ON THE USE OF EVOLUTIONARY ALGORITHMS IN DATA MINING[1]

Erick Cantú-Paz and Chandrika Kamath
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
7000 East Avenue, Livermore, CA 94550
cantupaz,kamath2@llnl.gov

ABSTRACT

With computers becoming more pervasive, disks becoming cheaper, and sensors becoming ubiquitous, we are collecting data at an ever-increasing pace. However, it is far easier to collect the data than to extract useful information from it. Sophisticated techniques, such as those developed in the multi-disciplinary field of data mining, are increasingly being applied to the analysis of these datasets in commercial and scientific domains. As the problems become larger and more complex, researchers are turning to heuristic techniques to complement existing approaches. This survey paper examines the role that evolutionary algorithms (EAs) can play in various stages of data mining. We consider data mining as the end-to-end process of finding patterns starting with raw data. The paper focuses on the topics of feature extraction, feature selection, classification, and clustering, and surveys the state of the art in the application of evolutionary algorithms to these areas. We examine the use of evolutionary algorithms both in isolation and in combination with other algorithms including neural networks, and decision trees. The paper concludes with a summary of open research problems and opportunities for the future.

INTRODUCTION

Data mining is increasingly being accepted as a viable means of analyzing massive data sets. With commercial and scientific datasets approaching the terabyte and even petabyte range, it is no longer possible to manually find useful information in this data. As the semi-automated techniques of data mining are applied in various domains, it is becoming clear that methods from statistics, artificial intelligence, optimization, etc., that comprise data mining, are no longer sufficient to address this problem of data overload. Often, the data is noisy and has a high level of uncertainty. It could also be dynamic, with the patterns in the data evolving in space and time. To address these aspects of data analysis, we need to incorporate heuristic techniques to complement the existing approaches.

In this paper, we survey the role that one category of heuristic algorithms, namely, evolutionary algorithms (EAs), plays in the various steps of the data mining process. After a brief definition of both the data mining process and evolutionary algorithms, we focus on the many ways in which these algorithms are being used in data mining. This survey is by no means exhaustive. Rather, it is meant to illustrate the diverse ways in which the power of evolutionary algorithms can be used to improve the techniques being applied to the analysis of massive data sets.

[1] Accepted for publication in "Data Mining: A Heuristic Approach", H. A. Abbass, R.A. Sarker, and C. S Newton, Eds., Idea Group Publishing, 2001.

Following a survey of current work in the use of EAs for data mining tasks such as feature extraction, feature selection, classification, and clustering, we describe some challenges encountered in applying these techniques. We conclude with the exciting opportunities that await future researchers in the field.

AN OVERVIEW OF DATA MINING

Data mining is a process concerned with uncovering patterns, associations, anomalies and statistically significant structures in data (Fayyad et al., 1996). It typically refers to the case where the data is too large or too complex to allow either a manual analysis or analysis by means of simple queries. Data mining consists of two main steps, data pre-processing, during which relevant high-level features or attributes are extracted from the low level data, and pattern recognition, in which a pattern in the data is recognized using these features (Figure 1.). Pre-processing the data is often a time-consuming, yet critical, first step. To ensure the success of the data-mining process, it is important that the features extracted from the data are relevant to the problem and representative of the data.

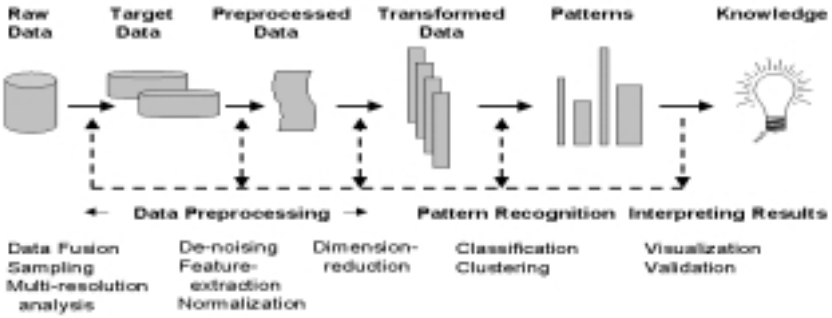

Figure 1: Data Mining - an iterative and interactive process

Depending on the type of data being mined, the pre-processing step may consist of several sub-tasks. If the raw data is very large, we could use sampling and work with fewer instances, or use multi-resolution techniques and work with data at a coarser resolution. Next, noise in the data is removed to the extent possible, and relevant features are extracted. In some cases, where data from different sources or sensors are available, data fusion may be required to allow the miner to exploit all the data available for a problem. At the end of this first step, we have a feature vector for each data instance. Depending on the problem and the data, we may need to reduce the number of features using feature selection or dimension reduction techniques such as principal component analysis (PCA) (Jackson 1991) or its non-linear versions. After this pre-processing, the data is ready for the detection of patterns through the use of algorithms such as classification, clustering, regression, etc. These patterns are then displayed to the user for validation. Data mining is an iterative and interactive process. The output of any step, or feedback from the domain experts, could result in an iterative refinement of any, or all, of the sub-tasks.

While there is some debate about the exact definition of data mining (Kamath 2001), most practitioners and proponents agree that data mining is a multi-disciplinary field, borrowing ideas from machine learning and artificial intelligence, statistics, high performance computing, signal and image processing, mathematical optimization, pattern recognition, etc. What is new is the confluence of the mature offshoots of these technologies at a time when we can exploit them for the analysis of massive data sets. As data mining has been applied to new problem domains, this technology mix has grown as well. For example, the growth of the Internet and the World Wide Web has resulted in tasks such as clustering text documents, multi-media searches, or mining a user's web surfing patterns to predict what page they are likely to visit next or to target the advertising on a web page. This has added natural language processing and privacy issues to the technological mix that comprises data mining.

Data mining techniques are being applied for the analysis of data in a variety of fields including remote sensing, bio-informatics, medical imaging, astronomy, web mining, text mining, customer relationship management, and market-basket analysis. While much of the focus in the data mining process tends to be on pattern recognition algorithms, the data pre-processing steps are more influential in the success of the data-mining endeavor (Langley and Simon, 1995; Burl et al., 1998). Unfortunately, the pre-processing steps often depend on the domain and problem. As a result, given the space limitations of this chapter, any discussion of the role of evolutionary algorithms in data pre-processing is likely to be limited in scope. Rather than ignore this important subject altogether, we will discuss aspects of this subject that are broadly applicable to several problem domains.

AN OVERVIEW OF EVOLUTIONARY ALGORITHMS

Evolutionary algorithms are randomized search procedures inspired by the mechanics of genetics and natural selection. EAs are often used as optimization algorithms, and this is the role that they play in most data mining applications. EAs work on a population of individuals that represent possible solutions to a problem in their chromosomes. Each individual can be as simple as a string of zeroes and ones, or as complex as a computer program. The initial population of individuals may be created entirely at random, or some knowledge about previously known solutions may be used to seed the population. The algorithm evaluates the individuals to determine how well they solve the problem at hand with an objective function, which is unique to each problem and must be supplied by the user. The individuals with better performance are selected to serve as parents of the next generation. Evolutionary algorithms create new individuals using simple randomized operators that are similar to sexual recombination and mutation in natural organisms. The new solutions are evaluated, and the cycle of selection and creation of new individuals is repeated until a satisfactory solution is found or a predetermined time limit has elapsed.

There are several major types of evolutionary algorithms: genetic algorithms (GAs), genetic programming (GP), evolution strategies (ES), and evolutionary programming (EP). All

evolutionary algorithms share the same basic concepts, but differ in the way they encode the solutions and on the operators they use to create the next generation.

Evolutionary algorithms are controlled by several inputs, such as the size of the population, and the rates that control how often mutation and crossover are used. In general, there is no guarantee that the evolutionary algorithm will find the optimal solution to an arbitrary problem, but a careful manipulation of the inputs and choosing a representation that is adequate to the problem increase the chances of success.

There are many ways to encode a potential solution as a chromosome, and there are many variations of selection methods, crossover, and mutation operators. Some of these choices are better suited to a particular problem than others, and no single choice is the best for all problems. Traditionally, genetic algorithms use chromosomes composed of zeroes and ones, but other encodings may be more natural to the problem and may facilitate the search for good solutions. Genetic programming encodes solutions as computer programs. ES and EP use floating-point numbers, which may be more suitable for function optimization problems where the parameters to optimize are real numbers, but may be an awkward match to a problem of finding the shortest route between multiple cities.

The choice of encoding is related to the operators that are used to produce new solutions from the selected ones. The simplest operator is mutation, and it acts by randomly changing a short piece of the chromosome. For example, when applied to strings of binary digits, it randomly chooses a location in the chromosome of an individual and flips a bit from zero to one or vice-versa. ES and EP use more sophisticated mutation operators.

Taking a cue from nature, genetic algorithms do not use mutation very often. The primary mechanism in GAs to create new individuals is crossover. In its simplest form, crossover randomly chooses two individuals from the pool that were selected to be parents, and exchanges segments of their two chromosomes around a single randomly-chosen point. The result is two new individuals, each with a segment of chromosome from each parent. Other variants of crossover exchange material around more than one point, and some researchers have experimented with recombining chromosomes from more than two parents. Some of the new solutions will be more fit than the parents, but others will be less fit. Evolutionary algorithms cannot avoid creating solutions that turn out to be unfit, but the selection process eliminates the bad solutions and keeps the best.

The selection of the parents can occur in many ways, but all selection methods have the same objective of preserving good individuals and discarding the less fit ones. Roughly, there are two kinds of selection: hard and soft. Soft selection methods assign to each individual a probability of survival based on their fitness, so that individuals with high fitness are more likely to be selected than individuals with low fitness. The soft selection methods then use the probabilities to select the parents. The hard methods do not involve any probabilities; they choose deterministically a fixed number of the best solutions available.

THE ROLE OF EVOLUTIONARY ALGORITHMS IN DATA MINING

After the brief overview of data mining and evolutionary algorithms, we next discuss the important role these algorithms can play in the various steps of data mining. In the following sections, we discuss how evolutionary algorithms can be used to improve the robustness and accuracy of the more traditional techniques used in feature extraction, feature selection, classification, and clustering.

In our survey, we view data mining as a multi-step process, focusing on the role that EAs can play in each step. However, we would be remiss if we did not include the work of those authors who blur the separation between the different steps, and use EAs to perform data mining as a whole on the input data. For example, in an early paper, Tackett (1993) identifies targets in a cluttered image by combining simple features extracted from the segmented image through linear and non-linear operations. If the resulting single value at the root of the tree is greater than zero, the object is classified as a target. Stanhope and Daida (1998) use a similar approach in their work on target classification using Synthetic Aperture Radar (SAR) images. Sherrah, Bogner, and Bouzerdoum (1996) also use non-linear pre-processing functions to create new features from primitive features. In addition, they associate one of three simple classifiers with each individual. The objective function is to minimize the number of errors made by each individual (a parse tree + a classifier) on the training data, with smaller trees being favored as a tie-breaker. In the process, the classifier is selected automatically.

EVOLUTIONARY ALGORITHMS IN FEATURE EXTRACTION

The process of extracting features that are relevant to the problem being addressed in data mining is very problem- and data-dependent. In some types of data, the features are relatively easy to identify. For example, in text data, the features are the words in the text, and in market basket analysis, the features are the items bought in a transaction. In each case, some processing of these raw features may be required. In text mining, words that do not represent the content of the text (e.g., articles) are removed and stemming of words performed so that similar words such as “computers” and “computing” are not considered as different (Frakes and Baeza-Yates, 1992). In market-basket analysis, we may need to convert the units so that all items bought by weight are measured in ounces.

While some types of data lend themselves easily to feature extraction, this task is more difficult in other cases. A typical example is image data, where feature extraction is far more challenging. In the past, image data was restricted to a few domains such as astronomy and remote sensing; however, it is now becoming more pervasive. With data mining being applied to domains such as medical imaging, multi-media on the web, and video images, it is important that we have robust techniques to identify features representing an image. Since images tend to vary widely, even within a domain, the adaptive nature of evolutionary algorithms can be exploited very effectively to address this important and difficult problem of feature extraction in image data.

An image is a rectangular array of pixels, where each pixel has either a gray-scale value, or a real value representing some physical quantity. In image mining, the first task is to identify an object in the image, followed by extraction of features that represent the object. Object identification is

often the more difficult of these two tasks, as it involves the conversion of the low-level representation (i.e., pixels) into a higher-level representation (i.e., objects). It is here that evolutionary algorithms can be used very efficiently and effectively. Two techniques that are traditionally used to identify an object in an image are *segmentation*, where the image is separated into several regions based on some desired criteria, and *edge detection*, where edges or contours in an image are identified (Weeks, 1996).

Several authors have exploited the use of evolutionary algorithms for image segmentation to deal with large and complex search spaces where limited information is available about the objective function. As Bhanu, Lee, and Ming (1995) point out, a key challenge in image segmentation is that most algorithms require the selection of several control parameters for optimal performance. This results in a high-dimensional search space, where the interactions between the parameters are complex and non-linear. Further, variations between images could cause the objective function representing the quality of segmentation to vary from one image to another. The problem is worsened by the fact that there is no single, universally accepted measure of the quality of the segmented image. To address these problems, Bhanu and Lee (1994) have explored the use of genetic algorithms to adaptively find the optimal set of control parameters for the Phoenix segmentation algorithm. The genetic algorithm selects an initial set of parameters based on the statistics of an image along with the conditions under which the image was obtained (time of day, cloud cover, etc.). The performance is evaluated using multiple measures of segmentation quality that include both global characteristics of the image and local features of the object. The system is adaptive as a global population of images, their associated characteristics, and the optimal control parameters, is maintained and used to seed the population each time a new image is analyzed. This global population is also constantly updated with higher strength individuals. Using scene images, Bhanu, Lee, and Ming (1995) show that their approach provides high quality results in a minimal number of cycles.

Another approach to segmentation using genetic algorithms is the work done in three-dimensional medical imaging by Cagnoni, Dobrzeniecki, Poli, and Yanch (1997). They too observe that the extreme variability of the features in biological structures causes the solutions generated by general-purpose algorithms to be unacceptable. As a result, some degree of adaptivity is required when segmenting medical images. Their approach identifies the contours of an object by first identifying the edge points using a filter whose parameters are optimized by a GA. These edge points are then used to seed an interpolation process, where the interpolation parameters are also generated by a GA. The fitness function is proportional to the degree of similarity between the contours generated by the GA and the contours identified in manually generated training examples. These filter and interpolation parameters are obtained for each new class of problems. Results on three-dimensional MRI images show that the GA–based techniques are insensitive to significant changes in shape across a sequence of images as well as the inter- and intra-slice variability in the contours, thus illustrating the power of these techniques.

The task of edge detection can also benefit from the use of evolutionary algorithms. Most edge detectors use simple first- and second-order derivatives to identify an edge. However, these operators are sensitive to noise and are not very general. In addition, they identify a pixel as an edge pixel based on the response of the edge detector at that pixel, ignoring the edge structure around the pixel. To overcome this disadvantage, several authors, including Tan, Gelfand, and

Delp (1989) and Bhandarkar, Zhang, and Potter (1994) have proposed an approach based on cost minimization, where the cost takes into account issues such as local edge structure, continuity of the edge, and fragmentation. This lends itself very naturally to the use of genetic algorithms for minimizing the cost. Bhandarkar et al. (1994), first define edge pixels as those that satisfy certain constraints, and then define the corresponding cost functions based on the local edge structure. Since the data is an image, the most natural representation of a chromosome is a two dimensional sequence of zeroes and ones, where an edge pixel is a one, and a non-edge pixel is a zero. The crossover operator is defined in two dimensions, with two-dimensional sub-images swapped between individuals. Their results show that both simulated annealing and an integrated GA (which includes elitism, intelligent mutation etc.) are better at detecting edges than a local search or a simple GA for both noisy and noise-free images.

This idea of using evolutionary algorithms to find an optimal set of parameters has also been used for image registration, where points in one image are mapped to corresponding points in another image of the same scene taken under different conditions. For example, Mandava, Fitzpatrick, and Pickens (1989) use GAs to find the parameters of a non-linear transformation that warps the four corners of one sub-image and maps them to another sub-image. To reduce the time, the quality of the transformation is evaluated using only a select sample of pixels in the sub-image.

In addition to genetic algorithms, several authors have used genetic programming to address image-processing problems. In particular, GP is often used for constructing image-processing operators for specific tasks. The idea is to start with a set of basic primitive functions such as a median filter applied to an image or the square of an image, and use GP to create a new operation. The fitness of the parse tree is usually evaluated by comparison with training examples, where the task to be achieved has been performed manually. Ebner and Zell (1999) describe how this approach can be used to measure optical flow, which requires the establishment of corresponding points between one image and the next. Brumby et al. (1999) use a similar approach for finding open water, such as rivers and lakes, amidst vegetation in remote sensing images. Their approach implements several checks to reduce unnecessary computation, and also gives credit for finding the anti-feature, that is, everything but the open water. Poli (1996) illustrates how GP can be used to find effective filters for medical images. He considers several ways of specifying the fitness function to account for the fact that any algorithm that uses filters for tasks such as image segmentation will give rise to false positives and false negatives. Depending on the application, the fitness function could assign weights to each, thus emphasizing appropriately the costs associated with either the false positives or the false negatives.

EVOLUTIONARY ALGORITHMS IN FEATURE SELECTION

Once the relevant features representing the data items have been extracted, it is often helpful to reduce this set of features. There are several reasons for this. In many situations, it is not possible to know a priori which features extracted from the data will be relevant to the problem at hand. Including features that are irrelevant not only increases the time complexity of many algorithms, but also increases the time needed to extract the features. Further, as the number of examples

needed for learning a concept is proportional to the dimension of the feature space, fewer training examples will be required if the number of features is reduced. In addition, some features may have costs or risks associated with them, and these should be weighted accordingly during the process of data mining. This leads to the problem of feature subset selection which is the task of identifying and selecting a useful subset of features to be used to represent patterns from a larger set of often mutually redundant, possible irrelevant, features with different associated measurement costs and risks (Yang and Honavar, 1997). Note that we use the term feature to indicate the attributes that represent an object or a data instance — these may be obtained directly from the original data, or derived by processing the original data.

The simplest way to remove irrelevant features is to apply domain knowledge. For example, if we are interested in clustering text documents, it is obvious that articles, such as "a", "an", and "the" are irrelevant variables (Frakes and Baeza-Yates, 1992). However, this approach is feasible only when a domain scientist can easily identify irrelevant attributes, which is rarely the case. More complex techniques such as principal component analysis can also be used to obtain linear combinations of attributes by projecting them along the directions of the greatest variance. We next discuss the ways in which evolutionary algorithms can be used to address the problem of feature selection.

The evolutionary approach most often used for feature selection is to combine the selection with the learning algorithm, in what is referred to as the wrapper approach. In this approach, the fitness of the feature subsets obtained during the evolutionary computation is evaluated using the learning algorithm itself. While this is more computationally intensive than selecting the features independent of the learning algorithm, it preserves any inductive and representational biases of the learning algorithm. Early work by Siedlecki and Sklansky (1989) with genetic algorithms identified an individual in the population as a series of zeros and ones, where a one indicated that a feature was included in the classification, and a zero indicated that it was not. The k-nearest-neighbor algorithm was chosen to evaluate how good each individual was based on its classification accuracy and the number of the features (i.e. ones) used. Others have applied the same basic binary encoding to select features in classification problems using neural networks (Brill, Brown, & Martin, 1990; Brotherton & Simpson, 1995)

Punch et al. (1993) extended the simple binary feature selection idea by representing an individual by a series of weights between zero and ten, thus weighting some features as more important than others. They found that their extension appeared to work better than the zero/one approach of Siedlecki and Sklansky (1989) on noisy real world datasets. Vafaie and DeJong (1998) also investigated a similar approach to feature selection using decision trees for classification. However, in their work, instead of just weighting each feature, they allowed the combination of existing features to form new features through simple operations such as add, subtract, multiply, and divide. This adaptive feature-space transformation led to a significant reduction in the number of features and improved the classification accuracy. Other related work in this area is that of Yang and Honavar (1997) who used neural networks as the classifier and a simple zero/one strategy for weighting each feature.

A very different use of genetic algorithms in feature selection is in the generation of ensembles of classifiers. Recent work by several authors (see, for example, Dietterich 2000) has shown that

it is possible to improve classification accuracy by combining the prediction of multiple classifiers. These ensembles of classifiers differ in the ways in which the classifiers are generated and their results are combined. Early work of Ho (1998), which used a random selection of features to create an ensemble, was extended by Guerra-Salcedo and Whitley (1999). They replaced the random selection with a more intelligent approach using genetic algorithms, and showed empirically that their idea was more accurate.

EVOLUTIONARY ALGORITHMS IN CLASSIFICATION

In this section, we describe how evolutionary algorithms can be used in conjunction with classification algorithms such as rule-based systems, neural networks, and decision trees.

Rule-Based Systems

Representing concepts as sets of rules has long been popular in machine learning, because, among other properties, rules are easy to represent and humans can interpret them easily. In EAs there are two main ways to represent rule sets. In the "Michigan" approach (Holland, 1975; Booker, Goldberg and Holland, 1989), each individual in the population represents one fixed-length rule, and the entire population represents the target concept. In contrast, in the "Pittsburgh" approach (Smith, 1980, 1983; DeJong, Spears, Gordon, 1993) each variable-sized individual represents an entire set of rules. The two representations have their merits and drawbacks and have been used successfully in classifier systems, which are rule-based systems that combine reinforcement learning and evolutionary algorithms.

The basic loop in a classifier system is that the system is presented with inputs from the environment, the inputs are transformed into messages that are added into a message list, and the strongest rules that match any message in the list are fired (possibly adding more messages to the list or acting on the environment). Rules are assigned a fitness value based on a reward returned by the environment. A genetic algorithm is used as the discovery component of the system, creating new rules based on the current best.

This is not the place to describe classic classifier systems or their relatives in detail. The interested reader should consult the book by Goldberg (1989) for a good introduction to classic CS, or the papers by Wilson (1995; 2000a) that describe some extensions. Wilson and Goldberg (1989) present an early critical review of classifier systems, and Wilson (2000b) presents a summary and outlook of research on XCS.

Classifier systems are commonly used as control systems in changing or uncertain environments, where there may not be sufficient or clear expert knowledge to produce a more conventional control (e.g., Goldberg, 1983). Closer to our interests in data mining, classifier systems have been used to learn Boolean functions (Wilson, 1995), which are of significance because they illustrate the ability of the system to learn complex non-linear concepts. Other applications include the classification of letters (Frey, 1991), and breast cancer diagnosis (Wilson, 2000a).

In classifier systems, the left side of rules is a conjunctive expression. This limits the descriptive power of the rules compared to, for example, first-order logic statements. First-order logic is important because it permits to express relationships between entities in databases. As Augier et al. (1995) noted, most of the machine learning algorithms that use first-order logic discover new rules using deterministic or heuristic approaches that can get trapped in local optima. To address this problem one can try to use EAs. A critical problem is to represent the rules, so that the evolutionary operators can act on them effectively and produce rules that make sense. Giordana and Neri (1995) proposed to use a user-defined template to specify the predicates. The EA finds the specific values that will be used in the rules. Their scheme has the advantage that the EA does not require modifications, because chromosomes are of fixed length and all combinations form valid rules. They also proposed two specialized crossover operators that are designed to promote specialization and generalization.

Another advantage of Giordana and Neri's system is also one of its main disadvantages: the dependence on the user to supply a template for the rules. Although this permits the incorporation of domain knowledge into the algorithm, the user must have a rough idea of the desired result. Augier et al. (1995) proposed an algorithm that addresses this issue by manipulating both the predicates and their values. The algorithm begins with a single rule that matches a single example. Specialized evolutionary operators modify the rule and create offspring that are added to the population until a limit is reached. The best rule after the execution of the EA is selected to form part of the final rule set, and the examples covered by the rule are deleted from the training set. The algorithm is repeated until there are no examples left.

Evolutionary Algorithms and Neural Networks

Genetic algorithms and artificial neural networks (ANNs) have been used together in two major ways. First, EAs have been used to train or to aid in the training of ANNs. In particular, EAs have been used to search for the weights of the network, to search for appropriate learning parameters, or to reduce the size of the training set by selecting the most relevant features. The second major type of collaboration has been to use EAs to design the structure of the network. The structure largely determines the efficiency of the network and the problems that it can solve. It is well known that to solve non-linearly separable problems, the network must have at least one layer between the inputs and outputs; but determining the number and the size of the hidden layers is mostly a matter of trial and error. EAs have been used to search for these parameters, as well as for the pattern of connections and for developmental instructions for the network. The interested reader may consult the reviews by Branke (1995), Whitley (1995) or Yao (1999).

Training an ANN is an optimization task with the goal of finding a set of weights that minimizes some error measure. The search space has many dimensions and it is likely to contain multiple local optima. Some traditional network training algorithms, such as backpropagation, use some form of gradient search, and may get trapped in local optima. In contrast, EAs do not use any gradient information, and are likely to avoid getting trapped in a local optimum by sampling simultaneously multiple regions of the space.

A straightforward combination of evolutionary algorithms and neural networks is to use the EAs to search for weights that make the network perform as desired. In this approach, each individual in the EA is a vector with all the weights of the network. Assessing the fitness of each network involves measuring the accuracy of classification or regression on the training set, so for each fitness evaluation, the training set is passed through the network. This can be inefficient if the training set is large, but the fitness may be estimated using a sample of the training set. Although the fitness would change over different samples, EAs are known to search well using such noisy evaluations.

There are three main variants of the training method:

- Start from a random population and use the weights found by the EA in the network without any further refinement (Caudell and Dolan,1989; Montana and Davis, 1989; Whitley and Hanson, 1989). This method may be particularly useful when the activation function of the neurons is non-differentiable.

- Use backpropagation or other methods to refine the weights found by the EA (Kitano, 1990; Skinner & Broughton, 1995). The motivation for this approach is that EAs quickly identify promising regions of the search space, but they do not fine-tune parameters very fast. So, EAs are used to find a promising set of initial weights from which a gradient-based method can quickly reach an optimum. This involves additional passes through the training data (for each epoch of backpropagation, for example), extending the processing time per individual, but sometimes the overall training time can be reduced because fewer individuals may need to be processed.

- Use the EA to refine results found by a NN learning algorithm. Although EAs do not refine solutions very fast, there have been some attempts to seed the initial population of the EA with solutions found with backpropagation (Kadaba & Nygard, 1990).

These approaches suffer from several problems. First, the length of the individuals grows rapidly with the size of the network. Since adjacent layers in a network are usually fully connected, the total number of weights that need to be represented is $O(n^2)$ (where n is the number of neurons). Longer individuals usually require larger populations, which in turn result in higher computational costs. For small networks, EAs can be used to search for good weights efficiently, but this method may not scale up to larger networks.

Another drawback is the so-called permutations problem (Radcliffe, 1990). The problem is that if the order of the hidden nodes is permuted, the representation of the weights would be different, so functionally equivalent networks can be represented in various ways. Some orderings may not be very suitable for EAs that use recombination because it might disrupt some favorable combinations of weights. To ameliorate this problem, Thierens et al. (1991) suggest that incoming and outgoing weights of a hidden node should be encoded next to each other. Hancock (1992) has done some analysis that suggests that the permutation problem is not as hard as it is often presented. Later, Thierens (1995) presented an encoding that completely avoids the permutations problem.

There are two basic approaches to using EAs to design the topology of an ANN: use a direct encoding to specify each connection of the network or evolve an indirect specification of the connectivity. The resulting network may be trained with a traditional learning algorithm (e.g., backpropagation), or the EA may be used to search the configuration and the weights simultaneously.

The key idea behind direct encodings is that a neural network may be regarded as a directed graph where each node represents a neuron and each edge is a connection. A common method of representing directed graphs is with a binary connectivity matrix: the (i, j)-th element of the matrix is one if there is an edge between nodes i and j, and zero otherwise. The connectivity matrix can be represented in the EA simply by concatenating its rows or columns. Several researchers have used this approach successfully (e.g., Miller, Todd, and Hegde (1989); and Belew, McInerney, and Schraudolph (1990)). Using this method, Whitley, Starkweather, and Bogart (1990) showed that the EA could find topologies that learn faster than the typical fully-connected feedforward network. The EA can be explicitly biased to favor smaller networks, which can be trained faster. However, since each connection is explicitly coded, the length of the individuals is $O(n^2)$, and the algorithm may not scale up to large problems.

Although direct encoding is straightforward to implement, it is not a good analogy of the way things work in nature. The genome of an animal does not specify every connection in its nervous system. Instead, the genome contains instructions that—in conjunction with environmental factors—determine the final structure of the network. Many interesting combinations of EAs with NNs imitate nature's indirect specification of nervous systems, and use a developmental approach to construct the networks.

A simple method to avoid specifying all the connections is to commit to a particular topology (feedforward, recurrent, etc.) and a particular learning algorithm, and then use the EA to set the parameters that complete the network specification. For example, with a fully-connected feedforward topology the EA may be used to search for the number of layers and the number of neurons per layer. Another example would be to code the parameters of a particular learning algorithm such as the momentum and learning rate for backpropagation (Belew, McInerney, & Schraudolph, 1990; Marshall & Harrison, 1991). By specifying only the parameters for a given topology, the coding is very compact and well suited for a evolutionary algorithm, but this method is constrained by the initial choice of topology and learning algorithm.

A more sophisticated approach to indirect representations is to use a grammar to encode rules that govern the development of a network. Kitano (1990) introduced the earliest grammar-based approach. He used a connectivity matrix to represent the network, but instead of coding the matrix directly in the chromosome, he used a graph rewriting grammar to generate the matrix. The chromosomes contain rules that rewrite scalar matrix elements into 2 x 2 matrices. To evaluate the fitness, the rules are decoded from the chromosomes, and the connectivity matrix is created applying all the rules that match non-terminal symbols. Then, the connectivity matrix is interpreted to build a network, which is trained by backpropagation, and the fitness is measured. Perhaps the major drawback in this approach is that the size of the network must be 2^i (where i is any non-negative integer that represents the number of rewriting steps), because after each rewriting step the size of the matrix doubles in each dimension.

Another example of a grammar-based developmental system is the work of Boers and Kuiper (1992). Each individual contains the rules for one Lindenmayer system (L-system), which are parallel string rewriting grammars (every applicable rule is used at each derivation step). L-systems have been used to model the development of living organisms. To evaluate the fitness, the system uses the rules of the L-system to generate a string that represents the structure of a neural network. Then, the network is trained using backpropagation and the fitness is determined by combining the accuracy of the classifications on separate training and testing sets.

Gruau (1992) invented a "cellular encoding" method to evolve the topology and the weights of the network simultaneously. His objective was to produce a coding for modular networks that would scale up to large and interesting problems naturally. Gruau (1994) proved that cellular encoding has many desirable properties for a neural network representation. For example, all possible networks are representable, and only valid networks result after applying the genetic operators. Each cell in the network has a copy of a grammar tree (a grammar encoded as a tree), a read head, and some internal registers. The development of the network starts with a single cell. The grammar tree contains instructions that make the cell divide, increment or decrement its bias or some weights, cut a connection, and stop reading the tree. At each step, every cell executes the instruction pointed to by its head, and the development finishes when all the cells reach stop instructions. Gruau solved large parity and symmetry problems, and his approach compares favorably to direct encoding (Gruau, Whitley, & Pyeatt, 1996).

Nolfi, Elman, and Parisi (1994) developed another grammar-based encoding. Their objective was to simulate cell growth, migration and differentiation, three processes involved in the development of natural neural networks. Their networks may contain up to 16 types of cells, and for each type there is a rule that governs how the cell reproduces. The rules are encoded in the chromosome, and they specify the types of the daughter cells and their relative spatial locations. After a fixed number of divisions, the cells grow artificial axons to reach other cells. Cells live in a two-dimensional space that is partitioned into three regions. The developmental process begins with a cell placed near the center. The neurons that end up in the lower and upper regions serve as the inputs and outputs, respectively. The cells in the middle region function as hidden units.

The grammar-based methods share several properties. First, the developmental process begins with a single cell, just as in nature. Second, all the methods are very sensitive to changes in parts of the genome that govern early development (e.g., the initial cell's type or the first rule to be applied).

Decision Trees and Evolutionary Algorithms

Decision trees are a popular classification method because they are easy to build and experts can interpret them easily. The internal nodes represent tests on the features that describe the data, and the leaf nodes represent the class labels. A path from the root node to one of the leaves represents a conjunction of tests. Since genetic programming traditionally uses trees to represent solutions, it seems well suited for the task of finding decision trees. Koza (1992) offered an early example of this use of GP in classification, where the fitness of each decision tree is based on its accuracy on a training set. Nicolaev and Slavov (1997) extended the fitness measure to include terms

related to the tree size, and determined that GP could find small trees that were comparable in accuracy to those found by C4.5 in several test cases. Folino, Spizzuti, and Spezzano (2000) demonstrate that a fine-grained GP system can find trees that are smaller and comparatively accurate to those found with C4.5 on several test problems. Their system was designed with the intention of implementing it on a parallel computer to shorten the computation time.

The trees considered above used tests on a single attribute of the data. These tests are equivalent to hyperplanes that are parallel to one of the axes in the attribute space, and therefore the resulting trees are called axis-parallel. Axis-parallel trees are easy to interpret, but may be complex and inaccurate if the data is partitioned best by hyperplanes that are not axis-parallel. Oblique decision trees use linear combinations of attributes in the tests in each of the internal nodes. Cantú-Paz and Kamath (2000) used evolution strategies and genetic algorithms to find the coefficients for the tests. They used the traditional top-down construction method, where the algorithm determines the test of each node, splits the data according to the test, and applies itself recursively to each of the resulting subsets. Cantú-Paz and Kamath compared their methods against axis-parallel and other oblique tree algorithms. They found that when the data was best split by oblique hyperplanes, the evolutionary methods were in general faster and more accurate than the existing oblique algorithms, but when the target concepts were well represented by axis-parallel hyperplanes, the existing methods were superior.

Other approaches to build oblique decision trees consider the entire tree at a time, just as Koza's original method. Bot and Langdon (2000) use traditional GP complemented with a multi-objective selection method that attempts to minimize the tree size and the classification errors simultaneously. When compared to other algorithms, the classification accuracy results were mixed, but GP was consistently slower.

Venturini et al. (1997) presented an interactive evolutionary algorithm that permits the user to evaluate combinations of the attributes that describe the data. The objective of the system is to find new variables that can describe the data concisely and that can be used in a traditional classification algorithm afterwards. Each individual in the algorithm uses two GP trees to represent new variables that are a transformation of the original attributes. The two new variables can be regarded as new axes on which the training set is projected and the result is displayed as a scatter plot. All the individuals are processed in this way and presented to the user who decides which projections show some interesting structures. The selected individuals undergo crossover and mutation, and the cycle is repeated. Venturini et al. (1997) present mixed results on several data sets from the UCI repository, but suggest several interesting extensions of their system, such as allowing the user to create rules directly by specifying thresholds on the screen.

EVOLUTIONARY ALGORITHMS IN CLUSTERING

We can distinguish two major methods to apply evolutionary algorithms to clustering problems. In the first method, each position in the chromosome represents an item in the training set. The task of the EA is to find the right cluster for each data item. If the number of clusters, k, is known a priori, each position in the chromosomes can take a value in [1,k]. This method is somewhat analogous to the direct encoding of neural nets. It is easy to implement, as there is no need for

special evolutionary operators, but it suffers from a severe scalability problem: the length of the individuals is exactly the size of the training set, and for large problems this option may not be practical. Examples of this approach include the work by Murthy and Chowdhury (1996).

Park and Song (1998) created a variation of the direct representation. They recognized that the clustering problem could be cast as a graph-partitioning problem. The objective is to consider the items in the data set as nodes in a graph and the objective is to use a GA to find connected subgraphs that represent clusters. Each data item has a corresponding position in the chromosomes, but the alleles are not the cluster labels, but the indices of other data items. So if position i contains the value j, there is a link in the graph between the nodes that represent items i and j. The values for each position are limited to the nearest neighbors of each data item, and the number of neighbors is an input parameter to the algorithm. Park and Song tested their algorithm on the problem of generating a thesaurus of word meanings and compared their results to other clustering algorithms. An advantage of their algorithm is that the number of clusters does not have to be specified in advance. The problem of scalability is still present as the individual's length is the size of the data set, and since this algorithm computes the nearest neighbors of all the data items, the algorithm may not be very efficient on data sets with many dimensions.

Another use of EAs in clustering is to identify the cluster centroids. Hall, Ozyurt and Bezdek (1999) described an evolutionary approach where the individuals represent the coordinates of the centers of the k desired clusters. They used a standard genetic algorithm, trying both floating point and binary representations, but did not observe a clear advantage to either approach. Their study considered both fuzzy and hard clustering, and their fitness functions included terms to penalize degenerate solutions (with fewer than k clusters). Hall et al. compared their algorithm to conventional clustering algorithms (FCM/HCM) and observed that their evolutionary approach usually found solutions as good as the other methods, and avoided degenerate solutions when the other methods did not. They experimented with adaptive methods to set the parameters of the algorithm and found the results encouraging. This is important because it facilitates the use of the evolutionary algorithm in practice. However, Hall et al. also reported that the execution time of the evolutionary method can take up to two orders of magnitude more than FCM/HCM. Despite the efficiency problem, Hall et al. noted that the evolutionary approach could be useful to evaluate other clustering fitness functions for which no optimization method has been devised. A similar approach is to use the EA to search for the optimal initial seed values for the cluster centroids and then run a clustering algorithm (Babu and Murty, 1993).

As in other problems, in clustering we can use domain knowledge in several ways to try to improve the performance of the algorithm. For example, we could design specialized evolutionary operators or we can hybridize the evolutionary algorithm with a conventional clustering algorithm. Fränti et al. (1997) tried both approaches. Their clustering algorithm represented the coordinates of the centroids. They used five different crossover methods (three of their own invention) and after crossover each new individual underwent two iterations of the k-means clustering algorithm. Later they extended the algorithm to include self-adaptation of parameters and automatic choice of operators (Kivijärvi, 2000). Fränti et al. (1997) observed that adding the k-means iterations was critical for obtaining good results, and although there can be a considerable increase of the computation time if many iterations are used, their experiments suggest that only a few iterations are needed. Along these lines, Krishna and Murty (1999) used a

single k-means iteration. The hybridization raises the question of how to allocate the computing time: should we use many generations of the EA and a few iterations of the local methods, or run the EAs for a few generations and use the local methods to improve the solutions considerably?

As we saw in the neural networks section, another way to use domain knowledge in GAs is to initialize the population with good known solutions. One way to do this in clustering problems would be to use the output of independent runs of the k-means algorithm to create at least part of the initial population (Murthy and Chowdhury, 1996).

In principle, the centroid-based representation has the advantage that the individuals are shorter, because they only need to represent the coordinates of the k centroids. This means that the length of the individuals is proportional to the dimensionality of the problem and not to the size of the training set as in the partitioning-based encoding. In addition, using the GA to assign the right cluster labels to each data item allows more flexibility in the shape of the clusters. For example, non-adjacent regions of the data space can belong to the same cluster.

PERFORMANCE OF EVOLUTIONARY ALGORITHMS

Evolutionary algorithms are proving themselves in solving real problems in data mining, especially in cases where the data is noisy, or requires the solution of a multi-objective optimization problem. However, they are not without their drawbacks.

A key concern expressed by several authors is that evolutionary algorithms can be very time consuming. For example, Poli (1996) comments that the tremendous computational demands of fitness evaluations in the use of genetic programming for image processing has prevented researchers from doing an extensive study of the behavior of these algorithms in solving real problems. A similar sentiment is expressed by Ebner and Zell (1999) who observe that the evolution of an image processing operator typically takes several days to complete on a single PC, making it difficult to use their algorithm in an adaptive vision system that adapts to changing environmental conditions.

Several approaches have been proposed to address this need for enormous computational resources. For example, Mandava, Fitzpatrick, and Pickens (1989) and Poli (1996) suggest that, in image processing, instead of using all pixels in an image to evaluate the fitness of an operator, only a small sample of pixels could be used in order to reduce the time required. Other authors, such as Bhanu, Lee, and Ming (1995) keep a global population of fit individuals, which can be used to seed the genetic algorithm for each image. This not only makes the system adaptive, but also reduces the computation time. Bhandarkar, Zhang, and Potter (1994) propose exploiting the inherent parallelism in genetic algorithms to reduce the time for edge detection operators in image analysis.

Researchers using evolutionary algorithms for feature selection also echo this need for extensive computer resources. Since the approach requires the classification step to be performed for each

fitness evaluation, it can be time consuming. A common solution in this case is the use of parallel processing (Punch et al., 1993).

Of course, sampling and parallel processing can also aid in classification and clustering problems. In addition, in previous sections we also hinted that using representations that are more appropriate for the problems at hand or designing custom operators could result in a more scalable algorithm. For example, directly encoding each weight in a neural network or each possible assignment of a data item to a cluster will not scale up to large and interesting problems.

RESOURCES FOR EVOLUTIONARY ALGORITHMS IN DATA MINING

With evolutionary algorithms rapidly gaining acceptance in data mining, there are a variety of resources that the interested researcher can refer to for the most recent advances in the field. There are several conferences held in the various topics covered in this chapter, including the EvoIASP conferences organized by the Working Group on Evolutionary Algorithms in Image Analysis and Signal Processing (2001), Knowledge Discovery and Data Mining (KDD), International Conference on Machine Learning (ICML), and the Genetic and Evolutionary Computation Conference (GECCO). The journals Evolutionary Computation, Genetic Programming and Evolvable Machines, IEEE Transactions on Systems, Man, and Cybernetics, and the IEEE Transactions on Evolutionary Computation are also excellent resources. There are several resources available on the Internet as well. A comprehensive bibliography on genetic algorithms by Alander (2000) includes their use in classifier systems, image processing, signal processing, neural networks, etc.

SUMMARY

In this survey paper, we have shown that evolutionary algorithms can complement many existing data mining algorithms. They can extract and select features, train neural networks, find classification rules, and build decision trees. Evolutionary algorithms are particularly useful when the problems involve the optimization of functions that are not smooth and differentiable, or functions where the objective value changes over time, which can happen in data mining as more data becomes available or if sampling is used to reduce the computation time.

While evolutionary algorithms enable us to solve some difficult problems, they come at a price, namely a need for high computational resources. However, with processors becoming faster and the increasing acceptance of parallel systems, we hope that this problem will be minimized in the future.

ACKNOWLEDGEMENT

UCRL-JC-141872. This work was performed under the auspices of the US Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

REFERENCES

Alander, J. (2000) Indexed bibliography of genetic algorithms and artificial intelligence. Technical Report No. 94-1-AI. University of Vaasa, Department of Information Technology and Production Economics. ftp://ftp.vaasa.fi/cs/report94-1/gaAIbib.ps.Z.

Augier, S., Venturini, G., Kodratoff, Y. (1995). Learning first order rules with a genetic algorithm. In *Proceedings of the First International Conference on Knowledge Discovery in Databases*. (pp. 21-26). Menlo Park, CA: AAAI Press.

Babu, G. P., & Murty, M. N. (1993). Clustering with evolution strategies. *Pattern Recognition*, 27 (2), 321-329.

Belew, R., McInerney, J., & Schraudolph, N. (1990). Evolving networks: Using the genetic algorithm with connectionist learning (Tech. Rep. No. CS90-174). San Diego: University of California, Computer Science and Engineering Department.

Bhandarkar, S., Zhang, Y., and Potter, W. (1994). An edge detection technique using genetic algorithm based optimization. *Pattern Recognition* 27, 1159-1180.

Bhanu, B. and Lee, S. (1994). *Genetic learning for adaptive image segmentation*. Boston, MA: Kluwer Academic Publishers.

Bhanu, B., Lee, S. and Ming, J. (1995). Adaptive image segmentation using a genetic algorithm. *IEEE Transactions on Systems, Man, and Cybernetics*, 25, 1543-1567.

Boers, J. W., & Kuiper, H. (1992). Biological metaphors and the design of modular artificial neural networks. Unpublished Master's Thesis, Leiden University, The Netherlands.

Booker, L. B., Goldberg, D. E., & Holland, J. H. (1989). Classifier systems and genetic algorithms. *Artificial Intelligence*, 40 (1/3), 235-282.

Bot, M.C.J., Langdon, W.B., Application of genetic programming to induction of linear classification trees. In *European Conference on Genetic Programming*, (pp. 247-258). Berlin: Springer-Verlag.

Branke, J. (1995). Evolutionary algorithms for neural network design and training (Technical Report). Karlsruhe, Germany: Institute AIFB, University of Karlsruhe.

Brill, F.Z., Brown, D.E., & Martin, W.N. (1990) *Genetic algorithms for feature selection for counterpropagation networks.* (Tech. Rep. No. IPC-TR-90-004). Charlottesville, VA: University of Virginia, Institute of Parallel Computation.

Brotherton, T.W., & Simpson, P.K. (1995). Dynamic feature set training of neural nets for classification. In McDonnell, J.R., Reynolds, R.G., & Fogel, D.B. (Eds.) *Evolutionary Programming IV* (pp. 83-94). Cambridge, MA: MIT Press.

Brumby, S**.,** Theiler, J., Perkins, S., Harvey, N., Szymanski, J., Bloch, J. and Mitchell, M., (1999). Investigation of image feature extraction by a genetic algorithm. Bellingham, WA: *Procedings of the International Society for Optical Engineering*, vol. 3812, 24-31

Burl, M., Asker, L., Smyth, P., Fayyad, U., Perona, P., Crumpler, L, & Aubele, J. (1998). Learning to recognize volcanoes on Venus. *Machine Learning*, 30, 165-195.

Cagnoni S., Dobrzeniecki, A., Poli, R., and Yanch, J. (1997). Segmentation of 3D medical images through genetically-optimized contour-tracking algorithms. Univ. of Birmingham School of Computer Science Tech. Report CSRP-97-28.

Cantú-Paz, E., & Kamath, C. (2000). Using evolutionary algorithms to induce oblique decision trees. In Whitley, D., Goldberg, D. E., Cantú-Paz, E., Spector, L., Parmee, L., & Beyer, H.-G. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference 2000* (pp. 1053-1060). San Francisco, CA: Morgan Kaufmann Publishers.

Caudell, T. P., & Dolan, C. P. (1989). Parametric connectivity: Training of constrained networks using genetic algorithms. In Schaffer, J. D. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 370-374). San Mateo, CA: Morgan Kaufmann.

De Jong, K. A., Spears, W. M., & Gordon, D. F. (1993). Using genetic algorithms for concept learning. *Machine Learning*, 13, 161-188.

Dietterich, T., (2000). An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization**.** *Machine Learning,* 40 (2), 139-158.

Ebner, M. and Zell, A. (1999). Evolving a task specific image operator. In Poli, R. et al. (ed.), *Evolutionary Image Analysis, Signal Processing and Telecommunications*, First European Workshop (pp.74-89). Berlin: Springer-Verlag.

Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R. (1996). *Advances in knowledge discovery and data mining*. Menlo Park, CA: AAAI Press/ The MIT Press.

Folino, G., Pizzuti, C. & Spezzano, G. (2000). Genetic programming and simulated annealing: A hybrid method to evolve decision trees. In Poli, R., Banzhaf, W., Langdon, W. B., Miller, J., Nordin, P., & Fogarty, T. C. (Eds.), *Genetic Programming: Third European Conference* (pp. 294-303). Berlin: Springer-Verlag.

Frakes, W.B. & Baeza-Yates, R. (1992). *Information Retrieval: Data Structures and Algorithms*. Englewood Cliffs, NJ: Prentice Hall.

Fränti, P., Kivijärvi, J., Kaukoranta, T., & Nevalainen, O. (1997). Genetic algorithms for large-scale clustering problems. *The Computer Journal*, 40 (9), 547-554.

Frey, P. W., & Slate, D. J. (1991). Letter recognition using Holland-style adaptive classifiers. *Machine Learning*, 6 , 161-182.

Giordana, A., & Neri, F. (1995). Search-intensive concept induction. *Evolutionary Computation*, 3 (4), 375-416.

Goldberg, D. E. (1983). Computer-aided gas pipeline operation using genetic algorithms and rule learning. Dissertation Abstracts International, 44 (10), 3174B. Doctoral dissertation, University of Michigan.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Reading, MA: Addison-Wesley.

Gruau, F. (1992). Cellular encoding of genetic neural networks (Tech. Rep. No. 92-21). Lyon Cedex, France: Ecole Normale Superieure de Lyon.

Gruau, F. (1994). Neural network synthesis using cellular encoding and the genetic algorithm. Unpublished doctoral dissertation, L'Universite Claude Bernard-Lyon I.

Gruau, F., Whitley, D., & Pyeatt, L. (1996). A comparison between cellular encoding and direct encoding for genetic neural networks. *In Proceedings of the First Annual Conference on Genetic Programming* (pp. 81-89). Cambridge, MA: MIT Press.

Guerra-Salcedo, C. and Whitley, D. (1999). Genetic approach to feature selection for ensemble creation. *Proceedings of the Genetic and Evolutionary Computation Conference*, pp 236-243.

Hall, L., Ozyurt, B., & Bezdek, J. (1999). Clustering with a genetically optimized approach. *IEEE Transactions on Evolutionary Computation*, 3(2), 103-112.

Hancock, P. J. B. (1992). Recombination operators for the design of neural nets by genetic algorithm. In Männer, R., & Manderick, B. (Eds.), *Parallel Problem Solving from Nature, 2* (pp. 441-450). Amsterdam: Elsevier Science.

Ho, T. (1998). The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20 (8), pp 832-844.

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

Jackson, J. E. (1991). *A user's guide to principal components.* New York, NY: John Wiley.

Kadaba, N., & Nygard, K. E. (1990). Improving the performance of genetic algorithms in automated discovery of parameters. *Machine Learning: Proceedings of the Seventh International Conference*, 140-148.

Kamath, C. (2001). On mining scientific data sets. To appear in *Data Mining in Scientific and Engineering Applications*, Norwell, MA: Kluwer Academic Publishers.

Kitano, H. (1990). Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 4 (4), 461-476.

Kivijärvi, J., Fränti, P., & Nevalainen, O. (2000). Efficient clustering with a self-adaptive genetic algorithm. In Whitley, D., Goldberg, D. E., Cantú-Paz, E., Spector, L., Parmee, L., & Beyer, H.-G. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference 2000* (pp. 377). San Francisco, CA: Morgan Kaufmann Publishers.

Koza, J. R. (1992). *Genetic programming: on the programming of computers by means of natural selection*. Cambridge, MA: The MIT Press.

Krishna, K., & Murty, M. N. (1999). Genetic k-means algorithm. *IEEE Transactions on Systems, Man, and Cybernetics-Part B* , 29 (3), 433-439.

Langley, P. & Simon, H. A. (1995). Applications of machine learning and rule induction. *Communications of the ACM*, 38 (11), 55-64.

Mandava, V., Fitzpatrick, J., and Pickens, D. (1989). Adaptive search space scaling in digital image registration. IEEE Transactions on Medical Imaging, 8, 251-262.

Marshall, S.J., & Harrison, R.F. (1991) Optimization and training of feedforward neural networks by genetic algorithms. In *Proceedings of the Second International Conference on Artificial Neural Networks and Genetic Algorithms* (pp. 39-43). Berlin: Springer-Verlag.

Miller, G. F., Todd, P. M., & Hegde, S. U. (1989). Designing neural networks using genetic algorithms. In Schaffer, J. D. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 379-384). San Mateo, CA: Morgan Kaufmann.

Montana, D. J., & Davis, L. (1989). Training feedforward neural networks using genetic algorithms. In *Proceedings 11th International Joint Conference on Artificial Intelligence* (pp. 762--767). San Mateo, CA: Morgan Kaufmann.

Murthy, C. A., & Chowdhury, N. (1996). In search of optimal clusters using genetic algorithms. *Pattern Recognition Letters*, 17, 825-832.

Nikolaev, N. I., & Slavov, V. (1998). Inductive genetic programming with decision trees. *Intelligent Data Analysis*, 2 (1).

Nolfi, S., Elman, J. L., & Parisi, D. (1994). Learning and evolution in neural networks (Tech. Rep. No. 94-08). Rome, Italy: Institute of Psychology, National Research Council.

Park, Y., & Song, M. (1998). A genetic algorithm for clustering problems. In Koza, J. R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M. H., Goldberg, D. E., Iba, H., & Riolo, R. L. (Eds.), *Genetic Programming 98* (pp. 568-575). San Francisco: Morgan Kaufmann Publishers.

Poli, R., (1996). Genetic programming for feature detection and image segmentation. In Fogarty, T. (ed.), *Evolutionary Computing*, in Lecture Notes in Computer Science, number 1143, pp 110--125. Springer-Verlag.

Punch, W., Goodman, E., Pei, M., Lai, C., Hovland, P. and Enbody, R. (1993). Further research on feature selection and classification using genetic algorithms, In *Proceedings Fifth International Conference on Genetic Algorithms*, pp 557-564.

Radcliffe, N. J. (1990). Genetic neural networks on MIMD computers. Unpublished doctoral dissertation, University of Edinburgh, Scotland.

Sherrah, J., Bogner, R. and Bouzerdoum, B. (1996). Automatic selection of features for classification using genetic programming. In *Proceedings of the 1996 Australian New Zealand Conference on Intelligent Information Systems*, Adelaide, Australia, November 1996, pp 284 - 287.

Siedlecki, W. and Sklansky, J. (1989). A note on genetic algorithms for large-scale feature selection. *Pattern Recognition Letters* (10), pp 335-347.

Skinner, A., & Broughton, J.Q. (1995). Neural networks in computational material science: training algorithms. *Modeling and Simulation in Material Science and Engineering*, 3, 371-390.

Smith, S. F. (1980). A learning system based on genetic adaptive algorithms. Dissertation Abstracts International, 41 , 4582B. (University Microfilms No. 81-12638).

Smith, S. F. (1983). Flexible learning of problem solving heuristics through adaptive search. *In Proceedings of the 8th International Joint Conference on Artificial Intelligence* (pp. 422-425).

Stanhope, S. and Daida, J., (1998). Genetic programming for automatic target classification and recognition in synthetic aperture radar imagery. In *Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on Evolutionary Programming*, V.W. Porto, N. Saravan, D. Waagen, and A.E. Eiben (eds.). Berlin: Springer-Verlag, pp. 735-744.

Tackett, W., (1993). Genetic Programming for Feature Discovery and Image Discrimination, In *Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, pp 303 – 309.

Tan, H., Gelfand, S., and Delp, E. (1989). A comparative cost function approach to edge detection. *IEEE Transactions on Sytems, Man, and Cybernetics* 19, 1337-1349.

Thierens, D., Suykens, J., Vanderwalle, J., & Moor, B.D. (1991). Genetic weight optimization of a feedforward neural network controller. In *Proceedings of the Second International Conference on Artificial Neural Networks and Genetic Algorithms* (pp. 658-663). Berlin: Springer-Verlag.

Thierens, D. (1995). Analysis and design of genetic algorithms. Unpublished doctoral dissertation. Leuven, Belgium: Katholieke Universiteit Leuven.

Vafaie, H. and DeJong, K. (1998). Feature space transformation using genetic algorithms. *IEEE Intelligent Systems and their Applications*, 13(2): pp 57-65.

Venturini, G., Slimane, M., Morin, F., & Asselin de Beauville, J.-P. (1997). On using interactive genetic algorithms for knowledge discovery in databases. In Bäck, T. (Ed.), *Proceedings of the Seventh International Conference on Genetic Algorithms* (pp. 696-703). San Francisco: Morgan Kaufmann.

Weeks, A. (1996). *Fundamentals of electronic image processing*. Bellingham, WA: The International Society for Optical Engineering Press.

Whitley, D. (1995). Genetic algorithms and neural networks. In Winter, G., Periaux, J., Galan, M., & Cuesta, P. (Eds.), *Genetic Algorithms in Engineering and Computer Science* (Chapter 11, pp. 203-221). Chichester: John Wiley and Sons.

Whitley, D., & Hanson, T. (1989). Optimizing neural networks using faster, more accurate genetic search. In Schaffer, J. D. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 391-397). San Mateo, CA: Morgan Kaufmann.

Whitley, D., Starkweather, T., & Bogart, C. (1990). Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel Computing*, 14 , 347-361.

Wilson, S. W., & Goldberg, D. E. (1989). A critical review of classifier systems. In Schaffer, J. D. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 244-255). San Mateo, CA: Morgan Kaufmann.

Wilson, S. W. (1995). Classifier fitness based on accuracy. *Evolutionary Computation*, 3 (2), 149-175.

Wilson, S. W. (2000a). Mining oblique data with XCS. IlliGAL Technical Report No 2000028, University of Illinois at Urbana-Champaign.

Wilson, S. W. (2000b). State of XCS classifier system research. In Lanzi, P., Stolzmann, W., & Wilson, S. W. (Eds.), *Learning Classifier Systems: From Foundations to Applications.* Berlin: Springer-Verlag.

Yang, J. and Honavar, V. (1997). Feature subset selection using a genetic algorithm, In *Proceedings of the Second Annual Conference on Genetic Programming*, pp 380-385.

Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, 87 (9), 1423-1447.