Credit Assignment in Genetic Learning Systems

John J. Grefenstette
Navy Center for Applied Research in Artificial Intelligence
Naval Research Laboratory
Washington, DC 20375-5000, U.S.A.

Abstract

Credit assignment problems arise when long sequences of rules fire between successive external rewards. Two distinct approaches to rule learning with genetic algorithms have been developed, each approach offering a useful solution to a different level of the credit assignment problem. We present a system, called RUDI, that combines features from both approaches. Experimental results are presented that support the hypothesis that multiple levels of credit assignment can improve the performance of rule learning systems based on genetic algorithms.

1. Introduction

Systems that learn heuristic rules often proceed in two phases: first, existing rules are assessed in a problem solving context and, second, rules are modified in the hope of improving the performance of the system on similar tasks. The rule assessment phase usually gives rise to a *credit assignment problem:* If a sequence of rules fires before the system solves a particular problem, how can credit or blame be accurately assigned to early rules that set the stage for the final result? For example, in a chess playing program the decision that immediately precedes checkmate is not usually to blame for the final outcome. Rather, some rule that fired earlier in the game may be responsible for the fatal sequence of moves. It is often difficult to identify the responsible rule. If a complete, tractable domain theory is available then analytical learning techniques (Mitchell, Keller & Kedar-Cabelli, 1986) might be applied. If one can assume that an optimal solution path is known (or can be produced by the problem solving module), then an analysis of solution traces can provide positive and negative instance of rule applications (Sleeman, Langley & Mitchell, 1982; Langley, 1983; Mitchell, Utgoff & Banerji, 1983). If very little background knowledge is available and the problem environment provides a natural measure for the quality of outcomes, it is appropriate to view the problem of learning as a search for high performance knowledge structures, and to explore the capabilities of genetic algorithms (Holland, 1975) to perform the search. This work explores the latter approach.

Two rather distinct approaches to rule learning using genetic algorithms have been developed. In one approach, illustrated by the system called called LS-1 (Smith, 1983), each knowledge structure in the population represents a production system program represented as a list of rules. As a result of applying the knowledge structure to the problem solving task, a fitness measure is assigned to the entire program and is used to control the selection of structures for reproduction. Genetic search operators are applied to the selected structures to produce a new population of production system programs. In practice, LS-1 successfully learned to solve maze tasks and to play draw poker (Smith, 1983). Another approach is taken in *classifier systems* (Holland & Reitman, 1978; Holland, 1986; Riolo, 1987), in which the genetic operators are applied to single production rules, or *classifiers.* Each rule is assigned a measure, called its *strength,* that indicates the utility of the rule to the system's goal of obtaining external reward. New rules are discovered by genetic operators applied to existing rules selected on the basis of strength. The newly created rules must successfully compete with established rules in order to survive. Given an appropriate representation for the rules (Holland, 1986), the theory of genetic search predicts that the successful new rules will be plausible variants of their precursers, and that classifier systems can discover significant sets of cooperative rules. The classifier system approach has been implemented in several successful learning systems (Booker, 1982; Goldberg, 1985; Wilson, 1987a; Zhou, 1987).

The relative merits of these two approaches is a topic of active debate in the genetic algorithm research community (De Jong, 1987). This paper presents the view that each approach offers an interesting solution to a distinct aspect of the credit assignment problem. Section 2 describes the credit assignment performed by LS-1, as well as its shortcomings. Section 3 presents a comparison of various credit assignment techniques used in classifier systems. Section 4 outlines a new rule learning system, RUDI, that tries to exploit each of these techniques to its best advantage. An experimental comparison of the various techniques appears in Section 5.

2. Implicit Credit Assignment in LS-1

Smith (1983) has described a system called LS-1 that employs a genetic algorithm to search for high performance knowledge structures, each structure consisting of a list of rules. LS-1 maintains a population of knowledge structures that evolves over time as a result of the system's experiences. A fitness measure for each knowledge structure is obtained by observing the performance of a problem solving module that uses the given rules to solve a number of tasks from the problem domain. Once each knowledge structure in the population has been evaluated, a new population of structures is formed in two steps. First, a *selection* procedure chooses structures for reproduction by a stochastic process that ensures that the expected number of offspring associated with a given structure is proportional to the structure's observed performance relative to the other structures in the current population. As a result, high performance structures may be chosen several times for replication and low performance structures may not be chosen at all. The second step produces new plausible knowledge structures by *recombining* pairs of selected structures using idealized genetic operators. The primary genetic operator is *crossover,* which combines two parent structures to form two similar offspring.[1] Crossover operates in LS-1 by exchanging

[1] Random mutation plays a minor role in genetic algorithms as a background operator that maintains the reachability of all points in the search space.

segments of the string or list representations of the parents. For example, if the parents are represented by the lists:

(rule A, rule B, rule C, rule D, rule E) and
(rule a, rule b, rule c, rule d, rule e)

then crossover at the rule level[2] might produce the offspring

(rule A, rule B, rule c, rule d, rule e) and
(rule a, rule b, rule C, rule D, rule E) .

The combined effect of performance-biased reproduction and crossover is a sophisticated form of adaptive search through the space of knowledge structures described by the *Schema Theorem* for genetic algorithms, established by Holland (1975) and extended to encompass LS-1's operator set by Smith (1983). This theorem states that the number of structures in the knowledge base that share a given subset of components (e.g., a group of rules in LS-1) can be expected to increase or decrease over time at a rate proportional to the observed performance of the subset. This property is known as the *implicit parallelism* of genetic algorithms (Holland, 1975). Thus, even though LS-1 explicitly computes utility only for entire sets of rules, credit assignment operates implicitly at the level of much smaller groups of rules.

The implicit credit assignment in LS-1 is especially effective if related rules are *clustered,* since rules that are physically close together on the list representing the knowledge structure stand a good chance of being inherited as a group. For this reason, LS-1 includes an *inversion* operator (Holland, 1975) that reverses a randomly chosen contiguous set of rules on a given knowledge structure, thereby altering the rule combinations disrupted by crossover. A moderate rate of inversion produced slight performance improvements in LS-1 (Smith, 1983). However, it was found that random inversion is too weak a method to search the space of rule permutations and identify related subsets of rules. The clustering problem was partially addressed in LS-2, a system that performs classification of human gait data (Schaffer & Grefenstette, 1985), by assigning multiple performance measures for each rule set. Section 4 presents a new way of using individual rule utilities to cluster related sets of rules within the LS-1 framework. First, we describe how such rule utilities can be computed using techniques developed for classifier systems.

3. Explicit Credit Assignment in Classifier Systems

A primary difference between LS-1 and classifier systems is that classifier systems assign a utility measure called *strength* to individual rules rather than to entire production system programs. Just as the genetic algorithm in LS-1 implicitly operates on small groups of rules based on explicit fitness measures assigned to entire rule sets, the genetic algorithm in classifier systems implicitly exploits information about components of rules based on the strength assigned to individual rules (Holland, 1986). Here we focus on the explicit credit assignment mechanisms that operate on the rule level. We further limit the discussion to classifier systems that learn heuristic control rules for applying a set of known operators to a set of states. We assume that a single rules fires at each step, and that the probability of a matched rule being selected to fire is proportional to its strength.[3] When a rule fires, the operator specified by the right-hand-side is applied to the current state to produce the new state. If no rule matches the current state, a randomly chosen operator is used to produce the next state.

We first consider a strength updating scheme we call the *Profit Sharing Plan (PSP),* a simplified version of the strength updating scheme described by Holland and Reitman (1978). In this scheme, problem solving is divided into *episodes* delimited by the receipt of external reward. At the end of each episode, the strength of each active rule loses a fixed faction of its value and gains an amount equal to the reward obtained.[4] Under relatively consistent external rewards, the strength of each rule under the PSP rapidly converges to an equilibrium strength that predicts the level of reward that will be received at the end of the episode.

Most recent classifier systems (Booker, 1982; Goldberg, 1983; Riolo, 1987; Wilson, 1987a) have adopted a distributed, incremental credit assignment scheme, called the *Bucket Brigade Algorithm (BBA),* that is closely related to the *Temporal Difference Methods* analyzed by Sutton (1988). In its simplest form, the BBA requires that each time a rule fires, the rule pays a fixed fraction, called the *bid-ratio,* of its strength to the rule that fired on the previous time step. When external reward is obtained, it is paid to the final rule in the chain.

In many cases, the PSP and the BBA lead to identical results, but differences can arise when rules match more than one state. Consider the example shown in Figure 1. Assuming that state *A* arises equally as often as state *E*, Table 1 shows the equilibrium strength for the rules under each credit assignment scheme. This example shows that rules that frequently fire in sequence are assigned similar levels of strength by the BBA, whereas the PSP generally gives a better estimate of the expected external reward.

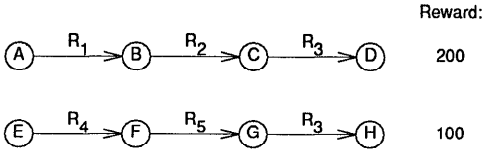

Fig. 1. State Space Fragment

Rule	PSP-Strength	BBA-Strength
R_1	200	150
R_2	200	150
R_3	150	150
R_4	100	150
R_5	100	150

Table 1. Effects of Different Strength Updating Schemes

[2] Crossover in LS-1 operates at multiple levels. Crossover might occur between individual symbols of two rules, producing new rules that inherit some conditions from each parent (Smith, 1983).

[3] In the general classifier system model (Holland, 1986; Riolo, 1987), conflict resolution may depend on the generality of the rules, strengths may be reduced by a variety of taxes, and more than one rule may fire on a single step. The effects of these factors on credit assignment is a topic for further research. The conflict resolution methods we discuss bear some similarity to those in ACT (Anderson & Kline, 1979).

[4] Holland and Reitman (1978) attenuate rewards so that rules firing earlier are usually rewarded less than those firing closer to the end of an episode.

4. RUDI: A Multilevel Credit Assignment System

We have seen that credit assignment in LS-1 operates on the level of groups of rules, but is hampered by its lack of knowledge about the performance of individual rules and by the inability of random inversion to create meaningful clusters of rules. The PSP and the BBA provides complementary utility information about individual rules. PSP-Strength provides a more accurate estimate of the utility of a rule in terms of its expected external reward. BBA-Strength indicates the dynamic associations among rules, with rules that fire in sequence converging to similar levels of BBA-Strength. This section describes one way to exploit all of these credit assignment techniques in a single system.

4.1. The Problem Solving Level

RUDI (for RUle DIscovery) is a system that combines features of LS-1 and classifier systems, as shown in Figure 2.

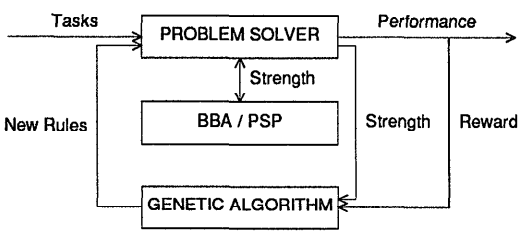

Fig. 2. RUDI: A Multilevel Genetic Learning System.

The problem solving level of RUDI consists of a simplified classifier system, as described in Section 3, that maintains both PSP-Strength and BBA-Strength for each rule. Since PSP-Strength provides an estimate of expected external reward, it is used for conflict resolution during problem solving. BBA-Strength is used by the learning level to cluster co-operative sets of rules, as described below. The problem solver reports to the learning level the average external reward received by each rule set during the solution of a sets of tasks from the problem domain, as well as the updated rule strengths.

4.2. The Learning Level

Like LS-1, the genetic learning algorithm in RUDI operates on a population of knowledge structures, each represented by a list of rules. The overall performance of the knowledge structures controls the selection of knowledge structures for reproduction. Modified structures are formed by applying crossover to the selected structures, as in LS-1, except that each rule in the offspring inherits the strengths associated with the corresponding rule in the parent structure.[5]

Unlike LS-1, the strengths of the individual rules influence the physical representation of the knowledge structures in RUDI, making it more likely that useful combinations of rules survive and propagate throughout the knowledge base. This is accomplished by a heuristic form of inversion called *clustering.* Clustering is performed just prior to crossover and involves two steps:

1) Sort the rules within the knowledge structure on the basis of their BBA-Strength.
2) Treating the knowledge structure as a ring, shift the sorted rules a random number of rule positions along the structure.

The shift phase is necessary in order to avoid a bias against certain distributions of building blocks. For example, the shift phase allows (but does not guarantee) an offspring to inherit all of the high BBA-Strength rules from each parent. Figure 3 shows an example of clustering a rule set.

BEFORE CLUSTERING:

Rule:	R_1	R_2	R_3	R_4	R_5
Strength:	225	50	250	30	300

1) SORT BY BBA-STRENGTH:

Rule:	R_5	R_3	R_1	R_2	R_4
Strength:	300	250	225	50	30

2) CIRCULAR SHIFT:

Rule:	R_4	R_5	R_3	R_1	R_2
Strength:	30	300	250	225	50

Fig. 3. Cluster Operator.

As shown in Section 3, rules that frequently occur in the same problem solving chain will tend to have similar levels of BBA-Strength. Clustering moves such rules closer together on the knowledge structure, and since crossover takes contiguous groups of rules from each parent, rules occurring frequently in the same chain will tend to be inherited as a group. This heuristic provides a way to form meaningful clusters of co-operative rules that was missing from LS-1.

5. An Experimental Study

This section describes experiments that compare RUDI's combination of credit assignment mechanisms with those mechanisms in isolation. Space permits the detailed discussion of only one set of experiments, but similar results have been achieved on other problems with various state-space topologies and distributions of rewards to final states. The test problem consisted of a state space containing 288 states as shown in Figure 4. There were 32 initial states and each traversal of the space required eight decision steps. Each non-final state was identified by an 8-bit feature vector indicating its position in the space. Three operators -- `left`, `straight`, and `right` -- mapped each state to its successors. The state space was cylindrical, so that `left`(0) = 63. The rewards associated with the final states ranged from 0 to 1000, with an average of 250. As shown in Table 2, the rewards were distributed around two hills of high reward, with payoff valleys between. The distribution of external rewards to the final states was chosen in order to require the discovery of correct heuristic rules for the early stages of each task in order to gain maximum reward. For example, to obtain maximum reward from starting state 0 (at the upper left corner of the state space), it was necessary to apply the operator `right` for at least seven of the eight steps in the task (in order to reach maximum reward at state 263 or 264).

[5] In the case of a new rule created by crossing in the middle of two rules, the new rule is assigned the average strength of the two parental rules.

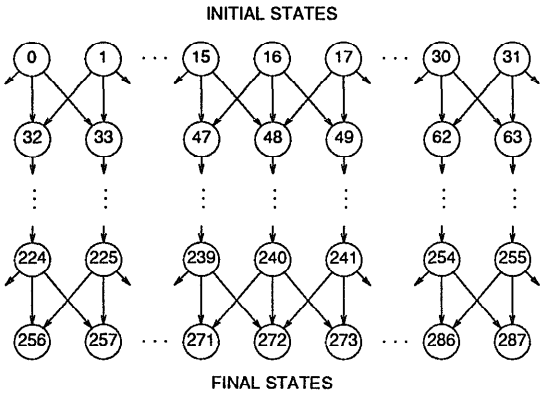

Fig. 4. Experimental State Space

State:	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271
Reward:	0	0	50	75	125	250	500	1000	1000	500	250	125	75	50	0	0
State:	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287
Reward:	0	0	50	75	125	250	500	1000	1000	500	250	125	75	50	0	0

Table 2. Distribution of External Rewards.

5.1. Rule Representation

The left hand side of each rule contains a pattern that matches a set of states, using the symbol # to indicate that the value of the corresponding feature is irrelevant.[6] The right hand side of each rule specifies a single operator, using a fixed mapping from integers to operators. For example, the rule

00000##1 → 0010

represents the heuristic

IF the current state is in the set {1, 3, 5, 7}
THEN apply operator `go-straight`.

Given this representation, there are a total of 19,683 distinct rules in the rule space. Recognizing that it is generally infeasible to consider all possible rules (or even all maximally specific rules), each system was required to learn rule sets that contained a maximum of 128 rules.

5.2. Experimental Results

A series of experiments was performed to test the effects of various credit assignment methods on the test problem. Although it is possible to inject available knowledge into genetic learning systems (Grefenstette, 1987), for the purposes of these experiments, all the learning systems began with randomly generated knowledge structures. Since genetic learning systems are stochastic processes, all plots show the average of five independent runs.

Figure 5 shows performance profiles for a random walk algorithm and for two simple classifier systems that used either the PSP or the BBA for conflict resolution and for rule reproduction. The system using PSP clearly dominated the system using BBA, but both left much room for improvement, since the maximum reward per episode was 1000. These results, while not conclusive, are consistent with previous studies in which classifier systems have had difficulty in performing successful credit assignment over chains of similar length, unless the BBA is augmented by specially designed *bridge classifiers* (Riolo, 1987).

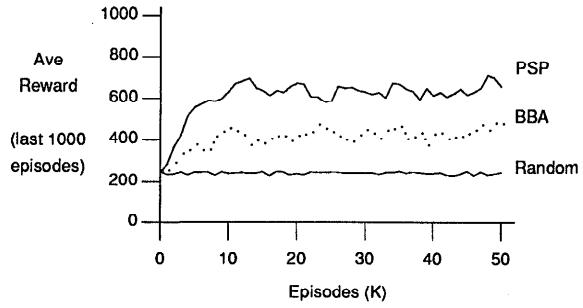

Fig. 5. Performance Profiles of Classifier Systems.

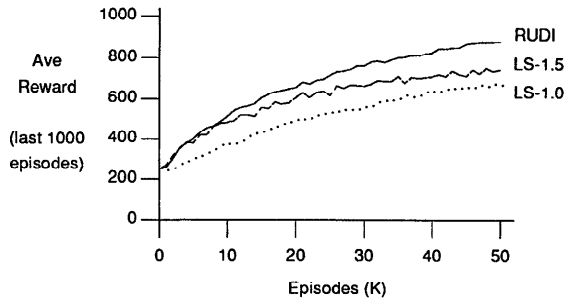

Fig. 6. Performance Profiles of LS-1 Style Systems.

Figure 6 shows the performance of three LS-1 style systems that used different forms of credit assignment. The system denoted LS-1.0 used random conflict resolution and no clustering at the learning level. The system denoted LS-1.5 used PSP for conflict resolution, but performed no clustering at the learning level. And, as described above, RUDI used PSP-Strength for conflict resolution and BBA-Strength for clustering at the learning level. Each system maintained a knowledge base of 50 knowledge structures, each consisting of 64 rules along with their associated strengths. Each structure was evaluated in 20 reward episodes, each starting at a randomly chosen initial state. Each run consisted of 2500 rule set evaluations (50 generations).

After each run, the rule set with the highest evaluation in the final population was subjected to a final evaluation consisting of 1000 reward episodes. The average results for all runs are shown in Table 3. The clear performance advantage of RUDI over the other systems supports the hypothesis that multiple levels of credit assignment can improve the performance of rule discovery systems based on genetic algorithms.

[6] Holland (1975) discusses the possibility of learning new features, but we do not pursue that approach here. Holland (1986) discusses appropriate pattern languages for classifier systems, and Booker (1982) discusses issues in matching.

System	Ave. External Reward
Optimal	1000
RUDI	943
LS-1.5	791
LS-1.0	724
PSP	632
BBA	468
Random	250

Table 3. Performance of Final Rule Sets.

6. Conclusions

This paper has examined issues of credit assignment in rule learning systems based on genetic algorithms. The classifier systems approach and the LS-1 approach each provides useful mechanisms for assigning credit. RUDI represents a new method of reconciling these two approaches, using the BBA designed for classifier systems to solve the clustering problem in LS-1. RUDI demonstrate the benefits of exploiting multiple levels of credit assignment. Further testing is needed to delimit the class of problems for which this approach is most valuable.

An important topics for further research involves the development of local strength updating schemes like the BBA that predict levels of external reward to the degree achieved by the PSP scheme. Such schemes would be especially useful in systems that allow parallel rule firing (Holland, 1986).

Wilson (1987b) has described a hierarchical form of classifier system in which strength is passed only among modules that operate at the same level of abstraction, thus keeping the chains at each level relatively short. This seems to be a promising approach that deserves further attention.

Acknowledgments

I want to thank Lashon Booker and Ken De Jong for many stimulating discussions on this topic.

References

Anderson, J. R., & Kline, P. J. (1979). A learning system and its psychological implications. *Proceedings of the Sixth International Joint Conference on Artificial Intelligence* (pp. 16-21). Tokyo: Morgan Kaufmann.

Booker, L. B. (1982). *Intelligent behavior as as adaptation to the task environment.* Doctoral dissertation, Department of Computer and Communications Sciences, University of Michigan, Ann Arbor.

De Jong, K. A. (1987). On using genetic algorithms to search program spaces. *Proceedings of the Second International Conference Genetic Algorithms and Their Applications* (pp. 210-216). Cambridge, MA: Lawrence Erlbaum.

Goldberg, D. E. (1985). Dynamic system control using rule learning and genetic algorithms. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 588-5925). Los Angeles, CA: Morgan Kaufmann.

Grefenstette, J. J. (1987). Incorporating problem specific knowledge into genetic algorithms. In *Genetic algorithms and simulated annealing.* D. Davis (ed.), London: Pitman Press.

Holland, J. H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor, MI: University of Michigan Press.

Holland J. H. (1986). Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R.S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos, CA: Morgan Kaufmann.

Holland, J. H., & Reitman, J. S. (1978). Cognitive systems based on adaptive algorithms. In D. A. Waterman, & F. Hayes-Roth (Eds.), *Pattern-directed inference systems.* New York: Academic Press.

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

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

Mitchell, T. M., Utgoff, P. E., & Banerji, R. (1983). Learning by experimentation: Acquiring and refining problem-solving heuristics. In R.S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 1). Palo Alto, CA: Tioga.

Riolo, R. L. (1987). Bucket brigade performance I: Long sequences of classifiers. *Proceedings of the Second International Conference on Genetic Algorithms and Their Applications* (pp. 184-195). Cambridge, MA: Lawrence Erlbaum.

Schaffer, J. D., & Grefenstette, J. J. (1985). Multi-objective learning via genetic algorithms. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 593-595). Los Angeles, CA: Morgan Kaufmann.

Sleeman, D., Langley, P., & Mitchell, T. M. (1982). Learning from solution paths: An approach to the credit assignment problem. *AI Magazine,* Spring 1982, 48-52.

Smith, S. F. (1983). Flexible learning of problem solving heuristics through adaptive search. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence* (pp. 422-425). Karlsruhe, Germany: Morgan Kaufmann.

Sutton, R. S. (1988). *Learning to predict by the methods of temporal differences.* (Technical Report TR87-509.1). Waltham, MA: GTE Laboratories Inc.

Wilson, S. W. (1987a). Classifier systems and the animat problem. *Machine Learning, 3(2),* 199-228.

Wilson, S. W. (1987b). Hierarchical credit allocation in a classifier system. In L. Davis (Ed.), *Genetic algorithms and simulated annealing.* London, UK: Pitman.

Zhou, H. H. (1987). *CSM: A genetic classifier system with memory for learning by analogy.* Doctoral dissertation, Department of Computer Science, Vanderbilt University, Nashville.