Proceedings of the International Multiconference on Computer Science and Information Technology pp. 141–148
ISSN 1896-7094

GA-Based Rule Extraction from Neural Networks for Approximation

Urszula Markowska-Kaczmar and Krystyna Mularczyk

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

Abstract. neural networks solving approximation problem. It is based on two hierarchical evolutionary algorithms with multiobjective Pareto optimisation. The lower level algorithm searches for rules that are optimised by the upper level algorithm. The conclusion of the rule takes the form of a tree whose inner nodes contain functions and operators, and leaves—identifiers of attributes and numeric constants. The details referring to the rules encoding, genetic operators and fitness function are described. Some preliminary results of experimental studies are presented, as well.

1 Introduction

Neural networks' inability to explain and justify their decisions inclines scientists towards developing methods of rule extraction. The prepositional rules are the most popular representation of knowledge extracted from neural networks because of their natural comprehensibility for human. A wide variety of network models as well as tasks performed by networks requires the use of different approaches to rule extraction. Generally, these methods may be divided into local ones that examine the structure of a network and global ones, independent of the network's architecture and based on training patterns [2]. A survey of the methods can be found in [10] or [3].

The most popular problems solved by networks include classification and approximation. The vast majority of methods of rule extraction deal with the first one. Methods dedicated to approximation, because of the problem's complexity, are very rare, and the local approach is usually taken.

Methods of rule extraction from networks that perform approximation are very rare. The main reason is that this problem, if compared to classification, is very complicated. There are practically no global methods in literature. It proves to be much more convenient to examine the structure of a network and calculate the dependencies between the inputs and outputs on the basis of the network's parameters, especially weights. Therefore local methods, such as REFANN [9] or [10], are preferred.

REFANN is a method dedicated to rule extraction from feed-forward networks. The analysis of weights and activation functions of individual neurons allows dividing the input space into subregions. Then, for each of these subregions, the algorithm finds the formula of a linear function that reflects the network's behaviour with the highest fidelity. REFANN, as every local method, tends to be efficient only for relatively small networks, that's why a network should be pruned before the algorithm is applied.

This paper describes a new method of rule extraction from networks that perform the task of approximation. The method is based on a multiobjective Pareto approach with using genetic algorithm. It is a modified version of the rule extraction method proposed for classification [6]. The results of preliminary experiments evaluating its efficiency are shown at the end of the paper.

2 Problem formulation

In the proposed method neural networks is perceived by the input patterns and outputs produced on this basis. It is so called global approach to rule extraction from the neural network that in this case performs approximation. Approximation consists in determining the formula of a function on the basis of its values in certain known points but when it is realised by a neural network the function is hidden in the parameters and architecture of the network.

Inputs and outputs produced by the network create the training set for the method of rule extraction. Let's assume that the training set consists of patterns of the form $\overline{x_i} = [x_{i1}, x_{i2}, \ldots, x_{in}, y_i]$, where $x_{ij} \in D_j$, $j \in \{1..n\}$, represent the coordinates of a given point that belong to domains D_j and y_i denotes the value of the function in this point produced by the network. We can say that the network realizes a function f that would satisfy the condition shown in (1).

$$(\forall i) f(x_{i1}, x_{i2}, \ldots, x_{in}) = y_i \tag{1}$$

Our aim is to find rules imitating behaviour of the neural network performing approximation. The rules produced by a method of rule extraction should be represented in a comprehensible way. The most popular notation is based on the 0-order logic (2). C_j, $j \in \{1..n\}$, are constraints imposed on individual attributes; if all of them are fulfilled, then the formula present in the conclusion applies.

$$C_1(x_{i1}) \wedge C_2(x_{i2}) \wedge \cdots \wedge C_n(x_{in}) \Rightarrow f(x_{i1}, x_{i2}, \ldots, x_{in}) \tag{2}$$

The form of a constraint depends on the type of the corresponding attribute. For logical (boolean) attributes, a premise (3) determines which of the two values is required by the rule. A constraint imposed on enumerable attributes defines a subset of acceptable values (4). Real attributes may be constrained by defining the minimum and maximum value (5).

$$x_i = val, \quad where\ val \in \{true, false\} \tag{3}$$

$$x_i \in V_i, \quad where V_i \in D_i \tag{4}$$

$$x_i \in [val_{min}, val_{max}] \tag{5}$$

Since not all attributes may be required when dividing the space into regions where the approximated function is continuous, some of them may be excluded from a given rule, thus making it more comprehensible. If a rule does not contain a premise corresponding to a given attribute, all its values are accepted.

An important issue in rule extraction is finding a compromise between the fidelity and comprehensibility of created rules. It stands to reason that describing complex behaviour of a network requires a large number of complicated rules. On the other hand, rules must be understandable for humans, therefore simplicity should be maintained. These goals are contradictory, which implies that rule extraction is a multiobjective problem. This fact should be taken into account when designing a method of extraction.

3 MulGEx as a new method of rule extraction

Rule extraction from neural networks can be seen as an optimization problem. If we take into account the existence of different type of attributes (enumerative, binary, continuous and discrete) arriving in the input patterns and different formulae of function that could be placed in the conclusion of a rule the need to find rules is similar to NP problem. Genetic programming or generally evolutionary algorithms are well known as a tool that allows searching for the optimal solution in a large, multidimensional space of possible solutions in an effective way. That is why they are used in MulGEx.

To improve efficiency, two algorithms working on two levels have been introduced, as in [7]. The lower level algorithm aims at evolving single rules and passing them to the upper level algorithm. The latter evaluates the performance of the entire set of rules and optimizes it if necessary. The general idea of MulGEx has been presented in Fig. 1.

3.1 The lower-level algorithm

The lower level algorithm is based on the Michigan approach, i.e. every individual encodes a single rule. This allows effective optimization of the conclusion, as well as adjusting the borders of the region where a given formula of a function applies.

Because more than one rule is usually necessary to describe the behaviour of a network, one must implement a mechanism of dealing with multimodality of the problem. The idea of sequential covering [11] has been used in MulGEx.

The Pareto approach introduced at this level (described in details in [6]) allows eliminating the problem of weight selection that arises when scalarisation is used. Because the number of required rules and the complexity of the function realised by the network is very difficult to determine beforehand, it is hard to find a configuration of weights that would give a satisfactory solution. It also improves efficiency, since the presence of a goal function in the form of a vector provides the algorithm with more information about every individual and allows detecting potentially good solutions. A classical algorithm evolves the individuals in one direction, which makes it more vulnerable to getting stuck at local optima.

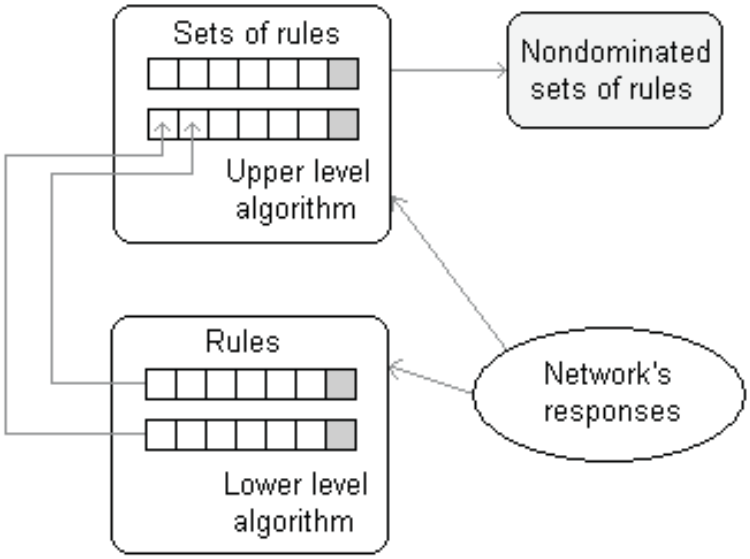

Fig. 1. The schema of MulGEx

The form of the chromosome The chromosome on the lower level encodes a single rule. It consists of a vector of premises followed by a conclusion that represents a function formula. Each premise is accompanied by a flag that determines if a given constraint is active, i.e. taken into account when determining which patterns match the rule.

Several types of premises have been implemented to represent different types of constraints:

- a single boolean value determines the acceptable value of a logical attribute;
- an array of logical values represents the subset of values for an enumerable attribute;
- two real values stand for the minimum and maximum value that a real attribute can take.

The conclusion may be constructed in various ways, which influences the number of different functions that can be represented, but also the complexity of the rule and the size of the space of possible solutions. The simplest version allows encoding only linear functions and takes the form of a vector of coefficients, as shown in Fig. 2 (MulGEx-Vector). The value of such an expression is calculated according to formula (6), where $x_1, \ldots, x_n$ are input attributes and $a_0, \ldots, a_n$ are numeric coefficients.

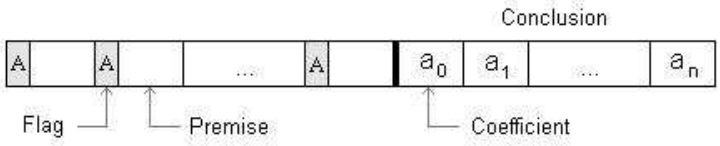

Fig. 2. The form of the chromosome on the lower level—conclusion as a vector

$$f(x_1, x_2, \ldots, x_n) = a_0 + \sum_{i=1}^{n} a_i x_i \tag{6}$$

Alternatively, the conclusion takes the form of a tree whose inner nodes contain functions and operators, and leaves – identifiers of attributes and numeric constants (MulGEx-Tree). The expression encoded in the tree represents more complicated dependencies between the neural network's inputs and its output.

The set of valid expressions may be defined recursively in the following way:

- an identifier of an attribute is an expression;
- a numeric constant is an expression;
- if e_1 and e_2 are expressions, then $(e_1 + e_2)$, $(e_1 - e_2)$, $(e_1 * e_2)$ and, if $e_2 \neq 0$, (e_1/e_2) are also expressions.

The form of the chromosome for this type of conclusion is presented in Fig.3.

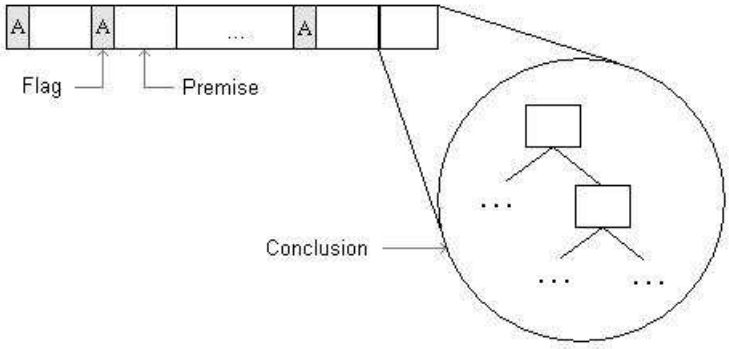

Fig. 3. The form of the chromosome on the lower level—conclusion as a tree

Genetic operators Genetic operators enable chromosome modifications as well as exchanging genetic information between individuals. Since the chromosomes are complex in MulGEx (they have variable length and consist of values of different types), specialised operators of crossover and mutation had to be designed.

Crossover is performed in a different way for the vector of premises within an individual and for the conclusion. Premises are exchanged in an uniform way, i.e. every gene is taken from a randomly selected parent. A gene is understood as a single real or boolean value, including activation flags. Therefore premises in the offspring are created in the following way:

- Logical premise – value copied from one parent; activation flag copied from one parent (selected independently);
- Enumerable premise – each value copied from a randomly selected parent; flag copied separately;
- Real premise – minimum and maximum values copied independently from the parents; if the minimum value in the offspring happens to be larger than the maximum value, the chromosome is corrected by swapping these values; flag copied separately.

The conclusion of the offspring, if presented as a vector of coefficients, is also created by means of uniform crossover. Every coefficient is taken from one of the parents. The conclusion in the form of a tree is created by selecting a subtree from each parent's conclusion and exchanging these subtrees.

The fitness function and the method of selection The following criteria are used for the evaluation of individuals:

- coverage – the number of patterns that fulfil the active constraints defined in the premises;
- error – the average difference between the neural network's output and the value of the conclusion, calculated for all covered patterns;
- complexity – the sum of the number of active premises and the number of elements (nonzero coefficients in MulGEx-Vector or nodes in MulGEx-Tree) in the conclusion.

These criteria are gathered into a single function (7) optimized by the algorithm.

$$f_{goal} = [coverage, error, complexity] \tag{7}$$

Because this function takes the form of a vector, the problem of comparing the individuals arises. Therefore fitness is calculated on the basis of domination [1], [4]. In a problem of minimisation, for two individuals x and y, such that $f_{goal}(x) = [f_1(x), f_2(x), \ldots, f_m(x)]$ and $f_{goal}(y) = [f_1(y), f_2(y), \ldots, f_m(y)]$, where m represents the number of objectives, x dominates y ($x \prec y$) only if the condition shown in (8) is met.

$$x \prec y \Leftrightarrow (\forall i = 1 \ldots m)\ f_i(x) \leq f_i(y) \wedge (\exists i = 1 \ldots m)\ f_i(x) < f_i(y) \tag{8}$$

The range assigned to an individual is equal to the number of other individuals that dominate it (9).

$$range(x) = count\{y \in Population | y \prec x\} \tag{9}$$

The best individuals (nondominated) receive the lowest ranges. Fitness is assigned to individuals on the basis of ranges.

The roulette wheel method is used during selection. This means that the probability of selecting a given individual is proportional to its fitness value.

If the conclusion of a rule takes the form of a tree, genetic programming is involved ([5]) and the efficiency of the algorithm is deteriorated. Therefore on the basis of the experimental studies one assumes that during reproduction only half of the population is created by means of crossover. Another 25% of individuals are the fittest individuals taken from the previous generation (elitism). This ensures that the best properties are preserved. The remaining 25% of individuals are created randomly, which aims at introducing new properties and maintaining diversity in the population.

3.2 The upper-level algorithm

The upper level algorithm is based on the Pitt approach, which means that entire sets of rules are evolved. This allows examining how well the rules cooperate in producing the answer as well as detecting superfluous or contradictory rules. Because this approach tends to be much less effective than Michigan, the initial population consists of sets of rules created by the lower level algorithm. This guarantees that high fidelity is achieved.

The Pareto approach allows creating an entire set of solutions with different properties in a single run of the algorithm. The final choice is left to the user who may decide whether he wants to obtain complicated rules with maximum accuracy or rather simple and understandable sets.

The form of chromosome The chromosome on the upper level consists of a vector of rules. A single rule is encoded in the same way as on the lower level. Because one cannot determine beforehand how many rules are necessary to describe the behaviour of a network, the length of the chromosome is variable. In order to avoid problems connected with variable-length chromosomes, an activation flag is added to every rule. The flag determines if a given rule belongs to the set. The length of the vector is equal to the number of rules evolved on the lower level and it is assumed that no new rules need to be added. However, they may be removed from the set, which improves comprehensibility.

Genetic operators Specialised genetic operators must be defined on the upper level to enable handling the complex chromosome.

Crossover is performed in a uniform way, where a single rule along with its flag is treated as one gene. As a result every rule in the offspring's chromosome is copied from a randomly selected parent.

Mutation is more complicated, since it must allow modifying the contents of a given set by adding or removing rules as well as the contents of individual rules. The first goal is achieved by changing the value of every activation flag with a small probability. The latter – by applying the mutation operator defined on the lower level to every rule.

The main purpose at this stage is to detect superfluous rules and premises and exclude them from the solution. However, minor adjustments of the constraints in the premises as well as values in the conclusion are also possible.

Evaluation of individuals The criteria of evaluation of rule sets are based on those defined for single rules. They include:

- coverage – the number of patterns recognized by the set (i.e. at least one active rule within the set);

– error – the sum of the average errors made by each active rule;
– complexity – the sum of the complexity of each rule increased by the number of active rules in the set.

The assignment of ranges and the fitness value is based on the Pareto approach and performed in the same way as on the lower level. The roulette wheel method is used during selection. Niching by means of a sharing function is implemented to maintain diversity in the population.

4 Experimental study

Experimental study aims at verifying whether the method is able to find satisfactory solutions for different datasets, regardless of the number of attributes and the formula of the approximated function. Datasets used during the tests have been gathered in Table 1. *Set-1* and *Set-2* have been created by the authors, *Pollution* is a popular benchmark set taken from [8].

Table 1. Datasets used during the tests

Set	Attributes	Number of patterns	Function
Set-1	8 real	20	$f(\overline{x}) = x_1 - 2 * x_2$
Set-2	2 real	50	$f(\overline{x}) = \begin{cases} x_1 + x_2 + 1 \; dla \, x_1 \geq 5 \\ x_1 + x_2 - 1 \; dla \, x_1 < 5 \end{cases}$
Pollution	15 real	60	unknown

All sets have been processed by a feed-forward neural network trained by means of the backpropagation method. A linear activation function has been used for *Set-1* and *Set-2* and a sigmoidal one for *Pollution*.

The effectiveness of rule extraction depends to a certain degree on the parameters of the algorithm. Genetic programming (MulGEx-Tree) requires relatively large populations, with the number of individuals set to at least 300, even for simple problems. The probabilities of crossover and mutation may be high, e.g. 0.9 and 0.1, respectively. This allows introducing new features to the chromosome. Since standard genetic programming is used in MulGEx-Vector, the number of individuals may be reduced to 50-100. The probability of mutation should be lower as well and adjusted to the number of input attributes, so that the chromosome does not change too rapidly.

The results of rule extraction for the simplest sets (*Set-1* and *Set-2*) have been gathered in Table 2.

Table 2. The results of rule extraction for simple sets (*Gener.* – the number of generations, *Compl.* – the complexity of the rule; $Vector$, $Tree$ – the form of the conclusion

Set-1				**Set-2**			
Vector		Tree		Vector		Tree	
Gener.	Compl.	Gener.	Compl.	Gener.	Compl.	Gener.	Compl.
33	3	211	5	126	3	431	13
51	3	329	5	93	3	647	9
26	3	87	7	81	3	982	13
86	3	354	5	164	3	475	9
37	3	91	7	132	3	783	15

The results indicate that the variant with a vector of coefficients proves to be much more efficient in the case of linear functions. The algorithm requires less generations to find a satisfactory result, also in the case of superfluous attributes. The reason is that, when evolving a tree, the algorithm must find the correct expression as well as adjust coefficients. The conclusion in the form of a vector eliminates the first problem – the type of function is known in advance and only constant values must be optimized.

The performance for much complcated sets is measured using the *Pollution* dataset. Table 3 presents example rules generated after 500 generations of the lower-level algorithm in MulGEx-Vector (real values have been truncated). Rules evolved by MulGEx-Tree are shown in table 4.

Table 3. Example rules created for *Pollution* by MulGEx-Vector (*Cov.* – coverage)

Rule	Cov.	Error
$JUT \in [0.00 - 0.97] \wedge OVR65 \in [0.062 - 0.99] \wedge$ $\wedge SO2 \in [0.00 - 0.77]$ $\Rightarrow 0.92 * JUT + 0.061 * HC + -0.006$	54	0.110
$PRE \in [0.31 - 0.94] \wedge JUT \in [0.31 - 0.79] \wedge$ $\wedge OVR65 \in [0.07 - 0.99] \wedge HUMID \in [0.00 - 0.90]$ $\Rightarrow 0.93 * JUT + 0.97 * NOX$	42	0.070
$PRE \in [0.31 - 0.94] \wedge JUT \in [0.39 - 0.79] \wedge$ $\wedge OVR65 \in [0.06 - 1.00] \wedge POPN \in [0.01 - 0.79] \wedge$ $\wedge SO2 \in [0.00 - 0.77]$ $\Rightarrow 0.93 * JUT + 0.004 * HUMID$	28	0.053

Table 4. Example rules created for *Pollution* by MulGEx-Tree (*Cov.* – coverage)

Rule	Cov.	Error
$WWDRK \in [0.054 - 0.98]$ $\Rightarrow ((3.55 + (POPN * 3.35)) * ((NOW + OVR65)/9.95))$	58	0.083
$JUT \in [0.22 - 0.80 \wedge HOU \in [0.10 - 1] \wedge$ $\wedge SO2 \in [0.01 - 0.85] \wedge HUMID \in [0.10 - 0.90]$ $\Rightarrow ((3.73 + (PRE * 3.28926)) * ((NOW + NUMID)/9.8))$	44	0.057
$PRE \in [0.33 - 0.70] \wedge EDUC \in [0.54 - 1.00] \wedge$ $\wedge SO2 \in [0.00 - 0.85] \wedge HUMID \in [0.00 - 0.91]$ $\Rightarrow ((3.66 + (PRE * 3.19)) * ((NOW + HUMID)/10))$	20	0.031

The results obtained for the *Pollution* dataset prove that the algorithm becomes less efficient when the number of input attributes is increased and the dependency between the neural network's inputs and output becomes complicated. The presence of many dimensions in the space of possible solutions as well as many local optima prevents the algorithm from finding reasonable and comprehensible rules.

The accuracy of rules evolved by MulGEx-Tree is usually higher that that of MulGEx-Vector. The reason is that the conclusion in the form of a tree may contain various types of functions, therefore it becomes easier to obtain a function that would match the responses of a network. On the other hand, MulGEx-Vector is much faster, since only the coefficients need to be optimized, and the rules are usually more comprehensible.

An important advantage is that thanks to the Pareto approach MulGEx is able to find rules with various properties in a single run. Some of them cover almost all input patterns and show the most general dependencies between the inputs and outputs, whereas other rules are more accurate, but also less understandable. The final choice of the best solution is left to the user.

5 Comparison to other methods

The comparison of the presented method with other methods of rule extraction is hardly possible. One of the reasons is that all methods dedicated to approximation known to the authors are based on the local approach and therefore may be used only with certain types of neural networks ([9]). Moreover, the results produced by algorithms of rule extraction are difficult to verify – without being able to interpret them and evaluate the correctness and usefulness of the rules, one cannot determine which method is better. It's worth noticing that MulGEx is more flexible than local methods and produces rules with various properties; this makes the solution more understandable for humans. On the other hand, local methods are more accurate, since the structure of the network may be examined.

6 Summary

A new global method of rule extraction from neural networks that perform the approximation tasks has been presented in the paper. Because approximation consists in searching for a formula that describes the dependency between a network's input and output, genetic programming has been used (MulGEx-Tree). Alternatively, an assumption has been made that the approximation is linear, which simplifies the form

of the conclusion (MulGEx-Vector). The specifics of the problem of rule extraction for approximation required introducing Pareto-optimization as a way of dealing with multiobjective problems.

MulGEx proves to be effective if the number of attributes is relatively small and the approximated function is simple; otherwise its efficiency deteriorates. It is also difficult to verify the correctness and usefulness of the obtained results.

Further work should be concentrated mainly on the improvement of the algorithm's effectiveness. In MulGEx-Vector it would be possible to evolve only the premise part (by means of a genetic algorithm) and calculate the best linear approximation of the function for a given area using a different algorithm. The problem with coefficients observed during the evolution of expressions in MulGEx-Tree might be solved by introducing an auxiliary genetic algorithm that would adjust constant values in conclusions. Implementing an additional mechanism of conclusion simplification would also be helpful, since some expressions in the tree may be reduced without changing their values. For example, subtrees that consist of operators and constant values only could be replaced with a single leaf.

References

1. C. Coello, D. A. Van Veldhuizen, and G. B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publisher, New York, 2002.
2. M. Craven. *Extracting Comprehensible Models From Trained Neural Networks*. PhD thesis, University of Wisconsin, Madison, 1996.
3. W. Duch, R. Setiono, and J. Zurada. Coputational intelligence methods for rule based data uderstandning. *Proceedings of the IEEE*, **92**(4), 2004.
4. C.M. Fonseca and P.J. Fleming. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In S. Forrest Ed., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 416–423, San Mateo, California, 1993.
5. J. Koza. *Genetic Programming*. MIT Press, London, 1996.
6. K. Markowska-Kaczmar, U. Mularczyk. Ga- based pareto optimization for rule extraction from neural networks. In Jin Y., editor, *Multiobjective Machine Learning*, Studies in Computational Intelligence, 2006.
7. U. Markowska-Kaczmar and R. Zagórski. Hierarchical evolutionary algorithms in the rule extraction from neural networks. Technical report, Fourth International Symposium on Engineering of Intelligent Systems, EIS, Funchal, Portugal, 2004.
8. R. C. McDonald, G. C.and Schwing. Instabilities of regression estimates relating air pollution to mortality. *Technometrics*, **15**:463–482, 1973.
9. R. Setiono, W. K. Leow, and J. M. Zurada. Extraction of rules from artificial neural networks for nonlinear regression. *IEEE Transactions On Neural Networks*, **13**(3), 2002.
10. A. Tickle, R. Andrews, M. Golea, and J. Diederich. The truth will come to light: Directions and challenges in extracting the knowledge embedded within trained artificial neural networks. *IEEE Trans on Neural Networks*, **9**(6):1057–1068, 1998.
11. A. Weijters and J. Paredis. Rule induction with genetic sequential covering algorithm (geseco). In *Proceedings of the second ICSC Symposium on Engineering of Intelligent Systems, (EIS 2000)*, pages 245–251, 2000.