Annales UMCS Informatica AI 4 (2006) 230-243

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI

http://www.annales.umcs.lublin.pl/

# Rule extraction from a neural network by hierarchical multiobjective genetic algorithm

Urszula Markowska-Kaczmar*, Krystyna Mularczyk

*Institute of Computer Science, Wrocław University of Technology,*
*Wybrzeże S.Wyspiańskiego 27, 50-370 Wrocław, Poland*

**Abstract**

The paper presents a method of rule extraction from the trained neural network by means of a genetic algorithm. The multiobjective approach is used to suit the nature of the problem, since different criteria (accuracy, complexity) may be taken into account during the search for a satisfying solution. The use of a hierarchical algorithm aims at reducing the complexity of the problem and thus enhancing the method performance. The overall structure and details of the algorithm as well as the results of experiments performed on popular benchmark data sets are presented.

## 1. Introduction

The problem of extracting rules from neural networks consists in finding the dependency between the network output and the properties of the presented input vector [1]. It is very important in the medical domain because it increases the trust of users to the system based on the neural network. The complexity of this task usually arises from a large number of dimensions of the input vectors. The usefulness of genetic algorithms results mainly from their ability of searching vast multidimensional spaces, as well as processing many potential solutions at a time. Moreover, unlike other methods they treat the network as a “black box" and concentrate only on the values of inputs and corresponding outputs, thus becoming architecture independent (e.g. decision trees), genetic algorithms prove to be more successful when dealing with real values of inputs. Several techniques, usually for networks whose task is to classify the input vectors, have already been developed, e.g. in [2,3] however, the multiobjective nature of the problem of rule extraction is rarely taken into account. A set of rules describing the functioning of a network may be optimised in several various ways. First of all, its accuracy may be improved by modifications that

*Corresponding author: *e-mail address*: urszula.markowska-kaczmar@pwr.wroc.pl

lead to maximizing the number of correctly classified patterns as well as reducing the number of errors made by the set. Another criterion usually taken into consideration is the complexity of the set, denoted by the number of rules and the number of premises in each rule [1], which in, turn may, be considered separately.

This implies the existence of several criteria of the evaluation of the solutions that tend to be contradictory. The standard approach in a genetic algorithm is based on the concept of aggregating functions that enable gathering all the values of the criteria into a single fitness function, usually in the form of a weighted sum. However, this method has two main drawbacks. The weights must be properly adjusted to allow the algorithm to find satisfactory solutions, which may result in the necessity of running the algorithm many times. Moreover, whenever the user decides to change the weights in order to concentrate on one criterion to a greater degree, the process of evolution must be repeated. This makes the use of classic genetic algorithms rather unwieldy and time-consuming.

During the process of rule extraction one may wish to take one of different approaches, depending on the purpose of extraction. One option is an attempt to obtain a relatively simple set of rules that contains only the most significant principles governing the functioning of the network, without focusing on exceptions. Another approach aims at developing a set that describes the network with the highest fidelity possible. This might lead to discovering "hidden" knowledge by finding rules that apply to small numbers of input vectors. Taking this into account, one may say that a perfect solution to the problem of rule extraction would produce results at a different level of accuracy and complexity to suit all the needs. This requirement is met by multiobjective genetic algorithms. Such a method has been proposed in [4] but, since the algorithm operates on entire sets of rules, its efficiency when dealing with complex problems is questionable.

## 2. Pareto optimality in multiobjective problems

In a multiobjective problem several criteria are used to evaluate each solution. The quality of a given solution cannot be easily expressed in terms of a single numerical value; hence the problem of comparing solutions and determining which of them proves to be the most satisfactory. The concept of Pareto optimality introduces a relation of quasi order on the set of solutions, called Pareto domination and *denoted* by the symbol $\prec$. For two solutions x and y, whose quality is measured by the two vectors consisting of the values of individual criteria, namely:

$$f(x) = \left[ f_1(x), f_2(x), ..., f_m(x) \right], \tag{1}$$

$$f(y)=\left[f_1(y),f_2(y),...,f_m(y)\right], \tag{2}$$

the relation of domination is defined as follows:

$$f(x)\prec f(y)\Leftrightarrow(\forall k=1,...,m)\quad f_k(x)\le f_k(y),$$
$$\wedge\exists\, f_k(x)<f_k(y). \tag{3}$$

It means that a given solution dominates another if it is better with regard to one criterion and at least equally good if any of the remaining criteria is taken into account. This implies that in a general case two solutions do not necessarily have to be bound by this relation, which makes them equally valuable. The optimisation in the Pareto sense, unlike the optimisation of a function, does not aim at producing a single satisfactory solution, but at finding an entire set of non-dominated solutions that would not only lie possibly close to the optimal values, but would also be distributed evenly in the space of solutions. The second condition guarantees that all criteria and various combinations of their relative ‘‘importance" are equally taken into account during the process of optimisation.

## 3. Basic concepts of the proposed method

The method of rule extraction proposed in the paper, we called MulGEx (**Mul**tiobjectve **G**enetic **Ex**tractor) is based on a genetic approach. It acquires a set of rules describing the performance of a network used for classification tasks. Although we concentrate on rule extraction from a neural network, since MulGEx treats a network as a black box (a global approach to rule extraction [1]), it can be used for extracting sets of rules immediately from raw data as well. The input vector presented to the network may consist of data of various types (logical, enumerable, real) whereas the only output is an integer representing the class that a given pattern belongs to. Because of great complexity of the problem, especially for large numbers of attributes and input vectors, the algorithm has been divided into two stages, which had been inspired by [2].

The lower-level genetic algorithm operates on single rules and aims at maximizing the number of correctly classified patterns without taking the rules’ complexity into account. The upper-level algorithm, on the contrary, respects all criteria mentioned in the introduction. Its task is to gather the rules evolved by the first algorithm into one set and refine them through further processing. The general structure of the algorithm has been shown in Fig. 1. The chromosome on the lower level represents a single rule consisting of a set of premises followed by the conclusion, as shown in Fig. 3a. A premise may be active or not, depending on the value of its flag. More detailed explanation of the purpose of flags is given in the next subsection. Similarly, a set evolved by the upper-level algorithm is a sequence of rules that may also be activated or disabled when the

corresponding flag is changed. The chromosome at this stage represents a set of rules. Its structure is presented in Fig. 3b.

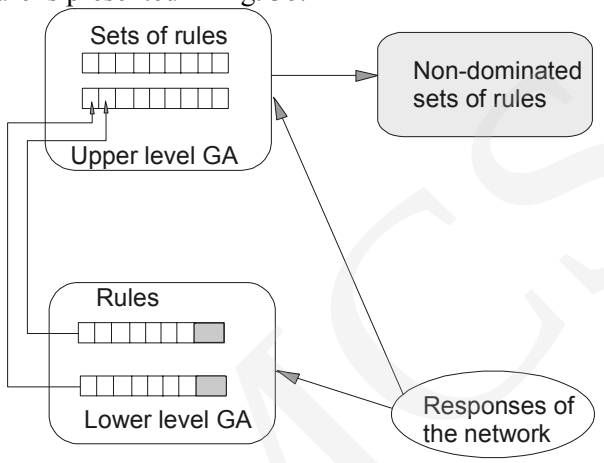


Fig. 1. The general structure of MulGEx

### 3.1. Description of the lower-level algorithm

The lower-level algorithm consists of several independently evolved populations, each responsible for a different class in the problem of classification. At this stage an individual represents a single rule in the form of a set of premises followed by the number of the appropriate class, as shown in 4.

$$IF \;\; p_1 \; AND \; p_2, \ldots \; AND \; p_m \;\; THEN \;\; k \tag{4}$$

Each premise $p_i$ corresponds to one element of an input vector presented to the network (the $i$-th input of the network)and denotes the constraint that the attribute must satisfy before the rule may be applied to the entire vector. Depending on the type of the attribute, this constraint may take one of the following forms.

- For logical attributes the constraint determines which of the two possible values must be taken by the attribute. Therefore the premise consists of a single Boolean value (Fig. 2a).
- For enumerable attributes the premise is expressed as an array of Boolean values ($V_i$) with one element per every value in the type. It determines the subset of values accepted by the constraint (Fig. 2b)
- Discrete and real attributes require defining the range of acceptable values, which is achieved by specifying its minimum and maximum
- Discrete and real attributes require defining the range of acceptable values, which is achieved by specifying its minimum and maximum (Fig. 2c).

Every rule contains a complete set of premises, i.e. imposes constraints on all elements of input vectors presented to the network. This prevents from

introducing chromosomes with variable lengths and thus facilitates defining the genetic operators. However, in many cases certain attributes do not influence the choice of a class during the process of classification, and therefore all their values could be accepted by a given rule. To achieve this, a Boolean flag is added to every premise. It determines whether the particular constraint remains active and is taken into account when the rule is applied to an input vector. By disabling some of the constraints one may obtain more general rules with lesser complexity, which is essential during the process of rule extraction.

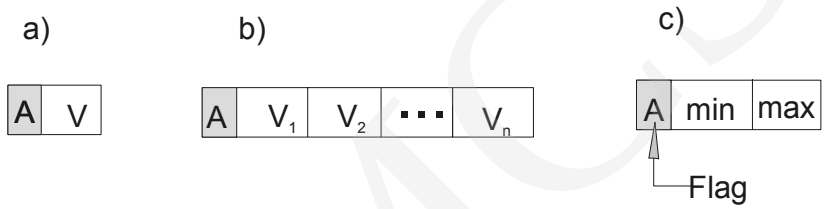


Fig. 2. Genes for different type of premises, which depending on the type of attribute, a) for logical, b) for enumerative, c) for real attribute

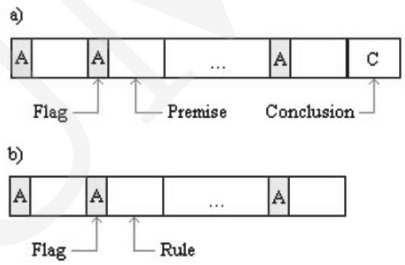


Fig. 3. A schema of chromosomes on both levels

The form of the chromosome (rule) requires defining special genetic operators responsible for crossover and mutation. Uniform crossover has been used, i.e. the process of exchanging genetic information between two individuals consists in choosing one of the parents with equal probability for each premise (gene) copied to the chromosome of the offspring. A premise is duplicated entirely, including the flag. The only exception occurs in the case of constraints imposed on real inputs, where each of the values determining the accepted range (namely the minimum and maximum) may be taken from a different parent. The conclusion of the rule may be copied from either of the parents, since the number of the class is identical for all individuals in a given population.

The effect of mutation depends on the type of the gene that it is applied to. In all cases the flag may be affected, which results in enabling or disabling a given gene. For the types consisting of Boolean values shown in Fig. 2a and 2b, mutation may cause changing a randomly chosen value to the opposite one. The

operator of mutation applied to a real value in a premise of the type shown in Fig. 2c changes it by a random amount (without exceeding the maximal range defined for the corresponding attribute), on the assumption that small modifications are more probable. Mutation does not affect the conclusion of a rule.

In order to compare both approaches (scalar and multiobjective), the value of the fitness function assigned to an individual in the lower-level algorithm may be either calculated as a weighed sum of the criteria or based on the relation of domination (the multiobjective approach). The first step in both cases consists in determining the number of correctly and incorrectly classified input vectors, i.e. the coverage and error, respectively. To this end the answer given by the neural network for each vector is compared to that given by the rule if the constraints have been satisfied. The weights used for fitness calculation in the first approach are chosen by the user and the function takes the form presented by Eq. 5.

$$Fitness_{lower} = W_c \cdot coverage - W_e \cdot error \tag{5}$$

If the value of the function happens to be negative, it must be modified, e.g. by adding a constant value to the fitness of every individual.

In the multiobjective approach, the fitness assignment is based on ranks given to the individuals. Several techniques may be used here – the most popular were those proposed by Goldberg in [5] and Fonseca and Fleming in [6]. Both of them are implemented in MulGEx, the latter with a modification introduced by Zitzler and Thiele (the SPEA algorithm [7]). The selection of individuals that participate in the process of reproduction is performed by means of the roulette wheel method, which implies that the probability of choosing a given individual for crossover is proportional to its fitness. Goldberg's algorithm of rank assignment is as follows:

*Temp := Population*
*n = 0*
*while(Temp ≠ 0)do*
{
$Nondominated = \{x \in Temp / (\neg \exists \in Temp) f(y) \prec f(x)\}$
$(\forall x \in Nondominated) rank(x) := n$
*Temp := Temp\Nondominated*
$n := n + 1$
}

Fitness assignment in SPEA is performed according to the following algorithm:

*Let N denote the size of the population*

*For each individual i in the external set* :

$$f_i = \frac{N}{N+q},$$

*where n stands for the number of individuals dominated by i* $f_i \in [0,1)$

*For each individual* j *in the population* :

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

*i.e.the sum of fitness values of all external individuals that dominate j increased by on*

$$f_j \in [1, N)$$

As in this method the lowest fitness values are assigned to the best individuals, the fitness function must be modified so that it can be used for the roulette wheel technique.

Since the complexity criterion is not taken into account during the fitness assignment at this stage, all genes of the individuals in the initial population are set to the disabled. During the process of evolution the premises are gradually activated by applying mutation, which allows to keep the complexity at a relatively low level and introduce only the necessary constraints. The duration of evolution is determined by the user who may choose either to create a fixed number of generations or to stop the algorithm when no improvement has been detected for a given amount of time (measured in generations). When the algorithm has stopped, the best rule in every population is stored in an external set and all vectors recognized by this rule are removed from the set that the algorithm works on. A good idea is to choose the rule with the highest coverage and noerror, but in some cases this might be impossible or inefficient. Afterwards a new initial population is created and all steps are repeated. This phase of rule extraction ends when there are no more vectors corresponding to a given class in the set, so that no new rules may be evolved. The external set containing rules stored during the entire course of evolution is passed to the upper-level algorithm for further processing and optimisation.

### 3.2. Description of the upper-level algorithm

The upper-level algorithm is used to refine and optimise the rules evolved by the previous algorithm. This is achieved by gathering them into a single set and evaluating the efficiency of their cooperation. As a result, superfluous rules may be eliminated and the remaining ones simplified. Because at this stage a multiobjective algorithm is used, the evolved sets of rules have different features; they vary both in accuracy and complexity. The most general structure of the algorithm resembles the one introduced at the lower level where, an individual represented a rule consisting of premises that could remain active or

inactive. In this case a chromosome codes a set of rules that may be disabled in the same way, which excludes them temporarily from participating in classification. Again, in order to avoid operating on chromosomes with variable length, each rule is accompanied by a flag determining its status in the set. In the initial population all individuals are exact copies of the set of rules obtained from the lower-level algorithm. This implies that the maximal accuracy is available without the necessity for adding new rules. The only condition that has to be met is that all genes remain active. Therefore the task of the algorithm is limited to searching for sets with reduced complexity. However, the improvement of this criterion should be accompanied by the lowest possible deterioration of accuracy.

The process of reproduction, just like at the earlier stage, consists in exchanging genes in a random way between the parents (uniform crossover). This means that every rule and its flag in a newly created individual are copied from the parent that has been selected randomly for this purpose. Mutation occurs on two levels. First of all, a randomly selected flag may be changed, which results in altering the activation status of the corresponding rule. The effect is that a potentially superfluous rule is excluded from the set or, in the oppositecase, a previously removed one is restored. The other type of mutation affects single premises inside the rules and is performed in the same way as in the lower-level algorithm, thus leading to the modification of constraints imposed on the attributes. Since in the previous algorithm some rules were evolved on the basis of a reduced set of input vectors, at this stage they have a chance to improve their quality through the use of the entire vector set as a ground for evaluation, as well as cooperation with other rules.

The upper-level algorit hm is multiobjective and therefore its fitness function is based on the relation of domination. The entire process of selection, including the rank-based method of assigning fitness to an individual as well as the technique of the roulette wheel, is identical to the one used in the lower-level algorithm. The only difference consists in the choice of criteria for the evaluation of individuals and the way of calculating their values. The coverage of a set is determined by the number of input patterns that are correctly classified by the set as a whole, i.e. by at least one of its active rules. The error, on the contrary, depends not only on the number of misclassified vectors, but also on the frequency of a given mistake. In other words, the error is calculated as the sum of errors for all active rules in the set. On the upper-level another criterion, namely the complexity of a set, is added. Its value is determined by the number of active rules in a set increased by the number of active premises in each of them. This criterion could be split into two, but, apart from complicating the algorithm, such a distinction would not result in a significant improvement of performance.

The process of evolution stops after evolving a given number of generations measured either from the beginning or since the last improvement observed (like in the previous algorithm). However, the definition of improvement in a multiobjective algorithm is not as straightforward as in classic algorithms, where the average fitness of the population may be easily compared in the course of evolution. In this case the quality of an individual is expressed as a vector consisting of several values. Average values for all criteria in the population may be calculated, but the comparison of different generations must be based on domination. In other words, a new generation is recognized as more fit only if the vector of those average values dominates a corresponding vector calculated for the previous generation. The output produced by the algorithm is a set of non-dominated solutions. The choice of the most appropriate one (in a given situation) is left to the user, who may apply certain weights to the criteria and obtain the solution with the highest weighed sum of their values. This might be done repeatedly, which results in presenting solutions with various features without the need for rerunning the algorithm with different parameters.

## 4. Experiments

The purpose of performing experiments is to verify whether the proposed method can be applied to various problems and whether it remains efficient regardless of the number and types of attributes in a set of input vectors. The use of popular benchmark data sets gathered in [8] during the tests facilitates comparing the method with other techniques developed for a similar class of problems. Table 1 presents the detailed information about the data sets that have been used for the tests, namely: *Iris*, *LED* – 24, *Monk* – 1 and *Quadrupoles*. A 2 % noise has been introduced to the *LED* – 24 data set.

Table 1. Data sets used for the tests

| Name (examples) | Type of attributes | Class | No of examples |
|---|---|---|---|
| Iris (150 ) | 3 continuous | Setosa | 50 |
| | | Versicolour | 50 |
| | | Virginica | 50 |
| LED-24 (1000) | 24 binary 2% noise | 10 classes | ≈100/ class |
| Monk-1 (432 ) | 6 enumerable | 0 | 216 |
| | | 1 | 216 |

Separate experiments were carried out directly on the data sets obtained from the repository [8], without having them processed by a neural network beforehand. One more data set (Quadrupeds) was added to verify the scalability

of the presented method. The reason is that a network tends to simplify data by eliminating noise at least partially. This should theoretically lead to reducing the complexity of rule sets and improving the efficiency of the algorithm, which has been observed during the tests. Extracting rules from the data processed by a network resulted in improved efficiency of the method and enhanced accuracy of the solutions.

The proposed genetic algorithm was tested on the data that had been processed by a multi-layer feed-forward neural network with one hidden layer, trained by the back-propagation algorithm. The accuracy achieved by the network in each case is presented in Table 2.

Table 2. Results of the network training

| data | Number of neurons | Accuracy |
|---|---|---|
| (set) | in the hidden layer | % |
| Iris | 3 | 98 |
| LED-24 | 5 | 95 |
| Monk-1 | 3 | 100 |

During the tests the lower-level algorithm worked with best effectiveness with a relatively low probability of crossover, not exceeding 50%, and a low probability of mutation, up to 1%. The desirable size of a single population depended strongly on the complexity of the problem, particularly on the number of attributes, and was usually set to 50 or 100 individuals. Increasing the size of a population (within reason) resulted in better efficiency of the search. On the upper level the probability of crossover and mutation was usually the same. Otherwise the changes in the chromosome became too rapid for the algorithm to be able to refine the rules through small modifications. At both stages of the algorithm its efficiency could be enhanced by adding the option of saving the best individuals in the population. In a multiobjective algorithm this is achieved by introducing an external population consisting of all non-dominated individuals found so far during the course of evolution, as described in [7] (SPEA algorithm). This permits to increase the probability of mutation without the risk of destroying the best individuals. In all cases the process of evolution was stopped when no improvement had been observed in the last 100 generations.

An example of rules for *Iris* evolved by the lower-stage algorithm for the data processed by the network is shown in Table 3. In this case the first rule in each class usually covers the majority of input patterns, whereas the remaining rules supplement it by covering all omitted examples. It is worth noticing that the number of patterns covered by each rule is calculated on the basis of a reduced set, after all vectors recognized by the previous rule have been excluded.

Therefore the actual number of examples covered by some of the rules may be larger than indicated. In all cases the number of misclassified examples is equal to zero, which results from the method of choosing the best rule (a very high weight assigned to error). The number of decimal places within the premises has been reduced for clarity. Table 4 presents the examples of sets obtained from the upper-level algorithm on the basis of the previously evolved rules. It is apparent that the improvement in accuracy is accompanied by increasing complexity of sets and rules – and vice versa.

Table 3. Rules for *Iris* evolved by the lower-level algorithm; *) indicates that coverage has been calculated on the basis of a reduced set of input patterns)

| No | Rule | Coverage |
|---|---|---|
| 1. | **IF** PetalLength ∈ [1.00; 2.90] **THEN** Setosa | 50 |
| 2. | **IF** PetalLength ∈ [1.97; 4.98] **AND** PetalWidth ∈ [0.96; 1.65] **THEN** Versicolour | 47 |
| 3. | **IF** SepalLenght ∈ [6.19; 7.17] **AND** SepalWidth ∈ [2.66; 3.32] **AND** PetalLength ∈ [1.00; 5.51] **AND** PetalWidth ∈ [0.88; 1.71] **THEN** Versicolour | 2* |
| 4. | **IF** PetalWidht ∈ [1.72;2.50] **THEN** Virginica | 46 |
| 5. | **IF** SepalLength ∈ [4.30; 6.12] **AND** SepalWidth ∈ [2.00;2.75] **AND** PetalLength ∈ [4.37; 5.76] **AND** PetalWidth ∈ [1.37; 2.50] **THEN** Virginica | 2* |
| 6. | **IF** PetalLenght ∈ [5.33;6.27] **THEN** Virginica | 1* |

Table 4. The two examples of sets of rules for *Iris* evolved by the upper-level algorithm (*cov* – stands for coverage, *compl* – stands for complexity)

| No | Rule | Cov. | Error | Compl. |
|---|---|---|---|---|
| 1. | **IF** PetalLength ∈ [1.00; 2.90] **THEN** Setosa<br>**IF** PetalLength ∈ [1.97; 4.98] **AND** PetalWidth ∈ [0.96; 1.65] **THEN** Versicolour<br>**IF** PetalWidht ∈ [1.72;2.50] **THEN** Virginica | 143 | 0 | 7 |
| 2. | **IF** PetalLength ∈ [1.00; 2.90] **THEN** Setosa<br>**IF** PetalLength ∈ [1.97; 4.98] **AND** PetalWidth ∈ [0.96; 1.65] **THEN** Versicolour<br>**IF** SepalLenght ∈ [6.19; 7.17] **AND** PetalWidth ∈ [0.88; 1.71] **THEN** Versicolour<br>**IF** PetalWidht ∈ [1.72;2.50] **THEN** Virginica<br>**IF** SepalWidht ∈ [2.00; 2.62] **AND** PetalWidht ∈ [1.37;2.50] **THEN** Viginica | 148 | 2 | 13 |

The general features of rule sets obtained by the method in question for the chosen benchmark data sets are gathered in Tables 5 and 6 – for the data passed through a network and for raw data, respectively. The algorithm has been run

five times for every data set and the average values of accuracy corresponding to particular sizes of rule sets have been calculated. In each case the algorithm evolved a wide range of sets with various features, including complex sets classifying correctly all the instances, as well as relatively simple solutions with a few rules for each category. This proves that the proposed method is suitable for different types of attributes. Moreover, very good results obtained for Quadrupeds are the evidence for the method scalability, i.e. the ability of solving problems with large numbers of attributes.

Table 5. The results of experiments (*size* – number of rules, *acc* – accuracy as the difference between coverage and error) performed on the data processed by a neural network

| Iris | | LED-24 | | Monk-1 | |
|---|---|---|---|---|---|
| *size* | *acc* | *size* | *acc* | *size* | *acc* |
| 1 | 50 | 1 | 106 | 1 | 72 |
| 2 | 97 | 5 | 507 | 2 | 144 |
| 3 | 141 | 10 | 926 | 3 | 216 |
| 5 | 147 | 15 | 943 | 5 | 360 |
| 8 | 150 | 20 | 953 | 7 | 432 |

Table 6. The results of experiments (*size*– number of rules, *acc* – accuracy as the difference between coverage and error) performed on the raw data

| Iris | | LED-24 | | Monk-1 | | quadrupeds | |
|---|---|---|---|---|---|---|---|
| *size* | *acc* | *size* | *acc* | *size* | *acc* | *size* | *acc* |
| 1 | 50 | 1 | 106 | 1 | 72 | 1 | 144 |
| 2 | 97 | 5 | 507 | 2 | 144 | 2 | 277 |
| 3 | 141 | 10 | 926 | 3 | 216 | 3 | 388 |
| 5 | 147 | 15 | 943 | 5 | 360 | 4 | 497 |
| 8 | 150 | 20 | 953 | 7 | 432 | 6 | 500 |

The results for Iris, LED-24 and Quadrupeds indicate that a set consisting of one rule per category may cover most of the instances although some level of misclassification is possible. Additional rules improve the accuracy in a relatively low degree. This applies especially to the LED-24 data set, where the accuracy of 96% and 92% was achieved by a set of 10 rules whereas a set classifying correctly all input vectors contained 23 rules for the data processed by the network and over 60 rules for the raw data (not presented in the table). The observed disproportion is caused by noise that requires very specific rules to deal with certain examples. A small increase in accuracy observed when new rules are added to a set indicates that the algorithm starts taking exceptions into account, which is usually not desirable – unless these specific rules lead to discovering "hidden" knowledge, i.e. unknown dependencies concerning small subsets of instances. Therefore the most complex rule sets evolved for Iris,

LED-24 and Quadrupeds have little value. This observation is possible due to the multiobjective approach used in the algorithm that allows to develop different solutions in parallel, thus facilitating their analysis and comparison. The results produced by the presented algorithm may be confronted with those obtained for a different multiobjective algorithm (GenPar) described in [2]. Since the same data sets have been used in both cases, it is possible to compare the effectiveness of the methods. However, this can be done to a certain degree only because of differences in the way of performing experiments. The most significant one is that the results presented in [4] and copied to Table 6 contain information on the number of correctly classified instances only (fidelity), without providing any data about the error made by a given set. Moreover, no superfluous attributes had been added to the *LED* data set.

The results for GenPar, shown in Table 7 prove to be less accurate than those for MulGEx. It is worth noticing that GenPar did not produce a set with maximal possible accuracy in any of these cases. This might be caused by the fact that it operates on entire sets only, which hinders individual rules from being intensively optimised (the lower-level algorithm in the proposed method is aimed solely at improving single rules), as well as by combining coverage and error into one criterion. The presented algorithm, on the contrary, considers them separately, thus gaining the ability of finding a solution with maximal coverage which is followed by modifications leading to the reduction of error.

Table 7. The results of experiments for GenPar (*size* – number of rules, *fid* –fidelity equivalent to coverage) on the basis of [4]

| 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 | - | - |

## 5. Conclusions

The results of the performed experiments indicate that the proposed method shows good effectiveness when extracting rules from various data, with different numbers and types of attributes. It enables producing a wide range of sets of rules, varying from those with perfect accuracy achieved at the cost of high complexity to relatively simple ones that cover only the most typical cases, as well as dealing with the problem of noisy data. This is obtained by combining a hierarchical genetic algorithm with a multiobjective approach based on the concept of Pareto optimality. The method is architecture independent. It can be used for any classification problem performed by a neural network regardless of

its structure, as long as the responses given by the network for the examined set of input vectors are known. The main weakness of this method consists in the necessity of adjusting parameters for both genetic algorithms in order to ensure the effectiveness of evolution. The optimal values of parameters may be different for each problem and no technique of their precise estimation has been developed. Another disadvantage is that the algorithm needs to examine the entire set of input vectors repeatedly to calculate the values of the criteria for each individual. Even if an effective data representation is implemented, large sets cause the deterioration of efficiency. In spite of these drawbacks present in many GA-based methods of rule extraction, the results produced by the proposed algorithm may be considered satisfactory.

## References

[1] Andrews R., *Survey and critique of techniques for extracting rules from trained artificial neural networks*. Knowledge-Based Systems, 8 (1995) 1.

[2] Markowska-Kaczmar U., Zagorski R., *Hierarchical evolutionary algorithm in the rule extraction from neural network*. ACM Computing SurveysInternational Symposium on Engineering of Intelligent Systems, (2004).

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

[4] Markowska-Kaczmar U., Wnuk-Lipinski P., *Rule extraction from neural network by genetic algorithm with pareto optimization*. In Rutkowski L., ed.: Artificial Intelligence and Soft Computing. Lecture Notes in Computer Science, Lecture Notes in Artificial Intelligence, (2004) 450.

[5] Goldberg D.E., *Genetic Algorithms in Search, Optimization and machine*. Addison-Wesley Publishing Company, Reading, Massachusetts, (1989).

[6] Fonseca C.M., Fleming P.J., *Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization*. In S. Forrest Ed., Proceedings of the Fifth International Conference on Genetic Algorithms, (1993) 416.

[7] Zitzler E., Thiele L., *Multiobjective evolutionary algorithms: A comparative case study and strength pareto approach*. IEEE Transaction on Evolutionary Computation, 3 (1999) 257.

[8] Blake C.C., Merz C., *Uci repository of machine learning databases*. University of California, Irvine, Dept. of Information and Computer Sciences, (1998).