A Review of Evolutionary Algorithms for Data Mining

Alex A. Freitas

University of Kent, UK, Computing Laboratory, A.A.Freitas@kent.ac.uk

Summary. Evolutionary Algorithms (EAs) are stochastic search algorithms inspired by the process of neo-Darwinian evolution. The motivation for applying EAs to data mining is that they are robust, adaptive search techniques that perform a global search in the solution space. This chapter first presents a brief overview of EAs, focusing mainly on two kinds of EAs, viz. Genetic Algorithms (GAs) and Genetic Programming (GP). Then the chapter reviews the main concepts and principles used by EAs designed for solving several data mining tasks, namely: discovery of classification rules, clustering, attribute selection and attribute construction. Finally, it discusses Multi-Objective EAs, based on the concept of Pareto dominance, and their use in several data mining tasks.

Key words: genetic algorithm, genetic programming, classification, clustering, attribute selection, attribute construction, multi-objective optimization

1 Introduction

The paradigm of Evolutionary Algorithms (EAs) consists of stochastic search algorithms inspired by the process of neo-Darwinian evolution (Back et al. 2000; De Jong 2006; Eiben & Smith 2003). EAs work with a population of individuals, each of them a candidate solution to a given problem, that "evolve" towards better and better solutions to that problem. It should be noted that this is a very generic search paradigm. EAs can be used to solve many different kinds of problems, by carefully specifying what kind of candidate solution an individual represents and how the quality of that solution is evaluated (by a "fitness" function).

In essence, the motivation for applying EAs to data mining is that EAs are robust, adaptive search methods that perform a global search in the space of candidate solutions. In contrast, several more conventional data mining methods perform a local, greedy search in the space of candidate solutions. As a result of their global search, EAs tend to cope better with attribute

interactions than greedy data mining methods (Freitas 2002a; Dhar et al. 2000; Papagelis & Kalles 2001; Freitas 2001, 2002c). Hence, intuitively EAs can discover interesting knowledge that would be missed by a greedy method.

The remainder of this chapter is organized as follows. Section 2 presents a brief overview of EAs. Section 3 discusses EAs for discovering classification rules. Section 4 discusses EAs for clustering. Section 5 discusses EAs for two data preprocessing tasks, namely attribute selection and attribute construction. Section 6 discusses multi-objective EAs. Finally, Section 7 concludes the chapter. This chapter is an updated version of (Freitas 2005).

2 An Overview of Evolutionary Algorithms

An Evolutionary Algorithm (EA) is essentially an algorithm inspired by the principle of natural selection and natural genetics. The basic idea is simple. In nature individuals are continuously evolving, getting more and more adapted to the environment. In EAs each "individual" corresponds to a candidate solution to the target problem, which could be considered a very simple "environment". Each individual is evaluated by a fitness function, which measures the quality of the candidate solution represented by the individual. At each generation (iteration), the best individuals (candidate solutions) have a higher probability of being selected for reproduction. The selected individuals undergo operations inspired by natural genetics, such as crossover (where part of the genetic material of two individuals are swapped) and mutation (where part of the generic material of an individual is replaced by randomly-generated genetic material), producing new offspring which will replace the parents, creating a new generation of individuals. This process is iteratively repeated until a stopping criterion is satisfied, such as until a fixed number of generations has been performed or until a satisfactory solution has been found.

There are several kinds of EAs, such as Genetic Algorithms, Genetic Programming, Classifier Systems, Evolution Strategies, Evolutionary Programming, Estimation of Distribution Algorithms, etc. (Back et al. 2000; De Jong 2006; Eiben & Smith 2003). This chapter will focus on Genetic Algorithms (GAs) and Genetic Programming (GP), which are probably the two kinds of EA that have been most used for data mining.

Both GA and GP can be described, at a high level of abstraction, by the pseudocode of Algorithm 1. Although GA and GP share this basic pseudocode, there are several important differences between these two kinds of algorithms. One of these differences involves the kind of solution represented by each of these kinds of algorithms. In GAs, in general a candidate solution consists mainly of values of variables – in essence, data. By contrast, in GP the candidate solution usually consists of both data and functions. Therefore, in GP one works with two sets of symbols that can be represented in an

individual, namely the terminal set and the function set. The terminal set typically contains variables (or attributes) and constants; whereas the function set contains functions which are believed to be appropriate to represent good solutions for the target problem. In the context of data mining, the explicit use of a function set is interesting because it provides GP with potentially powerful means of changing the original data representation into a representation that is more suitable for knowledge discovery purposes, which is not so naturally done when using GAs or another EA where only attributes (but not functions) are represented by an individual. This ability of changing the data representation will be discussed particularly on the section about GP for attribute construction.

Note that in general there is no distinction between terminal set and function set in the case of GAs, because GAs' individuals usually consist only of data, not functions. As a result, the representation of GA individuals tend to be simpler than the representation of GP individuals. In particular, GA individuals are usually represented by a fixed-length linear genome, whereas the genome of GP individuals is often represented by a variable-size tree genome – where the internal nodes contain functions and the leaf nodes contain terminals.

Algorithm 1: Generic Pseudocode for GA and GP

```
1: Create initial population of individuals
2: Compute the fitness of each individual
3: repeat
4:    Select individuals based on fitness
5:    Apply genetic operators to selected individuals, creating new individuals
6:    Compute fitness of each of the new individuals
7:    Update the current population (new individuals replace old individuals)
8: until (stopping criteria)
```

When designing a GP algorithm, one must bear in mind two important properties that should be satisfied by the algorithm, namely closure and sufficiency (Banzhaf et al. 1998; Koza 1992). Closure means that every function in the function set must be able to accept, as input, the result of any other function or any terminal in the terminal set. Some approaches to satisfy the closure property in the context of attribute construction will be discussed in Subsection 5.2. Sufficiency means that the function set should be expressive enough to allow the representation of a good solution to the target problem. In practice it is difficult to know *a priori* which functions should be used to guarantee the sufficiency property, because in challenging real-world problems one often does not know the shape of a good solution for the problem. As a practical guideline, (Banzhaf et al. 1998) (p. 111) recommends:

"An approximate starting point for a function set might be the arithmetic and logic operations: PLUS, MINUS, TIMES, DIVIDE, OR, AND, XOR. ... Good solutions using only this function set have been obtained on several different classification problems,. . . ,and symbolic regression problems."

We have previously mentioned some differences between GA and GP, involving their individual representation. Arguably, however, the most important difference between GAs and GP involves the fundamental nature of the solution that they represent. More precisely, in GAs (like in most other kinds of EA) each individual represents a solution to one particular instance of the problem being solved. In contrast, in GP a candidate solution should represent a generic solution – a program or an algorithm – to the kind of problem being solved; in the sense that the evolved program should be generic enough to be applied to any instance of the target kind of problem.

To quote (Banzhaf et al. 1998), p. 6:

> it is possible to define genetic programming as the direct evolution of *programs or algorithms* [our italics] for the purpose of inductive learning.

In practice, in the context of data mining, most GP algorithms evolve a solution (say, a classification model) *specific for a single data set*, rather than a *generic program* that can be applied to different data sets from different application domains. An exception is the work of (Pappa & Freitas 2006), proposing a grammar-based GP system that automatically evolves full rule induction algorithms, with loop statements, generic procedures for building and pruning classification rules, etc. Hence, in this system the output of a GP run is a *generic* rule induction algorithm (implemented in Java), which can be run on virtually any classification data set – in the same way that a manually-designed rule induction algorithm can be run on virtually any classification data set. An extended version of the work presented in (Pappa & Freitas 2006) is discussed in detail in another chapter of this book (Pappa & Freitas 2007).

3 Evolutionary Algorithms for Discovering Classification Rules

Most of the EAs discussed in this section are Genetic Algorithms, but it should be emphasized that classification rules can also be discovered by other kinds of EAs. In particular, for a review of Genetic Programming algorithms for classification-rule discovery, see (Freitas 2002a); and for a review of Learning Classifier Systems (a type of algorithm based on a combination of EA and reinforcement learning principles), see (Bull 2004; Bull & Kovacs 2005).

3.1 Individual Representation for Classification-Rule Discovery

This Subsection assumes that the EA discovers classification rules of the form "IF (conditions) THEN (class)" (Witten & Frank 2005). This kind of knowledge representation has the advantage of being intuitively comprehensible to the user – an important point in data mining (Fayyad et al. 1996). A crucial issue in the design of an individual representation is to decide whether the candidate solution represented by an individual will be a rule set or just a single classification rule (Freitas 2002a, 2002b).

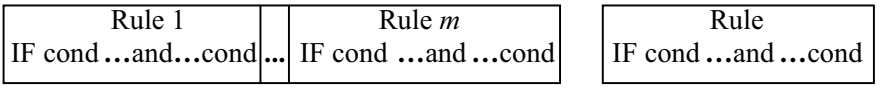
The former approach is often called the "Pittsburgh approach", whereas the later approach is often called the "Michigan-style approach". This latter term is an extension of the term "Michigan approach", which was originally used to refer to one particular kind of EA called Learning Classifier Systems (Smith 2000; Goldberg 1989). In this chapter we use the extended term "Michigan-style approach" because, instead of discussing Learning Classifier Systems, we discuss conceptually simpler EAs sharing the basic characteristic that an individual represents a single classification rule, regardless of other aspects of the EA.

The difference between the two approaches is illustrated in Figure 1. Figure 1(a) shows the Pittsburgh approach. The number of rules, m, can be either variable, automatically evolved by the EA, or fixed by a user-specified parameter. Figure 1(b) shows the Michigan-style approach, with a single rule per individual. In both Figure 1(a) and 1(b) the rule antecedent (the "IF part" of the rule) consists of a conjunction of conditions. Each condition is typically of the form <Attribute, Operator, Value>, also known as attribute-value (or propositional logic) representation. Examples are the conditions: "Gender = Female" and "Age < 25". In the case of continuous attributes it is also common to have rule conditions of the form <LowerBound, Operator, Attribute, Operator, UpperBound>, e.g.: "30K ≤ Salary ≤ 50K".

In some EAs the individuals can only represent rule conditions with categorical (nominal) attributes such as Gender, whose values (male, female) have no ordering – so that the only operator used in the rule conditions is "=", and sometimes "≠". When using EAs with this limitation, if the data set contains continuous attributes – with ordered numerical values – those attributes have to be discretized in a preprocessing stage, before the EA is applied. In practice it is desirable to use an EA where individuals can represent rule conditions with both categorical and continuous attributes. In this case the EA is effectively doing a discretization of continuous values "on-the-fly", since by creating rule conditions such as "30K ≤ Salary ≤ 50K" the EA is effectively producing discrete intervals. The effectiveness of an EA that directly copes with continuous attributes can be improved by using operators that enlarge or shrink the intervals based on concepts and methods borrowed from the research area of discretization in data mining (Divina & Marchiori 2005).

It is also possible to have conditions of the form <Attribute, Operator, Attribute>, such as "Income > Expenditure". Such conditions are associated

with relational (or first-order logic) representations. This kind of relational representation has considerably more expressiveness power than the conventional attribute-value representation, but the former is associated with a much larger search space – which often requires a more complex EA and a longer processing time. Hence, most EAs for rule discovery use the attribute-value, propositional representation. EAs using the relational, first-order logic representation are described, for instance, in (Neri & Giordana 1995; Hekanaho 1995; Woung & Leung 2000; Divina & Marchiori 2002).



(a) Pittsburgh approach (b) Michigan-style approach

Fig. 1. Pittsburgh vs. Michigan-style approach for individual representation

Note that in Figure 1 the individuals are representing only the rule antecedent, and not the rule consequent (predicted class). It would be possible to include the predicted class in each individual's genome and let that class be evolved along with its corresponding rule antecedent. However, this approach has one significant drawback, which can be illustrated with the following example. Suppose an EA has just generated an individual whose rule antecedent covers 100 examples, 97 of which have class c_1. Due to the stochastic nature of the evolutionary process and the "blind-search" nature of the generic operators, the EA could associate that rule antecedent with class c_2, which would assign a very low fitness to that individual – a very undesirable result. This kind of problem can be avoided if, instead of evolving the rule consequent, the predicted class for each rule is determined by other (non-evolutionary) means. In particular, two such means are as follows.

First, one can simply assign to the individual the class of the majority of the examples covered by the rule antecedent (class c_1 in the above example), as a conventional, non-evolutionary rule induction algorithm would do. Second, one could use the "sequential covering" approach, which is often used by conventional rule induction algorithms (Witten & Frank 2005). In this approach, the EA discovers rules for one class at a time. For each class, the EA is run for as long as necessary to discover rules covering all examples of that class. During the evolutionary search for rules predicting that class, all individuals of the population will be representing rules predicting the same fixed class. Note that this avoids the problem of crossover mixing genetic material of rules predicting different classes, which is a potential problem in approaches where different individuals in the population represent rules predicting different classes. A more detailed discussion about how to represent the rule consequent in an EA can be found in (Freitas 2002a).

The main advantage of the Pittsburgh approach is that an individual represents a complete solution to a classification problem, i.e., an entire set of rules. Hence, the evaluation of an individual naturally takes into account rule interactions, assessing the quality of the rule *set*. In addition, the more complete information associated with each individual in the Pittsburgh approach can be used to design "intelligent", task-specific genetic operators. An example is the "smart" crossover operator proposed by (Bacardit & Krasnogor 2006), which heuristically selects, out of the N sets of rules in N parents (where $N \geq 2$), a good subset of rules to be included in a new child individual. The main disadvantage of the Pittsburgh approach is that it leads to long individuals and renders the design of genetic operators (that will act on selected individuals in order to produce new offspring) more difficult.

The main advantage of the Michigan-style approach is that the individual representation is simple, without the need for encoding multiple rules in an individual. This leads to relatively short individuals and simplifies the design of genetic operators. The main disadvantage of the Michigan-style approach is that, since each individual represents a single rule, a standard evaluation of the fitness of an individual ignores the problem of rule interaction. In the classification task, one usually wants to evolve a *good set* of rules, rather than a set of *good rules*. In other words, it is important to discover a rule set where the rules "cooperate" with each other. In particular, the rule set should cover the entire data space, so that each data instance should be covered by at least one rule. This requires a special mechanism to discover a diverse set of rules, since a standard EA would typically converge to a population where almost all the individuals would represent the same best rule found by the evolutionary process.

In general the previously discussed approaches perform a "direct" search for rules, consisting of initializing a population with a set of rules and then iteratively modifying those rules via the application of genetic operators. Due to a certain degree of randomness typically present in both initialization and genetic operations, some bad quality rules tend to be produced along the evolutionary process. Of course such bad rules are likely to be eliminated quickly by the selection process, but in any case an interesting alternative and "indirect" way of searching for rules has been proposed, in order to minimize the generation of bad rules. The basic idea of this new approach, proposed in (Jiao et al. 2006), is that the EA searches for good groups (clusters) of data instances, where each group consists of instances of the same class. A group is good to the extent that its data instances have similar attribute values and those attribute values are different from attribute values of the instances in other groups. After the EA run is over and good groups of instances have been discovered by the EA, the system extracts classification rules from the groups. This seems a promising new approach, although it should be noted that the version of the system described in (Jiao et al. 2006) has the limitation of coping only with categorical (not continuous) attributes.

In passing, it is worth mentioning that the above discussion on rule representation issues has focused on a generic classification problem. Specific kinds of classification problems may well be more effectively solved by EAs using rule representations "tailored" to the target kind of problem. For instance, (Hirsch et al. 2005) propose a rule representation tailored to document classification (i.e., a *text* mining problem), where strings of characters – in general fragments of words, rather than full words – are combined via Boolean operators to form classification rules.

3.2 Searching for a Diverse Set of Rules

This subsection discusses two mechanisms for discovering a diverse set of rules. It is assumed that each individual represents a single classification rule (Michigan-style approach). Note that the mechanisms for rule diversity discussed below are not normally used in the Pittsburgh approach, where an individual already represents a set of rules whose fitness implicitly depends on how well the rules in the set cooperate with each other.

First, one can use a niching method. The basic idea of niching is to avoid that the population converges to a single high peak in the search space and to foster the EA to create stable subpopulations of individuals clustered around each of the high peaks. In general the goal is to obtain a kind of "fitness-proportionate" convergence, where the size of the subpopulation around each peak is proportional to the height of that peak (i.e., to the quality of the corresponding candidate solution).

For instance, one of the most popular niching methods is fitness sharing (Goldberg & Richardson 1987; Deb & Goldberg 1989). In this method, the fitness of an individual is reduced in proportion to the number of similar individuals (neighbors), as measured by a given distance metric. In the context of rule discovery, this means that if there are many individuals in the current population representing the same rule or similar rules, the fitness of those individuals will be considerably reduced, and so they will have a considerably lower probability of being selected to produce new offspring. This effectively penalizes individuals which are in crowded regions of the search space, forcing the EA to discover a diverse set of rules.

Note that fitness sharing was designed as a generic niching method. By contrast, there are several niching methods designed specifically for the discovery of classification rules. An example is the "universal suffrage" selection method (Giordana et al. 1994; Divina 2005) where – using a political metaphor – individuals to be selected for reproduction are "elected" by the training data instances. The basic idea is that each data instance "votes" for a rule that covers it in a probabilistic fitness-based fashion. More precisely, let R be the set of rules (individuals) that cover a given data instance i, i.e., the set of rules whose antecedent is satisfied by data instance i. The better the fitness of a given rule r in the set R, the larger the probability that rule r will receive the vote of data instance i. Note that in general only rules covering the

same data instances are competing with each other. Therefore, this selection method implements a form of niching, fostering the evolution of different rules covering different parts of the data space. For more information about niching methods in the context of discovering classification rules the reader is referred to (Hekanaho 1996; Dhar et al. 2000).

Another kind of mechanism that can be used to discover a diverse set of rules consists of using the previously-mentioned "sequential covering" approach – also known as "separate-and-conquer". The basic idea is that the EA discovers one rule at a time, so that in order to discover multiple rules the EA has to be run multiple times. In the first run the EA is initialized with the full training set and an empty set of rules. After each run of the EA, the best rule evolved by the EA is added to the set of discovered rules and the examples correctly covered by that rule are removed from the training set, so that the next run of the EA will consider a smaller training set. The process proceeds until all examples have been covered. Some examples of EAs using the sequential covering approach can be found in (Liu & Kwok 2000; Zhou et al. 2003; Carvalho & Freitas 2004). Note that the sequential covering approach is not specific to EAs. It is used by several non-evolutionary rule induction algorithms, and it is also discussed in data mining textbooks such as (Witten & Frank 2005).

3.3 Fitness Evaluation

One interesting characteristic of EAs is that they naturally allow the evaluation of a candidate solution, say a classification rule, as a whole, in a global fashion. This is in contrast with some data mining paradigms, which evaluate a partial solution. Consider, for instance, a conventional, greedy rule induction algorithm that incrementally builds a classification rule by adding one condition at a time to the rule. When the algorithm is evaluating several candidate conditions, the rule is still incomplete, being just a partial solution, so that the rule evaluation function is somewhat shortsighted (Freitas 2001, 2002a; Furnkranz & Flach 2003).

Another interesting characteristic of EAs is that they naturally allow the evaluation of a candidate solution by simultaneously considering different quality criteria. This is not so easily done in other data mining paradigms. To see this, consider again a conventional, greedy rule induction algorithm that adds one condition at a time to a candidate rule, and suppose one wants to favor the discovery of rules which are both accurate and simple (short). As mentioned earlier, when the algorithm is evaluating several candidate conditions, the rule is still incomplete, and so its size is not known yet. Hence, intuitively is better to choose the best candidate condition to be added to the rule based on a measure of accuracy only. The simplicity (size) criterion is better considered later, in a pruning procedure.

The fact that EAs evaluate a candidate solution as a whole and lend themselves naturally to simultaneously consider multiple criteria in the evaluation

of the fitness of an individual gives the data miner a great flexibility in the design of the fitness function. Hence, not surprisingly, many different fitness functions have been proposed to evaluate classification rules. Classification accuracy is by far the criterion most used in fitness functions for evolving classification rules. This criterion is already extensively discussed in many good books or articles about classification, e.g. (Hand 1997; Caruana & Niculescu-Mizil 2004), and so it will not be discussed here – with the exception of a brief mention of overfitting issues, as follows. EAs can discover rules that overfit the training set – i.e. rules that represent very specific patterns in the training set that do not generalize well to the test set (which contains data instances unseen during training). One approach to try to mitigate the overfitting problem is to vary the training set at every generation, i.e., at each generation a subset of training instances is randomly selected, from the entire set of training instances, to be used as the (sub-)training or validation set from which the individuals' fitness values are computed (Bacardit et al. 2004; Pappa & Freitas 2006; Sharpe & Glover 1999; Bhattacharyya 1998). This approach introduces a selective pressure for evolving rules with a greater generalization power and tends to reduce the risk of overfitting, by comparison with the conventional approach of evolving rules for a training set which remains fixed throughout evolution. In passing, if the (sub)-training or validation set used for fitness computation is significantly smaller than the original training set, this approach also has the benefit of significantly reducing the processing time of the EA.

Hereafter this section will focus on two other rule-quality criteria (not based on accuracy) that represent different desirables properties of discovered rules in the context of data mining, namely: comprehensibility (Fayyad et al. 1996), or simplicity; and surprisingness, or unexpectedness (Liu et al. 1997; Romao et al. 2004; Freitas 2006).

The former means that ideally the discovered rule(s) should be comprehensible *to the user*. Intuitively, a measure of comprehensibility should have a strongly subjective, user-dependent component. However, in the literature this subjective component is typically ignored (Pazzani 2000; Freitas 2006), and comprehensibility is usually evaluated by a measure of the syntactic simplicity of the classifier, say the size of the rule set. The latter can be measured in an objective manner, for instance, by simply counting the total number of rule conditions in the rule set represented by an individual.

However, there is a natural way of incorporating a subjective measure of comprehensibility into the fitness function of an EA, namely by using an *interactive* fitness function. The basic idea of an interactive fitness function is that the user directly evaluates the fitness of individuals during the execution of the EA (Banzhaf 2000). The evaluation of the user is then used as the fitness measure for the purpose of selecting the best individuals of the current population, so that the EA evolves solutions that tend to maximize the subjective preference of the user.

An interactive EA for attribute selection is discussed e.g. in (Terano & Ishino 1998, 2002). In that work an individual represents a selected subset of attributes, which is then used by a classification algorithm to generate a set of rules. Then the user is shown the rules and selects good rules and rule sets according to her/his subjective preferences. Next the individuals having attributes that occur in the selected rules or rule sets are selected as parents to produce new offspring. The main advantage of interactive fitness functions is that intuitively they tend to favor the discovery of rules that are comprehensible and considered "good" by the user. The main disadvantage of this approach is that it makes the system considerably slower. To mitigate this problem one often has to use a small population size and a small number of generations.

Another kind of criterion that has been used to evaluate the quality of classification rules in the fitness function of EAs is the surprisingness of the discovered rules. First of all, it should be noted that accuracy and comprehensibility do not imply surprisingness. To show this point, consider the following classical hypothetical rule, which could be discovered from a hospital's database: IF (patient is pregnant) THEN (gender is female). This rule is very accurate and very comprehensible, but it is useless, because it represents an obvious pattern.

One approach to discover surprising rules consists of asking the user to specify a set of general impressions, specifying his/her previous knowledge and/or believes about the application domain (Liu et al. 1997). Then the EA can try to find rules that are surprising in the sense of contradicting some general impression specified by the user. Note that a rule should be reported to the user only if it is found to be both surprising and at least reasonably accurate (consistent with the training data). After all, it would be relatively easy to find rules which are surprising and inaccurate, but these rules would not be very useful to the user.

An EA for rule discovery taking this into account is described in (Romao et al. 2002, 2004). This EA uses a fitness function measuring both rule accuracy and rule surprisingness (based on general impressions). The two measures are multiplied to give the fitness value of an individual (a candidate prediction rule).

4 Evolutionary Algorithms for Clustering

There are several kinds of clustering algorithm, and two of the most popular kinds are iterative-partitioning and hierarchical clustering algorithms (Aldenderfer & Blashfield 1984; Krzanowski & Marriot 1995). In this section we focus mainly on EAs that can be categorized as iterative-partitioning algorithms, since most EAs for clustering seem to belong to this category.

4.1 Individual Representation for Clustering

A crucial issue in the design of an EA for clustering is to decide what kind of individual representation will be used to specify the clusters. There are at least three major kinds of individual representation for clustering (Freitas 2002a), as follows.

Cluster description-based representation – In this case each individual explicitly represents the parameters necessary to precisely specify each cluster. The exact nature of these parameters depends on the shape of clusters to be produced, which could be, e.g., boxes, spheres, ellipsoids, etc. In any case, each individual contains K sets of parameters, where K is the number of clusters, and each set of parameters determines the position, shape and size of its corresponding cluster. This kind of representation is illustrated, at a high level of abstraction, in Figure 2, for the case where an individual represents clusters of spherical shape. In this case each cluster is specified by its center coordinates and its radius. The cluster description-based representation is used, e.g., in (Srikanth et al. 1995), where an individual represents ellipsoid-based cluster descriptions; and in (Ghozeil and Fogel 1996; Sarafis 2005), where an individual represents hyperbox-shaped cluster descriptions. In (Sarafis 2005), for instance, the individuals represent rules containing conditions based on discrete numerical intervals, each interval being associated with a different attribute. Each clustering rule represents a region of the data space with homogeneous data distribution, and the EA was designed to be particularly effective when handling high-dimensional numerical datasets.

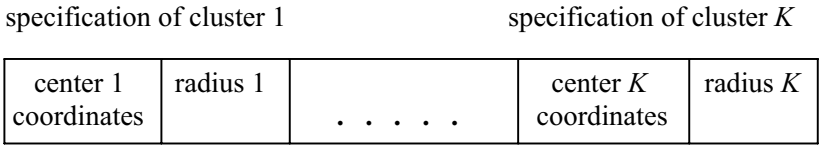


Fig. 2. Structure of cluster description-based individual representation

Centroid/medoid-based representation – In this case each individual represents the coordinates of each cluster's centroid or medoid. A centroid is simply a point in the data space whose coordinates specify the centre of the cluster. Note that there may not be any data instance with the same coordinates as the centroid. By contrast, a medoid is the most "central" representative of the cluster, i.e., it is the data instance which is nearest to the cluster's centroid. The use of medoids tends to be more robust against outliers than the use of centroids (Krzanowski & Marriot 1995) (p. 83). This kind of representation is used, e.g., in (Hall et al. 1999; Estivill-Castro and Murray 1997) and other EAs for clustering reviewed in (Sarafis 2005). This representation is illustrated, at a high level of abstraction, in Figure 3. Each data instance is assigned to the cluster represented by the centroid or medoid

that is nearest to that instance, according to a given distance measure. Therefore, the position of the centroids/medoids and the procedure used to assign instances to clusters implicitly determine the precise shape and size of the clusters.

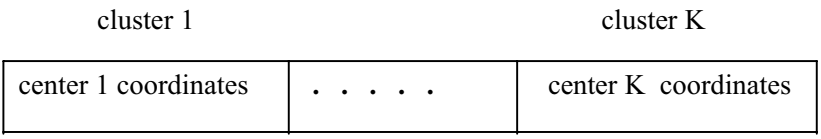


Fig. 3. Structure of centroid/medoid-based individual representation

Instance-based representation – In this case each individual consists of a string of n elements (genes), where n is the number of data instances. Each gene i, i=1,...,n, represents the index (id) of the cluster to which the i-th data instance is assigned. Hence, each gene i can take one out of Kvalues, where K is the number of clusters. For instance, suppose that $n = 10$ and K= 3. The individual <2 1 2 3 3 2 1 1 2 3> corresponds to a candidate clustering where the second, seventh and eighth instances are assigned to cluster 1, the first, third, sixth and ninth instances are assigned to cluster 2 and the other instances are assigned to cluster 3. This kind of representation is used, for instance, in (Krishma and Murty 1999; Handl & Knowles 2004). A variation of this representation is used in (Korkmaz et al. 2006), where the value of a gene represents not the cluster id of a gene's associated data instance, but rather a link from the gene's instance to another instance which is considered to be in the same cluster. Hence, in this approach, two instances belong to the same cluster if there is a sequence of links from one of them to the other. This variation is more complex than the conventional instance-based representation, and it has been proposed together with repair operators that rectify the contents of an individual when it violates some pre-defined constraints.

Comparing different individual representations for clustering – In both the centroid/medoid-based representation and the instance-based representation, each instance is assigned to exactly one cluster. Hence, the set of clusters determine a partition of the data space into regions that are mutually exclusive and exhaustive. This is not the case in the cluster description-based representation. In the latter, the cluster descriptions may have some overlapping – so that an instance may be located within two or more clusters – and the cluster descriptions may not be exhaustive – so that some instance(s) may not be within any cluster.

Unlike the other two representations, the instance-based representation has the disadvantage that it does not scale very well for large data sets, since each individual's length is directly proportional to the number of instances being clustered. This representation also involves a considerable degree of

redundancy, which may lead to problems in the application of conventional genetic operators (Falkenauer 1998). For instance, let $n = 4$ and $K = 2$, and consider the individuals <1 2 1 2> and <2 1 2 1>. These two individuals have different gene values in all the four genes, but they represent the same candidate clustering solution, i.e., assigning the first and third instances to one cluster and assigning the second and fourth instances to another cluster. As a result, a crossover between these two parent individuals can produce two children individuals representing solutions that are very different from the solutions represented by the parents, which is not normally the case in conventional crossover operators used by genetic algorithms. Some methods have been proposed to try to mitigate some redundancy-related problems associated with this kind of representation. For example, (Handl & Knowles 2004) proposed a mutation operator that is reported to work well with this representation, based on the idea that, when a gene has its value mutated – meaning that the gene's corresponding data instance is moved to another cluster – the system selects a number of "nearest neighbors" of that instance and moves all those nearest neighbors to the same cluster to which the mutated instance was moved. Hence, this approach effectively incorporates some knowledge of the clustering task to be solved in the mutation operator.

4.2 Fitness Evaluation for Clustering

In an EA for clustering, the fitness of an individual is a measure of the quality of the clustering represented by the individual. A large number of different measures have been proposed in the literature, but the basic ideas usually involve the following principles. First, the smaller the intra-cluster (within-cluster) distance, the better the fitness. The intra-cluster distance can be defined as the summation of the distance between each data instance and the centroid of its corresponding cluster – a summation computed over all instances of all the clusters. Second, the larger the inter-cluster (between-cluster) distance, the better the fitness. Hence, an algorithm can try to find optimal values for these two criteria, for a given fixed number of clusters. These and other clustering-quality criteria are extensively discussed in the clustering literature – see e.g. (Aldenderfer and Blashfield 1984; Backer 1995; Tan et al. 2006). A discussion of this topic in the context of EAs can be found in (Kim et al. 2000; Handl & Knowles 2004; Korkmaz et al. 2006; Krishma and Murty 1999; Hall et al. 1999).

In any case, it is important to note that, if the algorithm is allowed to vary the number of discovered clusters without any restriction, it would be possible to minimize intra-cluster distance and maximize inter-cluster distance in a trivial way, by assigning each example to its own singleton cluster. This would be clearly undesirable. To avoid this while still allowing the algorithm to vary the number of clusters, a common response is to incorporate in the fitness function a preference for a smaller number of clusters. It might also be desirable or necessary to incorporate in the fitness function a penalty term

whose value is proportional to the number of empty clusters (i.e. clusters to which no data instance was assigned) (Hall et al. 1999).

5 Evolutionary Algorithms for Data Preprocessing

5.1 Genetic Algorithms for Attribute Selection

In the attribute selection task the goal is to select, out of the original set of attributes, a subset of attributes that are relevant for the target data mining task (Liu & Motoda 1998; Guyon and Elisseeff 2003). This Subsection assumes the target data mining task is classification – which is the most investigated task in the evolutionary attribute selection literature – unless mentioned otherwise.

The standard individual representation for attribute selection consists simply of a string of N bits, where N is the number of original attributes and the i-th bit, i=1,...,N, can take the value 1 or 0, indicating whether or not, respectively, the i-th attribute is selected. For instance, in a 10-attribute data set, the individual "1 0 1 0 1 0 0 0 0 1" represents a candidate solution where only the 1st, 3rd, 5th and 10th attributes are selected. This individual representation is simple, and traditional crossover and mutation operators can be easily applied. However, it has the disadvantage that it does not scale very well with the number of attributes. In applications with many thousands of attributes (such as text mining and some bioinformatics problems) an individual would have many thousands of genes, which would tend to lead to a slow execution of the GA.

An alternative individual representation, proposed by (Cherkauer & Shavlik 1996), consists of M genes (where M is a user-specified parameter), where each gene can contain either the index (id) of an attribute or a flag – say 0 – denoting no attribute. An attribute is considered selected if and only if it occurs in at least one of the M genes of the individual. For instance, the individual "3 0 8 3 0", where $M = 5$, represents a candidate solution where only the 3rd and the 8th attributes are selected. The fact that the 3rd attribute occurs twice in the previous individual is irrelevant for the purpose of decoding the individual into a selected attribute subset. One advantage of this representation is that it scales up better with respect to a large number of original attributes, since the value of M can be much smaller than the number of original attributes. One disadvantage is that it introduces a new parameter, M, which was not necessary in the case of the standard individual representation.

With respect to the fitness function, GAs for attribute selection can be roughly divided into two approaches – just like other kinds of algorithms for attribute selection – namely the wrapper approach and the filter approach. In essence, in the wrapper approach the GA uses the classification algorithm to compute the fitness of individuals, whereas in the filter approach the GA does

not use the classification algorithm. The vast majority of GAs for attribute selection has followed the wrapper approach, and many of those GAs have used a fitness function involving two or more criteria to evaluate the quality of the classifier built from the selected attribute subset. This can be shown in Table 1, adapted from (Freitas 2002a), which lists the evaluation criteria used in the fitness function of a number of GAs following the wrapper approach. The columns of that table have the following meaning: *Acc* = accuracy; *Sens, Spec* = sensitivity, specificity; |Sel Attr| = number of selected attributes; |rule set| = number of discovered rules; *Info. Cont.* = information content of selected attributes; *Attr cost* = attribute costs; *Subj eval* = subjective evaluation of the user; |*Sel ins*| = number of selected instances.

Table 1. Diversity of criteria used in fitness function for attribute selection

Reference	Acc	Sens, Spec	|Sel Attr|	|rule set|	Info cont	Attr cost	Subj eval	|Sel ins|
(Bala et al. 1995)	yes		yes					
(Bala et al. 1996)	yes		yes		yes			
(Chen et al. 1999)	yes		yes					
(Cherkauer & Shavlik 1996)	yes		yes	yes				
(Emmanouilidis et al. 2000)	yes		yes					
(Emmanouilidis et al. 2002)		yes	yes					
(Guerra-Salcedo, Whitley 1998, 1999)	yes							
(Ishibuchi & Nakashima 2000)	yes		yes					yes
(Llora & Garrell 2003)	yes							
(Miller et al. 2003)		yes						
(Moser & Murty 2000)	yes		yes					
(Ni & Liu 2004)	yes							
(Pappa et al. 2002)	yes			yes				
(Rozsypal & Kubat 2003)	yes		yes					yes
(Terano & Ishino 1998)	yes			yes			yes	
(Vafaie & DeJong 1998)	yes							
(Yang & Honavar 1997, 1998)	yes					yes		
(Zhang et al 2003)	yes							

A precise definition of the terms used in the titles of the columns of Table 1 can be found in the corresponding references quoted in that table. The table refers to GAs that perform attribute selection for the classification task. GAs that perform attribute selection for the clustering task can be found, e.g., in (Kim et al. 2000; Jourdan 2003). In addition, in general Table 1 refers to GAs whose individuals directly represent candidate attribute subsets, but GAs can be used for attribute selection in other ways. For instance, in (Jong et al. 2004) a GA is used for attribute ranking. Once the ranking has been done, one can select a certain number of top-ranked attributes, where that number can be specified by the user or computed in a more automated way.

Empirical comparisons between GAs and other kinds of attribute selection methods can be found, for instance, in (Sharpe and Glover 1999; Kudo & Skalansky 2000). In general these empirical comparisons show that GAs, with their associated global search in the solution space, usually (though not always) obtain better results than local search-based attribute selection methods. In particular, (Kudo & Skalansky 2000) compared a GA with 14 non-evolutionary attribute selection methods (some of them variants of each other) across 8 different data sets. The authors concluded that the advantages of the global search associated with GAs over the local search associated with other algorithms is particularly important in data sets with a "large" number of attributes, where "large" was considered over 50 attributes in the context of their data sets.

5.2 Genetic Programming for Attribute Construction

In the attribute construction task the general goal is to construct new attributes out of the original attributes, so that the target data mining task becomes easier with the new attributes. This Subsection assumes the target data mining task is classification – which is the most investigated task in the evolutionary attribute construction literature.

Note that in general the problem of attribute construction is considerably more difficult than the problem of attribute selection. In the latter the problem consists just of deciding whether or not to select each attribute. By contrast, in attribute construction there is a potentially much larger search space, since there is a potentially large number of operations that can be applied to the original attributes in order to construct new attributes. Intuitively, the kind of EA that lends itself most naturally to attribute construction is GP. The reason is that, as mentioned earlier, GP was specifically designed to solve problems where candidate solutions are represented by both attributes and functions (operations) applied to those attributes. In particular, the explicit specification of both a terminal set and a function set is usually missing in other kinds of EAs.

Data Preprocessing vs. Interleaving Approach

In the data preprocessing approach, the attribute construction algorithm evaluates a constructed attribute without using the classification algorithm to be applied later. Examples of this approach are the GP algorithms for attribute construction proposed by (Otero et al. 2003; Hu 1998), whose attribute evaluation function (the fitness function) is the information gain ratio – a measure discussed in detail in (Quinlan 1993). In addition, (Muharram & Smith 2004) did experiments comparing the effectiveness of two different attribute-evaluation criteria in GP for attribute construction – viz. information gain ratio and gini index – and obtained results indicating that, overall, there was no significant difference in the results associated with those two criteria.

By contrast, in the interleaving approach the attribute construction algorithm evaluates the constructed attributes based on the performance of the classification algorithm with those attributes. Examples of this approach are the GP algorithms for attribute construction proposed by (Krawiec 2002; Smith and Bull 2003; Firpi et al. 2005), where the fitness functions are based on the accuracy of the classifier built with the constructed attributes.

Single-Attribute-per-Individual vs. Multiple-Attributes-per-Individual Representation

In several GPs for attribute construction, each individual represents a single constructed attribute. This approach is used for instance by CPGI (Hu 1998) and the GP algorithm proposed by (Otero et al. 2003). By default this approach returns to the user a single constructed attribute – the best evolved individual. However it can be extended to return to the user a set of constructed attributes, say returning a set of the best evolved individuals of a GP run or by running the GP multiple times and returning only the best evolved individual of each run. The main advantage of this approach is simplicity, but it has the disadvantage of ignoring interactions between the constructed attributes.

An alternative approach consists of associating with an individual a set of constructed attributes. The main advantage of this approach is that it takes into account interaction between the constructed attributes. In other words, it tries to construct the *best set* of attributes, rather than the set of *best attributes*. The main disadvantages are that the individuals' genomes become more complex and that it introduces the need for additional parameters such as the number of constructed attributes that should be encoded in one individual (a parameter that is usually specified in an ad-hoc fashion). In any case, the equivalent of this latter parameter would also have to be specified in the above-mentioned "extended version" of the single-attribute-per-individual approach when one wants the GP algorithm to return multiple constructed attributes.

Examples of this multiple-attributes-per-individual approach are the GP algorithms proposed by (Krawiec 2002; Smith & Bull 2003; Firpi et al. 2005). Here we briefly discuss the former two, as examples of this approach. In (Krawiec 2002) each individual encodes a fixed number K of constructed attributes, each of them represented by a tree, so that an individual consists of K trees – where K is a user-specified parameter. The algorithm also includes a method to split the constructed attributes encoded in an individual into two subsets, namely the subset of "evolving" attributes and the subset of "hidden" attributes. The basic idea is that high-quality constructed attributes are considered hidden (or "protected"), so that they cannot be manipulated by the genetic operators such as crossover and mutation. The choice of attributes to be hidden is based on an attribute quality measure. This measure evaluates the quality of each constructed attribute separately, and the best attributes of the individual are considered hidden.

Another example of the multiple-attributes-per-individual approach is the GAP (Genetic Algorithm and Programming) system proposed by (Smith & Bull 2003, 2004). GAP performs both attribute construction and attribute selection. The first stage consists of attribute construction, which is performed by a GP algorithm. As a result of this first stage, the system constructs an extended genotype containing both the constructed attributes represented in the best evolved individual of the GP run and original attributes that have not been used in those constructed attributes. This extended genotype is used as the basic representation for a GA that performs attribute selection, so that the GA searches for the best subset of attributes out of all (both constructed and original) attributes.

Satisfying the Closure Property

GP algorithms for attribute construction have used several different approaches to satisfy the closure property (briefly mentioned in Section 2). This is an important issue, because the chosen approach can have a significant impact on the types (e.g., continuous or nominal) of original attributes processed by the algorithm and on the types of attributes constructed by the algorithm. Let us see some examples.

A simple solution for the closure problem is used in the GAP algorithm (Smith and Bull 2003). Its terminal set contains only the continuous (real-valued) attributes of the data being mined. In addition, its function set consists only of arithmetic operators (+, –, *, %,) – where % denotes protected division, i.e. a division operator that handles zero denominator inputs by returning something different from an error (Banzhaf et al. 1998; Koza 1992) – so that the closure property is immediately satisfied. (Firpi et al. 2005) also uses the approach of having a function set consisting only of mathematical operators, but it uses a considerably larger set of mathematical operators than the set used by (Smith and Bull 2003).

The GP algorithm proposed by (Krawiec 2002) uses a terminal set including all original attributes (both continuous and nominal ones), and a function set consisting of arithmetical operators (+, –, *, %, log), comparison operators (<, >, =), an "IF (conditional expression)", and an "approximate equality operator" which compares its two arguments with tolerance given by the third argument. The algorithm did not enforce data type constraints, which means that expressions encoding the constructed attributes make no distinction between, for instance, continuous and nominal attributes. Values of nominal attributes, such as male and female, are treated as numbers. This helps to solve the closure problem, but at a high price: constructed attributes can contain expressions that make no sense from a semantical point of view. For instance, the algorithm could produce an expression such as "*Gender* + *Age*", because the value of the nominal attribute *Gender* would be interpreted as a number.

The GP proposed by (Otero et al. 2003) uses a terminal set including only the continuous attributes of the data being mined. Its function set consists of arithmetic operators (+, –, *, %,) and comparison operators ($\geq$, $\leq$). In order to satisfy the closure property, the algorithm enforces the data type restriction that the comparison operators can be used only at the root of the GP tree, i.e., they cannot be used as child nodes of other nodes in the tree. The reason is that comparison operators return a Boolean value, which cannot be processed by any operator in the function set (all operators accept only continuous values as input). Note that, although the algorithm can construct attributes only out of the continuous original attributes, the constructed attributes themselves can be either Boolean or continuous. A constructed attribute will be Boolean if its corresponding tree in the GP individual has a comparison operator at the root node; it will be continuous otherwise.

In order to satisfy the closure property, GPCI (Hu 1998) simply transforms all the original attributes into Boolean attributes and uses a function set containing only Boolean functions. For instance, if an attribute A is continuous (real-valued), such as the attribute *Salary*, it is transformed into two Boolean attributes, such as "Is *Salary* $> t$?" and "Is *Salary* $\leq t$?", where t is a threshold automatically chosen by the algorithm in order to maximize the ability of the two new attributes in discriminating between instances of different classes. The two new attributes are named "*positive-A*" and "*negative-A*", respectively. Once every original attribute has been transformed into two Boolean attributes, a GP algorithm is applied to the Boolean attributes. In this GP, the terminal set consists of all the pairs of attributes "*positive-A*" and "*negative-A*" for each original attribute A, whereas the function set consists of the Boolean operators {AND, OR}. Since all terminal symbols are Boolean, and all operators accept Boolean values as input and produce Boolean value as output, the closure property is satisfied.

Table 2 summarizes the main characteristics of the five GP algorithms for attribute construction discussed in this Section.

Table 2. Summary of GP Algorithms for Attribute Construction

Reference	Approach	Individual representation	Datatype of input attrib	Datatype of output attrib
(Hu 1998)	Data preprocessing	Single attribute	Any (attributes are booleanised)	Boolean
(Krawiec 2002)	Interleaving	Multiple attributes	Any (nominal attrib. values are interpreted as numbers)	Continuous
(Otero et al. 2003)	Data preprocessing	Single attribute	Continuous	Continuous or Boolean
(Smith & Bull 2003, 2004)	Interleaving	Multiple attributes	Continuous	Continuous
(Firpi et al. 2005)	Interleaving	Multiple attributes	Continuous	Continuous

6 Multi-Objective Optimization with Evolutionary Algorithms

There are many real-world optimization problems that are naturally expressed as the simultaneous optimization of two or more conflicting objectives (Coello Coello 2002; Deb 2001; Coello Coello & Lamont 2004). A generic example is to maximize the quality of a product and minimize its manufacturing cost in a factory. In the context of data mining, a typical example is, in the data preprocessing task of attribute selection, to minimize the error rate of a classifier trained with the selected attributes and to minimize the number of selected attributes.

The conventional approach to cope with such multi-objective optimization problems using evolutionary algorithms is to convert the problem into a single-optimization problem. This is typically done by using a weighted formula in the fitness function, where each objective has an associated weight reflecting its relative importance. For instance, in the above example of two-objective attribute selection, the fitness function could be defined as, say: "2/3 classification_error + 1/3 Number_of_selected_attributes".

However, this conventional approach has several problems. First, it mixes non-commensurable objectives (classification error and number of selected attributes in the previous example) into the same formula. This has at least the disadvantage that the value returned by the fitness function is not meaningful to the user. Second, note that different weights will lead to different selected attributes, since different weights represent different trade-offs between the two conflicting objectives. Unfortunately, the weights are usually defined in an ad-hoc fashion. Hence, when the EA returns the best attribute subset to the user, the user is presented with a solution that represents just one possible

trade-off between the objectives. The user misses the opportunity to analyze different trade-offs.

Of course we could address this problem by running the EA multiple times, with different weights for the objectives in each run, and return the multiple solutions to the user. However, this would be very inefficient, and we would still have the problems of deciding which weights should be used in each run, how many runs we should perform (and so how many solutions should be returned to the user), etc.

A more principled approach consists of letting an EA answer these questions automatically, by performing a global search in the solution space and discovering as many good solutions, with as much diversity among them, as possible. This can be done by using a multi-objective EA, a kind of EA which has become quite popular in the EA community in the last few years (Deb 2001; Coello Coello 2002; Coello Coello & Lamont 2004). The basic idea involves the concept of Pareto dominance. A solution s_1 is said to dominate, in the Pareto sense, another solution s_2 if and only if solution s_1 is strictly better than s_2 in at least one of the objectives and solution s_1 is not worse than s_2 in any of the objectives. The concept of Pareto dominance is illustrated in Figure 4. This figure involves two objectives to be minimized, namely classification error and number of selected attributes (No_attrib). In that figure, solution D is dominated by solution B (which has both a smaller error and a smaller number of selected attributes than D), and solution E is dominated by solution C. Hence, solutions A, B and C are non-dominated solutions. They constitute the best "Pareto front" found by the algorithm. All these three solutions would be returned to the user.

The goal of a multi-objective EA is to find a Pareto front which is as close as possible to the true (unknown) Pareto front. This involves not only the minimization of the two objectives, but also finding a diverse set of non-dominated solutions, spread along the Pareto front. This allows the EA to return to the user a diverse set of good trade-offs between the conflicting objectives. With this rich information, the user can hopefully make a more intelligent decision, choosing the best solution to be used in practice.

At this point the reader might argue that this approach has the disadvantage that the final choice of the solution to be used depends on the user, characterizing a subjective approach. The response to this is that the knowledge discovery process is interactive (Brachman & Anand 1996; Fayyad et al. 1996), and the participation of the user in this process is important to obtain useful results. The questions are *when and how* the user should participate (Deb 2001; Freitas 2004). In the above-described multi-objective approach, based on Pareto dominance, the user participates by choosing the best solution out of all the non-dominated solutions. This choice is made *a posteriori*, i.e., after the algorithm has run and has returned a rich source of information about the solution space: the discovered Pareto front. In the conventional approach – using an EA with a weighted formula and returning a single solution to the user – the user has to define the weights *a priori*, i.e., before

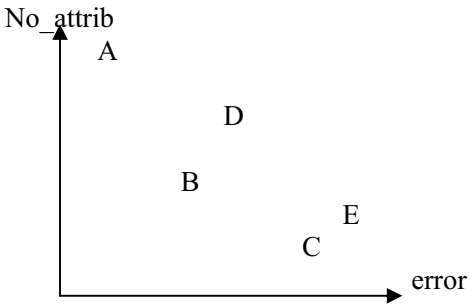


Fig. 4. Example of Pareto dominance

running the algorithm, when the solution space was not explored yet. The multi-objective approach seems to put the user in the loop in a better moment, when valuable information about the solution space is available. The multi-objective approach also avoids the problems of ad-hoc choice of weights, mixing non-commensurable objectives into the same formula, etc.

Table 3 lists the main characteristics of multi-objective EAs for data mining. Most systems included in Table 3 consider only two objectives. The exceptions are the works of (Kim et al. 2000) and (Atkinson-Abutridy et al. 2003), considering 4 and 8 objectives, respectively. Out of the EAs considering only two objectives, the most popular choice of objectives – particularly for EAs addressing the classification task – has been some measure of classification accuracy (or its dual, error) and a measure of the size of the classification model (number of leaf nodes in a decision tree or total number of rule conditions – attribute-value pairs – in all rules). Note that the size of a model is typically used as a proxy for the concept of "simplicity" of that model, even though arguably this proxy leaves a lot to be desired as a measure of a model's simplicity (Pazzani 2000; Freitas 2006). (In practice, however, it seems no better proxy for a model's simplicity is known.) Note also that, when the task being solved is attribute selection for classification, the objective related to size can be the number of selected attributes, as in (Emmanouilidis et al. 2000), or the size of the classification model built from the set of selected attributes, as in (Pappa et al. 2002, 2004). Finally, when solving the clustering task a popular choice of objective has been some measure of intra-cluster distance, related to the total distance between each data instance and the centroid of its cluster, computed for all data instances in all the clusters. The number of clusters is also used as an objective in two out of the three EAs for clustering included in Table 3. A further discussion of multi-objective optimization in the context of data mining in general (not focusing on EAs) is presented in (Freitas 2004; Jin 2006).

Table 3. Main characteristics of multi-objective EAs for data mining

Reference	Data mining task	Objectives being Optimized
(Emmanouilidis et al. 2000)	attribute selection for classification	accuracy, number of selected attributes
(Pappa et al 2002, 2004)	attribute selection for classification	accuracy, number of leafs in decision tree
(Ishibuchi & Namba 2004)	selection of classification rules	error, number of rule conditions (in all rules)
(de la Iglesia 2007)	selection of classification rules	confidence, coverage
(Kim et al. 2004)	classification	error, number of leafs in decision tree
(Atkinson-Abutridy et al. 2003)	text mining	8 criteria for evaluating explanatory knowledge across text documents
(Kim et al. 2000)	attribute selection for clustering	Cluster cohesiveness, separation between clusters, number of clusters, number of selected attributes
(Handl & Knowles 2004)	clustering	Intra-cluster deviation and connectivity
(Korkmaz et al. 2006)	clustering	Intra-cluster variance and number of clusters

7 Conclusions

This chapter started with the remark that EAs are a very generic search paradigm. Indeed, the chapter discussed how EAs can be used to solve several different data mining tasks, namely the discovery of classification rules, clustering, attribute selection and attribute construction. The discussion focused mainly on the issues of individual representation and fitness function for each of these tasks, since these are the two EA-design issues that are more dependent of the task being solved. In any case, recall that the design of an EA also involves the issue of genetic operators. Ideally these three components – individual representation, fitness function and genetic operators – should be designed in a synergistic fashion and tailored to the data mining task being solved.

There are at least two motivations for using EAs in data mining, broadly speaking. First, as mentioned earlier, EAs are robust, adaptive search methods that perform a global search in the solution space. This is in contrast to other data mining paradigms that typically perform a greedy search. In the context of data mining, the global search of EAs is associated with a better ability to cope with attribute interactions. For instance, most "conventional", non-

evolutionary rule induction algorithms are greedy, and therefore quite sensitive to the problem of attribute interaction. EAs can use the same knowledge representation (IF-THEN rules) as conventional rule induction algorithms, but their global search tends to cope better with attribute interaction and to discover interesting relationships that would be missed by a greedy search (Dhar et al. 2000; Papagelis & Kalles 2001; Freitas 2002a).

Second, EAs are a very flexible algorithmic paradigm. In particular, borrowing some terminology from programming languages, EAs have a certain "declarative" – rather than "procedural" – style. The quality of an individual (candidate solution) is evaluated, by a fitness function, in a way independent of how that solution was constructed. This gives the data miner a considerable freedom in the design of the individual representation, the fitness function and the genetic operators. This flexibility can be used to incorporate background knowledge into the EA and/or to hybridize EAs with local search methods that are specifically tailored to the data mining task being solved.

Note that declarativeness is a matter of degree, rather than a binary concept. In practice EAs are not 100% declarative, because as one changes the fitness function one might consider changing the individual representation and the genetic operators accordingly, in order to achieve the above-mentioned synergistic relationship between these three components of the EA. However, EAs still have a degree of declarativeness considerably higher than other data mining paradigms. For instance, as discussed in Subsection 3.3, the fact that EAs evaluate a complete (rather than partial) rule allows the fitness function to consider several different rule-quality criteria, such as comprehensibility, surprisingness and subjective interestingness to the user. In EAs these quality criteria can be directly considered during the search for rules. By contrast, in conventional, greedy rule induction algorithms – where the evaluation function typically evaluates a partial rule – those quality criteria would typically have to be considered in a post-processing phase of the knowledge discovery process, when it might be too late. After all, many rule set post-processing methods just try to select the most interesting rules out of all discovered rules, so that interesting rules that were missed by the rule induction method will remain missing after applying the post-processing method.

Like any other data mining paradigm, EAs also have some disadvantages. One of them is that conventional genetic operators – such as conventional crossover and mutation operators – are "blind" search operators in the sense that they modify individuals (candidate solutions) in a way independent from the individual's fitness (quality). This characteristic of conventional genetic operators increases the generality of EAs, but intuitively tends to reduce their effectiveness in solving a specific kind of problem. Hence, in general it is important to modify or extend EAs to use task specific-operators.

Another disadvantage of EAs is that they are computationally slow, by comparison with greedy search methods. The importance of this drawback depends on many factors, such as the kind of task being performed, the size of the data being mined, the requirements of the user, etc. Note that in some

cases a relatively long processing time might be acceptable. In particular, several data mining tasks, such as classification, are typically an off-line task, and the time spent solving that task is usually less than 20% of the total time of the knowledge discovery process. In scenarios like this, even a processing time of hours or days might be acceptable to the user, at least in the sense that it is not the bottleneck of the knowledge discovery process.

In any case, if necessary the processing time of an EA can be significantly reduced by using special techniques. One possibility is to use parallel processing techniques, since EAs can be easily parallelized in an effective way (Cantu-Paz 2000; Freitas & Lavington 1998; Freitas 2002a). Another possibility is to compute the fitness of individuals by using only a subset of training instances – where that subset can be chosen either at random or using adaptive instance-selection techniques (Bhattacharyya 1998; Gathercole & Ross 1997; Sharpe & Glover 1999; Freitas 2002a).

An important research direction is to better exploit the power of Genetic Programming (GP) in data mining. Several GP algorithms for attribute construction were discussed in Subsection 5.2, and there are also several GP algorithms for discovering classification rules (Freitas 2002a; Wong & Leung 2000) or for classification in general (Muni et al. 2004; Song et al. 2005; Folino et al. 2006). However, the power of GP is still underexplored. Recall that the GP paradigm was designed to *automatically* discover computer *programs*, or *algorithms*, which should be *generic "recipes"* for solving a given kind of problem, and not to find the solution to one particular instance of that problem (like in most EAs). For instance, classification is a kind of problem, and most classification-rule induction algorithms are generic enough to be applied to different data sets (each data set can be considered just an instance of the kind of problem defined by the classification task). However, these generic rule induction algorithms have been *manually* designed by a human being. Almost all current GP algorithms for classification-rule induction are competing with conventional (greedy, non-evolutionary) rule induction algorithms, in the sense that both GP and conventional rule induction algorithms are discovering classification rules for a single data set at a time. Hence, the output of a GP for classification-rule induction is a set of rules for a given data set, which can be called a "program" or "algorithm" only in a very loose sense of these words.

A much more ambitious goal, which is more compatible with the general goal of GP, is to use GP to *automatically* discover a rule induction *algorithm*. That is, to perform *algorithm induction*, rather than rule induction. The first version of a GP algorithm addressing this ambitious task has been proposed in (Pappa & Freitas 2006), and an extended version of that work is described in detail in another chapter of this book (Pappa & Freitas 2007).

References

Aldenderfer MS & Blashfield RK (1984) *Cluster Analysis* (Sage University Paper Series on Quantitative Applications in the Social Sciences, No. 44) Sage Publications.

Atkinson-Abutridy J, Mellishm C, and Aitken S (2003) A semantically guided and domain-independent evolutionary model for knowledge discovery from texts. *IEEE Trans. Evolutionary Computation 7(6)*, 546-560.

Bacardit J, Goldberg DE, Butz MV, Llora X, Garrell JM (2004). Speeding-up Pittsburgh learning classifier systems: modeling time and accuracy. *Proc. Parallel Problem Solving From Nature (PPSN-2004), LNCS 3242,* 1021-1031, Springer.

Bacardit J and Krasnogor N (2006) Smart crossover operator with multiple parents for a Pittsburgh learning classifier system. *Proc. Genetic & Evolutionary Computation Conf. (GECCO-2006)*, 1441-1448. Morgan Kaufmann.

Backer E (1995) *Computer-Assisted Reasoning in Cluster Analysis*. Prentice-Hall.

Back T, Fogel DB and Michalewicz (Eds.) (2000) *Evolutionary Computation 1: Basic Algorithms and Operators*. Institute of Physics Publishing.

Bala J, De Jong K, Huang J, Vafaie H and Wechsler H (1995) Hybrid learning using genetic algorithms and decision trees for pattern classification. *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI-95)*, 719-724.

Bala J, De Jong K, Huang J, Vafaie H and Wechsler H (1996) Using learning to facilitate the evolution of features for recognizing visual concepts. *Evolutionary Computation 4(3)*: 297-312.

Banzhaf W (2000) Interactive evolution. In: T. Back, D.B. Fogel and T. Michalewicz (Eds.) *Evolutionary Computation 1*, 228-236. Institute of Physics Pub.

Banzhaf W, Nordin P, Keller RE, and Francone FD (1998) *Genetic Programming ∼ an Introduction: On the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann.

Bhattacharrya S (1998) Direct marketing response models using genetic algorithms. *Proceedings of the 4th Int. Conf. on Knowledge Discovery and Data Mining (KDD-98)*, 144-148. AAAI Press.

Brachman RJ and Anand T. (1996) The process of knowledge discovery in databases: a human-centered approach. In: U.M. Fayyad et al (Eds.) *Advances. in Knowledge Discovery and Data Mining*, 37-58. AAAI/MIT.

Bull L (Ed.) (2004) *Applications of Learning Classifier Systems*. Springer.

Bull L and Kovacs T (Eds.) (2005) *Foundations of Learning Classifier Systems*. Springer.

Cantu-Paz E (2000) *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer.

Caruana R and Niculescu-Mizil A (2004) Data mining in metric space: an empirical analysis of supervised learning performance criteria. *Proc. 2004 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD-04)*, ACM.

Carvalho DR and Freitas AA (2004). A hybrid decision tree/genetic algorithm method for data mining. *Special issue on Soft Computing Data Mining, Information Sciences 163(1-3)*, pp. 13-35. 14 June 2004.

Chen S, Guerra-Salcedo C and Smith SF (1999) Non-standard crossover for a standard representation - commonality-based feature subset selection. *Proc. Genetic and Evolutionary Computation Conf. (GECCO-99)*, 129-134. Morgan Kaufmann.

Cherkauer KJ and Shavlik JW (1996). Growing simpler decision trees to facilitate knowledge discovery. *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD-96)*, 315-318. AAAI Press.

Coello Coello CA, Van Veldhuizen DA and Lamont GB (2002) *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer.

Coello Coello CA and Lamont GB (Ed.) (2004) *Applications of Multi-objective Evolutionary Algorithms*. World Scientific.

Deb K (2001) *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley.

Deb K and Goldberg DE (1989). An investigation of niche and species formation in genetic function optimization. *Proc. 2nd Int. Conf. Genetic Algorithms (ICGA-89)*, 42-49.

De Jong K (2006) *Evolutionary Computation: a unified approach*. MIT.

De la Iglesia B (2007) Application of multi-objective metaheuristic algorithms in data mining. *Proc. 3rd UK Knowledge Discovery and Data Mining Symposium (UKKDD-2007)*, 39-44, University of Kent, UK, April 2007.

Dhar V, Chou D and Provost F (2000). Discovering interesting patterns for investment decision making with GLOWER – a genetic learner overlaid with entropy reduction. *Data Mining and Knowledge Discovery 4(4)*, 251-280.

Divina F (2005) Assessing the effectiveness of incorporating knowledge in an evolutionary concept learner. *Proc. EuroGP-2005 (European Conf. on Genetic Programming), LNCS 3447,* 13-24, Springer.

Divina F & Marchiori E (2002) Evolutionary Concept Learning. *Proc. Genetic & Evolutionary Computation Conf. (GECCO-2002)*, 343-350. Morgan Kaufmann.

Divina F & Marchiori E (2005) Handling continuous attributes in an evolutionary inductive learner. *IEEE Trans. Evolutionary Computation, 9(1),* 31-43, Feb. 2005.

Eiben AE and Smith JE (2003) *Introduction to Evolutionary Computing*. Springer.

Emmanouilidis C, Hunter A and J. MacIntyre J (2000) A multiobjective evolutionary setting for feature selection and a commonality-based crossover operator. *Proc. 2000 Congress on Evolutionary Computation (CEC-2000)*, 309-316. IEEE.

Emmanouilidis C (2002) Evolutionary multi-objective feature selection and ROC analysis with application to industrial machinery fault diagnosis. In: K. Giannakoglou et al. (Eds.) *Evolutionary Methods for Design, Optimisation and Control*. Barcelona: CIMNE.

Estivill-Castro V and Murray AT (1997) Spatial clustering for data mining with genetic algorithms. *Tech. Report FIT-TR-97-10*. Queensland University of Technology. Australia.

Falkenauer E (1998) *Genetic Algorithms and Grouping Problems*. John-Wiley & Sons.

Fayyad UM, Piatetsky-Shapiro G and Smyth P (1996) From data mining to knowledge discovery: an overview. In: U.M. Fayyad et al (Eds.) *Advances in Knowledge Discovery and Data Mining*, 1-34. AAAI/MIT.

Firpi H, Goodman E, Echauz J (2005) On prediction of epileptic seizures by computing multiple genetic programming artificial features. *Proc. 2005 European Conf. on Genetic Programming (EuroGP-2005), LNCS 3447,* 321-330. Springer.

Folino G, Pizzuti C and Spezzano G (2006) GP ensembles for large-scale data classification. *IEEE Trans. Evolutionary Computation 10(5)*, 604-616, Oct. 2006.

Freitas AA and. Lavington SH (1998) *Mining Very Large Databases with Parallel Processing.* Kluwer.

Freitas AA (2001) Understanding the crucial role of attribute interaction in data mining. *Artificial Intelligence Review 16(3)*, 177-199.

Freitas AA (2002a) *Data Mining and Knowledge Discovery with Evolutionary Algorithms.* Springer.

Freitas AA (2002b) A survey of evolutionary algorithms for data mining and knowledge discovery. In: A. Ghosh and S. Tsutsui. (Eds.) *Advances in Evolutionary Computation*, pp. 819-845. Springer-Verlag.

Freitas AA (2002c). Evolutionary Computation. In: W. Klosgen and J. Zytkow (Eds.) *Handbook of Data Mining and Knowledge Discovery*, pp. 698-706.Oxford Univ. Press.

Freitas AA (2004) A critical review of multi-objective optimization in data mining: a position paper. *ACM SIGKDD Explorations, 6(2),* 77-86, Dec. 2004.

Freitas AA (2005) Evolutionary Algorithms for Data Mining. In: O. Maimon and L. Rokach (Eds.) *The Data Mining and Knowledge Discovery Handbook*, pp. 435-467. Springer.

Freitas AA (2006) Are we really discovering "interesting" knowledge from data? *Expert Update, Vol. 9, No. 1,* 41-47, Autumn 2006.

Furnkranz J and Flach PA (2003). An analysis of rule evaluation metrics. *Proc.20th Int. Conf. Machine Learning (ICML-2003).* Morgan Kaufmann.

Gathercole C and Ross P (1997) Tackling the Boolean even N parity problem with genetic programming and limited-error fitness. *Genetic Programming 1997: Proc. 2nd Conf. (GP-97)*, 119-127. Morgan Kaufmann.

Ghozeil A and Fogel DB (1996) Discovering patterns in spatial data using evolutionary programming. *Genetic Programming 1996: Proceedings of the 1st Annual Conf.*, 521-527. MIT Press.

Giordana A, Saitta L, Zini F (2004) Learning disjunctive concepts by means of genetic algorithms. *Proc. 10th Int. Conf. Machine Learning (ML-94)*, 96-104. Morgan Kaufmann.

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

Goldberg DE and Richardson J (1987) Genetic algorithms with sharing for multimodal function optimization. *Proc. Int. Conf. Genetic Algorithms (ICGA-87)*, 41-49.

Guerra-Salcedo C and Whitley D (1998) Genetic search for feature subset selection: a comparison between CHC and GENESIS. *Genetic Programming 1998: Proc. 3rd Annual Conf.*, 504-509. Morgan Kaufmann.

Guerra-Salcedo C, Chen S, Whitley D, and Smith S (1999) Fast and accurate feature selection using hybrid genetic strategies. *Proc. Congress on Evolutionary Computation (CEC-99)*, 177-184. IEEE.

Guyon I and Elisseeff A (2003) An introduction to variable and feature selection. *Journal of Machine Learning Research 3*, 1157-1182.

Hall LO, Ozyurt IB, Bezdek JC (1999) Clustering with a genetically optimized approach. *IEEE Trans. on Evolutionary Computation 3(2)*, 103-112.

Hand DJ (1997) *Construction and Assessment of Classification Rules.* Wiley.

Handl J and Knowles J (2004) Evolutionary multiobjective clustering. *Proc. Parallel Problem Solving From Nature (PPSN-2004), LNCS 3242,* 1081-1091, Springer.

Hekanaho J (1995) Symbiosis in multimodal concept learning. *Proc. 1995 Int. Conf. on Machine Learning (ML-95)*, 278-285. Morgan Kaufmann.

Hekanaho J (1996) Testing different sharing methods in concept learning. *TUCS Technical Report No. 71.* Turku Centre for Computer Science, Finland.

Hirsch L, Saeedi M and Hirsch R (2005) Evolving rules for document classification. *Proc. 2005 European Conf. on Genetic Programming (EuroGP-2005)*, *LNCS 3447,* 85-95, Springer.

Hu YJ (1998). A genetic programming approach to constructive induction. *Genetic Programming 1998: Proc. 3rd Annual Conf.*, 146-151. Morgan Kaufmann.

Ishibuchi H and Nakashima T (2000) Multi-objective pattern and feature selection by a genetic algorithm. *Proc. 2000 Genetic and Evolutionary Computation Conf. (GECCO-2000)*, 1069-1076. Morgan Kaufmann.

Ishibuchi H and Namba S (2004) Evolutionary multiobjective knowledge extraction for high-dimensional pattern classification problems. *Proc. Parallel Problem Solving From Nature (PPSN-2004), LNCS 3242,* 1123-1132, Springer.

Jiao L, Liu J and Zhong W (2006) An organizational coevolutionary algorithm for classification. *IEEE Trans. Evolutionary Computation, Vol. 10, No. 1,* 67-80, Feb. 2006.

Jin, Y (Ed.) (2006) *Multi-Objective Machine Learning.* Springer.

Jong K, Marchiori E and Sebag M (2004) Ensemble learning with evolutionary computation: application to feature ranking. *Proc. Parallel Problem Solving from Nature VIII (PPSN-2004), LNCS 3242*, 1133-1142. Springer, 2004.

Jourdan L, Dhaenens-Flipo C and Talbi EG (2003) Discovery of genetic and environmental interactions in disease data using evolutionary computation. In: G.B. Fogel and D.W. Corne (Eds.) *Evolutionary Computation in Bioinformatics*, 297-316. Morgan Kaufmann.

Kim Y, Street WN and Menczer F (2000) Feature selection in unsupervised learning via evolutionary search. *Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD-2000)*, 365-369. ACM.

Kim D (2004). Structural risk minimization on decision trees: using an evolutionary multiobjective algorithm. *Proc. 2004 European Conference on Genetic Programming (EuroGP-2004), LNCS 3003,* 338-348, Springer.

Korkmaz EE, Du J, Alhajj R and Barker (2006) Combining advantages of new chromosome representation scheme and multi-objective genetic algorithms for better clustering. *Intelligent Data Analysis 10* (2006),163-182.

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

Krawiec K (2002) Genetic programming-based construction of features for machine learning and knowledge discovery tasks. *Genetic Programming and Evolvable Machines 3(4)*, 329-344.

Krsihma K and Murty MN (1999) Genetic k-means algorithm. *IEEE Transactions on Systems, Man and Cyberneics - Part B: Cybernetics,* 29(3), 433-439.

Krzanowski WJ and Marriot FHC (1995) *Kendall's Library of Statistics 2: Multivariate Analysis - Part 2. Chapter 10 - Cluster Analysis*, pp. 61-94.London: Arnold.

Kudo M and Sklansky J (2000) Comparison of algorithms that select features for pattern classifiers. *Pattern Recognition 33(2000)*, 25-41.

Liu JJ and Kwok JTY (2000) An extended genetic rule induction algorithm. *Proc. 2000 Congress on Evolutionary Computation (CEC-2000).* IEEE.

Liu H and Motoda H (1998) *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer.

Liu B, Hsu W and Chen S (1997) Using general impressions to analyze discovered classification rules. *Proc. 3rd Int. Conf. on Knowledge Discovery and Data Mining (KDD-97)*, 31-36. AAAI Press.

Llora X and Garrell J (2003) Prototype induction and attribute selection via evolutionary algorithms. *Intelligent Data Analysis 7*, 193-208.

Miller MT, Jerebko AK, Malley JD, Summers RM (2003) Feature selection for computer-aided polyp detection using genetic algorithms. *Medical Imaging 2003: Physiology and Function: methods, systems and applications.* Proc. SPIE Vol. 5031.

Moser A and Murty MN (2000) On the scalability of genetic algorithms to very large-scale feature selection. *Proc. Real-World Applications of Evolutionary Computing (EvoWorkshops 2000). LNCS 1803*, 77-86. Springer.

Muharram MA and Smith GD (2004) Evolutionary feature construction using information gain and gene index. *Genetic Programming: Proc. 7th European Conf. (EuroGP-2003), LNCS 3003*, 379-388. Springer.

Muni DP, Pal NR and Das J (2004) A novel approach to design classifiers using genetic programming. *IEEE Trans. Evolutionary Computation 8(2)*, 183-196, April 2004.

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

Ni B and Liu J (2004) A novel method of searching the microarray data for the best gene subsets by using a genetic algorithms. *Proc. Parallel Problem Solving From Nature (PPSN-2004), LNCS 3242,* 1153-1162, Springer.

Otero FB, Silva MMS, Freitas AA and Nievola JC (2003) Genetic programming for attribute construction in data mining. *Genetic Programming: Proc. EuroGP-2003, LNCS 2610*, 384-393. Springer.

Papagelis A and Kalles D (2001) Breeding decision trees using evolutionary techniques. *Proc. 18th Int. Conf. Machine Learning (ICML-2001),* 393-400. Morgan Kaufmann.

Pappa GL and Freitas AA (2006) Automatically evolving rule induction algorithms. *Machine Learning: ECML 2006 – Proc. of the 17th European Conf. on Machine Learning, LNAI 4212,* 341-352. Springer.

Pappa GL and Freitas AA (2007) Discovering new rule induction algorithms with grammar-based genetic programming. *Maimon O and Rokach L (Eds.) Soft Computing for Knowledge Discovery and Data Mining*. Springer.

Pappa GL, Freitas AA and Kaestner CAA (2002) A multiobjective genetic algorithm for attribute selection. *Proc. 4th Int. Conf. On Recent Advances in Soft Computing (RASC-2002)*, 116-121. Nottingham Trent University, UK.

Pappa GL, Freitas AA and Kaestner CAA (2004) Multi-Objective Algorithms for Attribute Selection in Data Mining. In: Coello Coello CA and Lamont GB (Ed.) *Applications of Multi-objective Evolutionary Algorithms*, 603-626. World Scientific.

Pazzani MJ (2000) Knowledge discovery from data, *IEEE Intelligent Systems,* 10-13, Mar./Apr. 2000.

Quinlan JR. (1993) *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Romao W, Freitas AA and Pacheco RCS (2002) A Genetic Algorithm for Discovering Interesting Fuzzy Prediction Rules: applications to science and technology

data. *Proc. Genetic and Evolutionary Computation Conf. (GECCO-2002)*, pp. 1188-1195. Morgan Kaufmann.

Romao W, Freitas AA, Gimenes IMS (2004) Discovering interesting knowledge from a science and technology database with a genetic algorithm. *Applied Soft Computing 4(2)*, pp. 121-137.

Rozsypal A and Kubat M (2003) Selecting representative examples and attributes by a genetic algorithm. *Intelligent Data Analysis 7*, 290-304.

Sarafis I (2005) *Data mining clustering of high dimensional databases with evolutionary algorithms*. PhD Thesis, School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, UK.

Sharpe PK and Glover RP (1999) Efficient GA based techniques for classification. *Applied Intelligence 11*, 277-284.

Smith RE (2000) Learning classifier systems. In: T. Back, D.B. Fogel and T. Michalewicz (Eds.) *Evolutionary Computation 1: Basic Algorithms and Operators*, 114-123. Institute of Physics Publishing.

Smith MG and Bull L (2003) Feature construction and selection using genetic programming and a genetic algorithm. *Genetic Programming: Proc. EuroGP-2003, LNCS 2610*, 229-237. Springer.

Smith MG and Bull L (2004) Using genetic programming for feature creation with a genetic algorithm feature selector. *Proc. Parallel Problem Solving From Nature (PPSN-2004), LNCS 3242,* 1163-1171, Springer.

Song D, Heywood MI and Zincir-Heywood AN (2005) Training genetic programming on half a million patterns: an example from anomaly detection. *IEEE Trans. Evolutionary Computation 9(3)*, 225-239, June 2005.

Srikanth R, George R, Warsi N, Prabhu D, Petry FE, Buckles B (1995) A variable-length genetic algorithm for clustering and classification. *Pattern Recognition Letters 16(8)*, 789-800.

Tan PN, Steinbach M and Kumar V (2006) *Introduction to Data Mining*. Addison-Wesley.

Terano T and Ishino Y (1998) Interactive genetic algorithm based feature selection and its application to marketing data analysis. In: Liu H and Motoda H (Eds.) *Feature Extraction, Construction and Selection: a data mining perspective*, 393-406. Kluwer.

Terano T and Inada M (2002) Data mining from clinical data using interactive evolutionary computation. In: A. Ghosh and S. Tsutsui (Eds.) *Advances in Evolutionary Computing: theory and applications*, 847-861. Springer.

Vafaie H and De Jong K (1998) Evolutionary Feature Space Transformation. In: H. Liu and H. Motoda (Eds.) *Feature Extraction, Construction and Selection*, 307-323. Kluwer.

Witten IH and Frank E (2005) *Data Mining: practical machine learning tools and techniques* . 2nd Ed. Morgan Kaufmann.

Wong ML and Leung KS (2000) *Data Mining Using Grammar Based Genetic Programming and Applications*. Kluwer.

Yang J and Honavar V (1997) Feature subset selection using a genetic algorithm. *Genetic Programming 1997: Proc. 2nd Annual Conf. (GP-97)*, 380-385. Morgan Kaufmann.

Yang J and Honavar V (1998) Feature subset selection using a genetic algorithm. In: Liu, H. and Motoda, H (Eds.) *Feature Extraction, Construction and Selection*, 117-136. Kluwer.

Zhang P, Verma B, Kumar K (2003) Neural vs. Statistical classifier in conjunction with genetic algorithm feature selection in digital mammography. *Proc. Congress on Evolutionary Computation (CEC-2003)*. IEEE Press.

Zhou C, Xiao W, Tirpak TM and Nelson PC (2003) Evolving accurate and compact classification rules with gene expression programming. *IEEE Trans. on Evolutionary Computation 7(6)*, 519-531.