14

GA-Based Pareto Optimization for Rule Extraction from Neural Networks

Urszula Markowska-Kaczmar, Krystyna Mularczyk

Wroclaw University of Technology,
Institute of Applied Informatics
Wyb. Wyspianskiego 27, 50-370 Wroclaw, Poland
urszula.markowska-kaczmar@pwr.wroc.pl

Summary. The chapter presents a new method of rule extraction from trained neural networks, based on a hierarchical multiobjective genetic algorithm. The problems associated with rule extraction, especially its multiobjective nature, are described in detail, and techniques used when approaching them with genetic algorithms are presented. The main part of the chapter contains a thorough description of the proposed method. It is followed by a discussion of the results of experimental study performed on popular benchmark datasets that confirm the method's effectiveness.

14.1 Introduction

For many years neural networks have been successfully used for solving various complicated problems. The areas of their applications include communication systems, signal processing, the theory of control, pattern and speech recognition, weather prediction and medicine. Neural networks are used especially in situations when algorithms for a given problem are unknown or too complex. In order to solve a problem, a network does not need an algorithm; instead, it must be provided with training examples that enable it to learn the correct solutions. The most commonly used neural networks have the input and output data presented in the form of vectors. Such a pair of vectors is an example of the relationship that a network should learn. The training consists of a certain iterative process of modifying the network's parameters, so that the answer given by the network for each of the input patterns is as close to the desired one as possible.

The unquestionable advantage of neural networks is their ability of generalization, which consists in giving correct answers for new input patterns that had not been used for training. Their resistance to errors is equally important. This means that networks can produce the right answers also in the case of noisy or missing data. Finally, it should be emphasized that they are fast while processing data and relatively cheap to build. Nevertheless, a serious disadvantage is that neural networks produce answers in a way that is incomprehensible for humans. Because in certain areas of applications, for instance in medicine, the trustworthiness of a system that supports people's work is essential, the domain of extracting knowledge from neural networks has been developed since the 90s [1]. This knowledge describes in an understandable way the performance of a neural network that solves a given problem. Knowledge extraction may also be perceived as a tool of verifying what has actually been learned by a network.

Rapid development of data storage techniques has been observed in the last years. Data is presently perceived as a source of knowledge and exploited by a dynamically evolved area of computer science called *data mining*. Its methods aim at finding new and correct patterns in the processed data. The problems that data mining deals with include classification, regression, grouping and characteristics. The majority of them may be approached by means of neural networks. In this case knowledge extraction from a network could make its behavior more trustworthy.

The discovered knowledge, if presented in a comprehensible way, could be used for building expert systems as a source alternative or supplementary to the knowledge obtained from human experts. Rule extraction from neural networks might be used for this purpose as well as perceived as a method of automatic rule extraction for an expert system. The system itself could be built as a hybrid consisting of a neural network and a set of rules that describe its performance. Such a hybrid would combine the advantages of both components; it would work fast, which is typical of neural networks, and, on the other hand, it would offer a possibility of explaining the way of reaching a particular conclusion, which is the main quality of expert systems. Moreover, if the data changed, the system's reconstruction could be performed relatively quickly by repeating the process of training the network and extracting knowledge.

Skeptics could ask why knowledge should be extracted from a network and not directly from data. Networks' ability to deal with noisy data should be emphasized again. It tends to be much easier to extract knowledge after having the data processed by a network.

However, if we imagine a network with dozens of inputs with various types, the problem of finding restrictions that describe the conditions of belonging to a given class becomes NP–hard. Searching such a vast space of solutions may be efficiently performed by genetic algorithms – another nature–based technique, which became the basis for a method of rule extraction called MulGEx (**Mul**tiobjectve **G**enetic **Ex**tractor) that will be presented in this chapter.

14.2 The State-of-the-Art in Rule Extraction from Neural Networks

The research on effective methods of acquiring knowledge from neural networks has been carried out for 15 years, which testifies not only to the importance of the problem, but also to its complexity.

The majority of methods apply to networks that perform the classification task, although works concerning rule extraction for the regression task appear as well [17]. The developed approaches may be divided into three main categories: global, local and mixed. This taxonomy bases on the degree to which a method examines the structure of a network.

Global methods treat a network as a black box, observing only its inputs and responses produced at the outputs. In other words, a network provides the method with training patterns. The examples include VIA [21] that uses a procedure similar to classical sensibility analysis, BIO-RE [20] that applies truth tables to extract rules, Ruleneg [8], where an adaptation of PAC algorithm is applied, or [15], based on inversion. It is worth mentioning that in such approaches the architecture of a neural network is insignificant. Other algorithms in this group treat the rule extraction as a machine learning task where neural network delivers training patterns. In this case different machine learning algorithms are used, for example genetic algorithms or decision tree methods.

The opposite approach is the local one, where the first stage consists of describing the conditions of a single neuron's activation, i.e., in determining the values of its inputs that produce an output equal to 1. Such rules are created for all neurons and concatenated on the basis of mutual dependencies. Thus we obtain rules that describe the relations between the inputs and outputs for the entire network. As examples one may mention Partial RE [20], M of N [7], Full RE [20], RULEX [2]. The problem of rule extraction by means of the local approach is simple if the network is relatively small. Otherwise different approaches are developed in order to reduce the architecture of a neural network. Some methods group the hidden neurons' activations, substituting a cluster of neurons by one neuron, other optimize the structure of a neural network introducing a special training procedure or using genetic algorithms [16], [10].

The last group encompasses mixed methods that combine the two approaches described above.

Most of the methods concern multilayer perceptrons (MLP networks). However, methods dedicated to other networks are developed as well, e.g., [18], [5], [9].

Knowledge acquired from neural networks is represented as crisp prepositional rules. Fuzzy rules [14], first order rules and decision trees are used, too.

There are many criteria of the evaluation of the extracted rules' quality. The most frequently used include:

- fidelity,
- accuracy,
- consistency,
- comprehensibility.

In [1] these four requirements are abbreviated to FACC. *Accuracy* is determined on the basis of the number of previously unseen patterns that have been correctly classified. *Fidelity* stands for the degree to which the rules reflect the behavior of the network they have been extracted from. *Consistency* occurs if, during different training sessions, the network produces sets of rules that classify unseen patterns in the same way. *Comprehensibility* is defined as the ratio of the number of rules to the number of premises in a single rule. These criteria are discussed in detail by Ghosh and Taha in [20].

In real applications not all of them may be taken into account and their weight may be different. The main problem in rule extraction is that these criteria, especially fidelity and comprehensibility, tend to be contradictory. The least complex sets, consisting of few rules, cover usually only the most typical cases that are represented by large numbers of training patterns. If we want to improve a given set's fidelity, we must add new rules that would deal with the remaining patterns and exceptions, thus making the set more complicated and less understandable. Therefore rule extraction requires finding a compromise between different criteria, since their simultaneous optimization is practically infeasible. A good algorithm of rule extraction should have the following properties [21]:

- independence from the architecture of the network,
- no restrictions on the process of a network's training,
- guaranteed correctness of obtained results,
- a mechanism of accurate description of a network's performance.

A method that would meet all these requirements (or at least the vast majority) has not been developed yet. Some of the methods are applicable to enumerable or real data only, some require repeating the process of training or changing the network's architecture, some require providing a default rule that is used if no other rule can be applied in a given case. The methods differ in computational complexity (that is not specified in most cases). Hence the necessity of developing new methods. This work is an attempt to fill this gap for a network that solves the problem of classification.

14.3 Problem Formulation

The presented method of extracting rules from a neural network belongs to the group of global methods. It treats the network as a black box (Fig. 14.1)that provides it with training patterns. For this reason the architecture of a network, i.e., the way of connecting individual neurons, is insignificant.

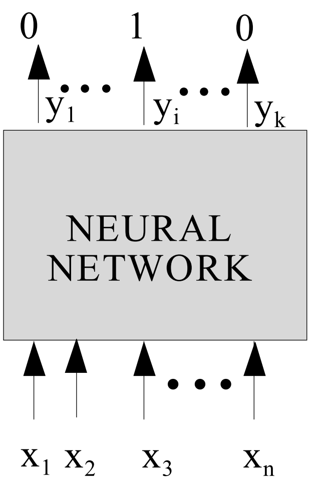

Fig. 14.1. Neural network as a black box.

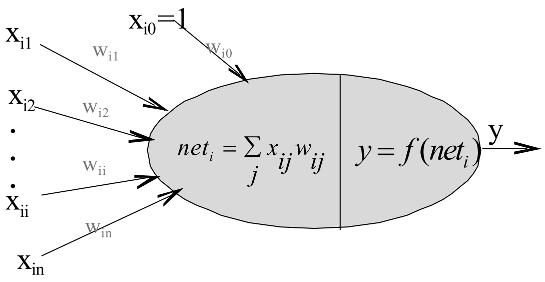

Fig. 14.2. Model of neuron.

Every neuron (Fig. 14.2) in a neural network performs simple operations of addition and multiplication by calculating its total input (for the i-th neuron $net_i = \sum x_{ij} w_{ij}$, where x_{ij} is the signal on the j-th input and w_{ij} is the weight of this connection), which is followed by applying the activation function $f(net)$ and producing the neuron's output.

However, because of a large number of neurons in a network and the parallelism of processing, it is difficult to describe clearly how a network solves a problem. Broadly speaking, the knowledge of this problem is encoded in the network's architecture, weights and activation functions of individual neurons.

Let's assume that a network solves a classification problem. This resolves itself into dividing objects (patterns) into k mutually separable classes C_1, C_2, ..., C_k.

Every p-th pattern will be described as an input vector $x_p = [x_{p1}, x_{p2}, ...,$ $x_{pm}]$ presented to the network. After the training the network's output indicates the class that the pattern belongs to. The class is encoded using the

"1 of k" rule, i.e., only one of k outputs may produce the value 1 and the remaining ones must be equal to 0.

Our goal is to describe the performance of such a network in a comprehensible way. Undoubtedly, the most popular way of representing the extracted knowledge is the 0-order logic, i.e., rules of the following form:

$$IF\, prem_1\, AND\, prem_2 ... prem_n\ THEN\, class_v, \tag{14.1}$$

A single premise $prem_i$ determines the constraints that must be imposed on the i-th input so that the network may classify the pattern into the class included in the conclusion. Each premise in the left part of the rule corresponds to a constraint imposed on the i-th attribute X_i. Depending on the type of the attribute, a constraint is defined in one of the following ways:

- *For a real type of attribute* (discrete and continuous) it introduces the bounds of the range of acceptable values, i.e., $X_i \in [Value_{min}, Value_{max}]$.
- *For enumerable attributes* – a constraint represents a subset of permissible values, i.e., $X_i = Value_1\, OR\ X_i = Value_2\ OR\ ...X_i = Value_k$, where $Value_j$ belongs to the set of all possible values defined within a given type.
- *For logical attributes* – it determines which of the two values they must take, i.e., $X_i = Value$, where $Value \in \{true, false\}$.

During the process of classification some of the attributes may prove to be insignificant. Therefore the conjunction of premises does not necessarily have to contain constraints for all attributes. Some of the premises may be eliminated, which indicates that all values of corresponding attributes are accepted by the rule.

The *IF...THEN...* notation is very natural and intuitive. One cannot expect every doctor, for instance, to be willing to acquaint themselves with the notation used in the first order logic (the calculus of predicates). This simple reason explains the popularity of the above-mentioned solution. The majority of classification problems, where there are no dependencies between the attributes, can be expressed by means of the 0-order logic. Therefore the purpose of a rule extraction algorithm is to produce one or more sets of rules, presented in an understandable way, e.g. in the above-mentioned notation, that would meet the criteria defined in the previous section. The most important features are that these sets reflect the performance of the network both for training patterns and the previously unseen ones, and that the number and complexity of the rules they consist of are kept at a relatively low level.

The problem of classification, even in the presence of one class only, is connected with multimodal optimization, since in a general case the patterns belonging to one class cannot be covered by one rule. On the other side, if we recall the criteria that an extracted set of rules should fulfill, the problem of rule extraction turns out to be multiobjective.

14.4 Pareto Optimization

The most common way of dealing with multiobjetive problems is weighted aggregation, but its main disadvantage consists in the necessity for choosing weights that determine the relative importance of the criteria. This decision rests entirely on the user and must be made before the algorithm is run. The weights are used for fitness calculation and influence the degree to which particular criteria are optimized, forcing the algorithm to respect certain priorities. For example, in rule extraction one must decide whether rules should be very accurate or rather more concise and comprehensible. The problem is that choosing the correct weights is not necessarily easy and can be done onlyapproximately. A significant role is played by intuition and therefore the results produced by the algorithm may prove not to be satisfactory. Another drawback is that depending on the purpose of rule extraction, different criteria may be important and different sets of weights may be required to meet all the needs. In both cases the algorithm requires rerunning with modified parameters, which, especially for complex problems, tends to be very time-consuming.

Pareto optimization is an alternative to scalarization that enables avoiding the operation of converting the values of criteria into a single value. All criteria are equally taken into account and therefore the purpose of the algorithm is to produce a whole set of solutions with different merits and disadvantages. Such an algorithm would create complex and accurate rules as well as more general and comprehensible ones, leaving the final choice of the best solution to the user. The method in question bases on the concept of Pareto domination – a relation of quasi order, denoted by the symbol $\prec$, that makes it possible to compare two solutions represented as vectors consisting of the values of individual criteria. Let's assume that there are two solutions – x and y, and a function f that is used for measuring the solutions' quality, such that:

$$f(x) = [f_1(x), f_2(x), ..., f_m(x)], \tag{14.2}$$

$$f(y) = [f_1(y), f_2(y), ..., f_m(y)]. \tag{14.3}$$

On the assumption that all criteria are minimized, the relation of dominance is defined in the following way:

$$f(x) \prec f(y) \quad \Leftrightarrow \quad (\forall k = 1, ..., m) \quad f_k(x) \leq f_k(y) \quad \wedge \quad (\exists k) \quad f_k(x) < f_k(y). \tag{14.4}$$

This means that $f(x)$ dominates $f(y)$ if two conditions are met, namely: all the criteria values (vector elements) in x are at least as good as the corresponding values in y, and at least one of them is better than in y. Its worth noticing that two solutions may not be bound by this relation at all, for example:

$$f_1(x) < f_1(y) \wedge f_2(x) > f_2(y) \quad \Rightarrow \quad \neg(f(x) \prec f(y)) \wedge \neg(f(y) \prec f(x)). \tag{14.5}$$

In such a case both solutions are considered equally valuable. A multiobjective algorithm, unlike a single objective one, aims at finding an entire set of non-dominated solutions that are located as close to the real Pareto front as possible. Moreover, an additional requirement is that these solutions should be distributed evenly in the space to ensure that all criteria are equally taken into account. Genetic algorithms are particularly suitable for performing multiobjective optimization, because they operate on a large number of individuals simultaneously, which facilitates optimizing a whole set of solutions. Besides, due to niching techniques the second condition is met automatically.

14.5 Multimodality and Multiobjectiveness in Genetic Algorithms

Genetic algorithms aim at finding the global extreme, which is a great advantage, but in some cases, when finding all the optima is the basis of a correct solution, may prove to be undesirable. That's why the mechanism of niching was developed, which enables individuals to remain at the local optima. Several methods are used for approaching such problems [13], among others:

- *Iterations* - a genetic algorithm is rerun several times; if all optima may be found with equal probability, one gets a chance of finding them due to independent computations.
- *Parallel computations* - where a set of populations is evolved independently.
- *Sharing-based methods* - the sharing function determines by how much an individual's fitness is reduced, depending on the number of other individuals in its neighborhood. The original fitness is divided by the value of the sharing function, which is proportional to the number of individuals surrounding a given one and inversely proportional to the distance from them.

The existence of several objectives in the problem to be solved is reflected in the method of evaluating individuals in a genetic algorithm. The objective function takes the form of a vector, therefore it can not be directly used as the fitness function. Such problems in GA-based methods are solved by applying scalarization, which is the most common approach. In this case the fitness function is a weighted sum of elements, where each represents one of the objectives. The problem becomes how to define the weights. Choosing the right weights may be a problem, because the objectives are usually mutually exclusive and several solutions representing different trade-offs between them may exist.

Pareto optimization may also be used in this case. It consists in comparing individuals by means of domination, which allows either rank assignment or performing the tournament selection. The first Pareto-based fitness assignment method, proposed by Goldberg in [6], belongs to the most commonly used ones. It consists in assigning the highest rank to all nondominated solutions. Afterwards, these solutions are temporarily removed from the population and the procedure is repeated for the remaining individuals that receive a lower rank value. The process of rank assignment is carried out as follows:

```
$Temp := Population$
$n = 0$
$while\,(Temp \neq 0) \quad do$
{
$Nondominated = \{x \in Temp/(\neg\exists y \in Temp) f(y) \prec f(x)\}$
$(\forall x \in Nondominated) \quad rank(x) := n$
$Temp := Temp \backslash Nondominated$
$n := n + 1$
}
```

Niching may be used at every stage of this algorithm to help preserve diversity in the population. This optimisation has been introduced into the NSGA algorithm [19].

Another method, proposed by Fonseca and Fleming in 1993, is based on the idea that the rank of an individual depends on the number of other individuals in the population that dominate it. Zitzler and Thiele modified it to develop the Strength Pareto Approach, which solves the problem of niching when elitism is introduced to a multiobjective genetic algorithm [23], [4]. In the case of multiobjective optimisation, elitism requires storing all nondominated individuals found during the course of evolution in an external set. These individuals participate in reproduction along with the ones from the standard population, but they are not mutated, which prevents them from losing their quality. Fitness assignment in the Strength Pareto Evolutionary Algorithm (SPEA) is performed in the following way (N denotes the size of the population):

- For each individual i in the external set do:

$$f_i = \frac{card\{j \in Population | i \prec j\}}{N+1}$$

- For each individual j in the population, do:

$$f_j = \sum_{i, i \prec j} f_i + 1.$$

Because in this method the lowest fitness values are assigned to the best individuals, i.e., $f_i \in [0,1)$ and $f_j \in [1,N)$, the fitness function must be modified so that selection may be carried out.

14.6 MulGEx as A New Method of Rule Extraction

MulGEx belongs to black box methods. It does not require any special network architecture nor use the information encoded in the network itself. Moreover, it does not impose any constraints on the types of attributes in the input patterns. Its main idea consists in introducing genetic algorithms working on two levels as in [12]. MulGEx is based on the concept of Pareto optimization, but scalarization has been implemented on both levels to allow comparison. In the case of the lower-level algorithm this is less important, for it always produces one solution at a time, but on the upper level the choice influences the algorithm's performance significantly. The difference has been shown by experimental study.

The following criteria have been used for evaluating the produced solutions:

- *coverage* - the number of patterns classified correctly by a rule or set,
- *error* - calculated on the basis of the number of misclassified patterns,
- *complexity* - depending on the number of premises in a single rule and rules in a set.

The first two correspond to *fidelity* and determine to what extent the answers given on the basis of the extracted rules are identical to those produced by the network. *Complexity* has been introduced to evaluate the solutions' *comprehensibility*. The remaining FACC requirements cannot be used as criteria during the process of rule extraction. *Accuracy* may be measured only after the algorithm has stopped and the final set of rules is applied to previously unseen patterns. *Consistency* requires running the algorithm several times and comparing the obtained results.

One of the most important decisions that needs to be made before implementing a genetic algorithm concerns the form of an individual, i.e., the way of encoding data in the chromosome. This influences not only the genetic operators, but also the entire algorithm's performance. Two alternative approaches are used in rule extraction, namely Michigan and Pitt [13]. The first one consists in encoding a single rule in each individual. This allows rules to be optimized efficiently, but gives no possibility of evaluating how the extracted rules work as a set. Moreover, one must implement one of the niching techniques as well as solve the problem of contradictory rules that might be evolved. In the Pitt approach an individual contains an entire set of rules, which eliminates the drawbacks of the previous method. However, because of the variable length of chromosomes, i.e., the changing number of rules in the evolved sets, more complicated genetic operators have to be implemented. The proposed method combines both these approaches. The main idea is shown in Fig. 14.3.

The first step consists in evolving single rules by a low-level genetic algorithm. Focusing on individual rules allows adjusting their parameters precisely.

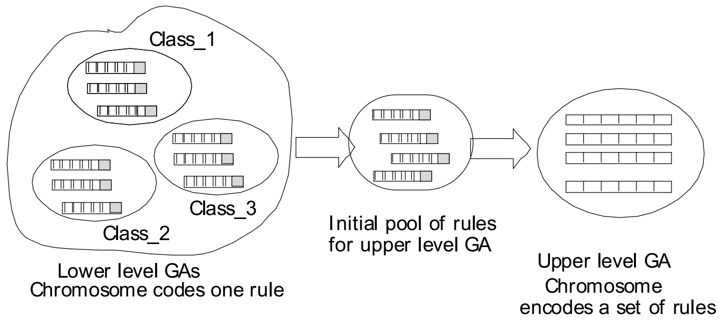

Fig. 14.3. The main idea of MulGEx.

The results are passed to the upper-level algorithm that performs further optimization by working on sets built of the obtained rules and evaluating how well these rules cooperate.

14.7 Details of the Method

The algorithms on both levels will be described in detail in this section. The most significant elements in the design of a GA-based application will be presented, namely the way of encoding individuals, genetic operators and fitness function.

14.7.1 The Lower-level Algorithm

The lower level algorithm delivers the initial pool of rules for the upper level algorithm. It operates on chromosomes representing single rules (the Michigan approach). In order to reduce the complexity of rule extraction, several independently evolved populations are introduced, one for each of the classes defined in the problem. This is a solution to the problem of multimodality that guarantees that the algorithm finds relatively precise rules for all classes, including those represented by a small number of training patterns. Each of the genetic algorithms on the lower level is run several times and produces one rule at a time. This sequential approach to rule extraction solves the problem of multimodality within the classes. The best rule evolved is stored in an external set (the upper level genetic algorithm's initial pool). Afterwards, all training patterns recognized by this rule are removed from the input set used by the algorithm for fitness calculation (sequential covering approach [22]). Then a new population is created in a random way and the algorithm is rerun.

At this stage the choice of the method of dealing with multiobjectiveness is of little significance, since only one rule is extracted at a time. Both approaches, i.e., scalarization and Pareto optimization, have been implemented and compared. Choosing the most appropriate weights is relatively easy, because complexity is rarely taken into account at this stage. In most cases it is reasonable to select rules with a very small error value or, if possible, with no error at all. This guarantees high fidelity of the final set evolved by the algorithm, even though the number of rules may be relatively large.

Nevertheless, Pareto optimization tends to be more efficient here, especially for multidimensional solution spaces and large numbers of classes. At an early stage of evolution the algorithm attempts to discover the areas in the space where patterns belonging to a given class are located. The first rules with coverage greater than 0 evolved have usually large error values and should be subsequently refined in order to reduce misclassification. A single objective algorithm combines *coverage* and *error* into one value and therefore does not take into account the potential usefulness of such rules. These rules are assigned low fitness values because of high error and may be excluded from reproduction, thus being eliminated from the population. This hinders the process of evolution. A Pareto-based genetic algorithm considers all objectives separately, therefore such rules are very valuable (as non-dominated) and will be optimized by means of genetic operators with a strong probability in the next generations. As a result, the initial error will be reduced. Therefore the number of generations needed to evolve satisfying rules tends to be lower if Pareto optimization is used. In this case, however, the choice of the best rule is not straightforward because of a limited possibility of comparing individuals. Because of this, having stopped the algorithm, one must temporarily apply certain weights to scalarize the fitness function.

The number of generations created within such an iteration is chosen by the user who may decide to evolve a constant number of generations every time or to remove the best rule when no improvement has been detected for a given period of time. Improvement occurs when an individual with the highest fitness so far appears in a single objective algorithm or when a new non-dominated solution is found in a multiobjective one.

The process of extraction for a given class completes when there are no more patterns corresponding to this class, or when the user decides to stop the algorithm after noticing that newly found rules cover too few patterns (which means that the algorithm may have started taking exceptions and noisy data into account). As soon as evolution in all the populations has stopped, all rules stored in the external set are passed to the upper-level algorithm for further processing.

The Form of the Chromosome

The chromosome on this level is presented in Fig. 14.4. An individual consists of a set of premises and a conclusion that is identical for all individuals within a given population. The conclusion contains a single integer - the number of the appropriate class. The form of a single premise depends on the type of the corresponding input in the neural network and may represent a constraint imposed on a binary, enumerable or real value.

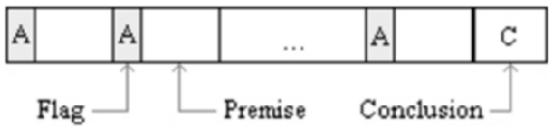

Fig. 14.4. A schema of chromosomes on the lower level.

The appropriate genes designed for all above-mentioned types of attributes are presented in Fig. 14.5a – 14.5c. Every premise is accompanied by a logical flag *A* that indicates whether the constraint is active. If the flag is set to *false*, all values of a given attribute are accepted by the rule.

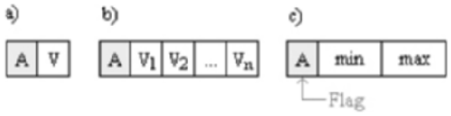

Fig. 14.5. Genes representing different types of premises, depending on the type of attribute.

A constraint defined for a *binary attribute* has been implemented as a single logical variable (Fig. 14.5a). In order to match a given rule, a pattern must have the same value of this attribute as the premise, unless the flag indicates that this particular constraint is inactive. An *enumerable attribute* requires an array of logical variables, where every element corresponds to one value within the type (Fig. 14.5b). All elements set to true represent together the subset of accepted values. A given rule can be applied to a pattern if the value of the attribute in the pattern belongs to the subset specified within the premise.

A constraint imposed on a *real value* consists of two real variables representing the minimal and maximal value of the attribute that can be accepted by the rule (Fig. 14.5c). The number of premises in a rule is constant and equal to the number of the network's input attributes. Premises are eliminated by deactivating their flags, which indicates that certain constraints are not taken into account when applying the rule to the patterns.

Genetic Operators

Having defined the form of the chromosome, one must introduce appropriate genetic operators to allow reproduction and mutation.

The process of exchanging genetic information between individuals is based on uniform crossover. Every logical or real variable in the chromosome is copied from one of the offspring's two parents. The choice of the parent is performed randomly, with equal probability. A logical premise is therefore copied entirely from one parent. The subset of values in an enumerable one is a random combination of those encoded in the parents' chromosomes. In the case of real premises, the newly created range may be either copied entirely from one parent or created as a sum or intersection of both parents' ranges. The conclusion of the rule may be copied from either of the individuals, since its value is the same within a given population.

Mutation consists of modifying individual premises in a way that depends on their type. Some of the logical variables in premises corresponding to binary or enumerable attributes, chosen randomly usually with a small probability, may be assigned the opposite value, which changes the set of patterns accepted by the rule. Constraints imposed on real attributes may be altered by modifying one of the bounds of the range of accepted values. This is done by adding a random value to a given real variable within the premise. The algorithm becomes more effective if small changes are more probable than larger ones, which prevents individuals from being modified too rapidly and losing their qualities. Mutation may also influence the flags attached to premises. This results in individual constraints being activated or deactivated and makes rules either more specific and complex or more general. For obvious reasons mutation cannot affect the conclusion that remains constant in the course of evolution.

The Fitness Function and the Method of Selection

The quality of an individual is measured in the following way: The first step consists in calculating the number of patterns classified correctly by the rule (*coverage*) as well as the number of misclassified patterns (*error*). This requires checking whether the neural network's response, given for every pattern that the rule can be applied to, matches the conclusion. Additionally, the number of active premises is determined in order to evaluate the *complexity* of the rule. These criteria of quality assessment have to be gathered into a single fitness value.

In single objective optimization, all of them are multiplied by certain weights and the results are summed up to produce a scalar value. Since fitness cannot be negative, the obtained values may need to be scaled. This is the simplest solution, however, the information concerning the values of individual criteria is lost. The weights used for fitness calculation are chosen by the user and the function takes the form presented by Eq. 14.6 (the objectives that are minimized are negated).

$$Fitness_{scalar} = W_c \cdot coverage - W_e \cdot error - W_x \cdot complexity \tag{14.6}$$

The multiobjective approach requires creating a vector containing all values of the criteria (Eq. 14.7). Comparing two individuals in this case must be based on the relation of domination. Fitness is calculated on the basis of Goldberg's method, unless the option of retaining the best individuals (elitism) has been chosen. In the latter case, the procedure of fitness assignment proposed in the Strength Pareto Approach is used.

$$Fitness_{Pareto} = [coverage, error, complexity] \tag{14.7}$$

The last element that needs to be defined when designing a genetic algorithm is the method of selection. Various possibilities have been proposed here. The roulette wheel technique has been implemented in MulGEx, which implies that all individuals have a chance to be chosen for reproduction, but the probability of them being selected is proportional to their fitness value.

14.7.2 The Upper-level Algorithm

The upper-level algorithm in MulGEx is multiobjective in the Pareto sense, although scalarization has also been implemented to allow comparison. It operates on entire sets of rules that are initially created on the basis of the results obtained from the lower-level one.

Every individual in the initial population consists of all the rules produced by the lower-level genetic algorithms (Fig. 14.3). This implies that the maximal fidelity achieved at the previous stage is retained and further optimization does not require adding new rules. The purpose of the upper-level algorithm consists mostly in improving comprehensibility by either excluding certain rules from the sets or eliminating individual premises. Naturally, this process should be accompanied by possibly low deterioration of *fidelity*, i.e., the algorithm must decide which rules are the least significant. Gradual simplification of the sets results in producing various solutions with different qualities, which is the main advantage of a multiobjective algorithm. Other modifications of the rules are also possible at this stage, but their influence on the quality of evolved solutions is usually of little importance.

Evolution stops after a given number of generations has been created since the beginning or since the last improvement. The result produced by the algorithm is a set of non-dominated solutions that correspond to sets of rules with different qualities and drawbacks. The final choice of the most appropriate one is left to the user and is usually done by applying weights to the criteria to enable comparing non-dominated solutions. Due to the multiobjective approach the user may examine several solutions at a time without having to rerun the algorithm with different parameters. This allows identifying noisy data. Normally, increasing the complexity of a set is accompanied by improved fidelity. However, if the observed improvement is very small after a new rule has been added, this may indicate that this rule covers exceptions that haven't been eliminated by a neural network. The multiobjective approach helps to determine the optimal size of a rule set, which is not possible in the case of scalarization, where weights have to be carefully adjusted for the algorithm to produce satisfying results.

As previously, designing the algorithm requires defining the form of an individual, followed by genetic operators, fitness evaluation and the method of selection.

The Form of the Chromosome

An individual takes the form of a set of rules that describes the performance of the network, which implies that rules corresponding to all classes are included in every set.

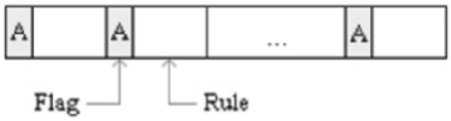

Fig. 14.6. A schema of chromosomes on the upper level.

Rules are accompanied by binary flags whose purpose is the same as in the chromosome on the lower level. Namely, setting a flag to *false* results in the temporary exclusion of a given rule from the set. This allows adjusting the size of the set without having to introduce variable-length chromosomes. The form of chromosome on this level is presented in Fig. 14.6.

Genetic Operators

The hierarchical structure of the chromosome requires defining specialized genetic operators.

Crossover exchanges corresponding rules between the parents in a uniform way. Rules are copied into the offspring entirely, including the flag.

The operator of mutation is more complicated, for it should enable modification on two levels - in relation to entire sets and individual rules. Therefore mutation is performed in two steps. First, every flag attached to a rule may change its value with a small probability, which results in adding or removing the rule from the set. Afterwards, premises inside the rule may be modified by applying the mutation operator used on the lower level. This helps to adjust the ranges of accepted values inside the rule or reduce the rule's complexity at the cost of fidelity.

It is worth noticing that in the lower-level algorithm some rules were evolved on the basis of a reduced set of patterns. At this stage the whole set is used for evaluation. Moreover, individual rules cooperate to perform the classification task and are assessed together. Because of these new conditions further optimization of rules might be possible and that is why the operator of mutation in the upper-level algorithm may alter the internal structure of rules that was developed before.

Evaluation of Individuals

The criteria used for the evaluation of individuals are calculated on the basis of the criteria introduced for rules on the lower level.

Coverage depends on the number of patterns classified correctly by the set as a whole, whereas *error* is the sum of errors made by each of the active rules. *Complexity* is defined as the number of active rules increased by the overall number of active premises within them.

The process of creating new generations resembles the one introduced on the lower level. Fitness is assigned to individuals depending on the type of the algorithm, according to one of the methods described in the previous subsection. Again, Pareto optimization requires creating a vector of the values of objectives (Eq. 14.7). Scalarization implemented for the purpose of comparison consists in applying weights to the objectives so that a single value is obtained.

The difference between genetic algorithms on both levels is that at this stage the purpose of the algorithm is to produce a whole set of solutions simultaneously, therefore niching proves to be very useful. To this end the technique of sharing function has been implemented to reduce the fitness of those individuals that are surrounded by many others. However, this is not necessary in the case of the SPEA algorithm, since niching is guaranteed by the method of fitness assignment.

14.8 Experimental Study

The purpose of experiments is to verify the algorithm's effectiveness and versatility. The method should be tested on various data, which aims at checking whether it has the following properties:

- independence on the types of attributes,
- effectiveness in the presence of superfluous attributes,
- resistance to noise,
- scalability.

A good practice is to perform experiments on well-known benchmark data sets, so that the results may be compared with those obtained by means of other methods. MulGEx has been tested on four sets from [3] and presented in Table 14.1. The data contain different types of attributes, which allows verifying the method's versatility. Resistance to noise was tested on the *LED-24* data set, where 2% noise had been added to make the classification task more difficult. Moreover, this data allows to observe whether the algorithm can detect superfluous attributes, since not all of them participate in determining the class that a given pattern belongs to (17 attributes are insignificant and have been chosen randomly). The evaluation of scalability requires patterns with a large number of attributes, hence the *Quadrupeds* set.

Table 14.1. Data sets used for the tests

Name (examples)	**Type of attributes**	**Class**	**No of examples**
Iris (150 instances)	3 continuous	Setosa	50
		Versicolour	50
		Virginica	50
LED-24 (1000 instances)	24 binary (17 superfluous), 2% noise	10 classes	about 100 per one class
Monk-1 (432 instances)	6 enumerable	0	216
		1	216
Quadrupeds (100 instances)	72 continuous	Dog	29
		Cat	21
		Horse	27
		Giraffe	23

All sets were processed by a neural network in order to reduce noise and eliminate unusual data. A multi-layer feed-forward network with one hidden layer was provided with the data and trained using the back-propagation algorithm. The results of the training are gathered in Table 14.2. The subsequent experiments performed to evaluate MulGex were conducted both on raw as well as processed sets, unless the network achieved maximal accuracy.

Table 14.2. Results of the network training

data set	**Number of neurons in the hidden layer**	**Accuracy %**
Iris	2	96,7
LED-24	3	95
Monk-1	3	100
Quadrupeds	2	100

The optimal values of the algorithm's parameters depend on the properties of the data set, especially on the number of attributes. During the experiments the best effectiveness of rule extraction was achieved when the probability of crossover was relatively low, for example 0.5. The reason is that even a small modification of a rule may change its *coverage* and *error* significantly. Therefore if the fittest individuals exchange genetic information during reproduction, the offspring may lose the most desirable properties. Low probability of crossover guarantees that some of the individuals are copied directly into the new population. Introducing elitism may also be helpful in this case, for it ensures that the best individuals are always present in the next generation. The probability of mutation should be inversely proportional to the number of attributes and during the tests was usually set not to exceed 0.01. The size of the population influences the efficiency of the algorithm as well. Theoretically, increasing the number of individuals reduces the number of generations needed to find satisfactory solutions. The more individuals, the more potential solutions may be examined and compared at a time. On the other hand, because the process of calculating *coverage* and *error* for each individual is very time-consuming, large populations cause deterioration in the algorithm's performance. Therefore a reasonable compromise must be found. Rule extraction for the above-mentioned data sets was successfully carried out when populations consisted of 20 - 50 individuals.

The results of the experiments performed both on raw data and sets processed by the network are gathered in Table 14.3 and Table 14.4, respectively. In each case multiobjective algorithms on both levels were stopped after having evolved 100 generations without improvement. MulGEx produced sets of non-dominated solutions with various properties. Then, solutions containing different numbers of rules were selected and their fidelity, i.e., the difference between *coverage* and *error*, was determined. Every test was carried out 5 times and the average result as well as the standard deviation were calculated.

The results of the experiments indicate that MulGEx is suitable for solving problems with various numbers and types of attributes, as well as with different sets of classes defined for a given problem. In the case of *Iris*, *Monk-1* and *Quadrupeds* the algorithm succeeded in finding a relatively small rule set that covered all training patterns without misclassification. This was not possible for *LED-24*, where the presence of noise resulted in increasing *complexity* or *error*, depending on the weights chosen by the user.

The experiments confirm the fact that the criteria used for evaluating solutions are contradictory. The simplest sets, consisting of few rules, cover only the most typical examples. When more rules are added to the set, its fidelity is improved at the cost of complexity.

At this point it should be mentioned that MulGEx satisfies the requirement that the produced solutions should be distributed evenly in the space.

Table 14.3. The results of experiments on raw data

Iris		LED-24		Monk-1		Quadrupeds	
Rules	*Fidelity*	*Rules*	*Fidelity*	*Rules*	*Fidelity*	*Rules*	*Fidelity*
1	50.0± 0.0	3	326.0± 1.0	1	108.0± 0.0	1	27.4± 2.5
2	96.8± 0.4	6	608.2± 6.4	2	180.0± 0.0	2	53.0± 5.0
4	143.6± 0.5	9	843.2± 12.6	4	324.0± 0.0	3	76.8± 2.8
6	147.4± 0.5	12	918.4± 5.0	6	396.0± 0.0	4	97.8± 2.9
8	149.4± 0.5	15	924.8± 4.7	7	432.0± 0.0	5	99.3± 1.2

Table 14.4. The results of experiments on data processed by a network

Iris		LED-24	
Rules	*Fidelity*	*Rules*	*Fidelity*
1	50.0± 0.0	3	336.2± 2.0
2	98.2± 0.4	6	642.0± 3.6
4	143.0± 0.7	9	897.6± 4.0
6	147.6± 1.1	12	981.8± 4.3
7	149.5± 0.9	15	987.4± 4.7

The evolved rule sets represent different trade-offs between the objectives and contain sets with maximal fidelity as well as sets consisting of one rule only and various intermediate solutions. The results obtained for *LED-24* for raw and processed data differ significantly. The *fidelity* of sets containing the same number of rules is higher for data passed through a neural network, which means that in this case a single rule covers statistically more patterns and misclassifies less. This is possible due to neural networks' ability of reducing noise, which facilitates rule extraction.

The Pareto approach in the upper-level algorithm has been compared with scalarization. To this end separate tests for the *Iris* data set have been performed. The lower-level algorithm was run 5 times and produced initial rule sets. In all cases Pareto optimization was used at this stage and rules with very high error weight were selected, so that the initial set could achieve maximal fidelity. Afterwards the upper-level one was executed several times with different settings. When the Pareto approach was used, it was run only once during each of the tests and a whole set of solutions was created. Then the user could apply weights to allow the algorithm to select the most appropriate solution. In the case of scalarization weights for the objectives had to be determined beforehand, therefore separate runs were necessary to produce results for all indicated weights' combinations.

The results (Table 14.5) show that for the *Iris* data set there is hardly any difference in the quality of solutions between the two approaches. However, it must be emphasized that scalarization requires rerunning the algorithm every time when the user decides to change the weights, which is very time-consuming. If we take into account the fact that adjusting weights is not easy, it proves to be much more convenient to find a set of solutions in a single run and then apply different weights in search of the most satisfying one.

Another advantage of Pareto optimization is that one may analyze all solutions simultaneously. Fig. 14.7 shows the values of *fidelity* and *complexity* for rule sets produced during one of the experiments. Before choosing a solution the user may evaluate which one would be the most satisfactory, based on the shape of the Pareto front. In the presented example very high *fidelity* may be achieved by a set with *complexity* equal to 6. If *complexity* is increased by adding new rules or premises, the improvement of *fidelity* is relatively low.

Table 14.5. The comparison of Pareto approach and scalarization (*Cov* stands for coverage, *Comp* – for complexity)

Weights			Pareto			Scalarization		
Cov.	*Error*	*Comp.*	*Cov.*	*Error*	*Comp.*	*Cov.*	*Error*	*Comp.*
1	1	1	143.6± 1.5	1.0± 0.0	8.6± 1.5	143.2± 1.5	1.0± 0.0	8.2± 1.6
1	1	10	143.0± 0.0	5.4± 0.9	6.0± 0.0	142.0± 1.7	6.0± 1.0	6.0± 0.0
1	10	1	141.2± 1.6	0.0± 0.0	9.2± 1.6	141.2± 1.6	0.0± 0.0	9.2± 1.6
10	1	1	150.0± 0.0	4.0± 2.1	18.4± 3.9	150.0± 0.0	2.0± 2.0	22.0± 2.9

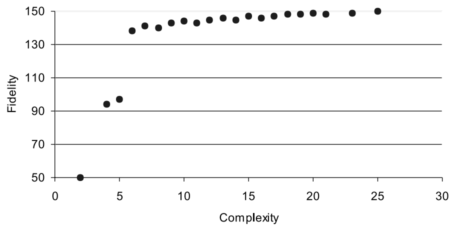

Fig. 14.7. The values of the criteria for rules sets obtained for *Iris*.

This may indicate that more complicated sets cover noisy data and are not valuable for the user. On the other hand, if *complexity* is reduced below 6, *fidelity* deteriorates rapidly, which suggests that an important rule has been eliminated. Therefore, having this additional information due to the Pareto approach, the user may choose the most appropriate weights when selecting one particular solution.

14.8.1 Evaluation of Rules by Visualization

The quality of rules produced by MulGEx for *LED-24* may be easily evaluated due to the possibility of visualizing the data. The first attributes in *LED-24* represent seven segments of a *LED* display presented in Fig. 14.8. The remaining ones are superfluous and are successfully excluded by the algorithm from participating in classification. The experiment was carried out for data that had been processed by a network.

The presented results were produced by the multiobjective lower-level algorithm and the following weights: *coverage* = 10, *error* = 10, *complexity* = 1 were used for selecting the most satisfactory solutions after the algorithm had stopped. The size of population was set to 20 and the probabilities of crossover and mutation to 0.9 and 0.01, respectively. The option of saving the best individuals (elitism) was chosen. Evolution was stopped after 100 generations without improvement and the best rule in each of the populations was selected. Table 14.6 and Fig. 14.9 show an example of rules produced for *LED-24*. In Fig. 14.9 thick black lines denote segments that are lit, missing lines stand for segments that are off and thin grey lines correspond to seg-

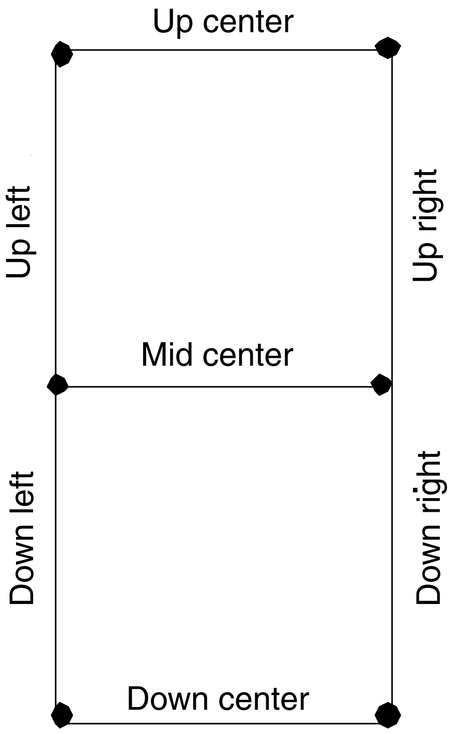

Fig. 14.8. The segments of *LED*.

ments whose state is insignificant when assigning a pattern to a given class (these segments are represented by inactive premises).

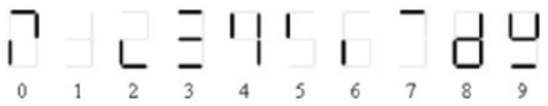

Fig. 14.9. Visualization of rules for *LED-24*.

The obtained rules presented in the form of a schema in Fig. 14.9 prove to be very comprehensible for humans. One may easily notice that the results resemble digits that appear on a LED display, which confirms the effectiveness of MulGEx. At the same time the algorithm succeeded in reducing the rules' complexity. The state of some of the segments proves to be insignificant when a certain pattern is classified as one of the digits.

Table 14.6. Rules produced by the lower-level algorithm for *LED-24*

Rule	Coverage	Error
IF Up center AND Up right AND NOT Mid center AND Down left THEN 0	114	0
IF NOT Up center AND NOT Up left AND NOT Down left THEN 1	83	0
IF NOT Up left AND Down left AND NOT Down right AND Down center THEN 2	118	0
IF Up center AND NOT Up left AND Mid center AND NOT Down left AND Down center THEN 3	110	0
IF Up left AND Up right AND NOT Down left AND Down right AND NOT Down center THEN 4	82	0
IF Up left AND NOT Up right AND NOT Down left THEN 5	92	3
IF NOT Up right AND Down left THEN 6	103	0
IF Up center AND NOT Up left AND NOT Mid center AND NOT Down left THEN 7	102	0
IF Up right AND Mid center AND Down left AND Down right AND Down center THEN 8	104	0
IF Up left AND Up right AND Mid center AND NOT Down left AND Down center THEN 9	81	0

14.9 Comparison to Other Methods

The vast majority of rule extraction algorithms are based on the scalar approach. Therefore comparing their effectiveness may not be appropriate because of different goals pursued by their authors. Nevertheless two different approaches have been implemented in MulGEx to allow evaluating their performance.

The results of experiments performed for MulGEx may be confronted with those obtained for a different multiobjective rule extraction algorithm, namely GenPar [11]. This is facilitated by the fact that the same data sets were used in both cases. Nevertheless the relative effectiveness of these methods may be evaluated only approximately because of certain differences in the way of performing experiments and the scope of collected information.

GenPar was tested on data processed by a neural network only. Moreover, during the course of evolution coverage and error were combined into a single objective, which reduced the number of objectives by one, and only coverage was measured during the tests. The results for GenPar, presented in [11] and shown in Table 14.7, contain information on the quality of evolved sets expressed in terms of coverage and the number of rules.

The results for GenPar indicate that its effectiveness is lower than that of MulGEx. Although fidelity achieved by solutions with the same numbers of rules are similar, in all cases GenPar failed to produce a solution with the

Table 14.7. The results of experiments for GenPar (*size* – number of rules, *fid* – fidelity equivalent to coverage) on the basis of [11]

Iris		LED-24		Quadrupeds	
size	*fid*	*size*	*fid*	*size*	*fid*
1	50	1	87	2	182
2	94	4	324	4	471
4	137	5	473	–	–
5	144	13	928	–	–

maximal fidelity possible. The reason may be that GenPar is based exclusively on the Pitt approach (entire rule sets are encoded in individuals). This implies that sets are evaluated as a whole and the performance of single rules is not taken into account, which prevents rules from being intensively optimized by adjusting values within the premises. The superiority of MulGEx lies in introducing a lower-level algorithm that evolves single rules before sets are formed and allows constant chromosome length on the upper-level.

14.10 Conclusion

Neural networks, in spite of their efficiency in solving various problems, have no ability of explaining their answers and presenting acquired knowledge in a comprehensible way. This serious disadvantage leads to developing numerous methods of knowledge extraction. Their purpose is to produce a set of rules that would describe a network's performance with the highest fidelity possible, taking into account its ability to generalize, in a comprehensible way. These objectives are contradictory, therefore finding a satisfactory solution requires a compromise.

Because the problem of rule extraction is very complex, genetic algorithms are commonly used for this purpose, especially for networks that solve the problem of classification. Since the fitness function is scalar, standard algorithms can deal with one objective only. Multiobjective problems are approached in many ways; the most common one is to aggregate the objectives into a single value, but this approach has many drawbacks that can be eliminated by introducing proper multiobjective optimization based on domination in the Pareto sense.

The proposed method, MulGEx, combines multiobjective optimization with a hierarchical structure to achieve high efficiency. It is network architecture independent and can be used regardless of the number and types of attributes in the input vectors. Although the algorithm has been designed to extract rules from neural networks, it can be also used with raw data. In this case, however, its efficiency may be deteriorated because the presence of noise in the data requires more complicated rule sets to describe the process of classification.

MulGEx consists of genetic algorithms working on two levels. The lower-level one produces single rules (independently evolved populations correspond to the classes defined for a given classification problem) and passes them to the upper-level one for further optimization. The latter algorithm evolves entire sets of rules that describe the performance of a network.

Due to Pareto optimization MulGEx produces an entire set of solutions with different properties simultaneously, varying from complex ones with very high fidelity to the simplest ones that cover only the most typical data. The usefulness of such solutions depends on the purpose of rule extraction. In most cases relatively general rules are considered valuable for a classification problem, for they provide information about the most important properties that objects belonging to a given class should have. Nevertheless, if we attempt to find hidden knowledge by means of *data mining*, we might be interested rather in more specific rules that cover a smaller number of patterns, but achieve high fidelity. The final choice of the most appropriate solution is left to the user who may select several solutions without the need for rerunning the algorithm. The effectiveness and versatility of MulGEx has been confirmed by experiments and the superiority of the Pareto approach over scalarization has been pointed out.

References

[1] Andrews, R., J. Diedrich, and A. Tickle: 1995, 'Survey and critique of techniques for extracting rules from trained artificial neural networks'. *Knowledge-Based Systems* **8**(6), 373–389.

[2] Andrews, R. and S. Geva: 1994, 'Rule extraction from constrained error back propagation MLP'. In: *Proceedings of 5th Australian Conference on Neural Networks, Brisbane, Quinsland.* pp. 9–12.

[3] Blake, C. C. and C. Merz: 1998, 'UCI Repository of Machine Learning Databases'. *University of California, Irvine, Dept. of Information and Computer Sciences.*

[4] Coello, Carlos A. Van Veldhuizen, D. A. and G. B. Lamont: 2002, *Evolutionary Algorithms for Solving Multi-Objective Problems.* Kluwer Academic Pblisher. New York.

[5] Fu, X. and L. Wang: 2001, 'Rule extraction by genetic algorithms based on a simplified RBF neural network'. In: *Proceedings Congress on Evolutionary Computation.* pp. 753–758.

[6] Goldberg, D. E.: 1989, *Genetic Algorithms in Search, Optimization and machine Learning.* Addison-Wesley Publishing Company, Reading, Massachusetts.

[7] Hayashi, Y., R. Setiono, and K. Yoshida: 2000, 'Learning M of N concepts for medical diagnosis using neural networks'. *Journal of Advanced Computational Intelligence* **4**(4), 294–301.

[8] Hayward, R., C. Ho-Stuart, J. Diedrich, and P. E.: 1996, 'RULENEG: Extracting rules from trained ANN by stepwise negation'. Technical report, Neurocomputing Research Centre Queensland University Technology Brisbane, Old.

[9] Jin, Y. and B. Sendhoff: 2003, 'Extracting interpretable fuzzy rules from RBF networks'. *Neural Processing Letters* **17**(2), 149–164.

[10] Jin, Y., B. Sendhoff, and E. Koerner: 2005, 'Evolutionary multi-objective optimization for simultaneous generation of signal-type and symbol-type representations'. In: *The Third International Conference on Evolutionary Multi-Criterion Optimization.* pp. 752–766.

[11] Markowska-Kaczmar, U. and P. Wnuk-Lipinski: 2004, 'Rule Extraction from Neural Network by Genetic Algorithm with Pareto Optimization'. In: L. Rutkowski (ed.): *Artificial Intelligence and Soft Computing.* pp. 450–455.

[12] Markowska-Kaczmar, U. and R. Zagorski: 2004, 'Hierarchical Evolutionary Algorithms in the Rule Extraction from neural Networks'. *Fourth International Symposium on Engineering of Intelligent Systems, EIS.* Funchal, Portugal.

[13] Michalewicz, Z.: 1996, *Genetic Algorithms + Data Structures = Evolution Programs.* Springer Verlag Berlin Heidelberg.

[14] Mitra, S. and H. Yoichi: 2000, 'Neuro-Fuzzy Rule Generation: Survey in Soft Computing Framework'. *IEEE Transactions on Neural Networks* **11**(3), 748–768.

[15] Palade, V., D. C. Neagu, and R. J. Patton: 2001, 'Interpretation of Trained Neural Networks by Rule Extraction'. *Fuzzy Days 2001, LNC 2206* pp. 152–161.

[16] Santos, R. Nievola, J. and A. Freitas: 2000, 'Extracting comprehensible rules from neural networks via genetic algorithms'. *Symp. on Combinations of Evolutionary Computation and Neural Network* **1**, 130–139.

[17] Setiono, R., W. K. Leow, and J. Zurada: 2002, 'Extraction of Rules from Artificial Neural Networks for Nonlinear Regression'. *IEEE Transactions on Neural Networks* **13**(3), 564–577.

[18] Siponen, M., J. Vesanto, O. Simula, and P. Vasara: 2001, 'An Approach to Automated Interpretation of SOM'. In: *In Proceedings of Workshop on Self-Organizing Map 2001(WSOM2001).* pp. 89–94.

[19] Srinivas, N. and K. Deb: 1993, 'Multiobjective optimizatio using nondominated sorting in genetic algorithms'. Technical report, Department of Mechanical Engineering, Indian Institute of Technology, Kanpur, India.

[20] Taha, I. and J. Ghosh: 1999, 'Symbolic Interpretation of Artificial Neural Networks'. *IEEE Transactions on Knowledge and Data Engineering* **11**(3), 448–463.

[21] Thrun, S. B.: 1995, 'Extracting rules from artificial neural networks with distributed representation'. In: G. Tesauro, D. Touretzky, and T. Leen (eds.): *Advances in Neural Information Processing Systems.*

[22] Weijters, A. and J. Paredis: 2000, 'Rule Induction with Genetic Sequential Covering Algorithm (GeSeCo)'. In: *Proceedings of the second ICSC Symposium on Engineering of Intelligent Systems, (EIS 2000).* pp. 245–251.

[23] Zitzler, E. and L. Thiele: 1999, 'Multiobjective Evolutionary Algorithms: A Comparative Case Study and Strength Pareto Approach'. *IEEE Transaction on Evolutionary Computation* **3**(4), 257–271.