Dynamic Classifiers: Genetic Programming and Classifier Systems

Patrick Tufts
Department of Computer Science
Volen Center for Complex Systems
Brandeis University
Waltham, Mass.
email: zippy@cs.brandeis.edu

Abstract

The Dynamic Classifier System extends the traditional classifier system by replacing its fixed-width ternary representation with Lisp expressions. Genetic programming applied to the classifiers allows the system to discover building blocks in a flexible, fitness directed manner.

In this paper, I describe the prior art of problem decomposition using genetic programming and classifier systems. I then show how the proposed system builds on work in these two areas, extending them in a way that provides for flexible representation and fitness directed discovery of useful building blocks.

Introduction

Learning systems that engage in sequential activity face the problem of crediting those steps that lead to a reward. When the reward is the result of many steps in a sequence, the problem becomes: what is the most efficient and reliable way to credit those steps?

In order to form solutions to problems where the payoff comes after a long sequence of actions, it is necessary to identify the steps that lead to reward. Classifier systems (Holland et al., 1987; Wilson, 1994b) do this using the bucket brigade mechanism (Holland et al., 1987). The bucket brigade works as follows. Each time step, the currently active rules (the action set) pass a fraction of their rewards to the rules that were active in the preceding time step. If a reward is received in a given time step, those rules that matched the input and advocated the action that the system took share the reward. Given a chain of rules that lead to a reward, each iteration through the chain passes some reward one step back. If the chain is n steps long, then it will take n iterations through the chain before the first classifier receives any reward.

The bucket brigade is closely related to temporal-difference (TD) methods (Sutton, 1988), and shares advantages with them over conventional prediction-learning schemes. Like TD learning methods, the bucket brigade learns incrementally. This means that it can perform prediction-learning tasks using less memory and less peak computation than systems that store all cases and then learn over the entire set at once (Sutton, 1988).

The bucket brigade has its shortcomings. For example, it favors short sequences of rules over long ones. A classifier system that uses the bucket brigade is thus unlikely to discover solutions that require more than four or five steps (Wilson & Goldberg, 1989).

I address this problem by developing a scheme for rule clustering (Wilson & Goldberg, 1989; Shu & Schaeffer, 1991; Wilson, 1987b; Wilson, 1994b) based on genetic programming (Koza, 1992). I call this hybrid, first proposed by Wilson (Wilson, 1994a), the Dynamic Classifier System (DCS), and show how clustering, as it emerges in DCS, may allow for longer chains and thus the solution of more complex tasks (Wilson & Goldberg, 1989; Wilson, 1994b).

Classifier Systems

A classifier system is a learning mechanism in which a collection of initial rules (possibly random) are updated by a genetic algorithm according to a fitness scheme. Figure 1 lists the basic elements as proposed by Holland (Holland et al., 1987).

A classifier is a condition/action pair with an associated fitness. All classifiers that match the current inputs form a match set. The system selects its action from one or more of these classifiers. Learning in a classifier system occurs through reinforcement, recombination, and reproduction of individual classifiers.

Problem

This section addresses the central problem for both genetic programming and classifier systems—the recognition of steps that solve a task. After showing how this problem affects learning systems from these two fields, I describe how the Dynamic Classifier System, which uses genetic programming within the framework

- finite population of classifiers
- set of inputs
- set of actions
- fitness/reward function
- discovery functions: crossover and mutation
- classifier reinforcement via the bucket brigade
- message list

Figure 1: Elements of a Basic Classifier System.

of classifier systems, can be powerful enough to efficiently recognize those steps that lead to reward.

Building blocks and Genetic Programming

In the genetic programming paradigm, programs are evolved in a manner analogous to the genetic algorithm. Where the genetic algorithm uses a fixed-length linear representation, genetic programming uses one that is hierarchical and extensible (Koza, 1992).

One problem with genetic programming is: how can useful subtrees in a population be identified and protected from crossover? Crossover acts to swap two randomly selected subtrees from two members of the population. If the swap point occurs within a good subsolution, the resulting program is likely to be less fit than its parents (Tufts & Juillé, 1994).

Several methods have been proposed to identify useful subtrees and protect them from crossover.

Automatically Defined Functions Automatically Defined Functions (ADFs), proposed by Koza (Koza, 1994), introduce a constrained syntax into genetic programming. The ADF template constrains members of the population to have a result-producing branch and a function definition branch. The difference between genetic programming without and with ADFs is as follows:

- Without ADFs, each member of the population produces a result that is then evaluated.
- With ADFs, each member produces a result that is evaluated, and also defines one or more functions.

The problem is that the experimenter must decide how much regularity to impose on the system in two ways:

1. the number of ADFs a program can create
2. the set of terminals and functions associated with each ADF

These two settings determine how tightly the learning system is constrained to a particular form of solution. As an example, in an experiment where the system was to learn a four-input multiplexor, Kinnear constrained each solution to have three ADFs, each with a *different* function and terminal set (Kinnear, 1994).

The approach proposed here allows the system to create modules as necessary to form reliable chains. ADF-style grouping of terminals and functions is expected to occur as part of the chaining process.

Module Acquisition Module Acquisition (MA) is a system for genetic programming in which random subtrees in a population of programs are selected, encapsulated, and placed in a global module library (Angeline & Pollack, 1993; Angeline, 1994; Kinnear, 1994). The encapsulation process replaces a subtree in a program with a function name, and adds the named function and definition to the library.

During encapsulation, the tree may be trimmed to an arbitrary depth, creating a parameterized module. The nodes below the trim point become the arguments to the newly defined function.

Modules in the library do not evolve. They are kept in the library as long as they are used by some member of the population. The initial encapsulation (Koza, 1992, pp. 110–112) results in a single reference to the encapsulated module: the reference in the function from which the subtree was taken. If the module provides a benefit to new programs when brought in through the crossover operator, then the number of references to it in the population will increase.

Module acquisition may be inferior to ADFs for one learning problem—even four-parity (Kinnear, 1994). Because module acquisition encapsulates random subtrees in the population (that is, there is no fitness-based selection), it may be less likely to discover useful subcomponents.

The Dynamic Classifier System is potentially more efficient at discovering modules because it can identify the building blocks of those modules through chaining. Measuring the utility of pieces and creating larger ones from them may be a better approach than forming entire solutions and then randomly decomposing them into potential building blocks.

Chain formation in Classifier Systems

The bucket brigade is closely related to Q-learning (Watkins, 1989) and is in the family of temporal-difference methods (Sutton, 1984; Sutton, 1988). One of the important aspects that the bucket brigade shares with these methods is that it is memory efficient. The bucket brigade operates locally, reinforcing connections between current rules and their temporally immediate predecessors. It does this by keeping track of all classifiers active during the current and the previous time steps. Classifiers that are in the current action set pass a fraction of their strengths back to the classifiers that were in the action set of the previous time step. When an action results in a reward, the current action set is reinforced. At every time step, the current action set is taxed. This tax is given to the classifiers from the previous time step via the bucket brigade. This translates into a wave of diminishing reward that passes back along the chain of classifiers. For a more detailed description of the reinforcement phase, see (Wilson, 1994b).

Other reinforcement schemes used in learning systems require a great deal of information about rule activation. The reinforcement mechanisms in LEX and ACT*, for example, keep track of all activated rules preceding a reward (Wilson, 1987b). Keeping track of all activated rules may not be feasible when a learning system is operating in a complex, unpredictable domain (autonomous agents, for example).

Maintaining long chains in a classifier system is important for problems that require a large number of steps before receiving a payoff. Some examples of domains where a large number of steps must be performed before payoff are: food foraging (Wilson, 1987a), chess (Samuel, 1981), and backgammon (Tesauro, 1992). There are two weaknesses of long chains: they are hard to maintain (the reinforcement problem), and they are hard to generate in the first place (Wilson & Goldberg, 1989). Furthermore, the genetic component of the classifier system and the bucket brigade may interact in unpredictable ways, leading to the destabilization of correct solutions (Compiani et al., 1991).

Prior art dealing with long chains

In this section I describe research into using various rule-clustering schemes in classifier systems.

Hierarchical Credit Allocation

Hierarchical Credit Allocation (HCA), proposed by Wilson (Wilson, 1987b), is a scheme where classifiers post and match messages that occur at different *levels*. By making the message system into a hierarchy, Wilson brings in a way for higher level classifiers to invoke lower level ones.

There is no notion of a genetic operator working on hierarchies of classifiers. The hierarchy is only exploited in activation and reinforcement. The classifiers fall into hierarchies depending on what messages they respond to. There is no operator that can in general take a sub-hierarchy and insert it into another hierarchy.

HCA is additionally constrained to the same fixed-length representation used by other classifier systems. The system proposed here allows the representation to change dynamically, adapting to the particular problem. In this respect, the proposed system is similar to the one proposed in (Wilson, 1994b).

- useful subtree identification
- efficient agglomeration of subtrees into fit modules
- caching of evaluated results of frequently used subtrees
- cooperation between members of population

Figure 2: Potential Benefits for GP

Hierarchically Structured Classifier System

In a Hierarchically Structured Classifier System (HCS), described in (Shu & Schaeffer, 1991), classifiers are grouped into clusters through a relationship operator. Crossover can occur between individuals in a cluster and also between two clusters. Crossover between clusters is given a higher probability than crossover within a cluster.

Individuals are fixed-length representations, and the initial grouping of classifiers is done randomly. The fitness of a classifier is directly proportional to the sum of the fitnesses of classifiers within its cluster.

There is no mutation in HCS. If there are no schemata in the initial random population to cover the solution space, there is no way for them to arise.

Additionally, HCS assumes that clustering is beneficial to solving a problem. If applied to tasks in which clustering is of no benefit (the Boolean Multiplexor, for instance (Goldberg, 1989, p.293)), HCS would spend much time on unnecessary operations: calculating the fitness of clusters and performing crossover between clusters.

Dynamic Classifiers

The Dynamic Classifier System extends traditional classifier systems and provides potential benefits for genetic programming (Figure 2).

The idea is that classifier systems are good at identifying short chains of rules, while genetic programming

1. initialize population with Lisp classifiers.
2. set sense vector to current inputs
3. put classifiers that match sense vector into match set (Section).
4. select action based on classifiers in match set
5. apply reward (if any) to those classifiers in match set that advocated the selected action
6. pass reward back using bucket brigade
7. apply genetic programming to expressions in the current match set
8. jump to step 2

Figure 3: DCS Execution Loop

suffers from not having a mechanism to give credit to useful subtrees.

At the same time, genetic programming is good at recognizing the fitness of trees of arbitrarily large size, while classifier systems using the bucket brigade algorithm have trouble with the analogous task of recognizing the fitness of rule chains of arbitrary length.

Implementation of Dynamic Classifiers

DCS is modeled after ZCS (Wilson, 1994a), which in turn decends from the standard classifier system as proposed by Holland. DCS differs from predecessors in that it uses a tree representation for the condition side of the classifiers. Each classifier has a single action associated with it. There are other ways to hybridize genetic programming and classifier systems—multiple actions per rule, for instance—but this paper will focus on the model closest to Holland's.

The genetic operators of crossover and mutation are the same as those from genetic programming. These operators are applied to the condition side of the classifiers, producing more complex match rules over time.

The execution loop of the system is shown in Figure 3. Note that this is essentially the same as that of ZCS (Wilson, 1994b). The only changes are in the representation of the classifier conditions and the genetic operators that act on them.

Some sample DCS classifiers are shown in Figure 4. These classifiers are from an application of DCS to the Animat problem (Wilson, 1987a; Wilson, 1991). Clustering arises from the interaction between crossover on S-expressions and reinforcement from the bucket brigade.

```
(if (not nw-v2) (go-ne))
(if (and s-v2 e-v1) (go-sw))
(if (or se-v1 nw-v1) (go-e))

(if (or (or (or s-v1
                (not (not ne-v1)))
            nw-v2) nw-v2)
    (go-sw))
```

Figure 4: Some DCS classifiers for the Animat problem

Theoretical improvements of hierarchical clustering

Dynamic Classifiers are a form of hierarchical rule clustering. As such, they alter the properties of reinforcement within classifier systems by grouping rules together into single units, clusters. A cluster behaves as a single rule for the purposes of reinforcement.

The Bucket Brigade

It takes $O(n)$ time to form a chain of length n (Wilson, 1987b). A chain here means a sequence of classifiers in which a reward passes from one end to the other. The *length* of the chain is the number of classifiers it contains, and the *degree of coupling* is given by the following equation.

Let $C_1 \dots C_n$ be the classifiers in a chain, and $S(C_i)$ be the strength of a given classifier i. Furthermore, let M be the maximum observed strength of all classifiers in a population (not just those in the chain) at a given time step.

Then the theoretical degree of coupling for a given chain is:

$$\frac{S(C_n) - S(C_1)}{M} \tag{1}$$

The *imbalance* of a chain is the difference in strengths of the first and last classifiers in a chain.

The time to bring the first step in an n step chain to within 90% of the last step as follows (the derivation given below is based on Wilson's (Wilson, 1987b)).

Let a bucket brigade sequence of length n be represented by a list of n elements, each initialized to a strength of zero. We represent one repetition of the whole sequence by having each element i pass a fraction of its strength to the previous element in the chain:

$$S_i(t+1) = S_i(t) - cS_i(t) + cS_{i+1}(t) \tag{2}$$

where the strength of the last element n, which receives a payoff R from the environment, is given by

$$S_n(t+1) = S_n(t) - cS_n(t) + R \tag{3}$$

Bucket Brigade with Clustering

If clustering occurs between two classifiers after r reinforcements, then the time to reinforce a chain of length n becomes

$$t = r + (n - 1) \tag{4}$$

That is, it takes r steps for C_n and C_{n-1} to merge into a single node. The remaining nodes, C_{n-k} ($k = 2 \ldots 9$) have been reinforced $r - k$ times, respectively. Each time step after $t = r$, an additional classifier will cluster with the ones that preceded it.

Going back to the equation from (Wilson, 1987b), it will now take $r + (n - 1)$ steps for the classifiers to form a single unit—a cluster, here, as opposed to a chain.

Since $r + (10 - 1) \leq 150$ for $r \leq 141$, whenever $r < 141$ a sequence of 10 classifiers will cluster faster than the bucket brigade can form a chain.

By introducing clustering, it is possible for a chain of classifiers to become *highly coupled* and *balanced* (as defined earlier).

Summary

In this paper, I described the difficulty of problem decomposition using classifier and genetic programming systems. I then introduced the Dynamic Classifier System (DCS, first proposed by Wilson in (Wilson, 1994a)). I showed how DCS deals with task decomposition by building on prior work, sketched its implementation, and surveyed its potential advantages.

References

Angeline, Peter J. (1994). Genetic programming and emergent intelligence. In Kinnear, Jr., Kenneth E. (Ed.), *Advances in Genetic Programming*, chapter 4, pp. 75-97. MIT Press.

Angeline, Peter J. & Pollack, Jordan (1993). Evolutionary module acquisition. In *The Second Annual Conference on Evolutionary Programming*, La Jolla, California.

Compiani, M., Montanari, D., & Serra, R. (1991). Learning and bucket brigade dynamics in classifier systems. In Forrest, Stephanie (Ed.), *Emergent Computation (Special Issue of Physica D)*, pp. 202–212. MIT/North-Holland.

Goldberg, David E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley.

Holland, John H., Holyoak, Keith J., Nisbett, Richard E., & Thagard, Paul R. (1987). *Induction: Processes of inference, learning, and discovery.* MIT Press.

Kinnear, Jr., Kenneth E. (1994). Alternatives in automatic function definition: A comparison of performance. In Kinnear, Jr., Kenneth E. (Ed.), *Advances in Genetic Programming*, chapter 6, pp. 120–141. MIT Press.

Koza, John R. (1992). *Genetic Programming: on the programming of computers by means of natural selection.* MIT Press.

Koza, John R. (1994). Scalable learning in genetic programming using automatic function definition. In Kinnear, Jr., Kenneth E. (Ed.), *Advances in Genetic Programming*, chapter 5, pp. 99–117. MIT Press.

Samuel, A. L. (1981). Some studies in machine learning using the game of checkers. In Feigenbaum, Edward A. & Feldman, Julian (Eds.), *Computers and Thought*, pp. 71–105. McGraw-Hill. Reprint of the 1963 edition.

Shu, Lingyan & Schaeffer, Johnathan (1991). HCS: Adding hierarchies to classifier systems. In Belew, Richard K. & Booker, Lashon B. (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 339–345, San Mateo, CA. Morgan Kauffmann.

Sutton, Richard. S. (1984). *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, UMass Amherst.

Sutton, Richard. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44.

Tesauro, Gerald (1992). Practical issues in temporal difference learning. *Machine Learning*, 8:257–277.

Tufts, Patrick & Juillé, Hugues (1994). Evolving non-deterministic algorithms for efficient sorting networks. Submitted.

Watkins, Christopher John Cornish Hellaby (1989). *Learning from Delayed Rewards.* PhD thesis, King's College, Cambridge University.

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

Wilson, Stewart W. (1987b). Hierarchical credit allocation in a classifier system. In Davis, Lawrence (Ed.), *Genetic Algorithms and Simulated Annealing*, Research Notes in Artificial Intelligence, chapter 8, pp. 104–115. Morgan Kauffmann, Los Altos, CA.

Wilson, Stewart W. (1991). The animat path to AI. In Meyer, Jean-Arcady & Wilson, Stewart W. (Eds.), *From Animals to Animats*, pp. 15–21. MIT Press.

Wilson, Stewart W. (1994a). Classifier fitness based on accuracy. Technical report, The Rowland Institute for Science. http://netq.rowland.org/art0.html.

Wilson, Stewart W. (1994b). ZCS: A zeroth level classifier system. *Evolutionary Computation*, 2(1):1–18.

Wilson, Stewart W. & Goldberg, David E. (1989). A critical review of classifier systems. In Schaffer, J. David (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, CA. Morgan Kauffmann.