Learning Problem Solving:  
A Proposal for Continued Research

Bruce W. Porter  
Artificial Intelligence Laboratory  
Computer Sciences Department  
March 1985

AI-TR-85-03
Learning Problem Solving:
A Proposal for Continued Research

Bruce W. Porter
Computer Sciences Department
University of Texas at Austin

Abstract
Many of the tasks which people perform involve problem solving: applying a sequence of operators to solve a problem. This research explores how efficient problem solutions are discovered from the myriad of less efficient alternatives. Results in machine learning are applied both to explain findings from psychological experimentation and to expand the utility of computers.

Learning to problem solve requires acquiring multiple forms of knowledge. Problem solving is viewed as a search of a state-space formulation of a problem. With this formalism, operators are applied to states to transit from the initial state to the goal state. The learning task is to acquire knowledge of the state-space to guide search. In particular, three forms of knowledge are required: why each operator is useful, when to apply each operator, and what each operator does. This research builds on an existing PROLOG system that learns problem solving in the domains of simultaneous linear equations and symbolic integration. First the current learning system is described. Then new research directions are proposed. These include:

- critically comparing machine learning techniques demonstrated in a variety of problem solving domains.
- using learned knowledge to guide the acquisition of further learning.
- dynamically re-defining the concept description language by discovering useful descriptors from the training.

Introduction
In a weak form, problem solving can be modelled as state space search. Viewed this way, a problem solving task is represented by an initial state (the problem to be solved), a set of goal states (solved problems), and a set of operators which can be applied to solve the problem. This defines a state-space, as shown in figure 1. Problem solving in this model consists of searching for a sequence of operators which transit from the initial state to one of the goal states.

Much research in computer science has focused on efficient methods of searching a state space [NILS80, pp 53–128]. Uninformed search simply explores the search space by random selection of operators to apply. The problem, of course,
is that the search space may be so large as to prohibit uninformed search. An alternative is informed search which selects operators to apply based on information gleened from the current state in the search. The operator selection knowledge is embedded in heuristics which may directly recommend an operator, block an operator judged useless, or select an operator based on the estimated quality of resulting states. Any task which can be formulated with a state space representation is a candidate for informed search.

The approach to learning problem solving proposed here is to replace search by knowledge. The effect of this on problem solving is that operators are selected to apply to states based on knowledge of the search space rather than by trying alternatives with little domain knowledge to distinguish among them. Specifically, there are three types of search space knowledge needed:

1) Knowledge of Solution Paths – As shown in figure 2, solution paths are sequences of operators which are applied serially. These sequences transit from an initial state to a goal state. Knowledge of solution paths improves problem solving because search is eliminated. The solution path serves as a procedure which dictates the solution with no guess-work. Learning solution paths is the subject of section 2.1 which proposes a technique called episodic learning. Episodic learning builds solution paths by learning why each operator is useful – i.e., the role of each operator in solution paths.
2) Clusters of States – As shown in figure 3, clusters are groups of states which can be regarded as a single state. This is critical to problem solving because the state space is potentially infinite. For problem solving purposes, clustering replaces an infinite set of states by a finite set of clusters. Clusters are formed by grouping states which play the same role in a solution path. That is, if a solution path transits from state1 to a goal then the group of states in the state space for which the same solution path is effective defines a cluster which includes state1. Learning clusters of states is the subject of section 2.2 which discusses the technique of perturbation. Perturbation is an automated method of discovering when operators should be applied – i.e., the cluster of states in which each operator is effective.

3) Operator Representations – As shown in figure 4, there is a spectrum of operator representations. At one extreme are opaque representations which cannot be easily analyzed. At the other extreme are transparent representations which can be analyzed. The world abounds with opaque operators. With opaque representations, the particular state to which the operator is applied and the state resulting from the application can be observed. But the general transformation performed by the operator is hidden. Transparent operator representations, on the other hand, are unnatural in the world. These representations make explicit the transformation performed by the operator. The advantage of transparent representations is that a problem solver can reason with the operator “semantics.” Since it is unreasonable to
Figure 3

Clusters of States

Figure 4

A Spectrum of Operator Representations

assume that operators are represented transparently, section 2.3 discusses a technique for learning transparent representations from examples of application of opaque operators. The representations learned are called relational models. Relational models explicitly represent what individual operators do.

This knowledge is useful for solving any problem which can be formulated as a state space search. In general, this knowledge cannot be built into a problem solver because it is difficult to obtain and encode (the knowledge acquisition problem).

This research explores a method for learning this essential state space knowledge. The technique builds on the inter-connection of a problem solver and a
learner. The problem solver is proficient at applying operators to states. The learner observes the problem solver to compile knowledge which can be used for subsequent problem solving. Initially, the problem solver is presented with problems without any knowledge of how to proceed. In this naïve state, there are two alternatives: performing uninformed search or asking for help. Since search techniques are not the focus of this research, the latter alternative is adopted. Knowledge is learned when the teacher advises the problem solver of the appropriate action. The problem with the advice is that it is overly specific - i.e., it refers only to the current problem solving state. The learner discovers the general lesson from the specific advice by experimentation. The learner proposes experiments to the problem solver and draws conclusions from the results.

2. The Current Learning System

This section discusses the capabilities and limitations of the current learning system. The learning system has been implemented in PROLOG and is referred to as the PET system. Section 3 proposes five research directions for improving the current learning model and implementation of PET.

2.1 Episodic Learning

With the aid of a teacher, students learn to solve simultaneous linear equations and symbolic integration problems. This section discusses this learning process with emphasis on the role of the teacher, the student’s prior knowledge about math, and the result of the learning. These general comments are then made more specific with examples of episodic learning in the domain of symbolic integration.

The teacher plays the role of the environment in the model of learning introduced in section 1. To fill this role, the teacher performs the following two functions:

1) The teacher demonstrates effective operator sequences for solving specific problems in the problem domain. From this the student learns episodes for problem solving in this domain. The student must discover the purpose of each operator in the episode. This information is not made explicit by the teacher but is required for constructing episodes whose usefulness extends beyond the small set of examples worked out by the teacher.

2) The teacher presents specific examples of general concepts to the learning element, or student. From this the student discovers the general concepts by induction over the set of examples. This conforms to the model of learning by example. Advice to apply OP to problem state S is a specific example of the general concept "problem states in which OP is useful." As with (1), this form of generalization also extends the student’s knowledge beyond the limited training provided by the teacher.
The first of these learning tasks is the topic of this section. The second task is elaborated on in the next section. These issues are jointly addressed in the PET system, but it is useful to describe them separately here.

In machine learning research, there is a spectrum of levels of involvement by the teacher in the learning cycle. The teacher's role in the PET learning cycle is to provide advice on which operator to apply to a particular problem when the PET performance element is "stumped." This advice is followed and PET attempts to understand why it is appropriate. If PET can determine the purpose of the operator in this instance, then PET learns from the exercise. Otherwise, PET must "bear-with" the teacher until something understandable happens. In either case, PET continues with the solution path, requesting help when necessary, until the problem is solved.

A student learning to solve simultaneous linear equations or symbolic integration problems is assumed to have prior knowledge of certain mathematical principles and procedures. This knowledge is in three categories:

1) knowledge of operators in the domain – The student is assumed to know how to apply operators to problems in the domain. This involves pattern matching general operator definitions with specific problems and propagating bindings through the operator definition.

2) knowledge of goals in the domain – The student is assumed to know the general form of a solved problem in the domain. This serves to guide solution paths to the final goal.

3) knowledge of generalization – The student is assumed to be able to generalize rules from examples. This is not required for episodic learning and will be covered in the next section.

The task of the learning element is to discover knowledge about the problem space which improves problem solving in the space. Episodic learning contributes part of this knowledge: knowledge of why individual operators are useful in problem solving. That is, episodic learning is concerned with discovering the purpose of individual operators in operator sequences. The assumption is that knowledge of parts contributes to knowledge of the whole.

Episodic learning consists of acquiring two types of state space knowledge: state evaluations and useful solution paths. State evaluations are a measure of the quality of states in the state space. The quality of a state \( S \) is measured by the distance from \( S \) to a goal state, where low distances indicate high quality. In PET, this distance measure is called the score of a state. In particular, goal states have score zero.
Learning state evaluations is done by incrementally backing-up values from known states to "neighboring" states. This process starts with goal states. States $s_1$ and $s_2 \in S$ are neighbors if there is a single operator, OP, such that the application of OP to $s_1$ yields $s_2$. States which are neighbors of goal states are assigned a score of one. Then, states which neighbor states of score one (and are not goal states) are assigned a score of two, and so on.

There are two problems with this straightforward approach to state evaluation. First, the state space is potentially infinite and the set of states which need to be evaluated must be limited. Second, the state space does not provide explicit information on which states are neighbors. These two problems are both addressed by the second type of knowledge discussed above: knowledge of solution paths. Solution paths are states which are passed through by an operator sequence, or episode. Those states which are included in solution paths are exactly the set of states which PET evaluates. Also, solution paths make explicit neighboring states in the state space. So, knowledge of state evaluation is intimately connected with knowledge of solution paths.

PET incrementally builds solution paths backwards from the goal state. That is, a solution path of length $n$ states is defined by an episode (operator sequence) of length $n - 1$ operators. Consider the episode $E$ defined by:

$$s_1^{\text{op}_1} s_2^{\text{op}_2} \cdots s_{n-1}^{\text{op}_{n-1}} s_n$$

where $s_n$ is a goal state. PET is constrained to only learn from a state transition in an episode if the purpose of the transition can be understood from existing knowledge. Initially, PET only knows the state evaluation of goal states, which have a score of zero. Therefore, initially PET is restricted to learning transitions to goal states, in which the purpose of the transition is clear. Assuming PET has not learned any other episodes and $E$ is presented by the teacher, PET learns that state $s_{n-1}$ is a neighbor of $s_n$. This enables PET to assign a score of one to $s_{n-1}$. Further, PET learns the solution path of $\text{op}_{n-1}$ to transit from $s_{n-1}$ to $s_n$.

From this initial learning, PET expands its ability to understand the purpose of transitions. Now transitions which achieve state $s_{n-1}$ are recognized as useful. If episode $E$ is again presented by the teacher, PET learns that state $s_{n-2}$ is a neighbor of $s_{n-1}$. This enables PET to assign a score of two to $s_{n-2}$. Further, PET learns the solution path of $\text{op}_{n-2}$ to transit from $s_{n-2}$ to $s_{n-1}$. Thus, PET is incrementally learning $E$ backwards from the goal state.

State space knowledge is encoded in augmented production rules. The form of a production rule for a transition from $state_1$ to $state_2$ via operator $op$ is:

$$\text{score}: state_1 \rightarrow op$$
Figure 5

A Lattice of Solution Paths

where score is the state evaluation for state1. So, the knowledge derived from episode E above is:

1: \( s_{n-1} \rightarrow op_{n-1} \)
2: \( s_{n-2} \rightarrow op_{n-2} \)
\vdots
\( (n - 1): s_1 \rightarrow op_1 \)

As shown in figure 5, episodic learning constructs a lattice structure of solution paths. Each node of the lattice is a learned sub-goal. Arcs between nodes represent transitions which have proved useful in past problem solving. This knowledge of episodes is applied to new problem solving by navigating through the lattice starting with the initial state. If there are multiple successors of a node, then the problem solver selects the successor with the lowest score. Should the lattice not include the initial state, then weak problem solving methods are applied until a state is reached which is in the lattice. Problem solving with learned episodes replaces search with knowledge of past experience encoded in the lattice.

We now present a short example of episodic learning in the domain of symbolic integration. 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 there are eighteen operators in this domain, for present purposes, assume there are only two:

\[ \text{OP1} \quad \int x^n \, dx \rightarrow \frac{x^{n+1}}{n+1} + C \]
\[ \text{OP2} \quad \int a \, \text{poly}(x) \, dx \rightarrow a \int \text{poly}(x) \, dx \]

OP1 integrates a power of the variable of integration, \( x \). OP2 extracts a constant, \( a \), from the expression being integrated.
As before, PET initially has a single goal. In the domain of symbolic integration, the goal is to eliminate the integral. Incremental learning constrains learning to only those operators applied to states which yield a goal state. Suppose the teacher presents the training instance:

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

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

\[ 7 \int x^2 \, dx \]  
(State 2)

State 2 is not a goal state, so PET does not learn from the training.

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

\[ \frac{7x^3}{3} \]  
(State 3)

which is a goal state. PET backs-up evaluation from state 3 (the goal state) to state 2 and assigns state 2 the score 1. State 2 is recorded as a sub-goal in the problem solving search space and the heuristic rule:

\[ \text{State2} \hspace{1cm} 1: 7 \int x^2 \, dx \rightarrow OP1 \]

is added to the rulebase.

Now, the original training instance (state 1 with advice to apply OP2) can enable learning. Applying OP1 to state 1 yields state 2 which is a recognized subgoal. State 1 is assigned a score of 2 by backing up evaluation from state 2. From this PET learns the subgoal state 1 and the heuristic rule:

\[ \text{State1} \hspace{1cm} 2: 7 \int x^2 \, dx \rightarrow OP2 \]

It is important to note that the episode learned is "loosely packaged." That is, rules from the rulebase can be applied in any order so long as the scores of the rules in the sequence are non-increasing. This enables branching within episodes when a shorter path can be selected over a longer path (path length measured by state evaluation). At each state in the solution path, an operator is selected which most advances the progress to a goal.

In the domain of symbolic integration, PET formed episodes which included rules for eighteen operators. The longest episode was eleven rule applications from
\[
\int \sin^7 x \, dx \\
poly^{n+1} \rightarrow poly^n \cdot poly \\
\int \sin^6 x \sin x \, dx \\
poly^{2n} \rightarrow poly^{2n} \\
\int (\sin^2 x)^3 \sin x \, dx \\
\sin^2 x \rightarrow 1 - \cos^2 x \\
\int (1 - \cos^2 x)^3 \sin x \, dx \\
\text{substitute} (u = \cos x, \ du = -\sin x) \\
\int -(1 - u^2)^3 \, du \\
\int a \cdot poly(x) \, dx \rightarrow a \int poly(x) \, dx \\
-1 \int (1 - u^2)^3 \, du \\
\text{Binomial Theorem} \\
\vdots \\
-1 \int 1 - 2u^2 + u^4 \, du \\
\int (du + dv) \rightarrow \int du + \int dv \\
-1(\int du - \int 2u^2 \, du + \int u^4 \, du) \\
\int du \rightarrow u \\
-1(u - \int 2u^2 \, du + \int u^4 \, du) \\
\int a \cdot poly(x) \, dx \rightarrow a \int poly(x) \, dx \\
-1(u - 2\int u^2 \, du + \int u^4 \, du) \\
\int u^n \, du \rightarrow \frac{u^{n+1}}{n+1} \\
-1(u - 2\frac{u^3}{3} + \frac{u^5}{5})
\]

**Figure 6**

An Episode Linking Eleven Rules

initial state to goal state. The initial state was \( \int \sin^7 x \, dx \) and the solution sequence is shown in figure 6. The intermediate states in the sequence appear to be diverting from a simplified goal state. Episodes bridge these necessary "digressions."

Once learned, PET can re-play an episode in whole or in part. The limiting factor is that the episode is overly-specific. The next section discusses a technique for generalizing episodes so that they apply to a class of problems. The technique
utilizes the structures built by episodic learning to partially automate the role of the teacher in learning by examples.

2.2 Perturbation

Perturbation is a technique for reducing the teacher’s role in learning by example. The teacher has two responsibilities: generating training instances for a concept and classifying them as positive or negative examples of the concept. From this, the learning element forms a general concept description which is complete and consistent with respect to the training set.

Perturbation relies on inherent regularity in natural domains. Given a training instance $I$ for concept $C$ provided by the teacher, perturbation makes small changes to $I$. The inherent assumption is that $I$ is prototypical of $C$. This implies that an instance which is highly similar to $I$ will also be a positive example of concept $C$. Perturbation generates and classifies the state space “neighbors” of $I$ to form a state cluster of positive examples of $C$. This state cluster is then provided to a generalization algorithm to form a description of $C$.

Specifically, perturbation automatically generates a set of positive and negative examples of a concept by a simple two step algorithm. First the examples are generated then they are classified.

Perturbation generates examples by applying a set of perturbation operators to a single teacher supplied example, $I$. Each perturbation operator selects a single feature $f$ in $I$ and slightly modifies it. Modifications are of two types: replace $f$ by the null feature (effectively removing the feature altogether) or replace $f$ by a sibling of $f$ in a concept hierarchy tree containing $f$. The latter type of modification generates a set of perturbation operators since $f$ may either have multiple siblings or be in multiple trees. Each perturbation operator generates an example $I'$ which is highly similar to $I$.

Perturbation classifies examples by exploiting the fact that problem solving domains are reactive. Consider a training instance $I$ which is classified by the teacher as a positive example of the concept “states in which operator $OP$ is effective.” The generation step of perturbation creates a set of examples which includes example $I'$. The classification step of perturbation determines whether $I'$ is a positive or negative example of the concept by experimentation. Specifically, $I'$ is a positive example if and only if the effect of $OP$ on $I'$ is the same as the effect of $OP$ on $I$. The effect of an operator on a state is the transition achieved by the operator. So,

$$apply(OP, I) = apply(OP, I') \iff I' \text{ is a positive example}$$

An important advantage of the PET perturbation technique is that examples are efficiently classified. The LEX system [MICH83], by contrast, requires that the
Figure 7
Perturbation Generates and Classifies

Multiple Examples from a Single Positive Example

problem solver be applied to an example. This involves a full $n$-level expansion of the search space which terminates when a goal state is reached. If the example lies on the shortest solution path then it is classified as positive, otherwise it is negative. The perturbation technique simply performs a 1-level search to determine the immediate successor of the example.

Furthermore, examples are efficiently generated by perturbation. Although the efficiency ultimately depends on the instance description language, perturbation simply selects a feature of the instance, finds it in the concept hierarchy tree (a linear search of the leaves, at worst) and selects a sibling (requiring two arc transitions). The perturbation process is shown in figure 7.

The advantage of perturbation is that it removes irrelevant detail from a concept description. The relevance of each feature of an example is tested in two ways. First, the feature is tested to determine if it can be completely removed. Second, if the feature cannot be removed, it is tested to determine if it can be generalized. Thus, perturbation generalizes each positive example without teacher involvement before generalizing it with the current concept description.
In summary, perturbation is a technique for automatically generating and classifying near-examples and near-misses of a concept. Standard generalization techniques can then be applied. Perturbation semi-automates the learning process by removing spurious details from examples.

We now present a short example of perturbation in the domain of symbolic integration. Following section X, assume that there are only two operators in the domain:

\[
OP1 \quad \int x^n \, dx \rightarrow \frac{x^{n+1}}{n+1} + C
\]

\[
OP2 \quad \int a \, poly(x) \, dx \rightarrow a \int poly(x) \, dx
\]

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

The first rule formed by episodic learning in section 2.1 is:

\[
1:7 \int x^2 \, dx \rightarrow OP1
\]

This rule is overly specific and is generalized by PET.

First, perturbation operators are applied to generate a set of examples. Four of these examples are:

\[
\begin{align*}
E_1 & \quad \int \Box \, dx & \quad 7 \int \Box \, dx & \quad 8 \int x^2 \, dx & \quad 7 \int x^3 \, dx \\
E_2 & & \quad E_3 & & \quad E_4
\end{align*}
\]

\(E_1, E_3\) 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 OP1 with \(E_1, E_3\) and \(E_4\) yields the new rule:

\[
1: \int x^{positive} \, dx \rightarrow OP1
\]

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 OP1. Given the positive example:

\[
\int x^{-3} \, dx
\]
PET minimally generalizes to:

\[ 1: \int x^{\text{nonzero}} \, dx \rightarrow OP1 \]

2.3 Relational Models

The world abounds with opaque operators. Opaque operator representations hide the semantics of operators. The transformation performed by the operator is concealed by the operator representation. Examples of opaque operators are familiar to everyone who has learned a task by observing experts who are proficient at the task. Their actions are recognized as legal but the observed solution seems magical. Their instruction cannot be fully assimilated without an understanding of the transformation performed at each step in the solution path.

One of the requirements for gaining expert problem solving skills is to acquire transparent representations for operators. Transparent operator representations reveal the "inner-workings" of the operator. This enables reasoning with operator definitions. This section examines two issues:

1) how transparent operator representations can be learned from opaque representations.

2) how transparent operator representations can improve the process of acquiring problem solving heuristics.

Transparent operator representations are essential for reasoning about operator transformations. For example, Waldinger's planning system [WALD77] demonstrates an effective use of transparent operator representations. The planner assumes that operators are represented with lists of pre-conditions, delete-conditions and add-conditions, *ala* STRIPS. The planner solves multiple goals simultaneously and must address the problem of sub-goal conflicts. One approach to handling sub-goal conflicts is demonstrated by HACKER [SUSS73] and INTERPLAN [TATE75]. These planning systems simply backtrack when a conflict is encountered. A couple of goals are reordered and the planners try to solve the new problem description. However, Waldinger exploits the operator representation to discover goal conflicts before they arise in planning. Goal regression is used to back-up constraints from each of the goal descriptions. If any constraints conflict, then the planner tries goal re-ordering. This ensures that the resulting plan is free of goal conflicts. As demonstrated by Waldinger, goal regression is a powerful reasoning strategy. But, as will be discussed in section 2.3, it relies on transparent operator representations.

This section discusses a technique for learning transparent operator representations from examples of operator applications. These transparent representations are called relational models. The central issue in learning relational-models is the
utilization of existing "background knowledge" about the domain. Several techniques for incorporating background knowledge into an evolving concept description are discussed in the next two sections. This leads to the technique of learning relational models which is demonstrated with the PET system.

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 2.1. In section 2.1, heuristic rules are of the form:

$PRE$ state description $\rightarrow$ $OP$

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 $S$.

The key difference between the two forms of rules is that the latter form explicitly represents the state description which results from the operator transition. This follows the style of rules proposed by Amarel [AMAR68]. One of the advantages of this style is that the representation helps to constrain inappropriate operator applications during problem solving. In addition to limiting operator applications to those states which match preconditions, Amarel-style rules require that the postconditions match as well. This form of heuristic rule thereby represents an entire transition.

In PET, these heuristic rules represent 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 as the right hand side of the rule.

This form of heuristic rule is generalized using "standard" generalization techniques. For example, the generalization technique which is used with perturbation (section 2.2) forms generalizations of rules of the form \( PRE \rightarrow OP \). Applying the same algorithm to states resulting from \( OP \)'s application yields a generalization of \( POST \). For each operator \( OP \) in a problem solving domain, PET uses the dropping conditions and climbing hierarchy tree generalization operators to induce general forms both for states in which \( OP \) is recommended and for states resulting from recommended applications.

However, this generalization scheme can yield unusable generalizations. 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 the increment of the \( x \) exponent in \( PRE \).

2) the denominator in \( POST \) is the increment of the \( x \) exponent in \( PRE \).
Further, generalizing with a third positive example, \( \int y^4 \, dy \), yields the generalization:

\[
\begin{array}{c}
\int \\
\bigg( \\
\bigg) \\
\var \\
\bigg) \\
\bigg( \\
\bigg) \\
pos
\end{array}
\text{op}\rightarrow
\begin{array}{c}
\int \\
\bigg( \\
\bigg) \\
\bigg) \\
\var \\
\bigg) \\
pos
\end{array}
\]

and a third essential constraint is lost:

3) the variable in the numerator of \textit{POST} is the variable of integration in \textit{PRE}.

\begin{tabular}{|l|l|}
\hline
Relation & Semantics \\
\hline
equal(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 \\
\hline
\end{tabular}

Table 1

Background Knowledge for Symbolic Integration

Relational models are an augmentation of this form of heuristic rule. An important role of this augmentation is to explicitly represent constraints between terms in \textit{PRE} and \textit{POST} so that they are not lost during generalization. Background domain knowledge augments the heuristic rules to relate terms in \textit{PRE} with terms in \textit{POST}. The previous section suggested some useful background relations for simultaneous linear equations. PET uses the set of relations listed in table 1 in the domain of symbolic integration.

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

The augmentation alters the interpretation of an heuristic rule for operator $OP$. The augmentation is instantiated with terms from $PRE$ and $POST$ and the resulting rule is interpreted as:

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

One of the applications of relational models to learning problem solving was described in [PORT84A, PORT84B]. The application is a variant of goal regression and is called constraint back-propagation. Constraint back-propagation enables the learner to detect generality in one rule of an episode and to use this knowledge to guide the generalization of other rules in the episode. Using the technique, the learner efficiently refines the knowledge base with little teacher involvement.

Constraint back-propagation in the domain of symbolic integration is illustrated with an example from Utgoff [UTGO83]. The example demonstrates how PET automatically selects the single perturbation candidate from the enormous space of possibilities which enables useful concept generalization.
Assume that from prior training for operator

\[ OP1 : \sin^2 x \rightarrow 1 - \cos^2 x \]

PET has acquired the following relational model:

![Image of a diagram showing a relational model for \( OP1 \).]

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

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

\[ OP2 : \sin^n x \rightarrow (\sin^2 x)^{\frac{n}{2}} \]

PET applies the operator, yielding \( \int (\sin^2 x)^3 \sin x \, dx \). As described in section 2, 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:
Now that episodic learning has associated the relational models for \( OP1 \) and \( OP2 \), perturbation operators are applied to generalize the model for \( OP2 \). The relaxed constraint in \( PRE_{op1} \) is regressed through the episode with the potential of identifying a feature of \( PRE_{op2} \) which can be relaxed (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 the feature slightly modified.

PET generalizes the original training instance with examples generated by perturbation. The following relational model is the minimal generalization of this (2 member) training set:
Intelligent selection of training instances is of even greater consequence in "scaled-up" learning domains. Unguided perturbation becomes unmanageable as the number of descriptors in rules and the branching factor of the concept hierarchy trees increase. When a learning element is knowledge-poor, unguided perturbation is a useful technique for acquiring initial knowledge. But, as more knowledge is acquired, training instances must be intelligently selected.

3. Extensions to this Research

This section discusses several extensions to this research on learning problem solving. In each case, the existing PET system serves as a foundation for the extension.

3.1 Critical Comparison of Machine Learning Techniques

Research in machine learning is quite spotty because there are few researchers and many problem domains. An essential task in any maturing science is to study the central issues addressed by the researchers. This requires abstracting from the specific implementation of the research to a set of general techniques and results. This task is further complicated by the fact that research results are demonstrated in a variety of problem solving domains.

We propose to study and publish critical comparisons of research results in machine learning. The first study we intend to undertake involves a comparison of analytical and empirical learning techniques. While past research has emphasized empirical methods, recently there has been considerable interest in analytic methods. Utgoff [UTGO83] demonstrates the use of analytic goal regression to adjust the bias inherent in the concept hierarchy trees used in the domain of symbolic integration [MITC78]. Minton [MINT84] demonstrates analytic methods with effective
learning from a single training instance in two person games like chess. Mitchell [MITC85] demonstrates the use of analytic methods for generalizing training examples into useful rules in the domain of VLSI design. Since the PET system uses both analytic and empirical learning techniques, it should serve as a good testbed for studying the tradeoffs between these approaches.

3.2 Using Relational Models to Guide Generalization

An essential part of machine learning is performing induction over a set of training instances. The result of the induction is a single generalization which is complete and consistent with respect to the training instances. Finding a generalization is hampered by the enormity of the space of candidate generalizations.

Relational models may help reduce the complexity of the search for a generalization. A relational model represents the transformation performed by an operator. In particular, the model shows the effect of the operator on the features of a state which are relevant to useful applications of the operator. This is exactly the set of features which must be generalized by the induction element.

Relational models decompose a group of features into small groups. The augmentation of the model represents the transformation of each group. Relational models constrain the generalization of two rules by requiring that the augmentation of the rules match. This is a relatively simple step which defines a high-level, gross match of the two original rules. The details are filled in by matching corresponding features of the two rules. Relational models make these correspondences explicit in the decomposition. This mitigates the problem of multiple matchings, thereby reducing the complexity of induction.

Viewed abstractly, relational models are useful during induction because they impose structure on the objects being generalized. This structure (augmentation) constrains the set of candidate generalizations by sharply reducing the possible matchings. This research will test the feasibility and payoff of using relational models to guide generalization.

3.3 Improving the Concept Description Language

It has long been recognized in machine learning that the concept description language greatly influences learning capability. Traditionally, researchers have painstakingly defined the description language so that the concepts that they wanted their system to learn were easily expressable. This shortcoming restricts the applicability of machine learning to those domains which are well enough understood to permit this heavy infusion of domain knowledge. In other words, machine learning is restricted to domains in which learning is not necessary.

There has been some research in dynamically re-defining the concept description language [UTGO83]. The assumption of this research is that the description
language should contain descriptors which are adequate for describing useful compositions of concepts. Or, in problem solving domains, the description language should enable descriptions of recurring operator sequences.

The shortcoming of this prior work is that useful operator sequences are not explicitly learned. Also lacking is an explicit representation of operator transformations. We believe that the essential domain knowledge provided by episodic learning and relational modelling in the PET system will make a significant contribution to learning concept descriptors. This research will examine techniques for dynamically adjusting the concept description language so that domain concepts are expressable.

Conclusions

Problem solving is central to many of the tasks which humans perform well and machines perform poorly. This research involves studying efficient problem solving strategies with the goal of automatically learning to problem solve. This research builds on an existing learning system called PET. PET learns three types of problem solving knowledge: why individual operators are effective, when each operator should be used, and what transformation is performed by each operator.

The PET system is a framework for continued research on learning problem solving. Specifically we propose three directions of research:

- critically comparing machine learning techniques demonstrated in a variety of problem solving domains.
- using learned knowledge to guide the acquisition of further learning.
- dynamically re-defining the concept description language by discovering useful descriptors from the training.
REFERENCES


