Rule Creation and Rule Learning through Environmental Exploration

Wei-Min Shen*
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA. 15213, U.S.A.
email: shen@cs.cmu.edu

Herbert A. Simon
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA. 15213, U.S.A.
email: has@cs.cmu.edu

Abstract

The task of *learning from environment* is specified. It requires the learner to infer the laws of the environment in terms of its percepts and actions, and use the laws to solve problems. Based on research on problem space creation and discrimination learning, this paper reports an approach in which exploration, rule creation and rule learning are coordinated in a single framework. With this approach, the system LIVE creates STRIPS-like rules by noticing the changes in the environment when actions are taken, and later refines the rules by explaining the failures of their predictions. Unlike many other learning systems, since LIVE treats learning and problem solving as interleaved activities, no training instance nor any concept hierarchy is necessary to start learning. Furthermore, the approach is capable of discovering hidden features from the environment when normal discrimination process fails to make any progress.

1 Introduction

While solving problems in a new environment, as when we learn how to swim, a learning system must explore the environment to correlate its actions with its senses, to induce the laws of the environment, and to create feasible problem representations for problem solving. We refer to this task as *learning from environment,* and give its specification in Table 1.

Given:
- Actions: Effectors executable in the environment;
- Percepts: Objects' features or relations observable from the environment;
- Constructors: A set of primitive predicates (e.g. =, >), functions (e.g. +, *) and logical connectives (e.g. A,-¬, and 3);
- Environment: A process that provides the observable information and determines the effects of the actions;
- Goal: A state description of the environment;

Learn:
- The laws of the environment that predict actions' effects and reach the given goal.

Table 1: The Definition of Learning form Environment

For example, consider the Tower of Hanoi as an environment. Suppose that the learner has actions: (pick disk$_x$ peg$_r$) and (put disk$_x$ peg$_x$), percepts: (on disk$_x$ peg$_x$), (in-hand disk$_x$), and (size> disk$_x$ disk$_y$), and constructors: A, ¬, and 3. To solve problems, the learner must learn consequences of its actions and the laws of the Tower of Hanoi. The environment enforces its laws by refusing to carry out illegal actions. The specified learning task has a wide range of complexity. It becomes harder as the learner's actions and percepts become more environment independent and the environment contains larger laws and hidden features. For example, the Tower of Hanoi problem becomes much harder if (pick disk$_x$ peg$_x$) is replaced by more primitive actions like pick(), turn-hand(0) and slide-hand(d), and (on disk$_x$ peg$_x$) is replaced by objects' shape, size, and their locations.

This paper will focus on the rule creation and rule learning approach used by a learning system called LIVE that can solve the specified task with primitive percepts (objects' features) and primitive actions (moving a hand). The learned rules are similar to STRIPS's operators, but they are used for both prediction and problem solving, and their prediction does not change the world as STRIPS'S add and delete lists. With this approach, LIVE creates new rules by noticing the changes caused by actions, and later refines them by discriminating the failures and successes of their prediction. When the discrimination process fails, hidden features will be defined

"This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 4976, Amendment 20, monitored by the:
Avionics Laboratory
Air Force Wright Aeronautical Laboratories
Aeronautical Systems Division (AFSC)
United States Air Force
Wright-Patterson AFB, OHIO 45433-6543
under Contract F33615-87-C-1499.

by searching back in the history. After briefly describing some related work and a survey of the LIVE system (details of LIVE'S other components, such as exploring, planning with incomplete rules, integrating plan execution with learning, and constructing qualitative predicates from numerical percepts, will be discussed in another paper [Shen, 1989b]), we will discuss the three parts of the rule creation and rule learning method and demonstrate them with examples.

2 Related Work

There are many approaches to the problem of rule creation and rule learning. The candidate elimination algorithm [Mitchell *et a/.*, 1983] is well known for learning conjunctive rules with a given concept hierarchy. Quinlan's ID3 [1987] is probably the best example of a system for learning disjunctive rules from many examples when learning and problem solving are separated. For discrimination learning, Vere [1980] uses it as counterfactuals, and Langley [1987] utilizes it in his SAGE system and develops a theory. More recently, Falkenhainer [1988] uses discrimination as disanalogy, and Newell and Flynn [Newell, 1987] use it to modify incorrect productions in SOAR. Carbonell and Gil [1987] also use a similar method in learning by experimentation.

With respect to automatic creation of a problem space, Hayes and Simon [1974] have built a system capable of constructing problem spaces from written English, which is extended by Yost and Newell [Newell, 1987] in the SOAR system. However, few attempts have been made to create problem spaces from interactions with an environment, although Drescher [1987] did some interesting work on implementing early Piagetian learning. As for discovering hidden features, the BACON system [Langley *et al,* 1983] creates new terms to infer laws from given data. [Mitchell *et a/.*, 1983] and [Utgoff, 1986] present methods for detecting the insufficiency of a concept description language in problem solving and define new terms. [Dietterich and Michalski, 1983] gives a survey on some of the existing *constructive induction* systems for discovering hidden features in time-independent environments (see Section 6), and present their solutions.

3 Overview of LIVE

The LIVE system is an extension of the GPS problem solving framework with a learning component that creates and learns rules through environmental exploration. Each rule consists of three parts: condition, action and prediction, all constructed in terms of the given percepts, actions and constructors. When applied forward, a rule whose condition matches the current environmental state can predict the consequence of the action performed; when applied backward, a rule that predicts the current goal can be used to propose new subgoals as if it were a problem reduction rule or inference rule. LIVE is capable of learning disjunctive rules when constructors include logical connectives $\wedge, \neg$, and $\exists$. In this case, the form of the conditions and predictions is: $P_1 \wedge P_2 \wedge \ldots \wedge \neg\exists(P_3 \wedge P_4 \wedge \ldots) \wedge \ldots \wedge \neg\exists(P_5 \wedge P_6 \wedge \ldots)$ where the P_i are predicates.

```
1 Solution Planning: find the differences between
  the goals and the current state; for each differ-
  ence find a rule, order the differences by their
  rules;
2 If the first difference has no rule, propose an
  exploration plan;
3 If an exploration plan exists, then select an ac-
  tion from it, else select the first difference with
  its rule from the solution plan;
4 If the rule is not executable, then identify new
  differences, order them by their rules, and in-
  sert them in the solution plan, go to 3;
5 Make prediction and execute the action;
6 If the outcome is expected, then go to 3;
  If the outcome is surprising, then explain the
  surprise and revise the rule set;
  If the current action is from the solution plan,
  then go to 1; else go to 3.
```

Table 2: The Outline of LIVE'S Algorithm

As we can see in the algorithm outline in Table 2, when LIVE is solving problems in a new environment, it will alternate its attention between environment exploration and problem solving because no knowledge is given at the outset. The decision for alternation mainly depends on *surprises,* situations where an action's consequences violate its prediction. When no rule can be found for solving the problem, LIVE will generate and execute an *exploration plan,* or a sequence of actions, seeking for surprises to extend the rule set. When new rules are learned, problem solving is resumed and a solution plan may be constructed through means-ends analysis. Since the correctness of a solution plan cannot be guaranteed, learning is inevitable at execution time. When a rule's prediction fails during execution, LIVE will revise the rule set and then plan another solution for the problem.

To illustrate the rule representation and the algorithm, let us consider again the Tower of Hanoi environment, and suppose the initial state is (on diskl pegl) (on disk2 pegl) (on disk3 pegl). Since LIVE starts with no rules at all, it does not know how to reach the given goal stale: (on diskl peg3) (on disk2 peg3) (on disk3 peg3). So it generates an exploration plan, say, pick up a disk from pegl and then put it down on peg2. RuleO and Rulel are created in the exploration. Rulel is defined as follows:

Condition: (in-hand $disk_x$)
Action: (put $disk_x$ peg_x) [Rulel]
Prediction: (on $disk_x$ peg_x) $\wedge \neg$(in-hand $disk_x$)

After the exploration, LIVE plans to put disks on peg3 one by one, believing that the order is unimportant. As a result, diskl is successfully put on peg3, but not disk2. After (put disk2 peg3), LIVE is surprised that disk2 is still in the hand. An explanation for the surprise is then found (details will be discussed later), and LIVE *splits* the above rule into two with the help of the explanation (the 3 quantifier is necessary because the match process examines the action and the positive condition before any negative condition, and $disk_y$ is a free variable here):

Condition: (in-hand $disk_x$) $\wedge \neg\exists$(on $disk_y$ peg_x)
Action: (put $disk_x$ peg_x) [Rule1]
Prediction: (on $disk_x$ peg_x) $\wedge$ $\neg$(in-hand $disk_x$)

Condition: (in-hand $disk_x$)$\wedge$(on $disk_y$ peg_x)
Action: (put $disk_x$ peg_x) [Rule2]
Prediction: (in-hand $disk_x$)

After these rules are learned, LIVE begins to plan another solution but finds that the current rules cause unresolvable goal interactions: no matter which goal is achieved first, it will be destroyed in order to achieve other goals. At this point, LIVE switches itself from problem solving to exploration, in the hope that new surprises will arise for learning better rules.

4 Creating New Rules through Exploration

LIVE learns from a new environment by correlating its actions with its percepts. At the very beginning, it simply executes its actions and compares the states before and after. When noticing that facts disappear and emerge as actions are taken, the system will build a new rule, using the disappeared facts as conditions, and the emerged facts plus the negation of disappeared facts as predictions. For example, to figure out what (pick $disk_x$ peg_x) does, LIVE may try (pick diskl pegl), and then find diskl moved from pegl to its hand. RuleO is then built (Rulel is built in the same way):

Condition: (on $disk_x$ peg_x)
Action: (pick $disk_x$ peg_x) [Rule0]
Prediction:(in-hand $disk_x$) $\wedge$ $\neg$(on $disk_x$ peg_x)

The main objective for rule creation is to keep the new rule as general as possible. In the example above, the rule is indeed most general because the action happens to make one and only one change. But what if there is no change at all, or there are many changes? In the case of no change, as when a robot hand does put() when nothing is in its hand, the system will build a new rule with equal conditions and predictions made by the facts related to the action. For example, a new rule about the put() will use the position of the hand both as the condition and as the prediction because the hand is the actor. In the example above, if (pick diskl pegl) did not change anything, then (on $disk_x$ peg_x) will also be the prediction because $disk_x$ and peg_x appear in the action. In the case of many changes, the system will select necessary relational changes just adequate for specifying the action. For example, suppose that a robot hand is above a stack of disks ($disk_1$, $disk_2$, ..., disk,;), when it turns away all the relations (direction= hand $disk_i$) will become false. In this case, LIVE will not include all these relations in the rule but choose one, any one, of them.

Perhaps a more important question to ask here is what if there are no given relation predicates, and all that can be perceived are objects' features such as shape, size, and location. The answer is to construct new relation predicates by noticing the features that changed [Shen, 1989b], However, we will not discuss this topic further in this paper.

5 Splitting Rules by Explaining Surprises

The newly constructed rules, as we saw in the last section, are clearly over-general and incomplete, but they serve as a springboard for further exploration. Because of their generality, LIVE will have chances to make mistakes, to be surprised, and hence to increase its knowledge about the environment.

Incomplete rules can be easily identified in LIVE, because all rules predict and LIVE compares their predictions with the actual outcomes in the environment whenever a rule's action is executed. A surprise occurs when the actual outcomes falsify the predictions, and the rule that made the predictions is the faulty rule.

LIVE uses discrimination to revise its rules. It remembers each rule's latest application, which contains the rule index, a state, and the rule's variable bindings. Once a rule causes a surprise, LIVE will search for the rule's last application, find the difference (using Langley's method [1987]) between the state now and the state then, and use the differences found to split the rule into two.

For example, suppose the current state is (on diskl peg3) (on disk3 pegl) (in-hand disk2) (size> disk3 diskl) (size> disk2 diskl) (size> disk3 disk2), and LIVE tries to put disk2 on pegl with the prediction (in-hand disk2) made by Rule2 with variable bindings: *((*$disk_y$. disk3) *(*peg_x . pegl) *(*$disk_x$. disk2)). After executing (put disk2 pegl), LIVE is surprised because disk2 is now on pegl. To explain the surprise, the system fetches the rule's last application which contains a state: (on diskl peg3) (on disk3 pegl) (in-hand disk2) (size> disk3 diskl) (size> disk2 diskl) (size> disk3 disk2), and bindings: *((disky* . diskl) *(*peg_x . peg3) *(*$disk_x$. disk2)). Comparing these two applications, LIVE finds the difference to be (size> $disk_x$ $disk_y$) A (on $disk_y$ peg_x) (the details are omitted here because readers can find similar examples in [Langley, 1987]). Based on the difference, Rule2 is then split into two new rules (shown below): one is a variant of the old rule with the condition augmented by the difference; the other is a new rule, whose condition combines the old condition with the negation of the difference, and whose prediction is the surprising consequences (constructed by the same method used in rule creation). Note that LIVE keeps both rules after splitting, and this is one of the main distinctions between LIVE's discrimination process and the one employed by Langley [1987]; for if Rule2 were thrown away when splitting Rulel, as his method does, the good Rule3 might never be found.

Condition:(in-hand $disk_x$)$\wedge$(on $disk_y$ peg_x)
$\wedge$(size> $disk_x$ $disk_z$)$\wedge$(on $disk_z$ peg_x)
Action: (put $disk_x$ peg_x) [Rule2]
Prediction: (in-hand $disk_x$)

Condition:(in-hand $disk_x$)$\wedge$(on $disk_y$ peg_x)
$\wedge\neg\exists$((size> $disk_x$ $disk_z$)$\wedge$(on $disk_z$ peg_x))
Action: (put $disk_x$ peg_x) [Rule3]
Prediction: (on $disk_x$ peg_x) $\wedge$ $\neg$(in-hand $disk_x$)

Three additional details are worth mentioning here. First, what if the discrimination process returns a wrong reason and the rule later becomes too specific? Such rules will be wasted and new rules will be re-learned. To prevent learning wasted rules again, LIVE will reject an explanation if the result of splitting is equivalent to some existing rule. Second, it seems that even though the order of surprises may cause rules' conditions to be built differently, the rules learned are always useful and effective for LIVE to reach the given goal, as we have found in many running examples. Third, if the outcomes include more changes than predicted, no surprise arises, but LIVE will remember those extra changes. In further plan execution, if these changes violate the conditions of the later rules in the plan, LIVE will then insert the changes into the rule's predictions.

6 Discovering and Assimilating Hidden Features

Previous research in discrimination learning has developed many methods for finding the critical difference between two states, but what if the two states have no difference at all, as when two pairs of green peas look exactly the same but produce different offspring? In this case, we say that the environment has hidden features, something unobservable that yet can discriminate two states that appear identical.

LIVE has two ways to discover hidden features, depending on whether an environment is time dependent or not. In a time-independent environment, where states do not depend on the previous actions, LIVE discovers hidden features by applying its constructor functions to the existing features and testing whether the result discriminates the ambiguous states. For example, when predicting whether a balance-beam [Siegler, 1983] will tip or balance, LIVE discovers the invisible "torque" concept by multiplying distance and weight. (This kind of hidden features is normally categorized by the term *constructive induction.)* In time-dependent environments, where states do depend on the previous actions, LIVE discovers hidden features by searching back in its history to find differences in the states preceding the two indistinguishable states. We will give a detailed description of the discovery of genes in a later section.

Discovering hidden features is only the first half of the whole story; they have to be assimilated into the system to be useful in the future. In a time-independent environment, since hidden features are defined in terms of observables, the system can simply use the newly defined features as if they are visible because they are computable from the observables. For example, when the concept of torque is discovered by a system that perceives only objects' distances and weights, the concept will simply be added as another object feature. Any rule that needs torque to discriminate its condition can use it by computing the value *weight*distance.* In a time-dependent environment, since hidden features determine the observables, two additional things must be done before these features can be used: determine how the hidden features define the observables; and determine how the hidden features are inherited through actions. One strategy is used for both tasks: testing all the constructor functions to find one that is consistent with all the examples collected. We will have more to say about this later when LIVE attempts to discover genes.

Although LiVE's discovering method has been tested in both time-independent and time-dependent environments and the hidden features to be discovered can be quite complex in principle (previous discovered features can be used to discover new features), it depends heavily on the constructors that are given. For example, if * is not given, no features like torque can be discovered. In the future, we hope LIVE will start with a parsimonious, domain independent set of primitives from which necessary constructors like * can be constructed during the interaction with environment. Some early results along this direction have been reported in [Shen, 1989a].

7 Solving the Low-Level Tower of Hanoi

In this section, we complete the description of how LIVE learns a set of correct and useful rules in the Tower of Hanoi environment. We call it low-level, because LIVE's innate percepts and actions do not, include high-level concepts and actions that previous studies have used. In this environment, no matter what the initial state is, LIVE always creates a set of good rules (different rules may be learned in different runs) and reaches the given goal state. In the run we have been talking about so far, it takes LIVE 35 steps, including both actions and proposing subgoals, to solve the problem. We have explained how RuleO through Rule3 are created. In the later stage, LIVE meets three more surprises, the first surprise conies when it tries to pick up a disk when another disk is in the hand, which splits RuleO into RuleO and Rule4; the second surprise comes when it tries to put a bigger disk on a smaller one, which splits Rule3 into Rule3 and Rule5; the third surprise comes when it tries to pick up a bigger disk underneath a smaller disk, which spilts RuleO into RuleO and Rule6. The following is a list of all seven rules learned by LIVE:

Condition: (on $disk_x$ peg_x) $\wedge$ $\neg\exists$(in-hand $disk_y$) $\wedge$ $\neg\exists$((size> $disk_x$ $disk_z$) $\wedge$ (on $disk_z$ peg_x))
Action: (pick $disk_x$ peg_x) [Rule0]
Prediction: (in-hand $disk_x$) $\wedge$ $\neg$(on $disk_x$ peg_x)

Condition: (in-hand $disk_x$) $\wedge$ $\neg\exists$(on $disk_y$ peg_x)
Action: (put $disk_x$ peg_x) [Rule1]
Prediction: (on $disk_x$ peg_x) $\wedge$ $\neg$(in-hand $disk_x$)

Condition: (size> $disk_x$ $disk_z$) $\wedge$ (on $disk_z$ peg_x) $\wedge$ (in-hand $disk_x$) $\wedge$ (on $disk_y$ peg_x)
Action: (put $disk_x$ peg_x) [Rule2]
Prediction: (in-hand $disk_x$)

Condition: (in-hand $disk_x$) $\wedge$ (on $disk_y$ peg_x) $\wedge$ $\neg\exists$((size> $disk_x$ $disk_z$) $\wedge$ (on $disk_z$ peg_x)) $\wedge$ $\neg$(size> $disk_x$ $disk_y$)
Action: (put $disk_x$ peg_x) [Rule3]
Prediction: (on $disk_x$ peg_x) $\wedge$ $\neg$(in-hand $disk_x$)

Condition: (on $disk_x$ peg_x) $\wedge$ (in-hand $disk_y$)
Action: (pick $disk_x$ peg_x) [Rule4]
Prediction: (on $disk_x$ peg_x)

Condition: (in-hand $disk_x$) $\wedge$ (on $disk_y$ peg_x)
$\wedge \neg\exists$((size> $disk_x$ $disk_y$) $\wedge$ (on $disk_z$ peg_x))
$\wedge$ (size> $disk_x$ $disk_y$)
Action: (put $disk_x$ peg_x) [Rule5]
Prediction: (in-hand $disk_x$)

Condition: (on $disk_x$ peg_x) $\wedge \neg\exists$(in-hand $disk_y$)
$\wedge$ (size> $disk_x$ $disk_z$) $\wedge$ (on $disk_z$ peg_x)
Action: (pick $disk_x$ peg_x) [Rule6]
Prediction: (on $disk_x$ peg_x)

Note that because of the order of exploration, discussed in Section 5, some rules, such as Rule3, are not as perfect as others, such as Rule0. The conditions (on $disk_y$ peg_x) and $\neg$(size> $disk_x$ $disk_y$) in Rule3 are not necessary. Nevertheless, the rules taken together compose a good set for LIVE to solve any problems in the low-level Tower of Hanoi environment.

8 Discovering Genes

To demonstrate how LIVE discovers hidden features, we give it the task of discovering genes by hybridizing garden peas as Mendel [1865] did. For simplicity, a pea is initially denoted as $P^i_j(color)$, where i stands for its generation and j stands for its identification, and we assume that a pea's color, green as 1 and yellow as 0, is its only observable feature. The system is given one action: *artificial fertilization*, or AF(pea$_x$, pea$_y$), which we assume produces exactly four children peas and distributes the parent's genes evenly into the children. Two predicates: $=, >$, and three functions: $\sqcup$ (bit-or), $\sqcap$ (bit-and), and an even distributed pairing function *e-distr*((a b) (c d)) = ((a c) (a d) (b c) (b d)), are given as constructors. If there were a large set of constructors, LIVE might of course have to make a lengthier search, but the present set will illustrate the process. From the system's point of view, the task is no more than predicting the effect of its action.

The experiment starts with a set of purebred peas, $P^1_1(0)$, $P^1_2(0)$, $P^1_3(1)$, $P^1_4(1)$, $P^1_5(0)$, $P^1_6(1)$, $P^1_7(1)$, and $P^1_8(0)$, as the first generation. All their ancestors are known to have the same color. To make the second generation, the system fertilizes yellow with yellow, green with green, yellow with green, and green with yellow. When applying $AF(P^1_1(0), P^1_2(0))$ (with no prediction), the system observes that all the offspring, $P^2_1(0), P^2_2(0), P^2_3(0)$ and $P^2_4(0)$, are yellow, and constructs a new rule:

Condition: $P^a_i(c_i) \wedge P^a_j(c_j)$
Action: $AF(P_i(c_i),\ P_j(c_j))$ [Rule1]
Predict: $P^{a+1}_k(c_i) \wedge P^{a+1}_l(c_i) \wedge P^{a+1}_m(c_i) \wedge P^{a+1}_n(c_i)$

This rule predicts correctly that the hybrids of green $P^1_3(1)$ with green $P^1_4(1)$ will be all green: $P^2_5(1)$, $P^2_6(1)$, $P^2_7(1)$, $P^2_8(1)$, but fails to predict that the hybrids of yellow $P^1_5(0)$ with green $P^1_6(1)$ will be all green too: $P^2_9(1), P^2_{10}(1), P^2_{11}(1), P^2_{12}(1)$. The incorrect prediction is made because c_i is bound to yellow. This surprise causes the system to compare this application with the last one (green with green), finding a difference that $c_i = c_j$ was true previously but is not true now, and splits Rule1 in two:

Condition: $P^a_i(c_i) \wedge P^a_j(c_j) \wedge (c_i = c_j)$
Action: $AF(P_i(c_i),\ P_j(c_j))$ [Rule1]
Predict: $P^{a+1}_k(c_i) \wedge P^{a+1}_l(c_i) \wedge P^{a+1}_m(c_i) \wedge P^{a+1}_n(c_i)$

Condition: $P^a_i(c_i) \wedge P^a_j(c_j) \wedge \neg(c_i = c_j)$
Action: $AF(P_i(c_i),\ P_j(c_j))$ [Rule2]
Predict: $P^{a+1}_k(c_j) \wedge P^{a+1}_l(c_j) \wedge P^{a+1}_m(c_j) \wedge P^{a+1}_n(c_j)$

Rule2 is then applied to predict that all hybrids of green $P^1_7(1)$ with yellow $P^1_8(0)$ will be yellow (because c_j is bound to yellow), but the result is again a surprise: all $P^2_{13}(1)$, $P^2_{14}(1)$, $P^2_{15}(1)$, $P^2_{16}(1)$ are green. After comparing the two cases, the system concludes that the reason for the surprise is $c_i > c_j$ ($>$ is a given relational predicate), and splits Rule2 in two to reflect the fact that green dominates yellow, and that ends the second generation:

Condition: $P^a_i(c_i) \wedge P^a_j(c_j) \wedge \neg(c_i = c_j) \wedge \neg(c_i > c_j)$
Action: $AF(P_i(c_i),\ P_j(c_j))$ [Rule2]
Predict: $P^{a+1}_k(c_j) \wedge P^{a+1}_l(c_j) \wedge P^{a+1}_m(c_j) \wedge P^{a+1}_n(c_j)$

Condition: $P^a_i(c_i) \wedge P^a_j(c_j) \wedge \neg(c_i = c_j) \wedge (c_i > c_j)$
Action: $AF(P_i(c_i),\ P_j(c_j))$ [Rule3]
Predict: $P^{a+1}_k(c_i) \wedge P^{a+1}_l(c_i) \wedge P^{a+1}_m(c_i) \wedge P^{a+1}_n(c_i)$

The third generation of the experiment is made by self-fertilizing the second generation. We assume the pairs to be hybridized are: $(P^2_1(0)P^2_2(0))$, $(P^2_5(1)P^2_6(1))$, $(P^2_9(1)P^2_{10}(1))$, and $(P^2_{13}(1)P^2_{14}(1))$. Rule1 successfully predicts the results of the first two pairs, which produce $(P^3_1(0)P^3_2(0)P^3_3(0)P^3_4(0))$ and $(P^3_5(1)P^3_6(1)P^3_7(1)P^3_8(1))$ respectively. However, the hybridization of the third pair surprises the system because the children have different colors: $(P^3_9(1)P^3_{10}(1)P^3_{11}(1)P^3_{12}(0))$. In explaining the surprise, the last application (the second pair) is brought in to compare with the current application, but the system fails to find any relational difference because both pairs are green. This is a point where hidden features must be discovered. Since the experiment is time-dependent, LIVE traces back in history for previous difference states. Fortunately, a difference is found when comparing the parents of the second and the third pair: $c_i = c_j$ was true for $(P^2_5(1)P^2_6(1))$'s parents $[P^1_3(1)P^1_4(1)]$ but not for $(P^2_9(1)P^2_{10}(1))$'s parents $[P^1_5(0)P^1_6(1)]$. This difference indicates the existence of two hidden features from the grandparents, denoted as m and f in the following, that are invisible in the parental generation but are necessary to determine the grandchildren's color. The representation of the pea is then extended to include three features: (color, m, f).

As we noted before, two things must be done at this point in order to use the hidden features. One is to figure out how m and f are related to the color feature, and the other is to figure out how m and f are inherited through the action. For the first task, since we know color is determined by m and f, and we know many examples (because the first generation peas are purebred): $P^1_1(0\ 0\ 0)$, $P^1_3(1\ 1\ 1)$, $P^2_9(1\ 1\ 0)$, $P^2_{13}(1\ 0\ 1)$, etc., it is straightforward to search through the constructor functions and see that color $= m \sqcup f$, i.e. $0=0\sqcup 0$, $1=1\sqcup 1$, $1=1\sqcup 0$, and $1=0\sqcup 1$. In Mendel's words, the dominant determines the color. For the second task, since we know that parents'

m and f are inherited into all four children's m and f, and we have the examples of how the third generation is produced from the second generation (where surprises arise):

$$P_1^2(000)P_2^2(000) \rightarrow P_1^3(0??)P_2^3(0??)P_3^3(0??)P_4^3(0??)$$
$$P_5^2(111)P_6^2(111) \rightarrow P_5^3(1??)P_6^3(1??)P_7^3(1??)P_8^3(1??)$$
$$P_9^2(101)P_{10}^2(101) \rightarrow P_9^3(1??)P_{10}^3(1??)P_{11}^3(1??)P_{12}^3(0??).$$

Searching through the constructor functions, we find that *e-distr* fits the data. Based on the answers to these two questions, Rule1 is now split into the following two new rules:

$$\begin{aligned}
&\text{Condition: } P_i^a(c_i\ m_i\ f_i) \wedge P_j^a(c_j\ m_j\ f_j)\\
&\qquad \wedge (c_i = c_j) \wedge (m_i = f_i) \wedge (m_j = f_j)\\
&\text{Action: } AF(P_i^a(c_i\ m_i\ f_i),\ P_j^a(c_j\ m_j\ f_j)) \quad \text{[Rule1]}\\
&\text{Predict: } P_k^{a+1}(m_i \sqcup m_j\ m_i\ m_j)\\
&\qquad \wedge P_l^{a+1}(m_i \sqcup f_j\ m_i\ f_j)\\
&\qquad \wedge P_m^{a+1}(f_i \sqcup m_j\ f_i\ m_j)\\
&\qquad \wedge P_m^{a+1}(f_i \sqcup f_j\ f_i\ f_j)
\end{aligned}$$

$$\begin{aligned}
&\text{Condition: } P_i^a(c_i\ m_i\ f_i) \wedge P_j^a(c_j\ m_j\ f_j)\\
&\qquad \wedge (c_i = c_j) \wedge \neg(m_i = f_i) \wedge \neg(m_j = f_j)\\
&\text{Action: } AF(P_i^a(c_i\ m_i\ f_i),\ P_j^a(c_j\ m_j\ f_j)) \quad \text{[Rule4]}\\
&\text{Predict: } P_k^{a+1}(m_i \sqcup m_j\ m_i\ m_j)\\
&\qquad \wedge P_l^{a+1}(m_i \sqcup f_j\ m_i\ f_j)\\
&\qquad \wedge P_m^{a+1}(f_i \sqcup m_j\ f_i\ m_j)\\
&\qquad \wedge P_m^{a+1}(f_i \sqcup f_j\ f_i\ f_j)
\end{aligned}$$

When $P_{13}^2(1\ 1\ 0)$ is hybridized with $P_{14}^2(1\ 1\ 0)$, Rule4 predicts correctly that among four children, three will be green and one will be yellow. After three generations of hybridization, as in the experiments Mendel reported in his paper, LIVE has discovered the hidden features, genes.

9 Conclusion

In this paper, the problem of learning and problem solving in new environments is specified as a problem of correlating a learner's percepts and actions and inferring the rules from the given environment. A discrimination-based learning method is investigated that creates general rules by noticing the changes in the environment, and then specifies the rules by explaining their failures in prediction. The method has two distinct characteristics. First, when an over-general rule is split, the method will keep both new rules for further development. Second, when the discrimination process is unable to find any difference between a success and a failure, the method will define hidden features in terms of existing or historical features. We have been investigating the method in several very different domains and further studies are focusing on applying it to more complex exploration tasks and learning problem solving strategies.

Acknowledgement

We thank Jaime Carbonell, Tom Mitchell, Chuck Thorp and Jeff Schlimmer for their valuable comments, and Klaus Gross for the suggestion of using Mendel's discovery as a testing domain. We also appreciate the support from the World Modelers project upon which LIVE is originally developed.

References

[Carbonell and Gil, 1987] J.G. Carbonell and Y. Gil. Learning by experimentation. In Proceedings of 4th International Workshop on Machine Learning, 1987.

[Dietterich and Michalski, 1983] T.G. Dietterich and R.S. Michalski. A comparative review of selected methods for learning from examples. In Machine Learning. Morgan Kaufmann, 1983.

[Drescher, 1987] G.L. Dresc her. A mechanism for early Piagetian learning. In Proceedings of AAAI-87, 1987.

[Falkenhainer, 1988] B. Falkenhainer. The utility of difference-based reasoning. In Proceedings of AAA1-88, 1988.

[Hayes and Simon, 1974] JR. Hayes and H.A. Simon. Understanding written problem instructions. In Knowledge and Cognition. Lawrence Erlbaum, 1974.

[Langley et al., 1983] P. Langley, H.A. Simon, and G.L. Bradshaw. Rediscovering chemistry with the BACON system. In Machine Learning. Morgan Kaufmann, 1983.

[Langley, 1987] P. Langley. A general theory of discrimination learning. In Production System Models of Learning and Development. MIT Press, 1987.

[Mendel, 1865] G. Mendel. Experiments in plant-hybridization (translation). In J.A. Peters, editor, Classic Papers in Genetics. Prentice-Hall, 1967.

[Mitchell et al., 1983] T.M. Mitchell, P.E. UtgofT and R..B. Banerji. Learning by experimentation: Acquiring and refining problem-solving heuristics. In Machine Learning. Morgan Kaufmann, 1983.

[Newell, 1987] A. Newell. Unified theories of cognition, William James Lecture. Harvard University, (available on video cassette from Harvard), 1987.

[Quinlan, 1987] J.R. Quinlan. Generating production rules from decision trees. In Proceedings of 10th IJCAI, 1987.

[Shen, 1989a] W.M. Shen. Functional transformation in AI discovery systems. Artificial Intelligence, (to appear) 1989.

[Shen, 1989b] W.M. Shen. Learning from Environment Based on Actions and Percepts. PhD thesis, Carnegie Mellon University, 1989.

[Siegler, 1983] R.S. Siegler. How knowledge influence learning. American Scientist, 71, 1983.

[Utgoff, 1986] P.E. UtgofL Machine Learning of Inductive Bias. Kluwer Academic, 1986.

[Vere, 1980] S.A. Vere. Multilevel counterfactuals for generalizations of relational concepts and productions. Artificial Intelligence, 14, 1980.