A Genetics-Based Technique for the Automated Acquisition of Expert System Rule Bases

Clayton M. Johnson  Stefan Feyock

Department of Computer Science
The College of William and Mary
Williamsburg, Virginia  23185

1 Introduction

The Genetic Algorithm (GA) is a powerful search paradigm which combines elements from evolutionary biology with concepts from population genetics. Because they operate in a domain independent fashion, GAs have been successfully applied to a wide variety of optimization and learning problems. Despite these successes, however, GAs have not yet made the breakthrough into mainstream Artificial Intelligence (AI). One reason for this discrepancy is that genetic algorithmists have typically adopted simple binary representation schemes in their research. The larger AI community, on the other hand, has relied mainly upon more expressive semantic representations describing complex interrelationships between problem elements.

This paper presents a technique by which genetic algorithms can be adapted to operate upon the LISP-like production rules typically used in Expert Systems. The work described in this paper is part of an ongoing project being conducted at the NASA/Langley Research Center in Hampton, Virginia. The aim of this project is the automated discovery of an expert system rule base for the prediction and diagnosis of in-flight malfunctions occurring in aircraft. Before discussing how this might be accomplished, a brief overview of genetic algorithms and genetics-based learning is in order.

2 Genetic Algorithms

The evolutionary process is the only naturally occurring search algorithm known to exist. In the past three million years, only a small fraction of the potentially infinite number of genetic combinations have been examined, and yet a large variety of complex structures have been successfully developed. The chromosomes of simple vertebrate organisms, for example, are composed of tens of thousands of genes, each of which can take on at least two values. Ignoring the non-linear nature of gene interactions, this corresponds to a minimum of $10^{3000}$ phenotypic possibilities for each vertebrate species. The enormous complexity of the structures "discovered" by natural selection is evidence of the system's incredible search capabilities.

The genetic algorithm (GA) captures some of the power of natural search by closely following the evolutionary paradigm. As with nature, the GA conducts its investigation from a population of individuals. This sharply contrasts traditional point-by-point techniques which follow some heuristic guide to climb a functional peak until a maximal objective function value is reached. Because the GA is able to climb many peaks in parallel, the chances of isolating a local optimum as the best solution are greatly reduced.

In keeping with the genetic metaphor, GAs require that all search parameters be coded into finite length strings over a fixed alphabet called "chromosomes".

As a simple example, if we were interested in maximizing the quadratic function

$$f(x) = x^2 - 20x + 100$$

over the integers in the interval [0,31], we might use a five-bit unsigned integer ranging in value from 00000 (zero) to 11111 (thirty-one) to represent the problem's domain.

The GA works directly on a population of chromosomes without requiring additional knowledge about the problem. In fact, the only information a genetic algorithm needs is an objective function which it uses to gauge the merit of each potential solution. In the above example, this would be equivalent to the function which is being optimized.

The GA operates iteratively, testing each member of the population by means of the objective function, and then combining useful individuals to form successive generations. This closely parallels natural populations in which new generations of organisms are produced and then raised until they themselves are able to reproduce. Since the evaluation of each point in the population is an independent calculation, GAs are highly adaptable to parallel processors.

The simplest genetic algorithms consist of only three operators: reproduction, crossover and mutation.

2.1 Reproduction

Reproduction is an artificial "survival of the fittest" in which strings are copied according to their fitness (the objective function). There are many well-established methods for performing genetic reproduction. The easiest technique is "roulette wheel selection", which defines the probability of selecting a given
string for reproduction as the fitness of that string divided by the sum of all fitness values from the current population. A new population is then created by using these selection probabilities in successive spins of a weighted roulette wheel.

Turning once again to the quadratic maximization example, the GA might initially generate the following random population of four strings (listed together with their associated fitness values):

```
00011 49
10101 121
11010 256
01111 4
```

Dividing the fitness of each string by the summation of all fitness values in the population (430 = 49 + 121 + 256 + 4), the following probabilities for roulette wheel selection would be obtained:

```
00011 11.4%
10101 28.1%
11010 59.5%
01111 1.0%
```

Since the probability of selecting 11010 is over fifty percent, at least half of the individuals in the next generation should be copies of this string. The low performing string 01111, however, is unlikely to make it into any successive populations. In this manner, the GA is able to focus its search by emphasizing the best structures discovered.

2.2 Crossover

Crossover is a randomized recombination operator which proceeds in three steps. First, members of the new population created through reproduction are mated at random. A cross site along the two strings is then chosen uniformly at random along the length of the string. Finally, two new strings are created by swapping the substrings following the cross site. If for example, the following two strings were mated,

```
10110
11010
```

and position two was chosen as the locus for crossover as indicated by the 1 symbol, the two new strings

```
10010
11101
```

would be created. In many GA applications, crossover takes place only with a prespecified probability.

2.3 Mutation

Reproduction with crossover is an effective means of combining high quality solutions; occasionally, however, these operators may become overzealous and lose important genetic material at certain positions. Mutation acts as an insurance policy against such unrecoverable loss by occasionally flipping a bit during the reproductive process. Mutation plays only a secondary role in the GA, as mutation probabilities are typically set quite low and in inverse proportion to the population size.

3 Classifier Systems

A classifier system is a genetics-based technique for learning syntactically simple rules called classifiers to handle some arbitrary problem. As schematically depicted in Figure 1, a classifier system is composed of three basic elements.

3.1 Rule and Message-Passing System

The rule and message-passing system receives information from the environment through a set of detectors. These detectors encode the current world state into one or more fixed length messages, typically over the binary alphabet. Messages from the detectors are posted to a message list, where they may trigger any number of classifiers.

In its simplest form, a classifier is a simple production rule consisting of a single condition-action pair. The classifier condition is a pattern recognition device in which the "wild card" symbol # is added to the underlying alphabet. A message matches a condition if at every position a zero in the condition matches a zero in the message, a one matches a one, or a # matches either a zero or a one. For instance, the message 10011 would match classifiers with either a 1#111 or a #0#. 

![Classifier System](image)

Figure 1: Classifier System

When a classifier's condition is matched, it becomes a candidate to post its message to the message list during the next time step. This message might trigger other classifiers, or it may cause some action to be performed through a set of system effectors. In this way, a classifier system is able to make decisions based on both external cues from its detectors and its internal processing states.

3.2 Apportionment of Credit System

Since there are only a limited number of slots on the message list, some method must exist for determining which classifiers are allowed to post messages. Although there are many ways of accomplishing this task, the bucket brigade algorithm suggested by Holland [7] is by far the most widely used.

In order to implement the bucket brigade, each classifier is assigned a quantity called its strength which is periodically adjusted in response to how useful the
classifier has been in achieving results. Strength values are then used by the bucket brigade as the basis of competition between classifiers. During each time step, a matching classifier posts a bid based on both its strength and its relevance to the current situation. Only the highest bidding classifiers are then allowed to fire.

More formally, let \( s(C, t) \) be the strength of classifier \( C \) at time \( t \). If the condition of \( C \) is matched by a message currently on the message list, it posts a bid

\[
B(C, t) = b \cdot R(C) \cdot s(C, t)
\]

where \( b \) is a constant considerably less than one, and \( R(C) \) is the relevance of the classifier equal to the number of non-# elements in the classifier's condition. The size of this bid represents the probability that the classifier posts its message to the message list. This probabilistic auction is used to provide for the occasional testing of less favored and newly created classifiers.

The bucket brigade algorithm is modeled after an information economy in which the right to trade information is bought and sold. Each classifier is treated as a "middleman" in the economy, dealing only with its suppliers (classifiers sending messages satisfying its condition) and its consumers (classifiers whose conditions match the message it posts). When a classifier makes a bid, it pays out part of its strength as follows:

\[
s(C, t + 1) = s(C, t) - B(C, t)
\]

Its suppliers \( \{C'\} \), the classifiers sending messages matched by a winning classifier \( C \), receive payment in the form

\[
s(C', t + 1) = s(C', t) + 1/a \cdot B(C, t)
\]

where \( a \) is the number of members in \( C \). Strength is therefore treated as a kind of capital that measures a classifier's ability to turn a profit in the system. As a simple example of how a classifier system functions under the bucket brigade, consider the following four classifiers and their associated strengths:

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Strength</th>
<th>Message Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>130</td>
<td>#0#:#0111</td>
</tr>
<tr>
<td>C2</td>
<td>200</td>
<td>#1#:1110</td>
</tr>
<tr>
<td>C3</td>
<td>150</td>
<td>#11:1001</td>
</tr>
<tr>
<td>C4</td>
<td>100</td>
<td>#00#:1010</td>
</tr>
</tbody>
</table>

where the : symbol is being used to separate the classifier condition and action. If the message 1001 was just posted to the message list by C3 during the last time step, both C1 and C4 would match. Assuming that the message list is only of length one, and the bidding constant \( b \) is set to 0.1, the following bids would be made:

\[
B(C1, t) = 0.1 \cdot 130 \cdot 1 = 13
\]
\[
B(C4, t) = 0.1 \cdot 100 \cdot 2 = 20
\]

These amounts would then be subtracted from the strengths of the relevant classifiers during the next time step:

\[
S(C1, t + 1) = 130 - 13 = 117
\]
\[
S(C4, t + 1) = 100 - 20 = 80
\]

Assuming the C4 won the probabilistic bidding process, the strength of C3 would be increased to

\[
S(C3, t + 1) = 150 + 1/1 \cdot 20 = 170
\]

The resulting state of the classifier system after this one execution cycle would therefore be

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Strength</th>
<th>Message Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>117</td>
<td>#0#:0111</td>
</tr>
<tr>
<td>C2</td>
<td>200</td>
<td>#1#:1110</td>
</tr>
<tr>
<td>C3</td>
<td>170</td>
<td>#11:1001</td>
</tr>
<tr>
<td>C4</td>
<td>80</td>
<td>#00#:1010</td>
</tr>
</tbody>
</table>

As observed in many simulations, the bucket brigade encourages the development of "default hierarchies" in which general classifiers cover most of the common cases while fairly specific classifiers handle the exceptions. Booker [5] has also shown that, because payoff is shared between classifiers, appropriate sized subpopulations of classifiers for each decision are maintained.

### 3.3 The Genetic Algorithm

In addition to exploiting existing rules, the classifier system also discovers potentially new classifiers by means of a genetic algorithm. After a relatively large number of iterations, the entire classifier system is submitted to the genetic algorithm in order to create a new population. Classifier strength is used as the measure of fitness instead of an objective function, and only a subpopulation of low performance classifiers is replaced instead of the entire population. Otherwise, however, the genetic search adopted by the classifier system performs in essentially the same manner as a simple three-operator GA.

### 4 Rule Representation

There are several difficulties inherent to the simple binary encoding schemes used by most classifier systems. Since the number of rule interpretations is not always a power of two, classifier systems often contain redundant information which may unevenly weight the search while at the same time reducing the effectiveness of recombination operators. It is also difficult to describe interdependencies and relationships between features in the environment. This makes it complicated to manipulate and add built-in knowledge and world models. Since the intent of this research is the automated discovery of Expert System rule bases, a higher level representation language based on LISP has been used instead.

source code for solving the Prisoner's Dilemma. With the exception of Antonisse and Keller's proposal, these efforts have treated rules as inviolable elements to be exchanged between alternate solution sets. The technique described in this paper represents a departure from the above approaches in that individual rules are the "chromosomes" which are manipulated by genetic search. This process allows for the incremental discovery of new rule bases through the adaptation of existing members of the population.

Figure 2 describes a partial grammar defining rules which could be used by the system to dynamically control the power settings of a twin-engine aircraft in response to changing pressure, temperature and air-speed. Variables are bound by the input and output interfaces to the GA, and are not of immediate concern.

1: <rule> (if <condition> then <action>)
2: <condition> (not <condition>)
3: <condition> (and <condition> <condition>)
4: <condition> (or <condition> <condition>)
5: <condition> <attribute>
6: <action> (power <engine> <change>)
7: <attribute> (temperature <engine> <change>)
8: <attribute> (pressure <engine> <change>)
9: <attribute> (speed <change>)
10: <engine> enginel
11: <engine> engine2
12: <engine> <variable>
13: <change> increasing
14: <change> decreasing
15: <change> constant
16: <change> <variable>

Figure 2: Rule Grammar

In order to use this grammar effectively, each rule in the population is accompanied by a trace of its derivation. Each trace is a list of integers corresponding to the productions which were used when generating a rule. This list is additionally annotated by zeros to mark the completed expansion of each production during a leftmost derivation through the grammar. Each production therefore contributes two pieces of information to the trace - an unique integer marking its application and a zero marking its completion. Subexpressions can then be found easily by a simple parity count: beginning with any nonzero element, the trace is searched from left to right until the number of zero and nonzero elements examined is equal. This section of the trace represents a parse subtree of the rule in question. For example, the rule

(if (not (pressure X decreasing))
then (power X constant))

would have associated with it the trace

(1 2 5 8 12 0 14 0 0 0 0 6 12 0 15 0 0 0)

One possible subtree from this trace would be

(8 12 0 14 0 0)

which is equivalent to the rule subexpression

(pressure X decreasing)

As will be seen below, this zero-embedded representation allows the genetic operators to create new, syntactically correct rules while preserving the "building block" nature of the GA search.

5 Genetic Operators

5.1 Crossover

The standard crossover operator functions by exchanging genetic material between two rules. In traditional GAs, a position or locus is used to determine how much information each parent passes on to its offspring. The genes before this locus in the first parent are combined with the genes after this locus in the second in order to create a new rule.

A slightly modified version of this operation is needed to perform crossover in this representation. The derivation traces of both parents are first examined for common non-zero elements. Note that since all traces must contain the first production of the grammar, a common element is guaranteed to exist. One of these elements is then randomly selected as the locus for crossover. Finally, the amount of material to be exchanged is determined by a simple parity count as described above. Since this modified crossover process only permits an exchange of data between subtrees with common root elements, syntactically valid rules will always be generated.

As an example, consider the following two rules and their traces:

Rule 1
(if (not (pressure X decreasing))
then (power X constant))

(1 2 5 8 12 0 14 0 0 0 0 6 12 0 15 0 0 0)

Rule 2
(if (and (speed decreasing) (temperature engine1 X))
then (power engine1 increasing))

(1 3 5 9 14 0 0 0 5 7 10 0 16 0 0 0 6 10 0 13 0 0 0)

The two traces have the following elements in common:

(1 5 6 14)

If element 5 was then randomly selected as the locus for crossover, a new rule

(if (not (speed decreasing))
then (power X constant))

(1 2 5 9 14 0 0 0 0 6 12 0 15 0 0 0)

would be generated. Although some computation time is required to find elements common to both parents, new rules do not have to be parsed in the grammar. Their traces are automatically generated through the crossover operator, and it is then a simple matter to construct the corresponding IF-THEN rules.
5.2 Mutation

The crossover operator described above is sufficiently able to exchange information between existing rule structures in the GA; crossover alone, however, cannot introduce new material into the system. As is the case with traditional GAS, a mutation operator is necessary to accomplish this task. Unlike these systems, though, randomly miscopying a gene during reproduction is not enough. Some care must be taken to mutate the rules and still produce syntactically correct IF-THEN structures. Once again, this is accomplished by means of the derivation trace.

Once a rule has been chosen for mutation, a random non-zero element from its trace is selected. The entire subtree beginning with this element is determined as above and then removed from the trace. It is replaced by randomly creating a new derivation from the grammar beginning with the selected production. If, for instance, the following rule and its trace were chosen for mutation,

\[(\text{if} \ (\neg \ (\text{speed} \ \text{decreasing})) \ \text{then} \ (\text{power} \ X \ \text{constant}))\]

and production 6 was chosen as the locus, then the new rule

\[(\text{if} \ (\neg \ (\text{speed} \ \text{decreasing})) \ \text{then} \ (\text{power} \ \text{engine2} \ \text{decreasing}))\]

might be created. As can be seen, this modified process of mutation ensures the creation of syntactically valid rules in the grammar.

6 Expert Systems Applications

Now that we have described a method by which genetic algorithms can be applied to more descriptive rule languages, it is time to address how this technique can be used in the development of Expert Systems. The first step, of course, is the design of a rule grammar describing the domain of interest. This grammar must contain not only objects and their attributes, but also the possible relationships between these elements and the system's possible responses. Although at first glance this might seem like an added burden to impose upon the system designer, it is in effect not much different from the developmental work typically performed by knowledge engineers. The only difference is that the choice of representation is made more formal.

Once a grammar has been selected, the only other design consideration is the determination of rule "payoff". In order for a genetics-based learning system to discover useful decision rules, it is necessary to supply the system with a feedback mechanism by which rules may be evaluated. This process requires that a test bed of expert decisions be made available to the system during its learning phase. Payoff can then be awarded to the GA in all cases where its decisions correspond with those of the expert. Negative payoff can be associated with decision actions which would have a detrimental effect on the system if performed. Once the GA has reached a level or performance where it agrees with the expert an arbitrarily high percentage of the time, a good expert rule base will have been developed.

7 Acknowledgements

The work presented in this paper was supported in part by NASA grant NCC-1-122.

References


