A Genetic Algorithm for Expert System Rule Generation

John C. Determan
Idaho National Engineering and Environmental Laboratory

James A. Foster
Computer Science Department
University of Idaho

ABSTRACT

We applied genetic algorithms to fuzzy rule generation to compute expert system rules from data. We have attempted to improve on existing techniques for the automatic generation of fuzzy logic expert system rules with a method we call genetic data clustering (GDC). A genetic algorithm groups training data points by their degree of similarity, and fuzzy logic expert system rules are formed from statistics over the members of these groups. We designed mutation and crossover operators, and an objective function specifically for the task of data clustering. We employed variable mutation probabilities and variable chromosome lengths. We tested the algorithm against two data sets, both large and small, including the Anderson and Fisher Iris data, and a LANDSAT satellite image data set. Overall, the results indicate that the GDC algorithm achieved average performance compared to a suite of classification algorithms found in the literature. The rule representation format was the primary limitation to performance.

ACKNOWLEDGMENTS

This document was prepared for the U.S. Department of Energy Assistant Secretary for Environmental Management under DOE Idaho Operations Office Contract DE-AC07-99ID13727. The authors would like to thank Dr. Tim Roney, and Dr. James Seydel for their support of this work, and also thank Mr. Greg Becker for lending his expertise in the field of nondestructive assay.
1. INTRODUCTION

We developed a new genetic algorithm (GA) based data clustering algorithm for generating fuzzy logic (Zadeh, 1965) expert system rules (Determan and Foster, 2000). There are two major questions that rule generation algorithms must face: 1) how many clusters exist in the data, and 2) how are ambiguous regions of the data, where clusters may overlap, treated? The first point determines how many rules are found, and the second affects rule accuracy. Data clustering algorithms possess adjustable parameters that affect the number of clusters found, and the membership profiles of the clusters. Our initial work used GAs to determine values for these adjustable parameters, but with some experience, we realized that GAs could tackle data clustering in a more direct and efficient manner. Instead of representing algorithm parameters, the genome could represent the cluster membership profiles, and number of clusters in the data. An advantage of performing genetic data clustering is that GAs are probabilistic, and ambiguous data points have some probability of being assigned to any cluster. With a good objective function, and appropriate genetic operators, natural selection should find good data clusters. However, grouping problems are difficult for GAs, due to tight constraints on how the genome can be mutated and crossed. In grouping problems a GA must neither duplicate nor remove alleles from the genome, and ordinary genetic operators do not guarantee these properties. Therefore, the genetic data clustering problem could be expected to shed light on an area of interest in the Evolutionary Computing community: combinatorial or grouping problems.

1.1 PREVIOUS WORK

The subtractive clustering algorithm (Chiu, 1994), based on the mountain clustering method (Yager and Filev, 1994) can generate fuzzy logic expert system rules from training data. The clustering process locates potential data clusters and estimates their means and standard deviations in each dimension of the training data. Backpropagation against the training data refines these means and standard deviations. The technique then forms a fuzzy logic rule from each cluster, and estimates the rule output as the mean of the correct classification for each cluster. For this process to work, various parameters need to be defined, the
most important of which are the cluster radius and the backpropagation learning rate and convergence criteria. The choice of cluster radius affects the number of clusters located in the data, and therefore the number of rules. The other parameters affect the estimated means and standard deviations, and therefore, the accuracy of the rules.

### 1.2 Our Work

We introduce several innovations that improve both the speed of the rule generation process and the accuracy of the generated rules. Our work also contributes to the application of GAs to combinatorial problems in general. First, we inverted the rule generation process described above. Our genetic data clustering algorithm first locates data points in the training data that belong together, it then calculates exact means and standard deviations over these clusters, and finally, it builds fuzzy rules from the cluster means and standard deviations, as before. This procedure bypasses the time-consuming process of backpropagation. We designed crossover and mutation operators specifically for data clustering. We used self-adaptive techniques, including variable length chromosomes and a variable probability of mutation. We performed tests to examine the effect of the probability of crossover. A constant value of 0.8 performed the best.

Section 2 discusses some fundamental issues in automatic rule generation. Section 3 details the genetic data clustering approach to automatic rule generation. Section 4 describes our evaluation of this new approach and presents data. Section 5 presents our conclusions.
2. AUTOMATIC RULE GENERATION

Generating fuzzy rules from training data is a two-step process. The first step is to locate data clusters in the training data. The second step is to use parameters derived from the data clusters to form fuzzy rule membership functions. We will discuss the relationship between data clusters and fuzzy rules first, followed by a discussion of data clustering algorithms. Figure 1 is an overview of the rule generation process.

Figure 1. Overview of the rule generation process. Training data imply a set of clusters, and fuzzy logic expert system rules derive from parameters calculated from these clusters.

2.1 Data Clusters and Fuzzy Rules

Knowledge in an expert system can come in many forms. It may be that an expert has formulated a set of precise rules based on independent observations, and these can be elicited and encoded. It may also be that an expert must examine a set of features and make a more integrated judgement. In the latter case, the rules may be so inter-related that they cannot be easily extricated from one another and encoded in a simple fashion. The expert however can accurately classify a data set, and may even give rationales for these classifications based on relative magnitudes of various features in the data, and experience, which trains the expert to know which features in the data are of overriding importance. Mathematical procedures exist to generate a set of interrelated rules from expert classified training data. These rules employ fuzzy logic, and can accurately model the human decision making process.
Fuzzy rules match inputs to patterns described by fuzzy sets. A fuzzy set allows graded membership for elements of the set. Fuzzy sets represent many real world situations, such as the set of sunny days, or the set of acceptable candidates. Fuzzy sets are represented mathematically by fuzzy membership functions. For example, the set “All men with height exceeding seven feet” is not a fuzzy set, for any proposed member either is or is not a member of the set. We say this set has a crisp membership function. On the other hand, the set “All tall men” is a fuzzy set. Both six feet tall men and seven feet men would, in most contexts, be members of the set, but not to the same degree. Figure 2 illustrates membership functions for the sets “All men with height exceeding seven feet” and “All tall men”. The membership function for the latter set clearly depicts the property of graded membership. We say it has a fuzzy membership function.

**Figure 2.** The figure shows membership functions for both ordinary and fuzzy sets. The property of graded membership distinguishes fuzzy sets from crisp sets.

In data clustering, the training data are viewed as points in a hyperspace, the dimension of which is the total number of input and output elements. As indicated in Figure 1, the rule generation process involves identifying data clusters within the training data. A cluster of data points represents an
approximate relationship from input to output, and this relationship can be represented as a rule using fuzzy logic. Fuzzy logic naturally captures the approximate nature of this relationship.

Consider an example. A precisely known relationship could be represented as a single data point \((x_1^*, x_2^*, x_3^*, y^*)\), rather than a cluster of data points. We can interpret this data point as a rule of the form:

\[
(1) \quad \text{if } (x_1 = x_1^*) \text{ and } (x_2 = x_2^*) \text{ and } (x_3 = x_3^*) \text{ then } y = y^*
\]

In words, any data point \((x_1, x_2, x_3, y)\) with features that exactly match the features of the known data point \((x_1^*, x_2^*, x_3^*, y^*)\) has the same output as that data point. While we have simplified the concept to the point of triviality, we can use this example to generalize to a more realistic situation. When dealing with training data, there is typically not any one single data point that represents an exact relationship, but there exist clusters of related data points, and a cluster centroid can be viewed as an abstracted data point. This abstracted data point forms a rule in the same manner as the single data point did above, except that the relationship is no longer exact. Averaging the values of training data features in each dimension determines the cluster centroid, and averaging gives rise to variation. Thus, clusters of similar data points in training data define rules, but these rules are not exact and are naturally represented as fuzzy rules. Cluster statistical parameters determine each such fuzzy rule.

In general, for \(n\) input and \(m\) output elements in each datum, the generated fuzzy rules take the following form:

\[
(2) \quad \text{if } \left(x_{i1} \text{ matches } A_{i1}\right) \ldots \left(x_{im} \text{ matches } A_{im}\right) \text{ then } \left(y_{i1} \text{ is } B_{i1}\right) \ldots \left(y_{im} \text{ is } B_{im}\right) \text{ where } \\
i \text{ indicates the } i^{th} \text{ rule, } \\
x_{i1} \ldots x_{im} \text{ are normalized rule input values for the } k^{th} \text{ data point, } x_k \\
y_{i1} \ldots y_{im} \text{ are rule output values, } \\
A_{i1} \ldots A_{im} \text{ are exponential membership functions (described below), and } \\
B_{i1} \ldots B_{im} \text{ are symmetric membership functions (described below).}
\]
The use of symmetric output functions allows for a simpler solution, in that only the centroids of the output membership functions need be determined. The input membership functions for the $i$th rule are expressed as follows:

$$A_i(x_{k_j}) = \exp\left(-\frac{1}{2} \left( \frac{x_{k_j} - x^*_j}{\sigma_{k_j}} \right)^2 \right), \quad 1 \leq j \leq n,$$

where $x^*_j$ and $\sigma_{k_j}$ are estimated cluster means and standard deviations, and are discussed below.

For simplicity, the remainder of the discussion will focus on the more specific case of one output element in each rule, or $m = 1$.

The result of the calculation $A_i(x_{k_j})$ is referred to as the **match strength** of the input $x_{k_j}$ with the associated membership function $A_i$. The **match strength** of a rule with multiple inputs is the product of the component **match strengths**. The **rule strength** is given the symbol $\mu_i$ and is expressed as:

$$\mu_i(x_j) = \exp\left(-\frac{1}{2} \sum_{j=1}^{n} \left( \frac{x_{k_j} - x^*_j}{\sigma_{k_j}} \right)^2 \right)$$

In fuzzy inferencing, the **rule strength** modifies the rule output. Greater rule **match strength** lends greater weight to the consequence of the rule. Defuzzification algorithms account for this. Using a standard defuzzification method (the center of gravity algorithm), the output of the system is found from:

$$y = \frac{\sum_{i=1}^{n} \frac{\mu_i y^*_i}{\sum_{i=1}^{n} \mu_i}}{\sum_{i=1}^{n} \mu_i} = \frac{\sum_{i=1}^{n} \mu_i y_i}{\sum_{i=1}^{n} \mu_i},$$

where

- $n$ is the number of rules,
- $y^*_i$ is a vector of output membership function centroids, and
- $\frac{\mu_i}{\sum_{i=1}^{n} \mu_i}$ represents the normalized weight of each rule.
In words, the rule set output is the weighted sum of the individual rule outputs, where the weight on each rule is the rule match strength, normalized by the sum of all rule match strengths. Figure 3 diagrams the relationship between data clusters and fuzzy rules.

**Figure 3.** The figure shows an example of the relationship between data clusters and fuzzy rules. Cluster statistical parameters determine Gaussian membership functions.

\[
y_1^* = \text{average of } y \text{ values in cluster 1} \\
y_2^* = \text{average of } y \text{ values in cluster 2}
\]

2.2 Data clustering for Automatic Rule Generation

There are many data clustering algorithms (Bezdek, Li, Attikiouzel and Windham, 1997). Algorithms exist to obtain either fuzzy or crisp partitions of data. Whereas crisp clustering methods assign each point to exactly one cluster, fuzzy clustering algorithms determine the degree of membership of every data point with respect to each cluster. Our work in automatic rule generation began by building on the mountain (fuzzy) clustering method (Yager and Filev, 1994). The essence of mountain clustering is to divide the n-dimensional data space into an n-dimensional grid. A finer grid improves rule accuracy, but is also more computationally expensive. The nodes of the grid define potential cluster centers. The Mountain function, defined below, measures the worth of a node as a cluster center.
The mountain function estimates the density of data points in the region of a node. The node with the greatest value of the mountain function, \( M_1^* \), is chosen as the first cluster center, and denoted \( N_1^* \).

The mountain function at the remaining nodes is then revised to remove the influence of the selected node, using the following formula:

\[
M_i(N_y) = M(N_y) - M_1^* \exp\left(-\beta d(N_i^*, N_y)\right)
\]

where

- \( M_1^* \) is a constant,
- \( d(N_i^*, N_y) \) is an appropriate distance function,
- \( N_i^* \) is the node just identified as a cluster center, and
- \( M_1^* \) is the mountain function value of node \( N_1^* \).

Successive clusters are chosen as the node with the greatest value of the mountain function at each cycle, and the function values are again revised. Rather than fixing the total number of clusters, the algorithm uses a stopping parameter \( \delta \). When \( M_1^* < \delta \), too few data points exist in the vicinity of the currently selected node because the data point density is low in the region of the selected point, and further clustering is uninformative.

The accuracy of the procedure will improve as the grid size decreases; however, the number of steps that the procedure requires is exponential in the dimension of the problem, making a fine grid very expensive for high dimensional problems. Subtractive clustering (Chiu, 1994) is a refinement of mountain clustering that overcomes this limitation. In subtractive clustering, rather than using a grid, each data point becomes a potential cluster center. The algorithm proceeds as in mountain clustering by
calculating the potential (mountain function) of every node and selecting cluster centers as the nodes with the greatest potential at each step. Some additional criteria for ending the clustering process were also developed, involving a parameter termed the cluster radius. Increasing the cluster radius reduces the number of clusters that the algorithm will find.

In both mountain and subtractive clustering, the cluster radius is used to estimate an initial value of the standard deviation of each cluster. Thus, each cluster is initially characterized by estimates of the centroid coordinates \((x_i^*, y_i^*)\) and standard deviations \((\sigma_i)\). Gradient descent optimization is performed on the variables \(x_i^*, y_i^*\), and \(\sigma_i\) to fit them to the training data. This process is analogous to backpropagation training of neural networks. The following equation is the basic equation for backpropagation in neural networks and gradient descent optimization of fuzzy rule parameters:

\[
(8) \quad z_{\text{new}} = z_{\text{old}} - \alpha \frac{\delta E}{\delta z} \quad \text{where} \\
E = \frac{1}{2} e_k^2 
\]

The variable \(z\) could be an interconnection weight in a neural network, or any of the variables \(\sigma_i, x_i^*\) or \(y_i^*\). The term, \(e_k\), is the error between the predicted and assigned confidence value of a given assay, and \(\alpha\) is the learning rate. The learning rate \(\alpha\) is a value between zero and one, where low values accelerate convergence. Convergence is achieved when the root-mean-square error of the system with respect to the training data becomes relatively constant (i.e., the difference in successive iteration-averaged root-mean-square error values is below a specified criterion). The optimized parameters determine fuzzy rules, as discussed above.

As indicated in the discussion, we must specify various parameters to complete the process of automatic rule generation. These include the cluster radius, and the backpropagation learning rate and convergence criteria. Our initial work employed subtractive data clustering for fuzzy rule generation, and genetic algorithms (Goldberg, 1989) were used to determine the best set of values for these parameters.
The C++ genetic algorithm class library developed at the Massachusetts Institute of Technology called GAlib, version 2.4.2, was acquired and employed in this effort (Wall, 1996). The application of genetic algorithms in fuzzy-system design has been studied by a number of authors (Heider and Drabe, 1997) (Takagi and Lee, 1993) (Karr, 1991).

Our use of genetic algorithms was elementary at this stage. We encoded the set of three rule generation parameters as three real-valued genes in a chromosome. We used uniform crossover, defined an initializer to create a population of potential solutions over the space of valid rule generation parameters, and defined a mutator that varied the value of a gene by a small delta. Finally, we defined an objective function that scored a particular chromosome based on performance of the system with respect to a specified training set. The objective function first trained a set of rules using the rule generation parameters within the current chromosome, and then scored the derived set of rules with respect to the percent correct classifications on the training data.

The genetic algorithm required a few parameters. The population was small, six individuals, because the initial algorithm was slow. Two probabilities were also required: the probability of crossover, and the probability of mutation. The probability of crossover was 0.9, as crossover is the primary means of exploring the search space. The probability of mutation was 0.1 per allele, as some mutation was needed to thoroughly explore the search space, but too much mutation could result in the loss of a good solution from the population. A mutation probability of 0.1 may appear high, but it is appropriate to the number of unique individuals generated during the search. Six individuals per generation multiplied by 200 generations yields a maximum of 1200 individuals, a maximum that would not be reached, due to convergence. Were the mutation probability to be much lower than 0.1, the number of mutations would be miniscule.

An early modification to our initial application of genetic algorithms was the addition to the objective function of a term for estimating good clustering. Determining the cluster radius decides the
number of data clusters so it is important to recognize a good radius. Many cluster validity indices have
been proposed to deal with this issue (Bezdek et al., 1997). The Xie-Beni index (Xie and Beni, 1991), has
been proposed for evaluating the validity of fuzzy clusters. This index is the ratio of the compactness of
the clusters (a measure of the variance of the data points from the centers of the data clusters) to the
square of the minimum distance between the cluster centers. In general, small values of this index
indicate well-formed data clusters, provided the number of clusters does not approach the number of data
points in the training data (Chiu, 1994). Therefore, we used the Xie-Beni index as a secondary
component of the objective function.

Our work with genetic algorithms has proceeded beyond the simple application discussed above.
We have developed specialized initialization, mutation, and crossover operators for combinatorial
problems. We were motivated by two factors: 1) the initial approach was slow, and therefore required
that we use small populations, 2) the accuracy of the generated rules needed improvement. A faster
algorithm would allow larger populations, and this in turn would likely improve the quality of our
solutions. Emmanuel Faulkenaur (Faulkenaur 1998) touches on some similar approaches in “The
Grouping Genetic Algorithm.” Other researchers have also addressed this problem (Soule and Foster,
1997). Faulkenaur’s book provides a good background to the general topic of grouping items through
genetic means; our work deals more specifically with data clustering.
3. GENETIC DATA CLUSTERING

We modified the automatic rule generation process outlined in Section 3 to improve both speed of the rule generation algorithm and accuracy of the resulting rules. We replaced the combined steps of subtractive clustering and backpropagation with a new genetic data clustering (GDC) algorithm. We used subtractive clustering only in the initialization step of GDC, and completely eliminated backpropagation. We designed genetic operators specifically for the GDC algorithm. We used an objective function based on the Xie-Beni index, as in earlier work, in the GDC algorithm. After the algorithm identifies a set of clusters in the training data, it calculates the means and standard deviations in each dimension of the training data; finally, the algorithm constructs fuzzy rules from these values in the manner described above. This section discusses the GDC algorithm and its relationship to the process of automatic rule generation. Although our initial approach automatic rule generation initially employed fuzzy data clustering methods, the GDC is a crisp data clustering algorithm that generates fuzzy rules.

3.1 Genetic Representation for Data Clustering

We begin with a discussion of the genetic representation of a solution. The building blocks - the fundamental units of information - of a solution are clusters of data points thus, a good representation will allow members of a cluster to be contiguous genes. Figure 4 shows the representation used in the GDC algorithm. The representation is a simple one-dimensional list of integers. The first N integers in the list are unique values identifying the N data points. Integers representing the number of clusters and the size of each cluster follow the data point identifiers. The cluster size genes serve to break the data point identifier genes into groups representing the data clusters. For example, consider a training set consisting of 75 data points, and a given solution consisting of 9 clusters with a data point distribution of 10, 9, 7, 6, 8, 10, 8, 11, and 6 points per cluster. The encoding of this solution will begin with the integers 0 through 74 (the data point identifiers) in the first 75 positions, followed by the number 75, and then by the 9 cluster sizes. A modified form of the rule representation, described at the end of Section 4.2, requires
additional fit parameters at the end of the chromosome. Our mutation and crossover operators, discussed below, ensured that new offspring always have the correct number of data points and no duplications.

The number of clusters, M, varies between individuals, is randomly determined for each initial solution, and can vary by the operation of the crossover operator. This also implies that the chromosomes are of variable length. The algorithm enforced minimum values on the number of clusters and points per cluster to prevent divide by zero errors. We discuss these issues at greater length in the following sections.

Figure 4. Genetic representation for the GDC algorithm. The representation has variable size, and this size depends on both the number of training points, N, and the number of clusters in a given solution, M. Cluster membership is indicated by sequences of data point identifiers, while a separate sequence of integers represents cluster sizes. The k additional parameters at the end of the chromosome are needed in a modified form of the rule representation, described at the end of Section 3.2.

3.2 Genetic Operators for Data Clustering

Our initialization routine, the pseudocode for which is given in Figure 5, used subtractive clustering as an initial estimate of cluster membership for each proposed solution, based on a random value of the cluster radius. The GDC algorithm no longer used backpropagation, so it no longer required the learning rate and convergence criteria. The estimated cluster means were used to calculate the distance of each point from the estimated cluster centroids. Points were assigned to each cluster probabilistically, with the probability of assignment to a cluster inversely proportional to the square of distance between a point and the cluster centroid. Any given point had the greatest probability of being associated with the closest cluster centroid, but had some probability of being associated with any cluster. It may seem counter-intuitive to allow points to be associated with any cluster, but it is a sensible procedure when regions of overlap are considered. Data points in a region of overlapping data clusters are, of necessity, not associated with the closest cluster centroid. A population of proposed solutions

Comment: Could this be "are, of necessity, not associated" instead, or should it be saying "are not necessarily?"
initialized in this manner will tend to include varied estimations of the membership partition of the data points, each close to the correct solution, and each differing from each other primarily in the membership of questionable points from regions of overlap.

**Figure 5.** The pseudocode for the initialization algorithm. The algorithm estimates the cluster centers with subtractive clustering. Then, for each data point, the routine calculates the probabilities of that data point belonging to each cluster as inversely proportional to the square of the distance of that point from each cluster center. A single random number then selects the cluster to which the data point belongs.

**Perform subtractive clustering**
Repeat for all data points
  Select a data point
  Repeat for each cluster
    Calculate distance of selected data point from current estimated cluster mean
    square the distance
    Add squared distance to sum
  Repeat for each cluster
    Form probability of selected data point being bound to each cluster using:
    \[
    \text{probability}_c = (1 - \text{distance}_c^2/\text{sum of all distances}^2)(\# \text{ of clusters} - 1)
    \]
    Add probability to probability\textsubscript{\textit{n},1} and place in nth position of an array
  Generate one random number, compare to array, and assign point to selected cluster

To summarize:
  If \(0 < \text{random number} < \text{probability}_1\) then the point belongs to cluster 1
  If \(\text{probability}_1 < \text{random number} < \text{probability}_2\) then the point belongs to cluster 2
  
  ...
  
  If \(\text{probability}_n \leq \text{random number} < \text{probability}_n\) then the point belongs to cluster \(n\)

The formula yields \(\text{probability}_n = 1\)

The probability of a point being assigned to a cluster is inversely proportional to the square of the distance between the point and the cluster.

As we will discuss below, the probability of mutation was variable. Our initial mutation operator swapped data points between clusters of a given solution. Specifically, we select two clusters uniformly at random, then randomly choose a data point from each cluster with probability equal to the squared distance from the cluster centroid. We simply moved the first point into the second cluster with probability 0.15 changing the size of both clusters. While the value of 15% for this parameter was arbitrary, tests showed that the algorithm was not sensitive to this parameter. In our implementation, this process exchanges two data points from different fields in the first section of the chromosome.
The application of genetic data clustering to large data sets revealed weaknesses in both the initialization and mutation techniques. Initializing every member of the population with subtractive clustering is expensive and redundant. Each individual solution is based on a randomly determined cluster radius, and this causes the number of clusters and the exact cluster membership profiles in each solution to vary, but the effort is superfluous. It is much quicker to initialize one solution via subtractive clustering, and then clone and mutate each new solution in the initial population from the one that preceded it. The quicker procedure was adopted, as tests showed that neither produced significantly different results. The swap mutation operator does not vary the number of clusters, only the membership profiles of the clusters and cluster sizes. The crossover operator, however, described below, alters the number of clusters in the offspring it creates, such that the ultimate solution found by genetic data clustering is not artificially constrained by the nature of the initial solution.

The swap mutator described above was neither sufficiently focused nor “intelligent” enough to deal effectively with large data sets. Tests on the LANDSAT data indicated the classification error rate of the generated rules was about double that reported in the literature. Furthermore, fuzzy rules could be constructed on these data sets by a manual procedure (see Chapter 5) that gave error rates approaching those given in the literature, such that the problem was in the genetic algorithm, and not in the rule representation. Tests with population sizes up to 1000 members had shown that increasing population size did not significantly improve the results. Furthermore, early tests during initial algorithm development had shown considerable performance improvement with the addition of a swap mutation operator (to an algorithm involving only crossover). These tests had indicated that the mutation operator was largely responsible for finding new and improved solutions, so we focused our attention here. It was determined that while mutation was still occurring at a high rate as the error rate in the evolved rules leveled off, equal numbers of correctly classified points and incorrectly classified ones were being swapped out of their clusters. A kind of equilibrium was reached and no further progress was possible. The original method used only distance from the cluster center to choose which points to swap. New
information was needed to improve the equilibrium error rate. The fundamental problem, of course, is regions of overlap in the training data. When there are thousands of data points, the algorithm has to focus clearly on these regions. The natural solution is to use nearest neighbor information to determine probability distributions of a point being misclassified.

The simple data classification procedure known as “k nearest neighbors” determines the k “closest” data points in a training set of known classifications to any data point of unknown classification. The method uses a majority vote of these neighbors to classify any unknown data point. However, nearest neighbor information can also generate a probability distribution for cluster membership, which is useful in stochastic algorithms such as the GDC. This distribution can induce a variable mutation probability that allows the algorithm to focus on regions of greatest difficulty. In homogenous regions there is essentially zero probability of a point being misclassified, while in regions of overlap the probability of misclassification is defined by the proportions of the different classes in the region. If only data with integral classifications were considered this could be accomplished with a simple counting and averaging procedure over all classes represented in a region (a nearest neighbors type of calculation). But to accommodate continuous data values, we employed a procedure similar to the potential function of mountain clustering (Section 1.2). In our refined scheme, we use the misclassification probability, \( P_{\text{miss}} \), as one of two parameters that define a variable mutation rate.

\[
(9) \quad P_{\text{miss}} (d_i) = \left( 1 - \frac{1}{k} \sum_{j=1}^{k} \exp \left( - \frac{|c_i - c_j|}{sf} \right) \right) / b + r
\]

where

- \( d_i \) is the \( i^{th} \) data point,
- \( c_i, c_j \) are the classification values of the \( i^{th} \) point and its \( j^{th} \) nearest neighbor,
- \( sf \) is a significance factor (1 for integral values, 10% full scale for continuous values),
- \( k \) is the number of neighbors considered in a region,
- \( b \) is an arbitrary scaling factor (2 is a good value), and
- \( r \) is a small random value to support mutation (see below).
In a reasonably homogenous region, most of a point’s nearest neighbors will have very similar
classification values, the sum of exponential terms divided by $k$ will approach unity, and the
misclassification probability will go to zero (plus a small random value $r$). At the opposite extreme, an
isolated point distinct from its neighbors, the misclassification probability will go to about 0.5 (again, plus
$r$). The method has worked well on all data sets tried so far with $r$ set to be between 0.1 and 0.3. The
small random term is to ensure that all data points have some probability of being swapped to another
cluster, as any point might get put in the wrong cluster either by the initializer, the crossover operator, or
even by the mutation operator, and would need to be swapped elsewhere. In addition, the nearest
neighbor term in the probability is necessarily an estimate based on a finite neighborhood (and heuristic in
nature, as well) and should be hedged with a small random term.

Our refined algorithm also estimated a second probability as part of the variable mutation rate. The
second probability is called the retention probability, $P_{ret}$, and it estimates how well the classification of a
given data point agrees with the classification of the cluster it is part of:

$$ P_{ret}(d_i) = \exp\left( -\frac{|c_i - c_i^*|}{sf} \right) $$

where $c_i$ is the average classification value of the cluster, and $d_i$ is in the cluster $c_i$.

When $c_i$ and $c_i^*$ are similar (relative to $sf$) the retention probability approaches unity, and in the opposite
extreme will go to zero. Note that both probability calculations use the same “significance factor” to
normalize classification difference values, such that the exponential terms will have similar orders of
magnitude. To simplify “calibration” of the mutation operator, the adjustable terms were restricted to the
misclassification probability. Figure 6 shows both estimated probabilities as functions of “significance
units.” The misclassification probability in particular is a function of average significance units. A low
value indicates that the point in question is similar to most of its nearest neighbors, and a value near one
indicates that the point is about one significance unit away from most of its neighbors. We derived the
misclassification probability curve for one hypothetical situation, and it would vary in different particular situations. The main curve represents the nearest neighbor term and the error bar the maximum possible value attainable by the addition of the small random term \( r \). The retention probability curve is an absolute curve for any point that differs by some number of significance units from the cluster associated with it. Both curves may have distinct abscissa values during any particular calculation, such that the intersection does not have any particular significance. It is important that the retention probability may approach a value of unity, but the misclassification probability would have only a very slim chance of doing so.

Our algorithm induces a swap mutation when \( P_{\text{ref}} < P_{\text{miss}} \). Once a point becomes associated with the right cluster, it has essentially no chance of mutating again, although not every point will be so fortunate. At the other end of the scale, the algorithm will swap a point to another cluster when it is poorly classified.

**Figure 6.** Comparison of the probabilities of misclassification and retention as a function of normalized classification differences (significance units). The probability of misclassification curve is based on average significance units for a particular hypothetical situation, while the probability of retention curve is absolute and valid for any point and cluster.

The swap mutator steps through the chromosome testing each point in turn by comparing the probabilities described above and swapping the point in question to a randomly selected cluster if the point’s retention probability is less than its misclassification probability. The retention probability is dynamic and must be calculated at every step. While the misclassification probability involves some expensive calculations, it
is static (for a given value of $k$), calculated once only, and stored with each data point. While the algorithm may swap a point to an inappropriate cluster, future mutations will likely move each point to a reasonable cluster.

Data clustering requires a crossover method that, to the extent possible, picks whole clusters from two parents, resolves membership conflicts between clusters from different parents that possess a common data point, and ensures that the resulting child genome represents a valid and complete set of data points. Figure 7 outlines our algorithm, which we call cluster crossover. The algorithm randomly selects clusters from two parents and copies them into a child. With each cluster selection, the algorithm removes a cluster from the donor parent and the points of this cluster from the other parent. This algorithm ensures that only unused data points are available for inclusion in the child at each step, and that all data points are included in the child. We limited the least number of points allowed in a cluster because a cluster with only one data point has standard deviations of zero. This condition causes an error, since the standard deviations occur as divisors in the fuzzy membership functions. While there is little chance that problems with large data sets will produce single-point clusters, it is safest to establish a limit, and smaller data sets require this bound to prevent errors. We chose three points per cluster as a reasonable minimum. For small data sets, a minimum value of two points per cluster sometimes resulted in two-point clusters where at least one dimension in the data points possessed the same value. This situation resulted in a standard deviation of zero in that dimension, resulting in a divide-by-zero error.

Figure 7. Pseudocode for the cluster crossover operator.

Copy selected parents into temporary variables, as the process is destructive. Repeat until both parents are depleted or possess too few data points to form a cluster (a minimum cluster size of three data points is enforced throughout the algorithm).
BEGIN
  Randomly select a parent.
  Randomly select a cluster from this parent.
  Remove the cluster from the donor parent.
  Remove any points belonging to this cluster from the other parent.
  Add the cluster to the child.
  If the average number of points per cluster < the minimum cluster size then
    Consolidate points into a single cluster
END
Distribute left over points, if any, randomly among the existing clusters in the child.
Two factors are primary in designing an objective function for data clustering: *training set performance* and the *cluster validity index*. Training set performance is defined as the ratio of correctly classified training data points to the total number of training points. We use a separate test data set to verify the performance of the algorithm, but this set is not involved in the training procedure. While training set performance is a useful component of the objective function, the correct number of clusters is generally unknown, so optimizing only this parameter will tend to subdivide proper clusters and overfit of the training data. Therefore, the objective function should also include a cluster validity index. The Xie-Beni index evaluates the validity of fuzzy clusters, with a small index indicating better clustering than a large value, and has proven particularly useful in our application. Table 1 shows the results of tests performed to compare the performance of the objective function either with or without the Xie-Beni component. We repeated each of these tests 20 times. The case using the Xie-Beni index yielded considerably lower error rates than the case without the Xie-Beni index. The Iris data were used to test the objective function because the data set was small enough to perform numerous repetitions in a reasonable amount of time. Although the clusters are formed from crisp sets of data points, the formation of fuzzy rules allows them to be treated as fuzzy clusters.

**Table 1.** This table shows the results of tests performed to compare the performance of the objective function either with or without the Xie-Beni component. We repeated each of these tests 20 times. The case using the Xie-Beni index yielded considerably lower error rates than the case without the Xie-Beni index.

<table>
<thead>
<tr>
<th>Test</th>
<th>Minimum Error Rate (%)</th>
<th>Average Error Rate (%)</th>
<th>Maximum Error Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xie-Beni Component in objective function</td>
<td>2.7</td>
<td>5.1</td>
<td>11</td>
</tr>
<tr>
<td>No Xie-Beni Component in objective function</td>
<td>4.0</td>
<td>7.0</td>
<td>13</td>
</tr>
</tbody>
</table>

Our objective function combined these two metrics:
where \( c \) is a scaling constant that gave best performance at a value of 3. Figure 8 shows the results of tests on the Iris data with values for the scaling factor \( c \) set to 1, 2, 3, 4, 5, and 10. We repeated each of these tests 20 times. While the dependence is not strong, the lowest minimum, average, and maximum error rates were all obtained for the test with \( c \) set to three. We square the training performance score to intensify the difference between high and low scores.

To improve the accuracy of the generated rules, we modified the fuzzy rule membership function from equation 4 to contain a factor \( b \) in the denominator, as shown below:

\[
\mu_k(x_i) = \exp\left(-\frac{1}{2} \sum_{j=1}^{N} \frac{(x_{ij} - x_{kj})^2}{b\sigma_{ij}}\right)
\]

The factor improved the fit of the membership function to the data. The best setting for this parameter varied from one data set to the next, with the most common values being close to unity or 0.5. While this modification typically (about 75% of the time) improved performance by about 10%, tests performed on the LANDSAT satellite image data (described below) showed considerable degradation in performance as the number of components in each data point increased. Therefore, we gave each component in the data a distinct fit parameter, and the algorithm determined these genetically. We added extra genes to the genome to hold these parameters (see Figure 4), initialized randomly to the integer values 100 or 50. We also modified the swap mutator to randomly increment or decrement these parameters by 5, at a probability of mutation of 0.1, and we modified the cluster crossover operator to perform uniform crossover on this segment. The algorithm divided the parameter values by 100 to scale them back to the proper range before using them in the fuzzy matching strength calculation.
The results of these modifications are illustrated in Figure 9. The label “none” indicates that the fit parameter was not present, “fixed” means that the parameter was a constant (0.55, in this calculation) and the same for all components of the data, and “varied” means that each component of the data had distinct, genetically determined, fit parameters. Another way of stating these results is that the parameter $b$ makes the fitness function self-adaptive, and this considerably improves the algorithm.

**Figure 8.** The results of calculations on the Iris data to determine an appropriate value for the scaling factor $c$ (where $c = 1, 2, 3, 4, 5,$ and $10$). We repeated each test 20 times. While the dependence is not strong, we obtained the lowest minimum, average, and maximum error rates for the test with $c = 3$.

**Figure 9.** Comparison of % error rates between three calculations with slight variations in the rule representation form. The variations refer to the fit parameter in the denominator of the fuzzy membership functions. “None” indicates that the parameter was not present, “fixed” that the parameter was a constant (0.55, in this calculation) and the same for all components of the data, and “varied” that each component of the data had distinct fit parameters, with values that were genetically determined.
4. TEST PROCEDURES AND RESULTS

We tested the automatic rule generation algorithm on several data sets of varying size and discuss the results of those tests in this section. All of the data were preprocessed in one of two ways, by either centering and reducing the data to unit variance, or performing a principal components analysis on the data. To perform valid comparison between our method and others, we obtained and used two data sets available in the literature in preprocessed form. Our nondestructive assay data sets, which were not available in preprocessed form, were processed by principal components analysis. Table 2 summarizes the properties of the data sets that we used.

Table 2. Properties of the data sets used in our tests.

<table>
<thead>
<tr>
<th>Name</th>
<th># Features Used</th>
<th># Examples</th>
<th>Type of Preprocessing</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris Data</td>
<td>4</td>
<td>150</td>
<td>Centered &amp; Reduced</td>
<td>Small, but frequently used, classification data set</td>
</tr>
<tr>
<td>LANDSAT Image Data</td>
<td>4, 8</td>
<td>6000</td>
<td>Principal Components</td>
<td>Large classification data set</td>
</tr>
</tbody>
</table>

We used two data sets from a comparative study of neural network derived classification algorithms, the Enhanced Learning for Evolutive Neural Architecture (ELENA) project (Aviles-Cruz, et al. 1995). There are numerous links to the ELENA data on the web, such as the following University of Texas site: “http://lans.ece.utexas.edu/~lans/bench/top.html”. The ELENA data contained a number of large data sets with numeric feature values, making them appropriate for use in our algorithm. In addition, the data sets selected had comparison results available for varied numbers of data features. Thus, these data allowed us to assess the performance of our algorithm on data with a broad range of complexity. One data set that we used was a classification of Irises, the Anderson and Fisher Iris Data set, and the other was a collection of LANDSAT image data. The testing methodology used in the ELENA project (Blayo et al., 1995) was to partition the full data sets in half, and to use one half of the
data for training and the other half for testing. In order to reduce dependence of the results on any
particular choice of training data, we partitioned the data five separate times, and trained and tested the
methods being compared against each partition. The reported results, reproduced in the plots at the end of
this chapter, are in the form of classification errors averaged over the five different partitions of the data
set. We constructed confidence intervals (95%) constructed for the minimum and maximum error rates,
as in the ELENA project, to provide a statistical bound on the expected performance range of the GDC
algorithm on the comparison data sets. We used the following equation to calculate the confidence
intervals:

\[ E^* - 1.96 \sqrt{\frac{E^* \left(1 - E^*\right)}{N}} \leq E \leq E^* + 1.96 \sqrt{\frac{E^* \left(1 - E^*\right)}{N}} \]

where

- \( N \) is the number of training examples,
- \( E \) is the expected error rate,
- \( E^* \) is the apparent error rate (a specific error rate, such as the maximum or minimum error).

We duplicated these procedures in the comparison tests performed on selected data sets from the ELENA
project. In order to determine the error rates, we rounded the predicted integer-valued classifications to
the nearest integral value. Predicted real-valued classifications were considered correct if they were
within 10% (with respect to the full scale range of values that the classification variable could take) of
the correct answer.

Data sets used to test classification algorithms generally consist of data points labeled by some
integral set of values. It is a useful benchmark to treat these labeled data as already clustered (by
classification) and form fuzzy rules by taking the average and standard deviations of the data points in
these clusters. These rules represent the results that the GDC (Genetic Data Clustering) algorithm could
obtain with “perfect” clustering on a given data set. In other words, these results represent the order of
magnitude performance obtainable by any reasonable algorithm, given the form of the rule representation.
Any algorithm with significantly inferior results to these “ideal” results clearly needs improvement.
Likewise, to obtain significantly greater performance than these “ideal” results, one would have to address the form of the rule representation. The chosen rule representation – Gaussian membership functions – imposes a particular inductive bias on the solution to the problem. Hence the “ideal” solution is only ideal with respect to this inductive bias. Thus, as an approximate benchmark, these results help to indicate where researchers need focus. Variations in the different partitions cause these rule sets to vary in their performance, so these ideal rules will be presented in the ELENA format with average, minimum, and maximum holdout error rates. We will use these rule sets to indicate the attainable level of performance that should be possible with our system, for the case of a fixed uniform fit parameter value, and we will present them in our comparisons, where available.

Table 3 presents all of the genetic parameter settings used in the GDC calculations. We tested both the simple (non-overlapping populations) and steady-state genetic algorithms. Both gave nearly identical results in terms of maximum, minimum, and average scores over ten repetitions. We used the steady-state GA for the tests presented in this section, however, because the steady-state algorithm was the faster of the two by about 40%, for various levels of population replacement. We also examined the effect of the probability of crossover. Table 4 records the results of experiments on four constant values of the crossover probability, 0.6, 0.7, 0.8, and 0.9 on the Iris data. We repeated each test 20 times. On average, a constant value of 0.8 performed the best, and we used 0.8 in our tests, but the exact value used was probably not very significant. The population size, 100 for most of the data sets, is larger than in our initial work. Improvements in both hardware and algorithm have made the larger population size feasible. As discussed above, we used the cluster crossover and swap mutation operators for the GDC. A variety of standard scaling and selection functions (Goldberg, 1989) were tested, and the combination of linear scaling and tournament selection yielded the best performance. Tournament selection involved two individuals at a time, and the linear scaling technique used in GALib is the one due to Goldberg (Goldberg, 1989) with default parameter value of 1.2. We performed the tests on a Pentium II PC (450 MHz, 128 MB RAM).
Table 3. Genetic parameter settings used in the GDC calculations.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of Genetic algorithm</td>
<td>Steady-state GA</td>
</tr>
<tr>
<td>Replacement parameter</td>
<td>Population size – 1 (simulates elitism, but faster)</td>
</tr>
<tr>
<td>Probability of crossover</td>
<td>0.8</td>
</tr>
<tr>
<td>Population size</td>
<td>100 (except where otherwise stated)</td>
</tr>
<tr>
<td>Crossover operator</td>
<td>Cluster crossover, Section 4.2</td>
</tr>
<tr>
<td>Mutation operator</td>
<td>Swap mutation, Section 4.2</td>
</tr>
<tr>
<td>Selection operator</td>
<td>Tournament, 2 selections</td>
</tr>
<tr>
<td>Scaling operator</td>
<td>Linear, Goldberg, 1.2</td>
</tr>
<tr>
<td>Maximum generations</td>
<td>100 (smaller data sets) / 200 (larger data set)</td>
</tr>
</tbody>
</table>

Table 4. The results of tests to determine the optimal probability of crossover. The table records the results of experiments on four constant values of the crossover probability, 0.6, 0.7, 0.8, and 0.9 on the Iris data. We repeated each test 20 times. On average, a constant value of 0.8 performed the best, and we used 0.8 in our tests, but the exact value used was probably not very significant.

<table>
<thead>
<tr>
<th>Probability of Crossover</th>
<th>Minimum Error Rate (%)</th>
<th>Average Error Rate (%)</th>
<th>Maximum Error Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.6</td>
<td>2.7</td>
<td>6.1</td>
<td>11</td>
</tr>
<tr>
<td>0.7</td>
<td>2.7</td>
<td>6.9</td>
<td>12</td>
</tr>
<tr>
<td>0.8</td>
<td>2.7</td>
<td>5.7</td>
<td>13</td>
</tr>
<tr>
<td>0.9</td>
<td>2.7</td>
<td>6.1</td>
<td>13</td>
</tr>
</tbody>
</table>

4.1 The Anderson and Fisher Iris Data

The Iris data (Anderson, 1935 and Fisher, 1936) consists of three groups: Iris Setosa, Iris Versicolor, and Iris Virginica. Four features are included in the data: petal length and width, and sepal length and width. The Versicolor and Virginica groups are significantly overlapping, but the Setosa group is more nearly distinct, hence it is questionable whether one should identify two or three clusters in the Iris data. Figure 10 shows the scatter plot for the petal length and width features, as an example. These data were part of the ELENA comparative study and available in centered and reduced form. These data are frequently encountered in the classification literature and were chosen for their familiarity.
4.2 Satellite Image Data

The ELENA comparative study had among its larger data sets a satellite image data set. This data set was chosen both for its size and because image data is a potential application for the algorithm we are developing. The data set contains more than 6000 data points, where each point consists of pixel values over 4 spectral bands from a 3x3 region of the image, or 36 pixel intensities. Each point has associated with it a classification of the type of feature in the region. Table 5 shows these classifications. The data were available in principal component format, with all 36 components retained. We tested the GDC algorithm against this data using four and eight principal components.
Table 5. Classifications of the satellite image data set.

<table>
<thead>
<tr>
<th>Classification</th>
<th>Region Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Red soil</td>
</tr>
<tr>
<td>2</td>
<td>Cotton crop</td>
</tr>
<tr>
<td>3</td>
<td>Grey soil</td>
</tr>
<tr>
<td>4</td>
<td>Damp grey soil</td>
</tr>
<tr>
<td>5</td>
<td>Soil with vegetable stuble</td>
</tr>
<tr>
<td>6</td>
<td>Mixture (discarded as unreliable)</td>
</tr>
<tr>
<td>7</td>
<td>Very damp grey soil</td>
</tr>
</tbody>
</table>

4.3 Experimental Results

We first present comparisons to results obtained in the ELENA project, using the Iris and satellite image data sets. We compare rules produced from four variations of our algorithm to classifiers from the ELENA project with the lowest, highest, and average error rates:

- “Initial GDC” calculations use the initial genetic calculations with fixed and uniform fit parameter values to produce a rule set
- “Initial Ideal GDC” calculations are the so-called ideal rule sets, discussed above, with fixed and uniform values of the fit parameter (manually calculated)
- “Improved GDC” calculations use the genetic calculations with improved rule representation (variable and non-uniform fit parameter value) to produce a rule set
- “Improved Ideal GDC” calculations are the so-called ideal rule sets with the improved rule representation (fit parameter values taken from the “Improved GDC” calculation results, but the rules are manually calculated)

We also present results of GDC calculations performed on data sets with real-valued classification scores. These include three sets of neutron assay data. No formal comparison calculations were available on these data sets, so we adopted a simpler testing procedure than for the comparison calculations presented above. We repeated each calculation five times. The rule set producing the minimum error rate on each data set is considered the end product of the series of calculations. We also calculated the minimum, maximum, and average error rates, to indicate the degree of variation seen in smaller data sets.
4.3.1 Iris Data

We compared our GDC algorithm to three benchmark learning algorithms (in order of increasing error rate): the Gaussian Quadratic Classifier (GQC), the Iterative Radial Vector Quantization (IRVQ), and the Restricted Coulomb Energy (RCE) algorithms. The results from these algorithms span the range of results in the ELENA project for the Iris data, showing the best, worst and average performance. The fact that the GQC performs the best on the Iris data is an indicator that the data have a Gaussian distribution, and should also indicate that good performance is possible with the GDC (Gaussian membership functions are used). We performed our calculations with the fit parameter set to 1.0 and a neighborhood size of 25 for the “Initial GDC” calculation.

Figure 11 presents results obtained with the Iris data. The average error rate of the “Initial GDC” calculations agrees exactly with the “Initial Ideal GDC” value, and the “Improved GDC” calculation average value slightly exceeds them both. Hence, the genetic clustering procedure worked very well on the Iris data. Due to the close agreement in all variants of the GDC calculation on the Iris data, we did not perform the “Improved Ideal GDC” calculation. Furthermore, the GDC algorithm performed about 1% better than the IRVQ algorithm, or above average among the classification algorithms tested in the ELENA project. The GQC procedure slightly outperforms even the GDC ideal rule sets, hinting that one limitation of the current algorithm may be the form that the rules take.

4.3.2 LANDSAT Satellite Image Data

Each data point from the satellite image data has 36 fields – nine pixel intensities in 4 spectral bands. Processing these data with the principal components technique allows us to select only the most important components of the data as input to the system. As before, we selected for comparison the results with the lowest, average and highest error rates. These methods are, in order of increasing error rate, the k-Nearest Neighbors (kNN), the Gaussian Quadratic Classifier (GQC), and the Restricted Coulomb Energy (RCE) algorithms. We present the comparisons for two variants of the calculations: 1) calculations using the first four principal components of the data (referred to as the four component
calculation), and 2) calculations using the first eight principal components of the data (referred to as the eight component calculation). We performed our satellite image calculations with the fit parameter set to 0.5 for the fixed value, four component calculation, and 1.0 for the eight component calculation. We used a neighborhood size of 200 for both calculations. The four component calculations produced lower error rates for a population of 50 individuals than for a population size of 100, hence we present the results for the smaller population size. The four component calculations with both 50 and 100 individuals were performed out to 200 generations, and all calculations appeared to be stable. However, our calculations cannot rule out that the larger population would not have achieved the same results given more time. The eight component calculations performed better with a population of 100 individuals (again, within the constraints of 200 generations).

**Figure 11.** Comparison of % error rates for various classifiers studied in the ELENA project vs. the GDC algorithm on the Iris data. Holdout testing was used with 5 separate partitions into equally sized test and training sets. Sets of 3 data points represent the minimum, average and maximum error rates, respectively, over the 5 partitions. The error bars represent 95% confidence intervals on the minimum and maximum error rates. The GQC algorithm had the lowest error rate, the RCE the highest, and the IRVQ represents average results. Our GDC algorithm performs somewhat above average, and agrees well with the “Initial ideal GDC” performance.
Figures 12 and 13 present the error rate comparisons for the four component calculations. In both plots, sets of 3 data points represent the minimum, average and maximum error rates, respectively, over the 5 partitions. The error bars represent 95% confidence intervals on the minimum and maximum error rates. The fit parameter values for the “Improved Ideal GDC” calculation are taken, after the fact, from the rules found during the actual calculations. Figure 12 shows that for both the initial and improved representations the actual calculations significantly outperform the rule sets based on ideal clustering. The “Improved GDC” calculations achieved considerably lower error rates than the “Initial GDC”. Together these calculations indicate that the genetic procedure is working as well as the rule representation will allow, and even a little better. Figure 13 shows that the “Improved GDC” performed just below the average value seen in the ELENA project on the four component satellite image data.

**Figure 12.** Comparison of % error rates for actual and ideal rule sets and two variations of the rule representation on the satellite image data using the first 4 principal components. Holdout testing was used with 5 separate partitions into equally sized test and training sets. Sets of 3 data points represent the minimum, average and maximum error rates, respectively, over the 5 partitions. The error bars represent 95% confidence intervals on the minimum and maximum error rates. For both the initial and improved representations, the actual calculations significantly outperform the rule sets based on ideal clustering. The “Improved GDC” calculations achieved considerably lower error rates than the “Initial GDC” calculations.
Figures 14 and 15 present the error rate comparisons for the eight component calculations, and use the same conventions as in Figure 12 and 13. Figure 14 shows that the actual rule sets performed only slightly better than the ideal rule sets, whereas the “Improved GDC” calculations performed much better than the “Initial GDC calculations.” The fact that eight components are being used makes it more important to establish correct fit parameter values on each component since no single fit parameter is sufficient. Nonetheless, the performance of the eight component rule set slipped in comparison to the classifiers studied in the ELENA project. Figure 15 shows that the GDC algorithm performed just below the level of the RCE algorithm, when used with the first eight components of the data.

Figure 13. Comparison of % error rates for various classifiers studied in the ELENA project vs. the GDC algorithm on the satellite image data using the first 4 principal components. Holdout testing was used with 5 separate partitions into equally sized test and training sets. Sets of 3 data points represent the minimum, average and maximum error rates, respectively, over the 5 partitions. The error bars represent 95% confidence intervals on the minimum and maximum error rates. The kNN algorithm had the lowest error rate, the RCE the highest, and the GQC represents average results. Our GDC algorithm achieved slightly below average performance on the satellite image data.
Figure 14. Comparison of % error rates for actual and ideal rule sets and two variations of the rule representation on the satellite image data using the first 8 principal components. Holdout testing was used with 5 separate partitions into equally sized test and training sets. Sets of 3 data points represent the minimum, average and maximum error rates, respectively, over the 5 partitions. The error bars represent 95% confidence intervals on the minimum and maximum error rates. For both the initial and improved representations, the actual calculations slightly outperform the rule sets based on ideal clustering. The “Improved GDC” calculations achieved considerably lower error rates than the “Initial GDC” calculations.

Figure 15. Comparison of % error rates for various classifiers studied in the ELENA project vs. the GDC algorithm on the satellite image data using the first 8 principal components. Holdout testing was used with 5 separate partitions into equally sized test and training sets. Sets of 3 data points represent the minimum, average and maximum error rates, respectively, over the 5 partitions. The error bars represent 95% confidence intervals on the minimum and maximum error rates. The kNN algorithm had the lowest error rate, the RCE the highest, and the GQC represents average results. Our GDC algorithm performed just below the level of the RCE algorithm on the satellite image data.
5. CONCLUSIONS AND FUTURE DIRECTIONS

We have presented a new method for automatically generating expert system rules from data. Innovations include a genetic data clustering algorithm that involves crossover and mutation operators specifically designed for the task of data clustering. In addition, we used self-adaptive techniques, including variable length chromosomes and a variable probability of mutation.

The primary component of the genetic data clustering algorithm is the swap mutation operator that uses a pair of heuristically calculated probabilities to select data points to swap from one cluster to another. These probabilities are based on nearest neighbor calculations and cluster average classification values, and help to focus the attention of the mutation operator on data points most likely to be found in regions of overlap in the training data.

The cluster crossover operator, to the degree possible, acts to contribute whole clusters from each parent to the child solution under construction. Thus, the GDC algorithm maintains appropriate building blocks under cluster crossover.

Tests have shown that performance of the GDC algorithm is mixed. Specifically:

- We used the Fisher Iris data, and LANDSAT satellite image data to compute ideal classification rules by a simple manual procedure. We used these rules to show that the GDC algorithm computes rule sets that are as good as the rule representation format will allow.

- An improvement to the rule representation format considerably reduced the error rate of the GDC calculations on the satellite image data, and to a much lesser extent on the Iris data. This observation confirms that a primary limitation to performance of the GDC algorithm is the rule representation format.
Comparison of our results on the above mentioned data sets with results from the ELENA project indicate that classification rules generated by our method achieve about average results, but also indicate that the rule representation, not the performance of the GDC algorithm, is the primary limitation to performance.

It is notable that both the GQC and GDC algorithms work well on the Iris data, as alluded to previously, and likewise both do less well on the satellite image data. Both methods use Gaussian functions, indicating a similar inductive bias at work in these methods. The satellite image data is likely less amenable to classification by Gaussian functions. Future researchers should therefore investigate different rule representations to see what different results they obtain with different inductive biases. Geographic data, like the satellite image data, is possibly fractal in nature, and this consideration might lead to a whole different range of basis functions for rule representation.

Interested researchers could experiment with self-adaptive behavior by making the scaling factor in the probability of misclassification a genetic value of the representation. After experimenting with a few constant values, we set this parameter to a constant value of two. However, what may be more valuable than determining an appropriate constant value for this parameter is seeing if varying this value as the problem evolves is of benefit. Evolving this parameter will create a dynamic relationship between the two probabilities that trigger the occurrence of a mutation, possibly increasing the beneficial effects of swap mutation.

The interested researcher may wish to explore what benefits, if any, may accrue from applying set theoretic estimation techniques (Combettes, 1993) in combination with genetic algorithms. The goal of set theoretic estimation is to combine observed data and a priori knowledge and produce only those solutions that are consistent with all available knowledge. One suggestion is to explore how set theoretic estimation might be incorporated in the objective function.
Although we had intended to examine larger feature sets than the first four and eight principal components of the satellite image data, it became clear that other issues need be addressed first. The classifier algorithms tested in the ELENA project tended to show better performance when using eight principal components than four, whereas the performance of GDC degraded during this same transition. It would appear more profitable to pursue other improvements to the algorithm before experimenting with increased feature sets on the satellite image data.

One final suggestion for future work is that the neutron assay data sets could be classified with one or more of the algorithms studied in the ELENA project. This would generate performance data against which the GDC algorithm’s performance on neutron assay data could be compared.
6. REFERENCES


