentially to state 3 and the resulting state is:

(State 4) $\quad a : 6x + 3y = 12$
$\quad\quad\quad\quad b : 1y = 2$

Each operator achieved a state simplification, so rules (of score 1) are learned which are similar to the previous rule for combinex(*b*).

Once these heuristics are acquired by the system, the original training instance for the subtraction operator can be understood. Assume that the training instance for sub (*a,b*), which is labelled state 1, is re-presented by the teacher. (Note that PET could remember past training instances which were not understood in anticipation that subsequent learning will make them understandable.) As before, PET applies the sub (*a,b*) operator to state 1, yielding state 2. The episode learned previously applies to state 2 and achieves the goal of simplifying the problem. Therefore PET is able to understand the transition from state 1 to state 2 caused by sub (*a,b*).

From this training, PET learns a new heuristic rule:

$$\overbrace{\left\{\begin{array}{l} term(a,\ 6 * x), term(a,\ 3 * y), term(a,\ -\ 12), \\ term(b,\ 6 * x), term,(b,4 * y), term(b,\ -\ 14) \end{array}\right\}}^{State\ 1} \rightarrow sub(a,b)$$

This rule is assigned a score of two based on the local observation that the evaluation of state 1 is one plus the evaluation of state 2.

3.3 Examples of episodic learning in symbolic integration

In the domain of simultaneous linear equations, PET was given the goal of reducing the number of terms. In the domain of solving symbolic integration problems, PET

is given a more usual form of goal which requires a complete solution for its satisfaction. In particular, the goal is a problem-state which is free of an integral. The PET learning method works for both forms of goals.

We now present a short example of episodic learning in the domain of symbolic integration. As before, PET starts with a set of operators and knowledge of how to apply them, but an empty rulebase of heuristics to control when they are applied. While PET uses eighteen operators for problem solving in this domain, for present purposes, assume there are only two:

$$OP1: \int x^n \, dx \rightarrow \frac{x^{n+1}}{n+1} + C$$

$$OP2: \int a \, poly(x) \, dx \rightarrow a \int poly(x) \, dx$$

OP1 integrates a term consisting of the variable of integration, x; OP2 extracts a constant, a, from the expression being integrated.

Suppose the teacher presents the training instance:

(State 1) $\int 7x^2 \, dx$

with the advice to apply OP2. PET follows the advice by binding a to 7 and $poly(x)$ to x^2, yielding:

(State 2) $7\int x^2 \, dx$

Since state 2 is neither a goal state nor a state that PET knows leads to a goal state, nothing is learned from this piece of advice.

"Bearing with" the teacher, PET is advised to continue problem solving by applying OP1 to State 2. This yields:

(State 3) $\frac{7x^3}{3}$

which satisfies the goal. Consequently, PET is able to learn the following heuristic rule (with score of one):

$$\overbrace{7 \int x^2 \, dx}^{State\ 2} \rightarrow OP1$$

PET can now learn from the original training example. Applying OP2 to state 1 yields state 2, which is a recognized subgoal, *i.e.*, a state which enables a known

episode for achieving the goal. From this PET learns the following heuristic rule (with score of two):

$$\overbrace{\int 7x^2\,dx}^{State\ 1} \rightarrow OP2$$

With further experience this overly-specific rule will be replaced by a more general heuristic.

3.4. Summary of experience with episodic learning

Episodic learning is a method for acquiring sequences of heuristic rules for problem solving. In addition to improving problem solving efficiency, learned episodes span solution path segments which appear to be deviating from the goal. In the domain of simultaneous linear equations, PET formed a lattice of rules for the operators multiply, subtract, add, combine x-terms, combine y-terms, combine constant terms, and delete zero terms. The longest episode was seven rule applications from initial state to goal state. The "head" rule in this episode recommended "cross multiply," which requires two successive multiply operations. As this episode demonstrates, a technique for recording the sub-goals achieved by each operator is essential for our method. As discussed in the previous section, the problem is that some of the intermediate states in the solution path appear to be moving away from the goal.

We have considered adopting a more liberal learning policy in PET. Currently, PET does not form a heuristic rule unless that rule can be integrated into an existing episode by assigning it a score. The advantage of this conservative policy is that only those rules with known utility are added to the rule base. A more liberal learning policy allows rules to be learned from training even though their role in episodes is not known. While we believe that this policy could be adopted, we are concerned about a proliferation of rules which are not associated with known solution paths. Furthermore, by only learning heuristic rules which are known to lead to the goal, PET is not confused by bad advice from the teacher.

In the domain of symbolic integration, PET formed heuristics for its eighteen primitive operators. The longest episode was eleven rule applications from the initial state of $\int \sin^7 x\,dx$ to the goal state of $-1(u - 2\frac{u^3}{3} + \frac{u^5}{5})$. As in the domain of simultaneous linear equations, the intermediate states in the sequence appear to diverge from the goal state. Episodes bridge these necessary "digressions."

It is important to note that learned episodes are loosely packaged. During problem solving, rules from the rule base can be applied in any order if the scores of the rules in the sequence are non-increasing. Episodes suggest solution paths while permitting

"short cuts" to be pursued. During learning, the episode representation for operator sequences allows incremental learning of heuristic rules and integration of new rules into learned sequences.

The major limitation of episodic learning is that the heuristic rules are overly-specific. The next section discusses a technique for generalizing rules so that they apply to a class of problems. The technique uses the structures built by episodic learning to partially automate the role of the teacher.

4. Perturbation: automatic generation of training examples

Episodic learning forms heuristics with overly-specific left hand sides. Rather than rely on a teacher to provide additional appropriate examples for generalization, examples are generated automatically by perturbation (Kibler and Porter, 1983a). These examples are classified, as positive or negative, by the problem solver. Perturbation reduces the role of the teacher.

Typically, the teacher has two responsibilities in the learning process: *generating* training instances for a concept and *classifying* them as positive or negative examples of the concept. From this, the learner forms a general concept description which is complete and consistent with respect to the training set. Often the teacher has had to be careful in selecting training instances to ensure that the learner will succeed (*e.g.*, Winston, 1975; Sammut, 1983).

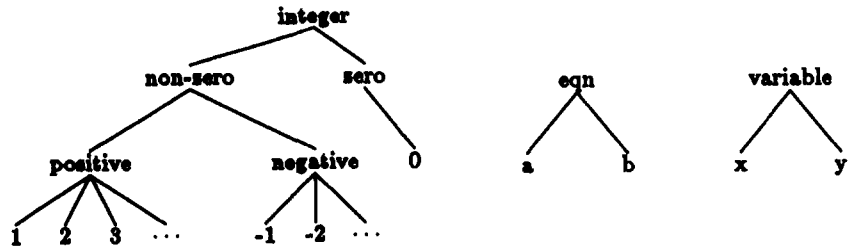


Figure 1. Concept hierarchy trees for simultaneous linear equations.

Perturbation automates the generation of training instances by making small changes to a single teacher supplied training instance, I. Each change is the result of applying a *perturbation operator* to I. A perturbation operation selects a single feature f in I and slightly modifies it. Modifications are of two types:

1) Replace f by the null feature, effectively removing the feature altogether.
2) Replace f by a sibling of f in a concept hierarchy tree for the problem solving domain. This type of modification generates a set of perturbation operators since f may either have multiple siblings or be in multiple trees. Figure 1 gives the con-

cept hierarchy trees used by PET in the domain of simultaneous linear equations. Similar trees are used for symbolic integration.

By applying these perturbation operators to the "seed" instance I, a set of instances which is "minimally different" from I is generated.

Now, each instance I' generated from I is classified. When I was presented by the teacher, with the advice to apply operator *OP*, it was (implicitly) classified as a positive example of the concept "states in which *OP* is effective." PET classifies I' as a positive example of the same concept if and only if the application of *OP* to I' yields the same subgoal (enables the same episodic segment) as the application of *OP* to I. In the terminology of explanation-based learning (Mitchell, *et al.*, 1986), I' is a positive example if the explanation that justified the classification of I as a positive example is also valid for I'. Here, the explanation is the sequence of operator applications, or episode, which leads to goal fulfillment.

An important advantage of the PET perturbation technique is that examples are generated and classified efficiently. Perturbation generates examples by selecting a feature of a given instance, finding it in the concept hierarchy tree (a linear search of the leaves, at worst) and selecting a sibling (requiring two arc transitions). Perturbation classifies examples by using the problem solver to apply a learned episode. This amounts to performing a highly directed search. The LEX system (Mitchell, *et al.*, 1983), by contrast, does not learn episodes. Therefore, the problem solver must perform an n-level expansion of the search space which is guided by (partially) learned search heuristics and terminates when a goal is reached. If the example lies on the shortest solution path then it is classified as positive. Negative examples are more difficult to classify. LEX performs exhaustive search to classify an example as negative (to ensure that the heuristic search did not miss a good solution path). While PET could also adopt this approach for classifying negative examples, we chose to ignore suspected negative examples and to reduce over-generalization by using small-step, conservative generalization operators. Relational models, which are discussed in Section 6, also reduce over-generalization errors.

In summary, perturbation tests the relevance of each feature of an example by collecting examples in which each feature is separately varied. In an empirically-based learning system, examples are provided by some undirected process. However, there is no assurance that the examples represent a useful coverage of the space of possibilities. By directing the generation of examples via perturbation, examples are systematically generated. This guarantees that a large and important class of examples will be considered. The role of experimental goal regression, as we discuss in Section 6, is to further improve data collection by guiding the application of perturbation operators.

4.1. The PET learning cycle with perturbation

This section describes the incorporation of the technique of perturbation into the PET learning cycle presented in Section 3. The only change is to the procedure *integrate.rule*, which generalizes a rule and adds it to the rulebase. The important feature is that PET uses perturbation to generate examples from those provided by a teacher. These examples are then classified and a general rule is learned by induction.

PET generalizes a rule of the form *problem-state* → *OP* in four steps. First, perturbation generates multiple training instances by making small changes to *problem-state*. Second, the operator *OP* is tried on each generated example. Third, each example is classified as positive or negative with respect to the effectiveness of *OP*. An example is classified as positive if *OP* allows the problem solver to complete a solution using learned episodes. Otherwise the example is classified, perhaps erroneously, as negative. Fourth, standard generalization operations are applied to the positive examples. These operations involve dropping conditions, turning constants to variables and climbing concept hierarchy trees, as defined by Michalski (1983).

The resulting generalized rule is then integrated in PET's rule base. This integration involves generalizing the new rule with existing rules which recommend the same operator *OP*. At this time, the allowed generalization operations are turning constants to variables and climbing concept hierarchy trees. If generalization cannot be performed then PET simply adds the new rule to the rulebase. Multiple rules which recommend the same operator correspond to a disjunctive description of the states in which the operator is effective.

The following is a more formal description of the rule generalization process which builds on the algorithm given in Section 3.1.

Subroutine integrate.rule (rulebase, newrule)

Decompose *newrule* into components:
  problem-state ← LHS (*newrule*)
  OP ← RHS (*newrule*)
Generalize *newrule* using perturbation:
  currgen ← *problem-state*
  LOOP UNTIL all perturbation operators have been applied
    SELECT perturbation operator *P*
    APPLY *P* to *problem-state*, yielding *problem-state'*
    IF *apply.episode* (*rulebase, newrule, problem-state'*)
      matches either a goal or the left hand side of rule (a subgoal)
    THEN *currgen* ← minimal generalization of *problem-state'* and *currgen*
  REPEAT
    (now *currgen* is a description of a state cluster of positive
    examples of the concept "states in which *OP* is effective.")

Integrate the new rule into the rulebase:
genrule ← (*currgen* → *OP*) with score of *newrule*
IF a member of *rulebase* can be generalized to cover *genrule*,
THEN replace member by generalization
ELSE add *genrule* to *rulebase*

4.2. Example of perturbation in simultaneous linear equations

This section illustrates the role of perturbation in forming heuristics. Recall that the first rule learned by PET for solving simultaneous linear equations recommends the operator combinex (*b*) in the state:

State (1)
$$a : 6x + 3y = 12$$
$$b : 6x - 6x + 4y - 3y = 14 - 12$$

From this, episodic learning forms the following rule (with score 1):

$$\overbrace{\left\{\begin{array}{l} term(a, 6 * x), term(a, 3 * y), term(a, -12), \\ term(b, 6 * x), (term(b, -6 * x), term(b, 4 * y), \\ term(b, -3 * y), term(b, -14), term(b, 12) \end{array}\right\}}^{State\ 1} \rightarrow combinex\ (b)$$

Perturbation is used to discover a general description of the states in which combinex (*b*) is effective. This single positive example provided by the teacher does not adequately restrict the space of candidate concept descriptions. From the example provided by the teacher, literally millions of different generalizations are possible. Perturbation helps guide the search by conducting a series of controlled experiments. Each experiment isolates a feature of the example while holding other features constant. An experiment is formed by a perturbation operator which selects a feature and either deletes it or replaces it by a sibling of that feature in a concept hierarchy tree.

In the domain of simultaneous linear equations, perturbation generates some 30 different near-examples, four of which are listed below. The boxed term indicates the feature that has been deleted or changed.

$a : 6x + \square = 12$
$b : 6x - 6x + 4y - 3y = 14 - 12$
E_1

$a : 6x + 3y = 12$
$b : \boxed{7}\ x - 6x + 4y - 3y = 14 - 12$
E_2

$a : 6x + 3y = 12$
$b : 6x - \square + 4y - 3y = 14 - 12$
E_3

$a : 6x + 3y = 12$
$b : 6x - 6x + 4y - 3y = \square - 12$
E_4

The second step is to classify each state as a positive or negative instance of the concept being learned. In this case, the concept is "states in which combinex (*b*) is effective." Effectiveness of combinex (*b*) is determined for each of the generated examples. The score of the rule for combinex (*b*) is one. This indicates that combinex (*b*) directly achieves the goal of reducing the number of terms in any problem for which the rule is applicable.

PET applies this effectiveness criterion to each of the four generated examples above. E_1 is classified as a positive example because combinex(*b*) reduces the number of terms in the expression. PET minimally generalizes the state description of the current rule for combinex with E_1, yielding a new rule:

$$\left\{\begin{array}{l} term(a,\ 6 * x),\ \boxed{},\ term(a,\ -\ 12), \\ term(b,\ 6 * x),term(b,\ -\ 6 * x),term(b,\ 4 * y), \\ term(b,\ -\ 3 * y),term(b,\ -\ 14),term(b,\ 12) \end{array}\right\} \rightarrow combinex(b)$$

The major effect is to delete the condition on the y-term of equation *a*. This is a spurious detail which is irrelevant to the general concept.

Generated example E_2 is also classified as positive. The minimal generalization of the state description of the current rule with E_2 yields the new rule:

$$\left\{\begin{array}{l} term(a,\ 6 * x),term(a,\ -\ 12), \\ term(b,\boxed{positive} * x),term(b,\ -\ 6 * x),term(b,\ 4 * y), \\ term(b,\ -\ 3 * y),term(b,\ -\ 14),term(b,\ 12) \end{array}\right\} \rightarrow combinex(b)$$

The major effect of this learning is to recognize that one of the x-coefficients of equation *b* can be *any* positive integer. Notice that the constraint on this coefficient is still overly constrained, since negative integers are excluded. Negative integers are not tested by perturbation in this example because they are not immediate siblings of the original coefficient 6. "Distant" siblings can be tested at the expense of an increase in the number of perturbation operators. Rather than incur this expense, PET relies on the teacher for subsequent training to further define the rule. This is one of many instances in which PET would benefit from intelligent selection of perturbation operators. This need is addressed by experimental goal regression (described in Section 6), which allows PET to test distant siblings with some assurance that the test will be fruitful.

Generated example E_3 is classified as negative because the combinex(*b*) operator is ineffective. This negative information is not used by the PET induction algorithm, since PET may have incorrectly classified the example. In many learning systems, negative examples are used for correcting over-generalization in concept definitions. PET avoids this problem by making conservative, minimal generalizations.

Example E_4 is classified as positive. This yields the concept description:

$$\left\{\begin{array}{l} term(a,\ 6 * x), term(a,\ -\ 12), \\ term(b,\ positive * x), term(b,\ -\ 6 * x), term(b,\ 4 * y), \\ term(b,\ -\ 3 * y), \square, term(b,\ 12) \end{array}\right\} \rightarrow combinex(b)$$

After the perturbation process, the heuristic rule for the operator combinex(b) is highly refined. For the single training instance in state 1, there are thirty perturbation operators. Fifteen of these test if a feature can be removed and fifteen test if a relevant feature can be generalized. The result of minimally generalizing over the generated positive examples is the rule:

$$\{term(b,\ positive_1 * x), term(b,\ positive_2 * x)\} \rightarrow combinex(b)$$

where $positive_1$ and $positive_2$ denote two positive integers.

This rule is significantly more refined than the original. One rough measure is the number of constraints in the state description. This count is reduced from nine to two by the perturbation process. Moreover, the two remaining constraints are quite general. More importantly, this learning does not involve teacher participation.

4.3. Examples of perturbation in symbolic integration

For the sake of clarity and completeness we illustrate the process of perturbation and subsequent generalization in the domain of symbolic integration. Since these ideas require no change, we will be brief. Consider the following operator:

$$OP1: \int x^n\, dx \rightarrow \frac{x^{n+1}}{n+1} + C$$

The first rule formed by episodic learning in Section 3.3 is:

$$7 \int x^2\, dx \rightarrow OP1$$

In order to generalize from this instance, perturbation operators are applied to generate a set of examples. Four of these examples are:

$$\underset{E_1}{\square \int x^2\, dx} \qquad \underset{E_2}{7 \int \square\, dx} \qquad \underset{E_3}{\boxed{8} \int x^2\, dx} \qquad \underset{E_4}{7 \int x^{\boxed{3}}\, dx}$$

E_1, E_3 and E_4 are classified as positive examples of the concept "states in which $OP1$ is effective." Since the current heuristic rule for $OP1$ has a score of 1, effectiveness is determined by a state transition to a goal state (a state not containing an integral). A minimal generalization of the state description of the current rule for

*OP*1 with E_1, E_3 and E_4 yields the new rule:

$$\int x^{positive}\, dx \rightarrow OP1$$

where *positive* represents any positive integer.

Only one more teacher supplied training instance is required for PET to converge on the final heuristic rule for *OP*1. Given the positive example:

$$\int x^{-3}\, dx$$

PET minimally generalizes to:

$$\int x^{nonzero}\, dx \rightarrow OP1$$

This generalization is errorful, as it would incorrectly suggest *OP*1 in the case of $\int x^{-1}$. The faulty generality is the result of an insufficiently precise concept description language. Within the given concept description language, the heuristic gives the best coverage of when to apply *OP*1. The last defense of this over-generalization is that heuristics are no guarantee of correctness.

4.4. Review of example generation with perturbation

We have found that perturbation is an effective technique for guiding generalization. In the domain of simultaneous linear equations, PET formed an effective set of heuristics for solving any problem. Perturbation uses a single training example provided by the teacher as a "seed" for automatic example generalization. Each generated example is a slight variant of the original training example and tests the relevance of an individual feature. Generated examples are then classified by guiding problem solving with rule sequences acquired by episodic learning.

While perturbation efficiently generates training examples, the process would be improved if useful perturbation candidates could be identified. This would increase the learning rate, since examples which most advance the search for a generalized concept description would be generated first. As we describe in Section 6, the role of experimental goal regression is to guide the perturbation process.

5. Relational models

The methods of the previous sections have made no assumptions about the representation of operators. In particular the techniques will work if operators have opaque representations, such as provided by procedures. However, in order to better direct

the learner, a more detailed representation of an operator is needed than that given by procedures. A relational model for a procedurally defined operator *OP* is an approximation of *OP* with a STRIPS-like rule. The advantage of this transparent representation is that the transformation performed by the operator is explicit. In particular, PET uses the relational model representation of operators in two ways: to detect and recover from over-generalization of heuristic rules and to guide the selection of perturbation operators.

STRIPS-like rules are commonly used in AI systems for learning and problem solving because the system can reason using the operator definitions (Minton, 1984; Waldinger, 1977). Rather than assume that transparent operator definitions are *a priori* knowledge, PET uses *relational modeling* to learn them from procedural operator definitions. Procedural representations simplify the user's task of defining operators and relational modeling allows the learner to accept the responsibility of shifting representation.

PET determines the input/output behavior of each procedurally-defined, opaque operator by applying the operator to a set of states. A relational description of this behavior is induced from the input/output examples. The search for this description requires incorporating domain-specific background knowledge into the evolving relational description. This section reviews related work on learning in the presence of background knowledge and discusses PET's algorithm for learning relational models.

5.1. Background knowledge

Vere's experiments with learning in the presence of background knowledge (Vere, 1977) are directly related to learning relational models. Vere demonstrates the use of background knowledge to restrict over-generalization of learned concepts. For example, in the domain of the card game poker, it is necessary to apply the background knowledge of card ranks in order to learn the concept of a "straight." Langley (1980) uses this method in the ACTG learning system. In particular a rule is recognized as an over-generalization if the right hand side contains variables which are not bound in the left hand side. In this event, ACTG conducts a search through a space of relations to find a path which links the left hand side of the rule to the right hand side. An earlier version of PET addressed the same over-generalization problem (Kibler and Porter, 1983b).

The contribution of this research is the test for over-generalization and the use of background knowledge to add needed constraints. A shortcoming of the approaches is that they provide little guidance of how much knowledge to incorporate. Vere allows only one predicate in background knowledge while ACTG searches through all relations. PET uses relational models to control the amount of background knowledge applied. Incorporating too little knowledge results in an overly-general

rule. Incorporating too much knowledge results in an overly-specific rule. PET uses perturbation to determine if a rule has become too specific.

Relational models extend Vere's and Langley's approach to rule augmentation by representing the transformation performed by an operator. Background knowledge is used to relate terms in the preconditions of the operator to terms in the postconditions of the operator. Multiple relations are usually required to define this transformation. As we describe in Section 6, these relations also serve to guide experimental goal regression.

5.2. Representing operators with relational models

PET's method for building relational models relies on background knowledge to connect hypothesized pre and post conditions of operators. In addition to augmenting concept descriptions to prevent over-generalization, relational models approximate the transformation performed by operators. In this section we formalize the notion of a relational model, which relies on a variant of typical production rules. Then we illustrate the use of relational models on several examples from the domain of symbolic integration.

A relational model of an operator *OP* is built on a heuristic rule for *OP*. This rule is a variant of the production rule representation for heuristics presented in section 2. Previously, we used heuristic rules of the form:

PRE state description → *OP*

with the interpretation:

If the current state S matches *PRE*
then operator *OP* is recommended in S.

A relational model is of the form:

PRE state description $\overset{op}{\rightarrow}$ *POST* state description

with the interpretation:

IF the current state S matches *PRE* **and**
the state resulting from applying *OP* to *PRE* matches *POST*
THEN *OP* is recommended in S.

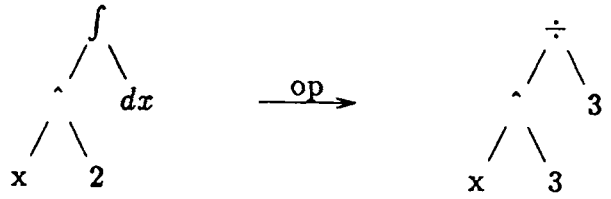
This form of a rule was first proposed by Amarel (1968) and has two advantages. By explicitly representing the effect of an operator, any difference between the desired outcome and the actual outcome can be noted. This is important in PET because relational models of operators are approximations which might be only partially learned. The Amarel-style rule allows for checking the post-conditions of an

operator application before recommending that the rule be used. The second advantage is the reduction in the size of the search graph generated during problem solving. Amarel-style rules offer one-step look ahead during the search. Rules which recommend operators that do not yield the desired *POST* conditions are pruned from consideration.

In the domain of symbolic integration, PET represents the *PRE* and *POST* state descriptions as parse trees. For example, the rule which recommends the operator

$$OP: \int x^n \, dx \rightarrow \frac{x^{n+1}}{n+1} + C$$

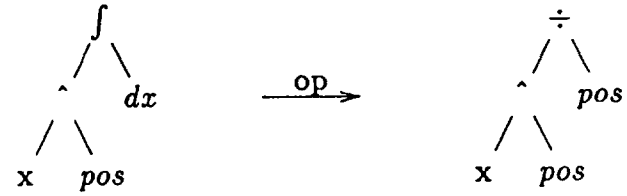
in state $\int x^2 \, dx$ is:



where "$+C$" is dropped for simplicity. Note that the state resulting from the operator application, *POST*, is explicitly represented in the right hand side of the rule.

This form of heuristic rule is generated using "standard" generalization techniques. For example, the induction algorithm used with perturbation forms generalized rules of the form *PRE* → *OP*. Applying the same algorithm to states resulting from *OP*'s application yields a generalization of *POST*. More specifically, PET generalizes *PRE* and *POST* by either removing conditions or replacing them by ones higher up in the concept hierarchy tree.

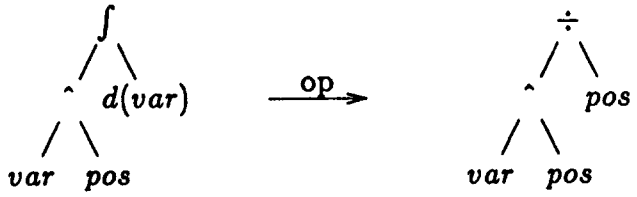
However, this generalization scheme can yield unusable rules. For instance, the rule above can be generalized with the positive training example $\int x^3 \, dx$. This yields the rule:



This rule is over-generalized since the critical relations are lost. There are two essential constraints in the original, instantiated rule that are lost in the generalization:

1) the x exponent in *POST* is one more than the x exponent in *PRE*.
2) the denominator in *POST* is one more than the x exponent in *PRE*.

Further generalization with a third positive example, $\int y^4\, dy$, yields the rule:



and a third essential constraint is lost:

3) the variable in the numerator of *POST* is the variable of integration in *PRE*.

Table 2. Background knowledge for symbolic integration.

Relation	**Semantics**
eq (X, Y)	X and Y are equal
suc (M, N)	N is the integer successor of M
sumof (L, M, N)	sum of L and M is N
product (L, M, N)	product of L and M is N
power (L, M, N)	L raised to the M-power is N
derivative (M, N)	derivative of M is N

Relational models provide an elaboration of this form of heuristic rule. In constructing a relational model for an operator, the descriptive language is augmented with background knowledge to explicitly represent constraints between terms in *PRE* and *POST* so that they are not lost during generalization. The background knowledge for symbolic integration is the set of relations listed in Table 2.

The augmentation is a list of relations from background knowledge which is instantiated with terms from the heuristic rule. These relations represent constraints between the terms so that they are not lost during rule generalization. For example, an augmentation of the rule given above forms the following relational model:

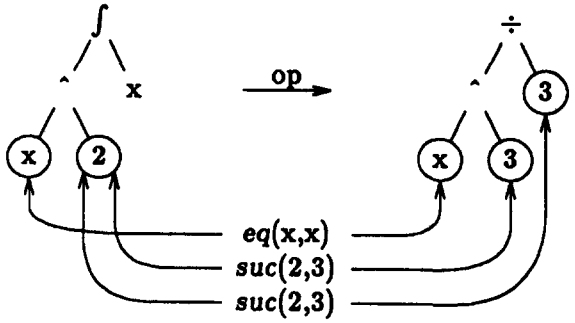


Diagram 1.

Now, if the rule is generalized as above, the constraints are not lost. The generalized augmented rule is:

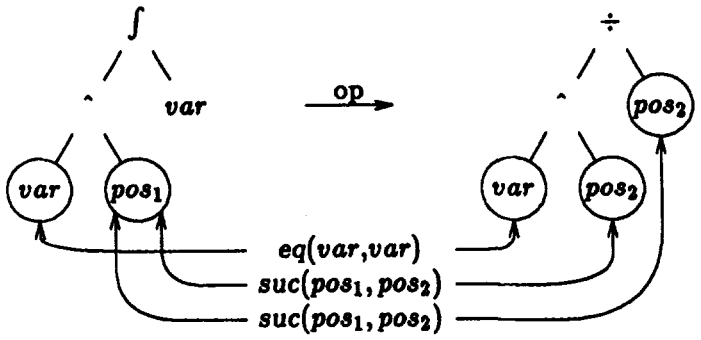


Diagram 2.

Augmentation extends the interpretation of a heuristic rule for operator *OP*. If the augmentation is instantiated with terms from *PRE* and *POST*, the resulting rule is interpreted as:

IF the current state, S, matches *PRE* and
the state resulting from applying *OP* to S matches *POST*
such that the relations in the augmentation hold,
THEN OP is recommended in S.

With this intuitive understanding, relational models can be formally defined. A relational model is a 4-tuple $\langle OP, PRE, POST, AUG \rangle$. The augmentation, *AUG*, is a set of relations $\{rel_1, rel_2, \ldots, rel_n\}$ from background knowledge. Each relation, $rel_i \in AUG$ has a relation name, or functor, and $m \geq 2$ arguments, $\{a_1, a_2, \ldots, a_m\}$. The purpose of the augmentation is to relate subexpressions of *PRE* with subexpressions of *POST*, thereby "linking" *PRE* to *POST*. To establish these links, each a_j is constrained to be a subexpression of either *PRE* or *POST*, such that not all a_j are from the same source.

Actually, this is an over-simplification. By allowing a_j to be a subexpression of an argument of another relation in *AUG*, composites of relational descriptors can be formed by "daisy-chaining" a link between *PRE* and *POST* through multiple descriptors. For example, the relation that *PRE* is the double derivative of *POST* is represented by:

derivative (*PRE*, X), derivative (X, *POST*).

5.3. *Learning relational models*

In earlier research (Porter and Kibler, 1984) we developed an algorithm for learning relational approximations of procedural operators. This algorithm is integrated into the PET system so that there are no *a priori* restrictions on the representation of problem solving operators. The input to the algorithm is an unaugmented heuristic rule. The learner then searches for the "best" augmentation of the rule. The output of the learner is a relational model in which the augmentaton is instantiated with terms from the original rule.

Augmentations are rated by measuring the coverage of *PRE* and *POST* by *AUG*. Intuitively, coverage is a measure of the number of nodes of *PRE* and *POST* which are arguments in *AUG*. The augmentation which achieves the best coverage is chosen for the relational model of the operator.

Given an unaugmented heuristic rule $R = \langle OP, PRE, POST \rangle$, a relational model of R is constructed by searching for the set of instantiated augmentation relations, *AUG*, which best covers *R*. This search is implemented in PET as a beam-search through the space of candidate augmentations. In this space, nodes are represented by the tuple $\langle AUG, Pool \rangle$ where *Pool* is the set of subexpressions of *PRE* and *POST* not covered by *AUG*. In particular, the initial state is $\langle nil, \{PRE \cup POST\} \rangle$. There is one operator in this search, which is described by:

Given a state $\langle AUG, Pool \rangle$,
  SELECT a relational descriptor, *D*, from the set of background concepts
  INSTANTIATE *D* with members of *Pool* or their sub-expressions
  REMOVE selected Pool members from *Pool*, yielding *Pool'*
  ADD instantiated descriptor to *AUG*, yielding *AUG'*.
Generate new state $\langle AUG', Pool' \rangle$.

This search ends when coverage fails to improve.

Built-in biases reduce the branching factor of the search for an augmentation with maximal coverage and minimal complexity. In the selection of a relational descriptor, preference is given to more primitive relations, such as *equal* and *suc*, over more complex relations, such as *product*. Further, there are semantic constraints on the subexpressions selected to instantiate a relation. For example, the first parameter in the *derivative* relation must contain a variable of differentiation. Finally, note that the algorithm tries large subexpressions from *PRE* and *POST* before small subexpressions, thereby maximizing the coverage of the augmentation. If two relational models have the same coverage, then the one with fewer relations is preferred.

5.4. Examples of relational models in symbolic integration

Relational models were not needed in the domain of simultaneous linear equations because all the operators had natural relational descriptions. However, in the domain of symbolic integration the operators are difficult to define relationally. For example, the operator which substitutes a new term for an expression in an integral appears to be most naturally defined procedurally. Given this procedural representation, PET learns a relational model which approximates the procedure. This section presents several of the generalized relational models learned by PET which correspond to operators in symbolic integration. The reader is directed to Porter (1984) for additional examples.

Each of the relational models uses generalized descriptors from the concept hierarchy tree for symbolic integration. This relational model illustrates PET's bias towards selecting large sub-expressions of *PRE* and *POST* when instantiating augmentation relations. In general, PET seeks an augmentation with maximum coverage of *PRE* and *POST*.

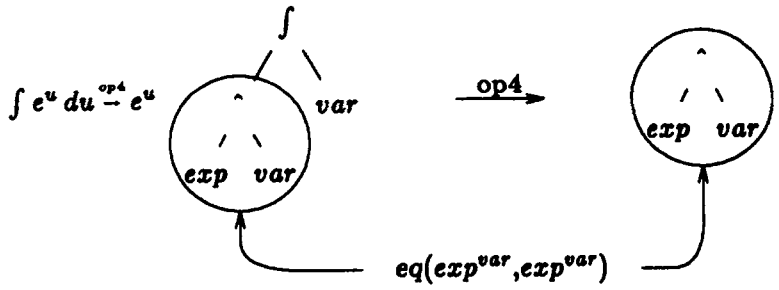


Diagram 3.

In the next example, $\int (2x + 3)^4\, dx$, PET deals with the *substitution* operator, which seems to require a procedural definition. Here the advice is to substitute u for $2x + 3$. In order to carry out this substitution, one must also divide the integrand by the derivative of u, namely 2. The state resulting from the substitution is $\int \frac{1}{2}u^4\, du$. The relational model built by PET for the operation is:

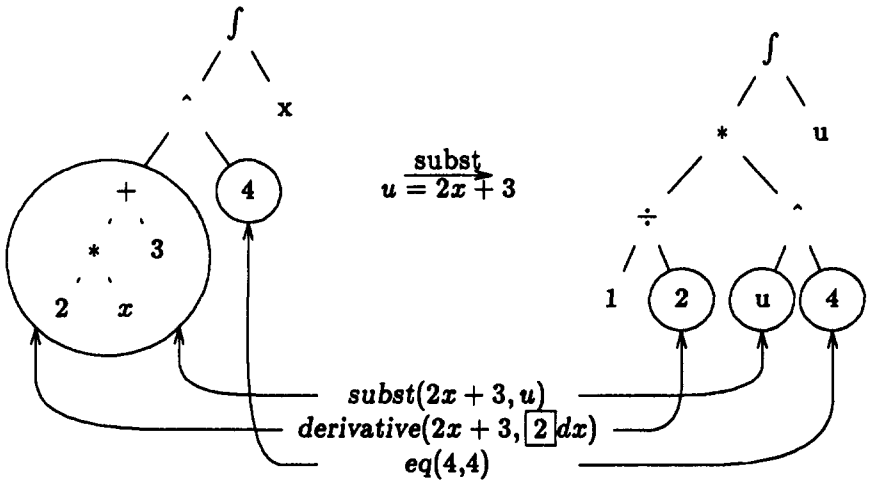


Diagram 4.

Notice that the relational descriptor *subst*, which describes the fact that u has replaced $2x + 3$, is distinguished from the general substitution operator, denoted by subst.

PET generalizes this relational model using perturbation and further teacher training. The generalized rule is:

$$\int poly^{int}\, dx \rightarrow \int \frac{1}{poly'}\, u^{int}\, du$$

More completely, the rule is:

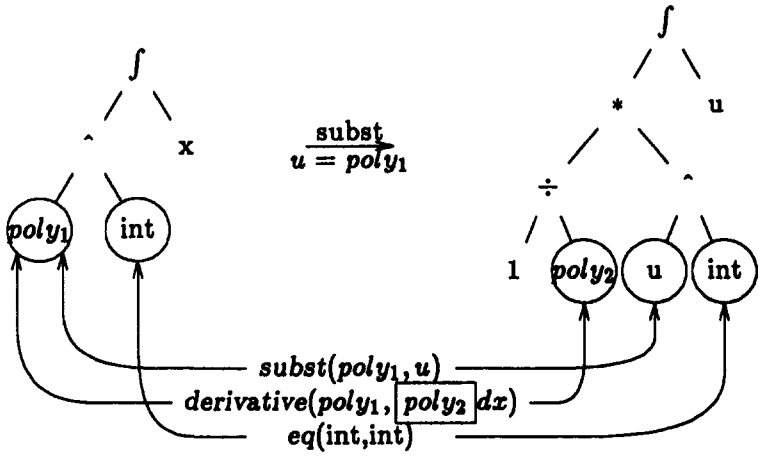


Diagram 5.

The relational model explicitly represents the constraint that $poly_2$ is the derivative of $poly_1$. The derivative relationship between these polynomials is not represented explicitly in LEX (Utgoff, 1983). Instead, the derivative relationship is hidden in the code for the substitution operator and constraint back propagation through the operator is impossible.

5.5. Summary of relational models

The advantage of representing operators relationally is that the transformation performed by the operator is explicit. In particular, PET uses relational models in two ways: to detect and recover from over-generalization of heuristic rules and to guide the selection of perturbation operators. The latter use is the topic of Section 6.

Much of the work in learning search heuristics assumes that operators are defined with transparent, STRIPS-like rules. Relational modeling extends the applicability of this work by removing *a priori* constraints on the form of the operators. Our algorithm for learning relational models has successfully learned a representation for each of the eighteen symbolic integration operators used by PET.

6. Experimental goal regression

This section describes a technique for improving the rate at which problem solving heuristics are learned. We call the technique *experimental goal regression* (EGR). EGR is an experientially based approximation to Waldinger's (1977) analytic goal regression. Goal regression is used to back-propagate goal conditions through operators to form sub-goal descriptions. Given a goal state G and an operator OP, a *goal regression product* is a description of a sub-goal state S, such that OP applied to S achieves G. The goal regression product corresponds to Dijkstra's (1975) notion of *weakest pre-condition*. According to Dijkstra, wp (OP, G) is the weakest constraint on a state S which guarantees that the application of OP to S yields a state satisfying G.

We first describe the roles of episodic learning, perturbation, and relational modeling in EGR. Then we present multiple examples of the application to EGR to improving the learning rate of search heuristics.

6.1. Integrating episodic learning, perturbation, and relational models

The rate at which problem solving heuristics are learned is improved by using relational models to select appropriate perturbation operators. As described in Section 4 PET applies perturbation operators to a single teacher-supplied training instance

to generate multiple similar examples. Perturbation automates part of the teacher's role, but not the task of *selectively* generating training instances which are most useful for concept convergence.

Experimental goal regression uses learned relational models and episodic segments to guide perturbation. Using EGR, perturbation candidates are generated which test specific features of a concept description that are believed to be overly specialized. Consider an episode E consisting of rule applications $r_1, r_2, \ldots, r_n$. Each rule r_i is represented with relational model $\langle OP_i, PRE_i, POST_i, AUG_i \rangle$. AUG_i represents the "intra-rule" links between PRE_i and $POST_i$. "Inter-rule" links are implicit in E. As described in Section 3, r_i is added to an episode if it enables r_{i+1}. This establishes an implicit link between $POST_i$ and PRE_{i+1}. Constraints imposed on r_i by r_j, $i < j$, are discovered by following inter-rule links through E and intra-rule links through relational models. These constraints suggest perturbation operators for r_i.

6.2. Examples from symbolic integration

We illustrate experimental goal regression in the domain of symbolic integration with an example from Utgoff (1983). The example demonstrates how PET automatically selects the single perturbation candidate from the space of possibilities which enables useful concept generalization.

Assume that from prior training for the operator:

$$OP1 : \sin^2 x \rightarrow 1 - \cos^2 x$$

PET has formed the following relational model:

$$\int \sin x(\sin^2 x)^n \rightarrow \int (1 - \cos^2 x)^n \sin x \, dx$$

summarizing the rule:

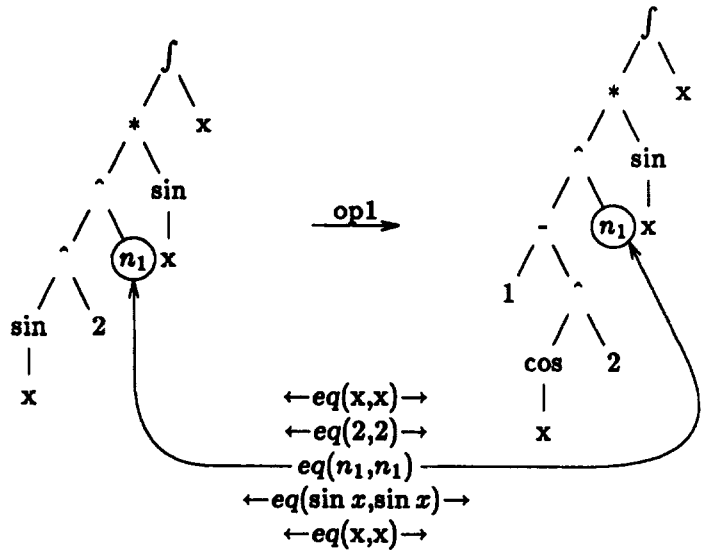


Diagram 6.

Note that this model has been generalized from instances such that PRE_{op1} matches states of the form $\int (\sin^2 x)^{nonzerointeger} \sin x \, dx$.

Now PET is presented the training instance $\int \sin^6 x \sin x \, dx$ with the advice to apply the opaque operator:

$$OP2 : \sin^n x \rightarrow (\sin^2 x)^{n/2}$$

PET applies the operator, yielding $\int (\sin^2 x)^3 \sin x \, dx$. As described in Section 3, PET can only learn a rule for this training instance if it achieves a known (sub)goal (allowing the rule to be integrated into an existing episode). In this example, the training instance achieves the subgoal defined by PRE_{op1}. The following relational model for the training instance is built by the beam-search algorithm of Section 5:

$$\int \sin^6 x \sin x \, dx \rightarrow \int (\sin^2 x)^3 \sin x \, dx$$

which is more accurately described by:

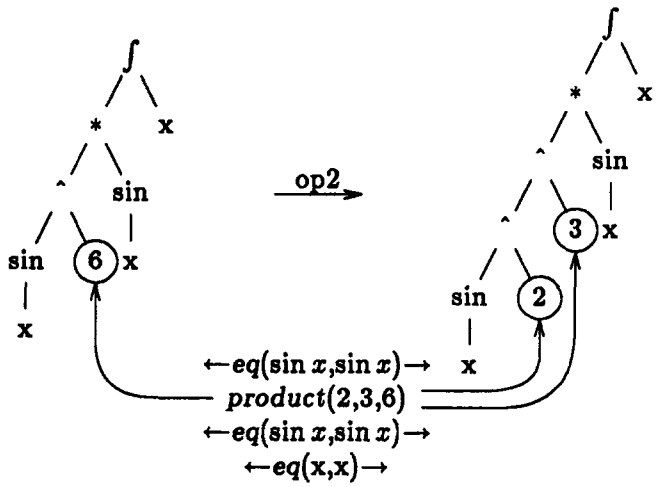


Diagram 7.

Now that episodic learning has associated the relational models for *OP*1 and *OP*2, perturbation operators are applied to generalize the model for *OP*2. The relaxed constraint in PRE_{op1} (i.e., "*nonzerointeger*") is regressed through the episode with the potential of identifying a feature of PRE_{op2} (i.e., "6") which can be generalized. The inter-rule link implicit in episodes connects the relational model of OP2 with the relational model of OP1. Matching $POST_{op2}$ with PRE_{op1} binds variable n_1 with 3. This suggests that the relational model for OP2 is overly-specific. Perturbation tests relaxing this constraint by generating a training instance with this feature slightly modified. This is done by traversing intra-rule links represented by the augmentation. Specifically, PET generates a useful training instance by the following steps:

1. Locate the relation $r \in AUG_{op2}$ with argument of 3 from $POST_{op2}$. In this case, r = *product* (2, 3, 6).
2. Perturb r to generate a slight variant, r'. This is done in three steps: First, replace the argument with a neighboring sibling in a concept hierarchy tree. In this case, replace 3 with 4. Second, locate an argument p in r such that p is a sub-expression of PRE_{op2} and replace it by free variable x. In this case, $p = 6$. Third, evaluate the resulting partially instantiated descriptor to uniquely bind x to p'. In this example, $p' = 8$ and $r' =$ product (2, 4, 8).
3. Generate PRE'_{op2}, a perturbation of PRE_{op2}, by replacing p by p'. In this example, $PRE'_{op2} = \int \sin^8 x \sin x \, dx$.
4. Classify PRE'_{op2} as an example or near-miss of a state in which *op*2 is useful. As discussed in Section 4, PRE'_{op2} is an example if apply (OP2, PRE'_{op2}) achieves the same subgoal as apply (OP2, PRE_{op2}). In this example, PRE'_{op2} is an example which achieves the subgoal of PRE_{op1}.

Finally, PET generalizes the original training instance with examples generated by perturbation. The following relational model is the minimal generalization of these instances:

$$\int \sin^{n2} x \sin x \, dx \rightarrow \int (\sin^2 x)^{n1} \sin x \, dx$$

or, more precisely:

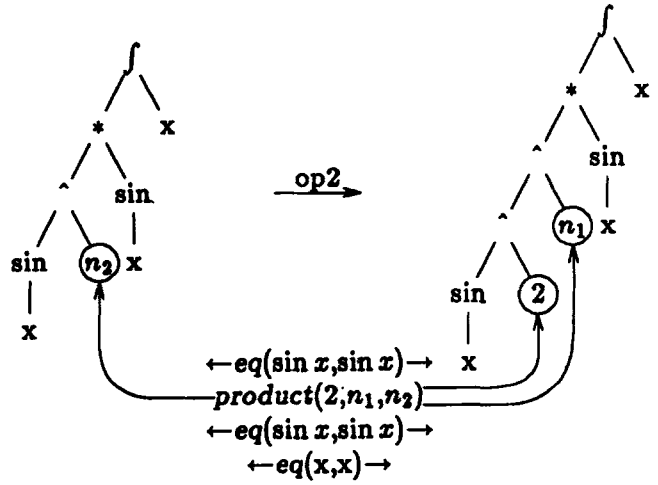


Diagram 8.

Note that the *product* (2, n_1, n_2) augmentation descriptor corresponds to the concept of even-integer (n_2). This is the correct constraint on the heuristic since this episode is only effective on integral expressions of the form $\int \sin^n x \sin x \, dx$, where n is even. As demonstrated with this example, constraint back-propagation through episodes and relational models can discover generalized descriptors which suggest new concepts. While it is not the focus of this research, these concepts could augment the description language for subsequent learning (see Utgoff, 1984).

We now present a second example of using relational models to guide the selection of perturbation operators. This example demonstrates that spurious features are easily identified in the relational model representation of heuristic rules. A spurious descriptor for a heuristic which recommends operator *OP* is one which is irrelevant to the effectiveness of *OP*. As with overly-specific features, perturbation can detect spurious features but the search is unguided.

Relational models allow PET to selectively guide the search for spurious descriptors. Rather than test the relevance of every descriptor, PET heuristically selects candidates. Given relational model ⟨*OP*, *PRE*, *POST*, *AUG*⟩ the heuristic states:

> Descriptors of *PRE* which are not transformed by *OP* may be irrelevant to the rule recommending *OP*.

Those descriptors of *PRE* which are not transformed are exactly those linked by the *eq* relation to descriptors of *POST*. This heuristic identifies candidate irrelevant descriptors which can be tested with perturbation. Consider the training instance:

$$7 \int x^2 \, dx$$

for the operator:

$$\int u^n \, du \xrightarrow{OP} \frac{u^{n+1}}{n+1}$$

PET forms a relational model for the rule:

$$7 \int x^2 \, dx \rightarrow 7 \frac{x^3}{3}$$

The heuristic for selecting perturbation candidates focuses on descriptors in *PRE* which are augmented with *eq* relations. With this guidance, perturbation generates two examples: $\int x^2 \, dx$ and $7 \int dx$. Of these, the first is classified by perturbation as a positive example and the second is classified as a negative example. The leading 7 is thereby recognized as a spurious descriptor and removed, yielding the generalized rule:

$$\int x^2 \, dx \rightarrow \frac{x^3}{3}$$

Of course, this technique is more useful on large rules with multiple spurious descriptors.

6.3. Review of experimental goal regression

Experimental goal regression is an experientially based approximation to analytic goal regression. In PET, experimental goal regression guides the selection of perturbation candidates. Candidates are selected which test features of heuristic rules which appear to be over-specialized. These features are found by back-propagating goal conditions to subgoal descriptions. This back-propagation is possible due to the explicit links in learned episodes relational models. This technique is especially effective at improving the learning rate as episodes grow in length since experimental goal regression may take place through episodes of arbitrary length.

7. A brief comparison

In this section we briefly compare PET both with learning programs that are based on an empirical or data-intensive mechanism and programs that are based on a reasoned or analytical mechanism. From this comparison we suggest that PET lies between empirical and analytic learning mechanisms on several dimensions.

7.1. Empirical learning methods

Methods that depend on large amounts of data typically share many characteristics. These systems have been called similarity-based learning systems (Mitchell, *et al*, 1986), but we prefer to use the term data-based or empirically based, since Langley's Sage.2 (1983) and Quinlan's ID3 (1986) are both data-intensive but focus on differences or discriminations rather than similarities.

Quinlan's (1986) ID3 program builds decision trees from numerous training instances to classify new instances. All states for which the application of operator *OP* leads to a solution are classified as Good-*OP*-states. Given training of this form, the decision tree learned by the ID3 algorithm represents a heuristic description of the general class of problems for which OP is effective. Since ID3 uses an attribute-value representation scheme, extensions would be required to represent operators for symbolic integration. However we guess that with appropriate data, much larger than PET would require, ID3 could form problem solving heuristics for simultaneous linear equation and symbolic integration.

Unfortunately, empirically based methods require huge numbers of data points, which are collected in an undirected and uncontrolled way. If the observed instances are representative of all instances, learning will be effective. However, a large number of training instances does not guarantee that the necessary examples for the correct generalization have been observed. In the end, the final generality of the learned concept relies on a fortuitous selection of training instances.

7.2. Explanation-based learning

Explanation-based learning systems accept a single instance of a concept (or a good application of an operator) and construct an explanation (often a proof) of why the instance satisfies the concept. The explanation is then used to form a generalization of the given instance. A number of explanation-based learning systems have been built and demonstrated in a variety of task domains (Mitchell, *et al.*, 1986; Minton, 1984; DeJong, 1981; DeJong, 1983).

Constructing a proof or an explanation takes different forms in the various systems. One form, which we call analytic goal regression, depends on computing,

as accurately as possible, the goal regression of a goal or subgoal through the operator. However, we have identified three requirements that limit the application of analytic goal regression (Porter and Kibler, 1985). These requirements are:

- Procedural descriptions of goals – In order to compute the inverse image of a goal with respect to an operator, the goal description must be a pattern, a relation, or a predicate. To our knowledge there is no means to compute the goal regression of a goal which has a procedural definition. Unfortunately, procedural descriptions abound (e.g. the definition of checkmate in chess).
- Operator invertibility – Even if the goal has a simple description, it is also necessary that the operator be described non-procedurally. STRIPS-like operator definitions are suitable candidates for goal regression. Unfortunately, many operators are best represented procedurally (e.g., an operator which replaces all occurrences of term T by term T′).
- Relative closure of concept language – Finally, assuming the goal and operator have simple descriptions, the computed goal regression may involve disjunctions or negations. For STRIPS-like operators this only occurs when the operator is not fully instantiated. The appearance of disjunctions or negations in the goal regression result usually takes the expression outside the original concept description language.

Each of these difficulties has been partially addressed by explanation-based techniques. In particular, one can approximate the procedural description of a goal by a pattern or relational description which satisfies the procedure. If the operators do not have computable inverses, one can provide the system with explicit definitions of the inverses, as is done in LEX2 (Utgoff, 1983). Finally, the problem with disjunctions is mitigated by choosing that disjunct which matches the current instance, as is done by Minton (1984) and Mahadevan (1985). Note that none of these difficulties occur with experimental goal regression as discussed in more detail by Porter and Kibler (1985).

8. Final remarks

In this paper we have described a collection of techniques which are integrated into a single, effective mechanism for learning problem solving heuristics. More specifically, we use perturbation to learn **when** operators should be applied, episodic learning to understand **why** operators are applied, and relational modeling to construct **what** the operator does and how the preconditions relate to the postconditions. This integration of techniques, which we call experimental goal-regression, lies between empirically and explanation-based learning on a number of dimensions.

Experimental goal regression achieves its power through **directed experimentation** rather than by summarizing massive amounts of data with empirically-based

methods or by careful and costly analysis with explanation-based methods. With respect to background knowledge, experimental goal regression requires a stronger domain theory than empirical learning methods, but a weaker domain theory than that required by explanation-based or analytic techniques. On this dimension, empirical methods are least restrictive in their domains of applicability, explanation-based methods are most restrictive, and experimental goal regression holds an intermediary position. With respect to the number of instances required to learn a heuristic, EGR requires more examples than explanation-based methods, but much less than that required by an empirical method. Finally, with respect to correctness of the heuristics formed, only explanation-based methods are guaranteed to produce correct heuristics, but EGR usually produces good heuristics. The effectiveness of EGR and the rules it generates has been demonstrated by the PET learning and problem solving system in the task domains of simultaneous linear equations and symbolic integration.

There are a number of serious questions about the learning of problem solving which this paper and other research on machine learning have not addressed. In particular, machine learning has not successfully dealt with:

- Organization of knowledge – Most systems have constructed few rules (heuristics) and have not needed to address the problem of organizing learned knowledge into a coherent, efficiently accessible whole.
- Concept Language – All learning systems depend on having the right, or nearly right, conceptual vocabulary. A search for a concept description is conducted through a space of combinations of these primitives. But where do these primitives come from? Analytical goal regression suggests one approach to this problem, but more work is called for.
- Strategy Acquisition – Nearly all research in learning problem solving has concentrated on generalizing state descriptions, typically forming heuristics for when an operator should be applied. While some work has been directed at forming sequences of operators, little work has focused on acquiring strategic and tactical knowledge.

The problem of concept formation, organization and integration is the focus of our further research. In particular, we are considering an exemplar based representation of concepts (Smith and Medin, 1981) which is organized in a hierarchy of frames. In this context, learning is the evolutionary addition and modification of slots, slot-fillers, and frames. This network of concepts will be used by an expert problem solver as we believe that the structure and content of knowledge both determines and is determined by its use.

Acknowledgement

Support for this research was provided by the Army Research Office, under grant number ARO DAAG29-84-K-0060, and by a gift from Hughes AI Research Center, Calabasas, California.

References

Amarel, S. (1968). On representations of problems of reasoning about actions. In D. Michie (Ed.), *Machine Intelligence 3*. Edinburgh University Press.

Anzai, Y. (1978). Learning strategies by computer. *Proceedings of the Second Canadian Society for the Computational Study of Intelligence* (pp. 181–190). Toronto, Canada.

Brazdil, P. (1978). Experimental learning model. *Proceedings of the Conference on Artificial Intelligence and Simulation of Behaviour* (pp. 46–50). Hamburg, West Germany.

DeJong, G. (1981). Generalization based on explanations. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence* (pp. 67–69). Vancouver, Canada.

DeJong, G. (1983). Acquiring schemata through understanding and generalizing plans. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence* (pp. 462–464). Karlsruhe, West Germany.

Dijkstra, E. (1975). Guarded commands, non-determinancy and formal derivation of programs. *Communications of the Association for Computing Machinery 18*, 453–457.

Iba, G. (1985). Learning by discovering macros in puzzle solving. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 640–643). Los Angeles, California.

Kibler, D., & Porter, B. (1983a). Perturbation: A means for guiding generalization. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence* (pp. 415–418). Karlsruhe, West Germany.

Kibler, D., & Porter, B. (1983b). Episodic learning. *Proceedings of the Third National Conference on Artificial Intelligence* (pp. 191–196). Washington, D.C.

Langley, P. (1980). Finding common paths as a learning mechanism. *Proceedings of the Third Conference of the Canadian Society for Computational Studies of Intelligence* (pp. 12–19). Victoria, B.C., Canada.

Langley, P. (1983). Learning effective search heuristics. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence* (pp. 419–421). Karlsruhe, West Germany.

Langley, P., & Ohlsson, S. (1984). Automated cognitive modeling. *Proceedings of the Fourth National Conference on Artificial Intelligence* (pp. 193–197). Washington, D.C.

Mahadevan, S. (1985). Verification based learning: A generalization strategy for inferring problem reduction methods. *Proceedings of the Ninth International Conference on Artificial Intelligence* (pp. 616–623). Los Angeles, California.

Michalski, R.S. (1983). A theory and methodology of inductive learning. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, (Eds.), *Machine learning*. Palo Alto: Tioga.

Minton, S. (1984). Constraint-based generalization: Learning game-playing plans from single examples. *Proceedings of the Fourth National Conference on Artificial Intelligence* (pp. 251–254). Austin, Texas.

Minton, S. (1985). Selectively generalizing plans for problem solving. *Proceedings of the Ninth International Conference on Artificial Intelligence* (pp. 596–599). Los Angeles, California.

Mitchell, T.M., Utgoff, P.E., Nudel, B., & Banerji, R. (1983). Learning by experimentation: Acquiring and refining problem solving heuristics. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, (Eds.), *Machine learning*. Palo Alto: Tioga.

Mitchell, T.M., Mahadevan, S., & Steinberg, L.I. (1985). LEAP: A learning apprentice for VLSI design.

Proceedings of the Ninth International Joint Conference on Artificial Intelligence (pp. 573–580). Austin, Texas.

Mitchell, T.M., Keller, R.M., & Kedar-Cabelli, S.T. (1986). Explanation-based generalization: A unifying view. *Machine learning, 1*, 47–80.

Newell, A., & Simon, H.A. (1961). The simulation of human thought. *Current trends in psychological theory*. Pittsburgh: University of Pittsburgh Press.

Neves, D.M. (1978). A computer program that learns algebraic procedures by examining examples and working problems in a textbook. *Proceedings of the Second Canadian Society for the Computational Study of Intelligence* (pp. 191–195). Toronto, Ontario.

Porter, B., & Kibler, D. (1984). Learning operator transformations. *Proceedings of the Fourth National Conference on Artificial Intelligence* (pp. 278–282). Austin, Texas.

Porter, B. (1984). Learning Problem Solving. PhD Dissertation, Department of Information and Computer Science, University of California, Irvine.

Porter, B., & Kibler, D. (1985). A comparison of analytic and experimental goal regression for machine learning. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 555–559). Los Angeles, California.

Quinlan, J.R. (1986). Induction of decision trees. *Machine learning, 1*, 81–106.

Sammut, C., & Banerji, R. (1983). Hierarchical memories: an aid to concept learning. *Proceedings of the Second International Machine Learning Workshop* (pp. 74–80). Allerton House, Illinois.

Samuel, A.L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development 3*.

Smith, E., & Medin, D. (1981). *Categories and concepts*. Cambridge: Harvard University Press.

Utgoff, P. (1983). Adjusting bias in concept learning. *Proceedings of the Second International Machine Learning Workshop* (pp. 105–109). Allerton House, Illinois.

Utgoff, P. (1984). Shift of Bias for Inductive Concept Learning. PhD Dissertation, Department of Computer Science, Rutgers University.

Vere, S.A. (1977). Induction of relational productions in the presence of background information. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence* (pp. 349–355). Cambridge, Massachusetts.

Waldinger, R. (1977). Achieving several goals simultaneously. In E.W. Elcock, and D. Michie, (Eds.), *Machine Intelligence 8*. New York: Halstead and Wiley.

Winston, P.H. (1975). Learning structural descriptions from examples. In Winston, P.H. (Ed.), *The psychology of computer vision*. New York: McGraw-Hill.