Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information Sciences 176 (2006) 691–724
INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL
www.elsevier.com/locate/ins

A comparison of classification accuracy of four genetic programming-evolved intelligent structures

Athanasios Tsakonas *

Department of Informatics, Aristotle University of Thessaloniki,
Artificial Intelligence and Information Analysis Laboratory, 54006 Thessaloniki, Greece

Received 13 October 2003; received in revised form 27 January 2005; accepted 16 March 2005

Abstract

We investigate the effectiveness of GP-generated intelligent structures in classification tasks. Specifically, we present and use four context-free grammars to describe (1) decision trees, (2) fuzzy rule-based systems, (3) feedforward neural networks and (4) fuzzy Petri-nets with genetic programming. We apply cellular encoding in order to express feedforward neural networks and fuzzy Petri-nets with arbitrary size and topology. The models then are examined thoroughly in six well-known real world data sets. Results are presented in detail and the competitive advantages and drawbacks of the selected methodologies are discussed, in respect to the nature of each application domain. Conclusions are drawn on the effectiveness and efficiency of the presented approach.

Keywords: Genetic programming; Context-free grammars; Decision trees; Artificial neural networks; Fuzzy rule-based systems; Fuzzy Petri-nets

* Tel.: +30 693 789 1399.
E-mail address: tsakonas@stt.aegean.gr

0020-0255/$ - see front matter
doi:10.1016/j.ins.2005.03.012

1. Introduction

Genetic Programming (GP) is a search methodology belonging to the family of evolutionary computation (EC) [15,47]. These algorithms nowadays have been applied in a wide range of real-world problems. Among successful EC implementations, GP retains a significant position due to its valuable characteristics, such as the flexible variable-length solution representation and the absence of population convergence tendency. Genetic programming in its canonical form enables the automatic generation of mathematical expressions or programs. Grammar-guided genetic programming (G^3P) [46] for knowledge discovery is an extension to the original GP concept and it makes possible the efficient automatic discovery of empirical laws. It relates to the Machine Discovery framework, originally described by Langley [17], which incorporated inductive heuristics and suffered from limitations regarding ill-conditioned data and large search spaces [27]. Genetic programming however can avoid these problems due to its stochastic nature. The grammar guided genetic programming provides a more systematic way to handle typing, and in this respect, it has more theoretical basis than strongly typed GP [20]. More importantly, the G^3P can constrain search space so that only grammatically correct individuals can be generated. The G^3P framework has already been applied in a variety of domains, such as the software project effort estimation [48] and the identification of macro-mechanical models [49]. Moreover, related techniques, based on G^3P, have been introduced, such as the grammar model-based evolution [50] and the tree-adjoining grammars [51].

In the present paper, the grammar-guided genetic programming paradigm is applied in a series of classification problems mainly corresponding to medical domains. Various computational intelligent approaches have been applied to medical problems in the past, including artificial neural networks (NN) and fuzzy systems [18] and evolutionary algorithms [23]. The problems addressed in this work have been used extensively as benchmarking data in the machine learning society. This data consists of six diagnosis and classification problems from the *Proben1* collection [25] of real-world data sets. We applied four methods for knowledge discovery using G^3P. The first approach generates a generalized form of decision trees. The second method creates fuzzy rule-based classifiers. The third approach generates a class of artificial neural networks. The fourth model creates fuzzy Petri-nets (FPN). For the latter two methodologies, we describe the *cellular encoding* paradigm in the GP grammar, in order to allow arbitrary network sizes and topologies. The models we selected to test have different properties for the discovery of empirical laws, a fact that makes useful the comparison of their effectiveness in dissimilar problems. Their difference is noteworthy both in terms of their *structure* (trees, competitive bases, networks) and in terms of *knowledge* elements (crisp rules, fuzzy rules, synapses). Specifically, the decision tree method creates a hierarchical crisp

rule-based classification tree. The fuzzy model generates competitive fuzzy rule-based classifiers. The neural network approach generates arbitrary massively parallel networks. The fuzzy Petri-nets methodology creates cooperative fuzzy rule networks.

The objective of this work is first to demonstrate the applicability of the G^3P for knowledge discovery in a range of different models and problems. Second, to compare the effectiveness of each of the four approaches and try to point out advantages and drawbacks of each approach in respect to the characteristics of each application domain. Additionally, the selected problems make possible for the reader to measure their relative success by further comparing the results to those found in literature [4,25].

The paper is organized as follows. In Section 2 we introduce the theoretical background of GP. We also present the G^3P principles, context-free grammars and current research in the field of cellular encoding. In Section 3 we give a short description of the data used in our experiments. Section 4 presents the experimental setup of our models. Our results regarding effectiveness and training time are presented in Section 5. The last section of this paper concludes with comparison of results and with a short discussion on future perspectives regarding this work.

2. Background

Evolutionary Computation has been an alternative methodology nowadays for many real-world problems [37,38]. Being a part of the evolutionary computation algorithms, the genetic programming advance [39] has valuable features, such as the variable length solution representation and population diversity maintenance [15]. In general, evolutionary models were inspired by the Darwinian theory of evolution. According to the most common implementations, a population of candidate solutions is maintained, and after a generation is accomplished, the population is expected fitted better for a given problem. Three genetic operators are mostly used in these algorithms:

- *reproduction*: copies an individual without affecting it,
- *recombination (crossover)*: exchanges genetic material between two individuals,
- *mutation*: exchanges a part of a randomly selected genetic material.

An evolutionary algorithm is summarized in the following steps [16]:

1. Initialize a population of individuals at random.
2. Evaluate randomly an individual and compare its fitness to other (this fitness determines how closely is an individual to the desired goal).

3. Modify an individual with a relatively high fitness using a genetic operator.
4. Repeat steps 2–3 until a termination criterion is met.

The described procedure is depicted in the flowchart of Fig. 1.

Usual termination criterions appear to be the accomplishment of a number of generations, the achievement of a desired classification error, etc. Genetic programming uses tree-like individuals that can represent mathematical expressions, making valuable the application of GP in symbolic regression problems. Such a GP individual is shown in Fig. 2.

2.1. Grammar-guided genetic programming

The prime advantage of genetic programming over genetic algorithms, is the ability to construct functional trees of variable length. This property

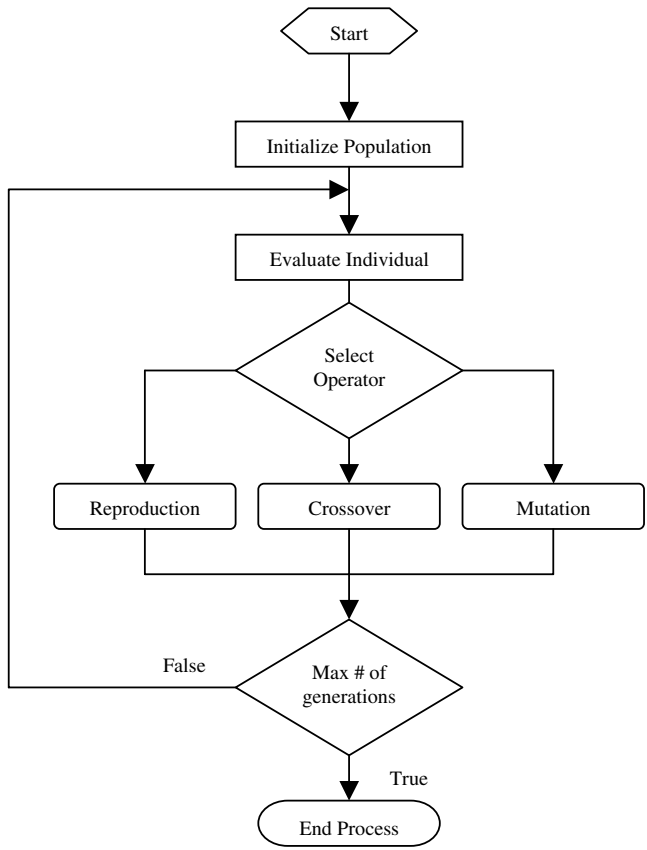

Fig. 1. Overview of a genetic programming run.

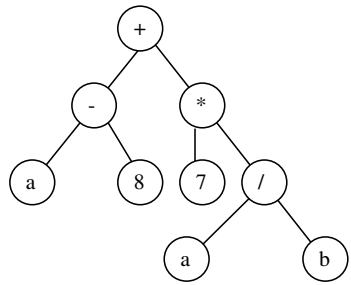

Fig. 2. Tree representation of the program (expression): $(a-8) + 7 * (a/b)$.

enables the search for very complex solutions that are usually in the form of a mathematical formula—an approach that is commonly known as *symbolic regression*. Later paradigms extended this concept to calculate any boolean or programming expression. Consequently, complex intelligent structures, such as *fuzzy rule-based systems* or *decision trees* have already been used as the desirable intention in genetic programming approaches [1,15,31–33]. The main qualification of this solving procedure is that the feature selection, and the system configuration, derive in the searching process and do not require any human involvement. Moreover, genetic programming, by inheriting the genetic algorithms' stochastic search properties, does not use *local search*—rather uses the *hyperplane search*—and so avoids driving the solution to any local minimum. The potential gain of an automated feature selection and system configuration is obvious; no prior knowledge is required and, furthermore, not any human expertise is needed to construct an intelligent system. Nevertheless, the task of implementing complex intelligent structures into genetic programming functional sets in not rather straightforward. The function set that composes an intelligent system retains a specific hierarchy that must be traced in the GP tree permissible structures. This writing offers two advantages. First, the search process avoids candidate solutions that are meaningless or, at least, obscure. Second, the search space is reduced significantly among only valid solutions. Thus, a *genotype*—a point in the search space—corresponds always to a *phenotype*—a point in the solution space. This approach—known as *legal searchspace handling method* [36]—is applied in this work using context-free grammars. As we will discuss in the next paragraph, the implementation of constraints using a grammar can be the most natural way to express a family of allowable architectures. While each intelligent system—such as a fuzzy system—has a functional equivalent—by means of being composed by smaller, elementary functions—what defines and distinguishes this system is its grammar.

2.2. Context-free grammars

Although powerful in its definition, the genetic programming procedure may be proved greedy in computational and time resources. Therefore, when the syntax form of the desired solution is already known, it is useful to restrain the genetic programming from searching solutions with different syntax forms [10,20]. The most advantageous method to implement such restrictions among other approaches [21], is to apply syntax constraints to genetic programming trees, usually with the help of a context-free grammar declared in the *Backus-Naur-Form* (BNF) [9,14,22,29]. The BNF-grammar consists of *terminal* nodes and *non-terminal* nodes and is represented by the set $\{N, T, P, S\}$ where N is the set of non-terminals, T is the set of terminals, P is the set of production rules and S is a member of N corresponding to the starting symbol. The use of the terms *terminal* and *non-terminal* in a BNF-grammar, does not correspond to what Koza defines as terminal and function. Rather, a function—a non-terminal node in terms of the GP tree architecture—is expressed as terminal in a BNF grammar. To avoid confusion, the use of the terms *GPFunction* and *GPTerminal*—instead of the ambiguous terms function and terminal—has been proposed [34] and is adapted throughout this paper. The construction of the production rules can be the most critical point in the creation of a BNF grammar, since these production rules express the permissible structures of an individual. An example grammar expressing a class of individuals, which can produce the program in Fig. 2, is composed by the following sets:

$$N = \{EXPR, OP\}$$

$$T = \{-, *, /, a, b, 7, 8\}$$

$$S = \langle EXPR \rangle$$

Then, P is expressed as shown in Table 1.

2.3. Cellular encoding

Although mapping decision trees or fuzzy rule-based systems to specific grammars can be relatively easy to implement, the execution of massively

Table 1
Grammar used for a simple example tree

Symbol	Rule
⟨EXPR⟩	::=⟨EXPR⟩ ⟨OP⟩ ⟨EXPR⟩|⟨VAR⟩|⟨NUMBER⟩
⟨OP⟩	::=- |*|/
⟨VAR⟩	::=a|b
⟨NUMBER⟩	::=7|8

parallel processing intelligent systems—such as the neural networks—is not forthright. In order to explore variable sized solutions, usually a kind of *indirect encoding* is applied. The most common one is the *cellular encoding* [8], in which a genotype can be realized as a descriptive phenotype for the desired solution. More specifically, within such a function set, there are elementary functions that modify the system architecture together with functions that calculate tuning variables. Current implementations include encoding for feedforward and Kohonen neural networks [8,12] and fuzzy Petri-nets [35]. In his original work, Gruau also used a context-free grammar—a BNF grammar—to encode indirectly the neural networks. On the other hand, in [35] a logic grammar—a context-sensitive one—is adapted to encode fuzzy Petri-nets. In our work, we show that as long as the depth-first execution of the program nodes of a GP tree is ensured—which is the default—a context-free grammar such as a BNF grammar is adequate for expressing neural networks. Gruau's original work has been facing some skepticism [11] on the ability to express arbitrarily connected networks. Later developments [10] seem to offer less restrictive grammar, though the *cut* function[1] still maintained bounded effect. In our approach, we inherit present grammar advances proposed in [35] in his logic grammar for fuzzy Petri-nets and we suggest a BNF grammar for neural networks that is more descriptive than previous works.

3. The data sets

The models are tested in three different setups of six data sets from the medical and biomedical domain. These data sets have been taken unmodified by a collection of real-world benchmark problems, the *Proben1* [25] that has been established for neural networks. The original data are derived by the *UCI Machine Learning Repository* [2]. In the Proben1 data set, modifications were applied for processing with NN and better comparability results. Specifically, first the values of every data attribute were standardized in the range $[-1, 1]$. Then, nominal values were substituted using binary encoding, incrementing this way the number of inputs or outputs (see Table 2). Missing values were substituted by standard—zero—values. Three different sequences of the examples were created. The outputs were represented using *1-of-n encoding* where n outputs were used to express n classes. This encoding facilitated the *winner-takes-all* approach that is also followed in our experiments. Table 2 shows the problem complexity of these data sets. The first problem to be addressed is the Wisconsin Breast Cancer data.

[1] An analytical explanation of the *cut* function is given in Section 4.

Table 2
Problem description

Problem	Attributes	Inputs		Classes	Records used
		Continuous	Discrete		
Cancer	9	9	0	2	696
Diabetes	8	8	0	2	764
Gene	60	0	120	3	3172
Heart	13	6	29	2	916
Horse	20	14	44	3	360
Thyroid	21	6	15	3	7196

The goal is to diagnose between benign and malignant breast tumor. The second problem is the Pima Indians Diabetes data, where we diagnose between positive or negative diabetes. The next problem comes from the bioscience field and is the classification of DNA sequences in genes between three classes: intron–exon, exon–intron or no boundary. The problem of the Heart disease follows, where the goal is to diagnose if the diameter of a heart vessel is reduced by more than 50% or not. The next problem is a veterinarian one. The aim is to diagnose if a horse with colic will die, survive or must undergo euthanasia. The last problem is to diagnose between thyroid hyperfunction, hypofunction or normal function. The overview of our testing approach is shown in Fig. 3.

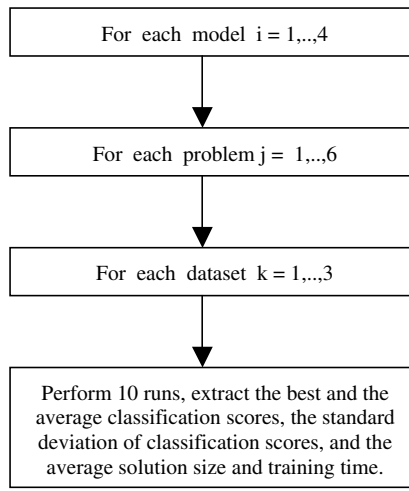

Fig. 3. Overview of the main procedure loop, which was followed for the execution of the experiments.

4. Design and implementation

Each data set was separated into a training set, a validation set and a test set. The training set consists of 50% of the data and the rest 50% is divided equally between the validation set and the test set. The separation of the examples into training, validation and test sets was performed in a loop manner. Specifically, the first two examples were assigned to the training set, then the next to the validation set and the fourth to the test set. This process was repeated until all the examples were assigned a set.

During the training phase, the validation set is typically used to avoid overfitting. A solution that has better classification score in the training set, is adapted as new best solution if and only if the *sum* of classification scores of both training and validation sets is the same or better than the best solution's respective score. We performed 20 runs for each data set. In all experiments, we used the same GP parameters.

It is accepted that the G^3P procedure may suffer size problems during initialization [27]. Although the fine-tuning of our algorithm was not the main concern of this paper, we investigated various initialization approaches. Without claiming optimality, the GP parameters are presented in Table 3.

This setup, together with function selection probability optimization, offered for the presented grammars stable and effective runs throughout experiments. As it can be observed, this setup denotes our preference for significantly high mutation rates, especially *shrink mutation* [30] that slows down the *code bloat* caused by crossover operations. The optimization of function selection probabilities is consisted of giving more selection probability to GPTerminals rather than GPFunctions. Although the initialization of the population is random, using this probability bias the algorithm is 'forced' to generate individuals of acceptable size. This optimization was decided after experimentation for each

Table 3
GP parameters for G^3P

Parameter	Value
Population	2000 individuals
GP implementation	Steady-state G^3P
Selection	Tournament with elitist strategy
Tournament size	6
Crossover rate	0.35
Overall mutation rate	0.65
Node mutation rate	0.4
Shrink mutation rate	0.6
Killing anti-tournament size	2
Maximum allowed individualsize	650 nodes
Maximum number of generations	100

of the four implementations, since it was not possible to obtain a general principle regarding the most proper probability values.

4.1. G^3P for decision trees

The classification procedures can be divided in two types, concerning the number of categories that are classified. The first classification type separates the data between only two classes (known as binary classification or two-class task), and the second type classifies the data among more than two classes (multi-class task). Multi-class tasks can be divided to more than one single-class tasks. One such approach is to build independent classification rules for each of the classes and then run these—competitive—rules simultaneously [3]. Another approach is addressed in [5], where the construction of two-class (or two class-sets) rules is performed in a hierarchical way, creating a cooperative crisp rule-base—rather a competitive one—which always results in one class.

The other way of handling multi-class tasks, in order to build a rule-base, is by using directly a multi-class approach, often with the incorporation of a niching scheme [7]. Here a multi-class approach is presented, which constructs crisp cooperative and hierarchical classification rule-trees, similar to Quinlan's inductive decision trees [26]. The popularity of these decision trees may be explained by the natural decision method that humans often follow. The latter conclusion is demonstrated for example, in the medical field, where physicians usually follow a form of a complex decision/classification tree [6], showing that, medical decision making often is similar to Quinlan's approach. Although fast and robust, Quinlan's model is however restricted in terms of each rule's (tree branch) premise set, where the expression evaluated is an inequality between an attribute (input variable) and a value (number). Apparently, a more generic methodology could involve, in the rules' premise sets, the incorporation of more complex comparisons, such as combinations of expressions including more than one attributes and values. In Table 4, we present the production rules for the implementation of such enhanced decision trees into the genetic programming architecture.

As seen by the grammar definition, a ⟨clause⟩ node may be either a ⟨class⟩ node or a ⟨if⟩ (⟨*if_less*⟩, ⟨if_equal⟩ etc.) node. If we examine more carefully this rule of the grammar, we may notice that this design enables the existence of one-node solutions, for example *CLASS3*. Although this case is normal (according to the grammar definition), it was proved in our experiments that it could delay the solution search. The reason is that this grammar may create many single-class individuals (as one such individual may have good fitness in the beginning of the run) and thus reduce the population diversity very early, leading to local optima. To smooth this phenomenon, we selected to apply two measures. First, we tuned the selection probability for operations, between the

Table 4
Production rules for decision trees

Symbol	Rule
⟨CLAUSE⟩	::=⟨CLASS⟩|⟨IF_LESS⟩| ⟨IF_EQUAL⟩|⟨IF_GREATER⟩
⟨IF_LESS⟩	::=**IF_LESS**⟨EXPR⟩⟨EXPR⟩⟨CLAUSE⟩ ⟨CLAUSE⟩
⟨IF_EQUAL⟩	::=**IF**_EQUAL⟨EXPR⟩⟨EXPR⟩⟨CLAUSE⟩ ⟨CLAUSE⟩
⟨IF_GREATER⟩	::=**IF_GREATER**⟨EXPR⟩⟨EXPR⟩⟨CLAUSE ⟩⟨CLAUSE⟩
⟨EXPR⟩	::=⟨NUMB⟩| ⟨OPER⟩
⟨OPER⟩	::=⟨PLUS⟩| ⟨MINUS⟩|...
⟨PLUS⟩	::=**+**⟨EXPR⟩⟨EXPR⟩
⟨NUMB⟩	::=float in [−1, 1]
⟨CLASS⟩	::=**CLASS1**|**CLASS2**|**CLASS3** |...

⟨clause⟩ representatives (i.e. we gave more selection probability to ⟨if⟩ rather than to ⟨class⟩ nodes). Second, we modified the fitness measure as shown in the following equations, in order not to promote very small-sized solutions. This modified fitness measure performs a penalty to small-sized solutions adapting the size of the solution examined. Although these measures do not guarantee that the run will not lose its diversity early, it was shown in our experiments that the algorithm performed statistically better (i.e. we did not encounter early convergence to local optima). This fitness measure is given by the equations:

$$F = a\phi \tag{1}$$

$$\phi = \sum_{t=0}^{n-1}(1 : f_t = Y_t | 0 : f_t \neq Y_t) \tag{2}$$

where F is the program fitness, t is a record in the training set, n is the number of training records, f_t is the program output for the record t, and Y_t is the value of the record t. The a factor, is given by the following equation:

$$a = \left(1 : S < 0.95 | 1 - \frac{2(S - 0.5) - 0.94}{0.05} : S \geqslant 0.95\right) \tag{3}$$

where S is the simplicity factor. This factor is presented in [3] and its value is:

$$S = -\frac{M - 0.5N - 0.5}{M - 1} \tag{4}$$

where M stands for the maximum size of trees allowed in our application (in nodes), N stands for the examined solution's size (in nodes). This value ranges from 0.5 to 1, producing 0.5 when the expression has the maximum size and 1 when the expression has only one node (the simpler case). This fitness measure will reduce the actual fitness value, when the simplicity's value of the solution is

greater than 0.95, in a linear manner, finally producing zero fitness value when the simplicity's value is one.

4.2. G^3P for fuzzy rule-based systems

A fuzzy if-then rule [13], can be in the form:

$$if\ x\ is\ A\ then\ y\ is\ B\ with\ C, \quad C \in [0,1]$$

where the "x is A" is the antecedent (or premise) set, "y is B" is the consequent (or conclusion) set, and C is the *certainty factor*. In fuzzy reasoning, the traditional two-valued logic, the *modus ponens*, is used in a generalized form. Namely, a fact may be more or less true, based on the truth of another fact. A fuzzy set is defined as:

$$A = \{(x, \mu_A(x))|x \in X\}$$

where, the $\mu_A(x)$ is a *membership function* for the fuzzy set. Fuzzy sets are seen an extension to the classic sets that have a crisp boundary, where the transition for a value from belonging to a set and not belonging to the set is gradual and characterized be the membership function. The membership functions are described as a mathematical formula. The X is called the *universe of discourse*, and it may be comprised by discrete or continuous values. When the universe of discourse X is a continuous space, several fuzzy sets are used, most times covering the X uniformly. These fuzzy sets often make use of linguistic terms such as "Small" or "Medium". Commonly, they are used in fuzzy rules, which may also be interpreted as fuzzy relations using *fuzzy reasoning*. Fuzzy reasoning contains inference rules that derive conclusions from a set of fuzzy rules and input data. In the Mamdani classifier model using the max–min composition, several steps are followed to perform fuzzy reasoning. Firstly, we compare the input data with the antecedent sets of the fuzzy rules and we get the degrees of compatibility (called *weights*) with respect to these antecedent sets. Then, we combine these degrees using fuzzy AND or OR to obtain a *firing strength*, which shows the degree that a rule is satisfied. The firing strength corresponds to the certainty factor presented above. The max–min criterion, when only AND operators are used, will assign as firing strength the smaller of the antecedent degrees of compatibility. Finally, we obtain the overall output between the consequent sets of the rules. When the max–min composition is used, the rule with the larger firing strength will be the system's output. The definition of the grammar we used is shown in Table 5 [1].

This grammar describes a fuzzy system model with four inputs and one output. The GPFunctions used to describe the fuzzy mechanism, correspond to the words with bold in Table 5. We suggested the working shown in Table 6, in order to simulate a Mamdani classifier.

Table 5
Production rules for fuzzy rule-based systems

Symbol	Rule
⟨TREE⟩	::=⟨RL⟩| ⟨RULE⟩
⟨RL⟩	::=**RL** ⟨TREE⟩⟨TREE⟩
⟨RULE⟩	::=**RULE** ⟨COND⟩⟨CLASS⟩
⟨COND⟩	::=⟨IF⟩| ⟨AND⟩
⟨IF⟩	::=**IF** ⟨INP⟩⟨FS⟩
⟨AND⟩	::=**AND** ⟨COND⟩⟨COND⟩
⟨CLASS⟩	::=**THEN** ⟨OUT⟩⟨CLASS_VALUE⟩
⟨FS⟩	::=**SMALL**|**MEDIUM**|**LARGE**
⟨INP⟩	::=**X1**|**X2**|**X3**|**X4**
⟨CLASS_VALUE⟩	::=**CLASS1**|**CLASS2**| **CLASS3**|...
⟨OUT⟩	::=**Y**

Table 6
Functions for the simulation of a Mamdani-model classifier

Function	Pseudo-code
RL(arg1, arg2)	If absolute(arg1) > absolute(arg2) then return arg1; else return arg2
RULE(arg1, arg2)	Return arg1 * arg2
IF(arg1, arg2)	Fuzzify (arg1), based on the (arg2) value, return weight
AND(arg1, arg2)	Return minimum(arg1, arg2)
THEN(arg1, arg2)	If arg1 = arg2 then return 1; else return −1
SMALL, MEDIUM, LARGE, etc.	Return a constant value (e.g. −1 for SMALL, 0 for MEDIUM, 1 for LARGE, etc.).
CLASS1, CLASS2, etc.	Return a constant value (e.g. 1 for CLASS1, 2 for CLASS2, etc.)
X1, X2, etc.	System inputs (assuming a numerical value)
Y	System output (assuming a numerical value)

The fuzzification is applied in *IF* nodes. The implementation uses Gaussian membership functions, and for a given Gaussian range a (standard for the *IF* nodes), a center $c = arg2$ and a value $x = arg1$, the function output will be the following:

$$n = \mathrm{e}^{-\frac{1}{2}\left(\frac{x-c}{a}\right)^2} \tag{5}$$

In order to offer more degrees of freedom, we selected to use nine (9) membership functions, which are presented in Fig. 4. The *THEN* node returns 1 if for the examining example the output (*arg1*), belongs to the class described by *arg2* and −1 otherwise. The reason to use this mechanism, together with the RL working, is to be able to know (when the tree evaluation is complete) whether the rule that fired was true or false. If the fired rule describes a false consequent set, the program value will be negative. While an individual represents a

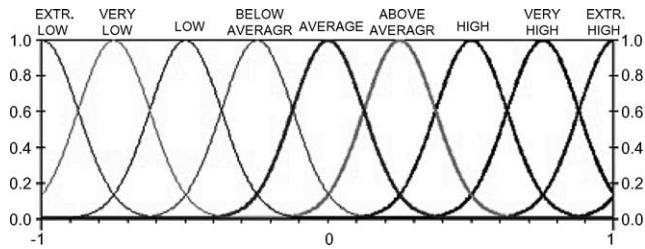

Fig. 4. Membership functions of the fuzzy rule-based system.

complete rule base, when examining an example during the training phase, this procedure will produce either positive or negative values indicating correct or wrong classification.

4.3. G^3P for artificial neural networks

In order to incorporate the architecture of feedforward neural networks into genetic programming, one has to decide whether he will use direct or indirect encoding. Direct encoding, although it describes fixed sized neural networks effectively, in most problem cases is inefficient, since the a priori knowledge of the neural network's best architecture is not available. Thus, it seems normal to prefer a learning process that uses variable sized neural networks such as cellular encoding, a variation of indirect encoding. The idea behind cellular encoding is that each individual program in the population is a specification for developing a neural network. As it can be seen from Fig. 5, to implement neural networks, we consider three types of places. First, *input places*, such as N_1 and N_2, associated with the system input values during run. Second, *intermediate places*, such as N_3 and N_4, and third, *output places*, such as N_5, that represents the system outputs.

The structure of a neural network for a particular problem must be restricted, in terms that if the problem requires a binary classification, then the network will have only two output places. The final output can follow

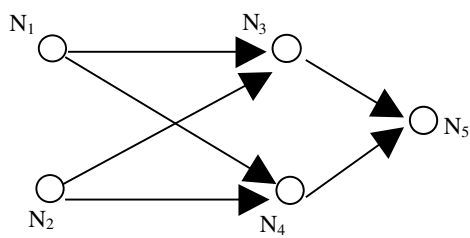

Fig. 5. A simple feedforward neural network.

Table 7
NN manipulating functions

	Name	Description	Number of arguments
Input place	sp1	Sequential division	3
	pp1	Parallel division	2
	in	Initialize the value	1
Intermediate place	sp2	Sequential division	3
	pp2	Parallel division	3
	stop	Terminate the modification	1
	lnk	Modify synapse (link)	4
	act	Activate	0

the *winner-takes-all* concept—the output with the larger value is assumed as the system's output. Table 7 presents the manipulating functions. Manipulating functions initialize the inputs and/or insert additional places into the developing network. These functions are classified into two types, since there are two types of modifiable places, the input places and the intermediate places. The implementation of a parallel processing system—such as neural networks—in a tree like the GP-tree that is executed *depth-first*, requires special handling of variables in order to emulate a *breadth-first* execution and simulate the parallel processing. Thus we selected to apply to all functions a parameter *passing by reference* for two variables: a parameter array Q and a parameter value V.

The array Q keeps synapse values and the value V handles activation results. In our implementation the parallel execution of each individual is ensured, by the proper handling of these variables. In order to avoid confusion, we should define explicitly the meaning of the *arguments* shown in Table 7. These arguments correspond to the permissible *sub-nodes* a parent *GP node* may have in a *GP tree*. On the other hand, the array Q and the variable V, correspond to the *internal implementation* of the neural network's elementary functions into the GP tree and they enable *variable sharing* throughout an individual's execution, that is needed to simulate the parallel processing. In the following paragraphs, a short presentation is given for each of the functions together with an explanation of the implementation. Emphasis is given mostly to implementation issues since the definition of modifying functions has been thoroughly described in existing literature [8,35].

Each function calls its arguments, passing the array Q and the variable V by reference, unless otherwise stated. For example, function *pp2* creates copies of the (input) array Q before passing them to its arguments.

The *sp1* function takes three arguments. The first and the third argument can be *pp1* or *sp1* (or *in*) functions. The second argument is a *weight value*. It calls sequentially the three arguments. Its application to the *developing* neural network is to add sequentially a node next to the node that is applied.

The *pp1* function has two arguments. They can be *pp1* or *sp1* (or *in*) functions. It feeds the arguments with copies of the array Q. It then saves the concatenation of them to array Q. It does not affect explicitly the variable V. Its modification to the developing neural network is to create a node in parallel to the node that is applied.

The *in* function has one argument. This argument is one of the network inputs. It initializes the array Q and the variable V to this value.

The *sp2* function has three arguments. The first and the third argument can be *pp2* or *sp2* (or *stop*) functions. The second argument is a *weight number*. It calls sequentially the three arguments. Its application to the developing neural network is to add sequentially a node next to the node that is applied similar to *sp1*.

The *pp2* function takes three arguments. The first and the third argument can be *pp2* or *sp2* (or *stop*) functions. The second argument can be a *lnk* or an *act* function. It feeds the three arguments with copies of the array Q. It then saves the concatenation of the first and the third argument to array Q. It does not affect explicitly the variable V. Its modification to the developing neural network is to create a node in parallel to the node that is applied, similar to *pp1*.

The *stop* function takes one argument. This argument is a *bias number*. It adds the bias number to the value V and performs hyperbolic tangent activation. The result is saved to variable V. The array Q is also initialized with the value V.

The *lnk* function takes four arguments. The first argument is a number that calculates the number of the synapse to be processed. The number derives by the application of the formula $Z \bmod N$, where Z is the number of existing synapses (kept in array Q) and N is the value of the argument. The second argument is a *weight number* and updates the weight of the selected synapse. The third argument is a *cut number* and cuts the selected synapse if and only if the cut value is 1 and the number of inputs to this place is greater than 1. The fourth argument can be again a *lnk* function or an *act* function.

The *act* function summarizes the elements (synapse inputs) in the array Q and returns the result to V. It takes no arguments.

As seen from the previous, there are additional functions that assist the selection of a number into a given range and a given precision. These functions are presented below.

The *weight number* function returns a float in the range $[-1, 1]$ with a precision of 0.00390625. The *bias number* function returns a float in the range $[-1, 1]$ with a precision of 0.000244140625. The *cut number* function returns an integer from the set $\{0, 1\}$. The *number* function returns an integer in the range $[1, 256]$. The selection of the precision for the *weight number* and the *bias number* is similar to the existing literature [10]. The BNF grammar production rules used for implementing the neural network are summarized in Table 8. This example grammar corresponds to a binary decision neural network—with two indepen-

Table 8
Production rules for artificial neural networks

Symbol	Rule
⟨CLAUSE⟩	**::=**⟨ANN⟩
⟨ANN⟩	**::=**⟨PROG⟩⟨ PROG⟩
⟨PROG⟩	**::=**⟨PLACE1⟩⟨WT⟩
⟨PLACE1⟩	**::=SP1**⟨PLACE1⟩⟨WT⟩⟨PLACE2⟩ |**PP1**⟨PLACE1⟩⟨PLACE1⟩ |⟨IN⟩
⟨IN⟩	**::=**⟨ATTR⟩
⟨WT⟩	**::=**float in $[-1, 1]$
⟨PLACE2⟩	::=**SP2**⟨PLACE2⟩⟨WT⟩⟨PLACE2⟩ |**PP2**⟨PLACE2⟩⟨SYNAPSE⟩⟨PLACE2⟩ |⟨STOP⟩
⟨STOP⟩	::=⟨BIAS⟩
⟨SYNAPSE⟩	::=⟨LNK⟩|⟨ACT⟩
⟨BIAS⟩	::=float in $[-1, 1]$
⟨LNK⟩	::=**LNK**⟨NUM⟩⟨WT⟩⟨CUT⟩⟨SYN APSE⟩
⟨ACT⟩	::=**ACT**
⟨NUM⟩	::=integer in $[1, 256]$
⟨CUT⟩	::=integer in $[0, 1]$
⟨ATTR⟩	::=data attribute (system input)

dent outputs. A tree starts with the ⟨CLAUSE⟩ symbol. As it can be seen from the design of this grammar, there is no limit on how many *cut* functions could be applied in a node. This is an upgrade from previous implementations—where the number of effective *cut* functions was limited—and enables to a larger set of neural networks to be expressed within the search process.

4.4. G^3P for fuzzy Petri-nets

A fuzzy Petri-net [19] can be seen as a network that is constructed by input places, transitions and output places. The topology of a simple fuzzy Petri-net is depicted in Fig. 6. The framework of a fuzzy Petri-net is finely correlated with the classification process of any pattern recognition task [24]. The input places are associated with the values of the features. These values are processed by the transitions of the network.

The levels of firing of the network, depend on the parameters that are associated with each transition. Consequently, an output place corresponds to a class. As with the case of NN in the previous paragraph, in order to handle a number of classes larger than two, this system adopts the *winner-takes-all* approach. Formally, a fuzzy Petri-net is described by the following 8-tuple:

$$FPN = (\boldsymbol{P}, \boldsymbol{T}, \boldsymbol{D}, \boldsymbol{I}, \boldsymbol{O}, cf, a, \beta)$$

where $\boldsymbol{P} = \{\boldsymbol{P}_1, \boldsymbol{P}_2, \ldots, \boldsymbol{P}_n\}$ is a finite set of places,

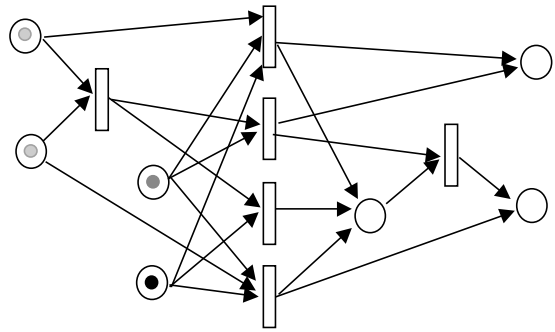

Fig. 6. A fuzzy Petri-net.

$\boldsymbol{T} = \{T_1, T_2, \ldots, T_n\}$ is a finite set of transitions,
$\boldsymbol{D} = \{D_1, D_2, \ldots, D_n\}$ is a finite set of propositions,
$\boldsymbol{P} \cap \boldsymbol{T} \cap \boldsymbol{D} = \emptyset$,
$\boldsymbol{I} : \boldsymbol{T} \to \boldsymbol{P}^{\infty}$ is the input function, a mapping from transitions to bags of places,
$\boldsymbol{O} : \boldsymbol{T} \to \boldsymbol{P}^{\infty}$ is the output function, a mapping from transitions to bags of places,
$cf : \boldsymbol{T} \to [0, 1]$ is an association function, a mapping from transitions to real values in the range $[0, 1]$,
$\boldsymbol{a} : \boldsymbol{P} \to [0, 1]$ is an association function, a mapping from places to real values in the range $[0, 1]$,
$\beta : \boldsymbol{P} \to \boldsymbol{D}$ is an association function, a mapping from places to propositions.

The incorporation of fuzzy Petri-nets into G^3P was originally presented by [35]. That implementation made possible the description of fuzzy Petri-nets of arbitrary size and topology. Unlike in [35], where a context-sensitive grammar is used to guide the genetic process, we present a context-free version of that grammar. The context-free equivalent, which applied in this work, satisfies the same descriptive rules. According to the methodology described in [35], manipulating functions are used to insert additional places and transitions. Since there are two types of modifiable places, the input and intermediate ones, we classify the place-manipulating functions into two types. These functions are presented in Table 9. Moreover, we have manipulating functions that are used on transitions. These transition-manipulating functions are shown in Table 10.

Two variables were used to assist the simulation of the *breadth-first* execution, as with the model of the previous paragraph: a parameter array Q and a parameter value V. The working of these functions can be summarized as follows:

Table 9
FPN place-manipulating functions

	Name	Description	Number of arguments
Input place	sp1	Sequential division	3
	pp1	Parallel division	2
	in	Initialize the value	2
Intermediate place	sp2	Sequential division	3
	pp2	Parallel division	3
	stop	Terminate the modification	0

Table 10
FPN transition-manipulating functions

Name	Description	Number of arguments
st	Sequential division	3
pt	Parallel division	2
cut	Remove one of the incoming edges	2
setcf	Set the certainty factor	1

The *sp1* function takes three arguments. The first and the third argument can be *pp1* or *sp1* (or *in*) functions. The second argument is a transition manipulating function such as *pt*, *st* or *cut*. It calls sequentially the three arguments. Its application to the developing fuzzy Petri-net is to add sequentially a new input place next to the place that is applied.

The *pp1* function has two arguments. They can be *pp1* or *sp1* (or *in*) functions. It feeds the arguments with copies of the array Q. It then saves the concatenation of them to array Q. It does not affect explicitly the variable V. Its modification to the developing fuzzy Petri-net is to create an input place in parallel to the place that is applied.

The *in* function has one argument. This argument is one of the FPN inputs. It fuzzifies the input and initializes the array Q and the variable V to this value. We used five (5) Gaussian membership functions, which are shown in Fig. 7.

The *sp2* function has three arguments. The first and the third argument can be *pp2* or *sp2* (or *stop*) functions. The second argument is a transition manipulating function (*pt*, *st*, etc.). It calls sequentially the three arguments. Its application to the developing FPN is to add sequentially a place next to the place that is applied (in a similar fashion of the *sp1*).

The *pp2* function takes three arguments. The first and the third argument can be *pp2* or *sp2* (or *stop*) functions. The second argument is a transition manipulating function. Initially, it feeds the three arguments with copies of the array Q.

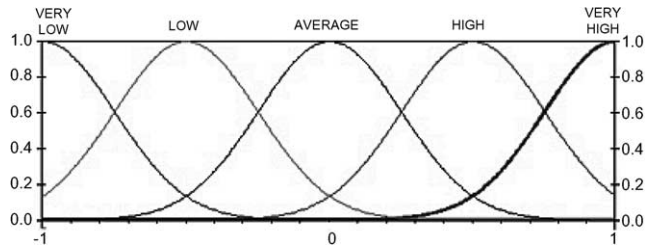

Fig. 7. Membership functions of the fuzzy Petri-net.

Table 11
Production rules for fuzzy Petri-nets

Symbol	Rule
⟨CLAUSE⟩	::=⟨FPN⟩
⟨FPN⟩	::=⟨PROG⟩⟨PROG⟩
⟨PROG⟩	::=⟨PLACE1⟩⟨TRAN⟩
⟨PLACE1⟩	::=**SP1**⟨PLACE1⟩⟨TRAN⟩⟨PLACE2⟩
	|**PP1**⟨PLACE1⟩⟨PLACE1⟩
	|⟨INIT⟩
⟨INIT⟩	::=⟨ATTR⟩
⟨PLACE2⟩	::=**SP2**⟨PLACE2⟩⟨TRAN⟩⟨PLACE2⟩
	|**PP2**⟨PLACE2⟩⟨TRAN⟩⟨PLACE2⟩
⟨TRAN⟩	::=**PT**⟨TRAN⟩⟨TRAN⟩
	|**ST**⟨TRAN⟩⟨PLACE2⟩⟨TRAN⟩
	|**CUT**⟨NUMBER⟩⟨TRAN⟩
	|**SETCF**⟨CF⟩
⟨NUMBER⟩	::=integer in [1, 256]
⟨CF⟩	::=real in [0, 1]
⟨ATTR⟩	::=data attribute (system input)

Then, it saves the concatenation of the first and the third argument to the array Q. Its modification to the developing FPN is to create a place in parallel to the place that is applied, similar to *pp1*. It does not affect explicitly the variable V.

The *stop* function has no arguments. It signals the end of further modification. The array Q is initialized with the value V.

The *st* function has three arguments. The first and the third argument are transition manipulating functions. The second argument is a place manipulating function. It calls sequentially the three arguments. Its application to the developing neural network is to add sequentially a transition next to the transition that is applied.

The *pt* function has two arguments. They can be transition manipulating functions. It feeds the arguments with copies of the array Q. Its modification

to the developing fuzzy Petri-net is to create an transition in parallel to the place that is applied.

The *cut* function has two arguments. The first argument is an integer. The second one is a transition manipulating function. It cuts the connection that corresponds to the (*I mod N)* value—where *I* is the number of inputs to this place and *N* is the integer—*iff* the number of inputs to this place is greater than 1. It passes down to the second argument the parameters *Q* and *V*.

The *setcf* function takes one argument. This argument is a *certainty factor* value. It performs the transition of the input value. The result is saved to variable *V*. The array *Q* is also initialized with the value *V*.

The BNF grammar is shown in Table 11. The starting symbol is the ⟨CLAUSE⟩ symbol. This grammar corresponds to a binary decision fuzzy Petri-net.

5. Results and comparison

5.1. Generalization performance

In Table 12 we present the training classification error (CE) for each model and data set. The presentation of these results is useful since, as stated previously, during the training phase the best solution is the one that has the lower sum of the classification errors in both training and validation set. The validation and test CEs are shown in Tables 13 and 14. As we may observe, the lower CE belongs on average to the G^3P for Decision Trees (DT) approach, whereas the higher to the Neural Networks model. However, we consider that this outcome should not lead to generalization on the general G^3P for NN model capabilities, since our results concern a given GP parameter setup.

This setup seems to be restrictive for the G^3P for NN model. More specifically, as it was expected, a high *correlation* between the size of the produced NN and the training classification result was noticed. For example, this value for the diabetes2 data set was 0.8483. Moreover, in order to ensure that these results are not related to the population size or the effective training time, we examined a larger model in Section 5.3 that follows. The fuzzy Petri-nets approach is the next to the NN model that suffers from relatively high CE. We consider that a model with a larger population size should be examined here too.

As it can be seen from Table 15, the average best solution size for fuzzy Petri-nets, in our configuration, varies in the middle between the size of NNs and the FRBSs. The G^3P for FRBS is the only methodology that competes in this configuration the G^3P for DT. Its success over the DT comes in the diabetes and the cancer data. This result seems to be very interesting since these data sets are the only sets that do not have discrete attributes (see Table 2). In

Table 12
Training classification error rates of G^3P in Proben1 data sets

Problem	G^3P for decision trees			G^3P for fuzzy rule-based systems			G^3P for artificial neural networks			G^3P for fuzzy Petri-nets		
	Best	Avg.	Stddev	Best	Avg.	Stddev	Best	Avg.	Stddev	Best	Avg.	Stddev
Cancer1	0.57	2.14	0.83	1.14	2.08	0.55	2.29	6.96	2.37	3.16	3.44	0.25
Cancer2	0.86	2.87	1.72	0.57	1.59	0.48	1.72	6.14	1.82	1.43	3.16	1.54
Cancer3	0.28	1.22	0.46	0.28	1.01	0.46	1.43	5.54	2.51	1.14	2.61	1.62
Diabetes1	15.70	20.98	2.97	15.70	17.25	1.11	24.08	24.74	0.41	20.41	23.62	1.96
Diabetes2	18.84	21.74	1.48	17.01	19.89	2.08	23.29	24.76	0.59	21.46	23.82	1.84
Diabetes3	18.06	22.82	3.17	16.49	19.80	2.23	22.51	23.32	0.49	21.72	23.76	2.03
Gene1	20.87	31.68	6.99	10.21	36.03	15.18	29.76	32.75	4.23	12.23	30.84	14.78
Gene2	25.59	34.16	6.13	13.80	35.42	12.17	28.49	34.16	5.20	12.35	24.05	16.54
Gene3	25.09	30.83	6.11	13.80	32.53	12.15	28.56	39.86	4.79	12.23	29.63	14.39
Heart1	14.41	16.26	1.13	14.62	16.24	1.52	21.83	21.83	0.00	19.43	20.18	0.99
Heart2	13.75	15.91	1.38	15.28	17.14	2.04	24.89	25.87	0.33	20.74	20.89	0.13
Heart3	11.79	13.70	1.36	15.06	15.65	0.77	23.58	23.94	0.25	21.17	21.25	0.16
Horse1	21.11	25.50	3.65	18.33	20.92	2.24	24.44	34.86	3.17	16.67	22.03	5.28
Horse2	16.11	22.41	4.58	15.55	20.13	3.96	16.66	26.11	3.02	19.44	22.96	3.94
Horse3	26.67	31.25	3.55	16.67	22.77	4.67	26.11	32.30	2.07	17.77	22.40	4.72
Thyroid1	2.83	5.77	0.85	5.33	6.66	1.09	5.78	5.98	0.29	6.19	7.19	0.86
Thyroid2	2.30	5.57	1.37	5.58	6.37	1.03	5.75	6.46	0.85	5.78	6.71	0.91
Thyroid3	1.25	2.22	5.35	5.80	6.75	1.33	5.78	6.27	0.81	5.78	6.41	0.74

Table 13
Validation and test classification error rates of G^3P for decision trees and fuzzy rule-based systems in Proben1 data sets

Problem	G^3P for decision trees						G^3P for fuzzy rule-based systems					
	Validation CE (%)			Test CE (%)			Validation CE (%)			Test CE (%)		
	Best	Avg.	Stddev	Best	Avg.	Stddev	Best	Avg.	Stddev	Best	Avg.	Stddev
Cancer1	2.29	3.76	1.29	2.29	3.79	1.01	1.72	2.87	0.83	2.29	4.39	1.42
Cancer2	2.87	5.05	1.58	1.72	4.68	2.18	2.87	4.48	0.84	1.72	4.45	1.23
Cancer3	3.44	6.23	1.88	2.29	4.39	1.36	4.02	5.08	0.61	3.44	4.90	0.83
Diabetes1	25.65	28.79	2.25	26.70	31.70	3.24	26.17	28.75	1.44	21.98	26. 47	3.40
Diabetes2	21.46	28.63	4.32	25.65	31.30	3.48	24.08	25.47	0.97	23.56	24. 78	1.22
Diabetes3	24.08	30.23	4.06	19.89	28.79	5.11	23.56	24.95	1.02	21.99	24.25	1.64
Gene1	22.32	32.53	5.93	22.32	33.03	6.70	11.85	36.83	14.99	11.97	37.12	14.99
Gene2	28.87	36.69	4.89	29.63	37.03	4.71	18.15	37.92	11.41	14.37	37.27	11.44
Gene3	26.16	31.49	5.80	26.48	32.21	5.06	12.35	36.60	11.46	11.97	37.04	11.81
Heart1	20.08	23.97	1.38	17.90	21.87	2.12	23.14	25.29	1.25	18.34	22.12	1.93
Heart2	15.72	18.27	1.35	17.90	21.68	1.81	16.59	18.55	1.44	20.96	23.68	1.89
Heart3	16.59	19.67	1.68	22.70	25.76	1.46	17.90	25.76	3.08	25.32	25.76	0.61
Horse1	31.11	36.05	2.84	28.88	33.55	2.78	34.44	40.37	6.78	38.88	41.48	4.49
Horse2	40.00	45.38	2.50	36.67	42.27	3.48	38.88	44.44	3.95	37.77	40.27	4.29
Horse3	35.55	40.67	2.56	28.89	35.94	3.04	33.33	37.22	4.11	27.77	35.28	5.00
Thyroid1	4.00	6.01	0.58	2.89	4.96	0.81	5.55	6.77	0.90	5.28	6.08	0.57
Thyroid2	2.39	4.84	1.00	2.66	5.73	1.25	5.94	6.54	0.82	5.44	5.95	0.52
Thyroid3	2.55	5.10	0.90	1.94	5.45	1.37	6.00	6.80	1.13	5.44	6.03	0.82

Table 14
Validation and test classification error rates of G^{3}P for ANN and fuzzy Petri-nets in Proben1 data sets

Problem	G^{3}P for artificial neural networks						G^{3}P for fuzzy Petri-nets					
	Validation CE (%)			Test CE (%)			Validation CE (%)			Test CE (%)		
	Best	Avg.	Stddev	Best	Avg.	Stddev	Best	Avg.	Stddev	Best	Avg.	Stddev
Cancer1	2.29	5.48	1.37	2.87	5.66	1.24	2.29	3.06	0.69	2.87	4.31	0.94
Cancer2	2.87	6.83	1.55	2.87	8.30	2.16	2.29	5.17	2.20	2.29	4.83	1.19
Cancer3	4.02	7.50	1.69	1.14	5.28	1.70	4.02	6.26	1.49	2.29	4.42	1.43
Diabetes1	24.60	26.11	0.71	22.51	24.54	1.26	24.08	27.53	2.26	23.03	26.82	2.56
Diabetes2	23.56	24.45	0.61	23.03	25.41	1.15	24.60	27.78	2.22	23.03	27.08	2.65
Diabetes3	27.22	29.13	0.94	24.08	28.76	1.84	26.17	28.63	2.19	24.08	28.21	2.16
Gene1	33.41	35.12	2.40	31.77	34.74	4.19	13.36	31.96	15.03	12.73	32.50	14.93
Gene2	30.64	41.26	3.50	31.27	41.48	4.08	18.03	27.42	13.28	20.55	29.12	12.12
Gene3	33.03	38.14	7.54	32.53	37.83	7.37	13.36	32.25	12.73	12.98	31.39	14.44
Heart1	26.20	26.20	0.00	23.14	23.14	0.00	24.89	25.76	0.79	23.14	25.63	1.44
Heart2	21.39	22.22	2.55	19.65	20.34	2.15	21.83	22.12	0.25	21.83	25.32	3.73
Heart3	19.65	22.37	1.69	25.32	28.60	1.59	21.39	22.85	2.15	24.89	25.90	1.76
Horse1	31.11	35.22	2.04	28.88	30.61	2.05	40.00	43.70	3.21	36.67	40.37	3.90
Horse2	43.33	47.22	3.39	38.88	43.50	2.55	37.77	42.96	4.49	35.55	38.88	4.84
Horse3	31.11	41.55	3.61	31.11	36.27	2.53	38.89	43.33	3.84	36.67	41.11	4.44
Thyroid1	6.00	6.14	0.19	5.44	5.50	0.80	6.28	7.17	0.77	5.55	6.26	0.69
Thyroid2	5.94	6.51	0.74	5.56	6.08	0.60	6.03	6.75	0.79	5.44	5.95	0.60
Thyroid3	5.89	6.39	0.69	5.39	5.72	0.50	6.17	6.51	0.61	4.22	5.49	0.84

general, classification competence of fuzzy systems over crisp systems is expected in data sets having continuous features, a principle that is verified in our experiments. On the other hand, data sets with discrete features seem to be better handled by our G^{3}P for DT approach.

5.2. Comparison on solution size and effective training time

In Table 16, we show the average effective generations for the models and the data sets. These results show that in more difficult problems—such as ill-conditioned (e.g. horse data) or large sized (e.g. gene data) problems—the algorithm needed slightly more training time to find the best solution. This is the same result found in [4] concerning the effective time of linearGP.

5.3. Comparison with larger models

In Section 5.1, we observed that the models for artificial neural networks and fuzzy Petri-nets succeeded generally in lower classification rates, than those of the fuzzy rule-based systems or the decision trees. This result, together with the fact that the cellular encoding scheme is used for the two models, led us to

Table 15
Average best solution size of G^3P in Proben1 data sets

Problem	G^3P for decision trees		G^3P for fuzzy rule-based systems		G^3P for artificial neural networks		G^3P for fuzzy Petri-nets	
	Avg.	Stddev	Avg.	Stddev	Avg.	Stddev	Avg.	Stddev
Cancer1	192	147	214	116	47	45	83	80
Cancer2	179	114	193	135	39	54	92	76
Cancer3	198	143	160	118	47	56	92	75
Diabetes1	325	176	274	125	44	44	45	26
Diabetes2	264	131	172	150	30	25	44	25
Diabetes3	217	153	207	208	17	10	51	17
Gene1	18	21	146	202	21	16	37	42
Gene2	13	22	130	197	25	10	44	36
Gene3	20	25	167	226	26	14	42	39
Heart1	233	120	378	139	8	3	45	11
Heart2	271	130	372	168	9	6	89	53
Heart3	315	118	315	119	10	4	64	31
Horse1	158	119	347	74	29	46	81	34
Horse2	91	84	311	74	28	27	84	43
Horse3	102	161	390	77	28	22	77	24
Thyroid1	14	20	165	212	36	27	39	20
Thyroid2	31	45	188	208	44	21	32	17
Thyroid3	37	59	193	178	39	30	34	22

further research on the model size. Thus, we experimented further using larger models for two cases. Specifically, we examined the problems of cancer1 and diabetes1 using a model with double population and running the algorithm for double number of generations. We performed five runs with models for ANN and FPN using the population of 4000 individuals for 200 generations. The results are shown in Table 17.

By comparing the results shown in Table 17, to those of Tables 12 and 14, we remark the following:

Increasing the model size resulted to apparent improvement of classification rate only for the test set of the ANN model in the cancer1 domain, and thus, we conclude that the larger model did not necessarily offered higher classification rates. On the other hand, it was observed, that using a larger model ensured higher *average* classification rates and lower error standard deviation values. The latter conclusion is depicted in all examples of Table 17, where the average error rates and the standard deviations have lower values than those of Tables 12 and 14. This remark implies that by using a larger model for the abovementioned approaches, we may expect better classification rates *on average*, and a sufficient number of runs is still required for the system to be given the opportunity to explore the search space.

Table 16
Effective training time of G^3P in Proben1 data sets (in generations)

Problem	G^3P for decision trees		G^3P for fuzzy rule-based systems		G^3P for artificial neural networks		G^3P for fuzzy Petri-nets	
	Avg.	Stddev	Avg.	Stddev	Avg.	Stddev	Avg.	Stddev
Cancer1	64	34	48	32	55	34	47	15
Cancer2	70	26	43	29	48	25	54	30
Cancer3	58	30	48	27	49	30	50	37
Diabetes1	87	14	71	25	41	33	45	27
Diabetes2	79	21	42	12	52	42	46	29
Diabetes3	72	21	40	31	75	26	52	30
Gene1	87	21	85	28	74	25	63	32
Gene2	86	32	85	23	80	26	55	40
Gene3	80	32	77	29	76	31	58	42
Heart1	74	19	77	24	94	34	30	26
Heart2	88	9	56	40	87	31	24	7
Heart3	82	22	40	23	67	46	30	11
Horse1	78	20	62	20	69	38	64	42
Horse2	74	29	74	23	66	40	60	51
Horse3	49	39	85	11	69	34	55	47
Thyroid1	52	46	84	30	49	41	52	46
Thyroid2	84	29	75	37	55	33	41	31
Thyroid3	84	25	85	21	59	34	34	43

Table 17
Error rates of G^3P for ANN and fuzzy Petri-nets using larger model

Problem	Training CE (%)			Validation CE (%)			Test CE (%)		
	Best	Avg.	Stddev	Best	Avg.	Stddev	Best	Avg.	Stddev
G^3P for artificial neural networks									
Cancer1	2.87	6.32	2.77	3.45	5.48	1.37	2.87	5.66	1.24
Diabetes1	23.82	24.19	0.30	25.13	25.86	0.60	23.56	23.66	0.23
G^3P for fuzzy Petri-nets									
Cancer1	3.16	3.74	0.41	2.30	3.45	0.81	3.45	4.89	1.00
Diabetes1	24.35	25.13	0.91	22.51	23.95	0.99	23.04	23.30	0.30

5.4. Comparison with other methodologies

The following tables include our results, in a comparative way to those included in literature. Although straightforward comparisons are not always efficient, these tables provide a useful way to extract valuable conclusions on the capabilities of the analyzed system.

Table 18
Comparative results for the breast cancer domain

Methodology	Error rate		Methodology	Error rate	
	Training	Test		Training	Test
G^3P-DT	0.0057	0.0210	2-pairs [45]	N/A	0.0650
G^3P-FRBS	0.0066	0.0248	3-pairs [45]	N/A	0.0410
G^3P-ANN	0.0181	0.0229	1-nearest neighbor [44]	N/A	0.0630
G^3P-FPN	0.0191	0.0248	Typical instances [44]	N/A	0.0780
C4.5 [52]	N/A	0.0494	Randomized C4.5 [52]	N/A	0.0353
Bagged C4.5 [52]	N/A	0.0367	Adaboosted C4.5 [52]	N/A	0.0310
Naïve Bayes [53]	N/A	0.0270	ID3 [53]	N/A	0.0480

5.4.1. Breast Cancer problem

For the breast cancer domain [40,41], we present our results together with those found in literature [44,45,52,53]. As it can be seen in Table 18, the methodology of this paper outperforms the approaches in [52,53]. Although the results in [44,45] are referred to a part of the data set encountered here, and thus a direct comparison is not practicable, they are provided in the table for clarity. The G^3P results in this table are comprised of the low classification rate for each approach *averaged* for the three data sets encountered (e.g. cancer1, cancer2 and cancer3 data sets).

5.4.2. Diabetes problem

For the diabetes domain [43], we compared our results to those given by StatLog [42] for a wide range of applications. For this, we considered the obtained lowest classification error *averaged* from the three datasets (diabetes1, diabetes2 and diabetes3). As it may be noticed in Table 19, the models of G^3P for decisions trees and fuzzy rule-based systems have the highest classification score among these methodologies. Moreover, only four approaches have better test classification score than those of the G^3P models of ANN and FPN.

5.4.3. Gene problem

For the gene domain [43], we present our results, together with results found in literature, concerning specific tasks performed using the same data set. More specifically, this data set has been commonly used in three separate tasks, each one being the binary distinction of one class from the other two. Although handling a multi-class classification problem into a single run is considered a significantly harder task, and often a N-class classification problem is decomposed into N two-class classification problems [54], and thus a direct comparison is not applicable, we provide in Table 20 the results from those two-class decomposition problems for clarity. As in previous paragraphs, the value referred for

Table 19
Comparative results for the diabetes domain

Methodology	Error rate		Methodology	Error rate	
	Training	Test		Training	Test
G^3P-DT	0.133	0.195	QuaDisc	0.237	0.262
G^3P-FRBS	0.150	0.215	Bayes	0.239	0.262
G^3P-ANN	0.234	0.227	C4.5	0.131	0.27
G^3P-FPN	0.204	0.233	IndCart	0.079	0.271
LogDisc	0.219	0.223	BayTree	0.008	0.271
Dipol92	0.22	0.224	LVQ	0.101	0.272
Discrim	0.22	0.225	Kohonen	0.134	0.273
Smart	0.177	0.232	Ac2	0	0.276
Radial	0.218	0.243	NewId	0	0.289
Itrule	0.223	0.245	Cn2	0.01	0.289
BackProp	0.198	0.248	Alloc80	0.288	0.301
Cal5	0.232	0.25	KNN	0	0.324
Cart	0.227	0.255	Default	0.35	0.35
Castle	0.26	0.258			

the G^3P generated systems, is the low classification error *averaged* for the three data sets tested (e.g. gene1, gene2 and gene3).

5.4.4. Heart problem

We present in Table 21, the results of the paper, together with those found in literature [55–57]. For this, we considered the obtained lowest classification error *averaged* from the three datasets (heart1, heart2 and heart3). As it can be seen, the proposed system maintains highly competitive classification scores.

Table 20
Comparative results for the gene domain

Methodology	3-Class classification		Classification of neither	Classification of exon–intron	Classification of intron–exon
	Training	Test	10-Fold cross validation	10-Fold cross validation	10-Fold cross validation
G^3P-DT	0.2385	0.2614	–	–	–
G^3P-FRBS	0.1260	0.1277	–	–	–
G^3P-ANN	0.2893	0.3185	–	–	–
G^3P-FPN	0.1227	0.1542	–	–	–
KBANN	–	–	0.0462	0.0756	0.0847
BACKPROP	–	–	0.0529	0.0574	0.1075
PEBLS	–	–	0.0686	0.0818	0.0755
PERCEPTRON	–	–	0.0399	0.1632	0.1741
ID3	–	–	0.0884	0.1058	0.1399
COBWEB	–	–	0.1180	0.1504	0.0946
Near. neighbor	–	–	0.3111	0.1165	0.0909

Table 21
Comparative results for the heart disease domain

Methodology	Error rate		Methodology	Error rate	
	Training	Test		Training	Test
G^3P-DT	0.1753	0.2741	Logistic regression [55]	N/A	0.230
G^3P-FRBS	0.1846	0.2251	NTgrowth [57]	N/A	0.230
G^3P-ANN	0.2329	0.2320	C4 [57]	N/A	0.252
G^3P-FPN	0.2120	0.2338	CLASSIT [56]	N/A	0.211
CDF [55]	N/A	0.210	CADENZA [55]	N/A	0.226

5.4.5. Horse problem

The horse-colic domain results, together with those found in literature [52,53], are shown in Table 22. As in previous paragraphs, the value referred for the G^3P generated systems, is the low classification error *averaged* for the three data sets tested (e.g. horse1, horse2 and horse3). In this domain, the

Table 22
Comparative results for the horse colic domain

Methodology	Error rate		Methodology	Error rate	
	Training	Test		Training	Test
G^3P-DT	0.2130	0.3148	C4.5 [52]	N/A	0.1561
G^3P-FRBS	0.1685	0.3480	Bagged C4.5 [52]	N/A	0.1481
G^3P-ANN	0.2240	0.3295	Randomized C4.5 [52]	N/A	0.1561
G^3P-FPN	0.1796	0.3230	Adaboosted C4.5 [52]	N/A	0.1825
Naïve Bayes [53]	N/A	0.1900	ID3 [53]	N/A	0.2170

Table 23
Comparative results for the thyroid domain

Methodology	Error rate		Methodology	Error rate	
	Training	Test		Training	Test
G^3P-DT	0.021	0.024	Conj.gradient + line search	0.055	0.072
G^3P-FRBS	0.056	0.054	Silva and Almeida	0.004	0.015
G^3P-ANN	0.057	0.055	SuperSAB	0.005	0.015
G^3P-FPN	0.059	0.051	Delta-Bar-Delta	0.008	0.017
Backprop	0.001	0.025	RPROP	0.004	0.019
BP (batch mode)	0.074	0.082	Quickprop	0.004	0.018
BP (batch mode) + Eaton + Oliver	0.076	0.083	Cascade colleration 10 units	0.002	0.016
BP + Darken and Moody	0.001	0.031	Cascade colleration 20 units	0.000	0.015
Salomon	0.054	0.068	Polak-Ribiere + line search	0.053	0.068
Chan and Fallside	0.053	0.068	Schmidhuber	0.017	0.028

results obtained by our system configuration, maintain lower classification scores than those found in literature.

5.4.6. Thyroid problem

For the diabetes domain, we compare our results with a number of neural network algorithms [58]. The methodology of this paper is antagonistic in terms of classification accuracy with many of the neural network models tested in [58]. Again here, we include as classification score for the G^3P generated systems, the average of the low error rates for the three sub-problems addressed (e.g. thyroid1, thyroid2 and thyroid3) (see Table 23).

6. Discussion

Considering the comparison within the G^3P models, the model for *decision trees* outperformed the other models in the thyroid problem. Also it has offered the highest classification scores in the cancer domain, followed by the model for *fuzzy-rule based systems*. The latter model, succeeded in high classification score in the gene domain, followed by the *fuzzy Petri-nets model*. In the domains of horse and diabetes there was not clear winner. On the other hand, the model for the *artificial neural networks* maintained a lower classification score in most of the problems. This led us to further research with larger population size and number of generations. The results showed that by using larger models, we might expect better classification scores only on *average*; the enhancement of the G^3P-ANN performance rather relies to the adoption of a larger solution size (e.g. size of the individual). The latter conclusion derives by the observation of high positive correlations between the solution size and the achieved classification score.

The comparison with the paper's work to the experimentation found in literature is not always a straightforward task. Although the data sets used here have been extensively applied in a wide range of techniques, the specific data setup and split into training and test sets, and the optional inclusion of a validation set, has not always been a well-recorded element. However, for an adequate number of experiments, the results obtained can still be compared, since cross-validation techniques have been used, and the data setup bias is expected to be low. From this point of view, the analyzed system of this paper has succeeded in achieving high classification scores in most of the domains tested. More importantly, the inherited features of the analyzed system (e.g. a genetic programming system) minimize the human involvement when a decision has to be taken on the solution architecture and tuning. These features, together with the highly competitive results presented here, can be used as a guide for the research in further application areas.

7. Conclusions and future work

This work presented a comparison of four grammar-guided genetic programming approaches in data classification. Namely, decision trees, fuzzy rule-based systems, feedforward neural networks and fuzzy Petri-nets were implemented using context-free grammars. Cellular encoding was adopted in order to describe arbitrary network topology for the neural networks and the fuzzy Petri-nets. Six data sets were used mainly from the medical domain. We performed 20 runs in three variations of these data for each of the four approaches. The results allow driving conclusions on the effectiveness of this approach.

For this work, we encountered data sets each with two or three classes. The other two approaches, G^3P for decision trees and G^3P for fuzzy rule-based systems worked competitively between each other, offering valuable results. Specifically, we observed that the DT approach performed better in data sets that had discrete features while the FRBS approach was more effective in data sets consisted of only continuous attributes. Furthermore, we consider that experiments with data sets having larger number of classes (such as the mushroom data from the *UCI Machine Learning repository*) would offer interesting observations on the relative performance of the models.

Research will follow this work regarding larger models that would enable to all methodologies to perform competitively. In the absence of a standard GP benchmarking data set, other real-world domains will be used in order to obtain transparent results on the classification success of this approach. More intelligent models—such as Kohonen networks—will be implemented in context-free grammars to offer more generalized knowledge discovery attempt. Finally, tuning properly the initialization procedure of a grammar-guided genetic programming will enable better exploration of the search space for a given size of the GP model.

References

[1] E. Alba, C. Cotta, J.M. Troya, Evolutionary design of fuzzy logic controllers using strongly-typed GP, in: Proceedings of the 1996 IEEE International Symposium on Intelligent Control, New York, NY, 1996, pp. 127–132.

[2] C. Blake, E. Keogh, C.J. Merz, UCI Repository of Machine Learning Databases. Department of Information and Computer Science, University of California, Irvine, CA. Available from: <http://www.ics.uci.edu/~mlearn/ML-Repository.html>.

[3] C.C. Bojarczuk, H.S. Lopes, A.A. Freitas, Genetic programming for knowledge discovery in chest-pain diagnosis, IEEE Eng. Med. Biol. July (2000) 38–44.

[4] M. Brameier, W. Banzhaf, A comparison of linear genetic programming and neural networks in medical data mining, IEEE Trans. Evol. Comp. 5 (1) (2001) 17–26.

[5] G. Dounias, A. Tsakonas, J. Jantzen, H. Axer, B. Bjerregaard, D.G. v.Keyserlingk, Genetic programming for the generation of crisp and fuzzy rule bases in classification and diagnosis of

medical data, in: Proceedings of the First International NAISO Congress on Neuro Fuzzy Technologies, NF-2002, Habana, Cuba, January 16, 2002.
[6] A.S. Elstein, S. Shulman Lee, S.A. Sprafta, Medical Problem Solving: an Analysis of Clinical Reasoning, Harvard University Press, Cambridge, 1978.
[7] A.A. Freitas, Genetic programming framework for two data mining tasks: classification and generalized rule induction, in: Genetic Programming 1997: Proceedings of the 2nd Annual Conference (Stanford University, July 1997), Morgan Kaufmann, 1997, pp. 96–101.
[8] F. Gruau, Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm, Ph.D. Thesis, Ecole Normale Superieure de Lyon, anonymous ftp:lip.ens-lyon.fr (140.77.1.11) pub/Rapports/PhD PhD94-01-E.ps.Z.
[9] F. Gruau, On using syntactic constraints with genetic programming, in: P.J. Angeline, K.E. Jinnear (Eds.), Advances in Genetic Programming, MIT, 1996.
[10] F. Gruau, D. Whitley, L. Pyeatt, A comparison between cellular encoding and direct encoding for genetic neural networks, in: J.R. Koza, D.E. Goldberg, D.B. Fogel, R.L. Riolo (Eds.), Genetic Programming 1996: Proceedings of the First Annual Conference, MIT Press, Cambridge, MA, 1996, pp. 81–89.
[11] T. Hussain, Cellular Encoding: Review and Critique, Technical Report, Queen's University, 1997. Available from: <http://www.qucis.queensu.ca/home/hussain/web/1997_cellular_encoding_review.ps.gz>.
[12] T. Hussain, R. Browse, Attribute grammars for genetic representations of neural networks and syntactic constraints of genetic programming, in: AIVIGI'98: Workshop on Evol. Comp., Vancouver BC, 1998.
[13] J.-S.R. Jang, C.-T. Sun, E. Mizutani, Neuro-fuzzy and Soft Computing, Prentice-Hall, Upper Saddle River, NJ, 1997.
[14] C.Z. Janikow, A methodology for processing problem constraints in genetic programming, Comput. Math. Appl. 32 (8) (1996) 97–113.
[15] J.R. Koza, Genetic Programming: on the Programming of Computers by Means of Natural Selection, MIT Press, Cambridge, MA, 1992.
[16] J.R. Koza, F.H. Bennett III, D. Andre, M.A. Keane, Genetic Programming III, Morgan Kaufmann Publ. Inc., 1999.
[17] P. Langley, H.A. Simon, G.L. Bradshaw, Rediscovering chemistry with the Bacon system, Machine Learning: an Artificial Intelligence Approach, vol. 1, Morgan Kaufmann, 1983.
[18] M.F. Abbod, D.G. von Keyserlingk, D.A. Linkens, M. Mahfouf, Survey of utilization of fuzzy technology in medicine and health care, Fuzzy Sets Syst. 120 (2:1) (2001) 331–349.
[19] C.G. Looney, Fuzzy Petri nets for rule-based decision-making, IEEE Trans. Systems Man Cybernet 18 (1988) 178–183.
[20] D.J. Montana, Strongly typed genetic programming, Evol. Comput. 3 (2) (1995).
[21] N. Paterson, M. Livesey, Evolving caching algorithms in C by GP, in: Genetic Programming 1997, MIT Press, 1997, pp. 262–267.
[22] P. Naur, Revised report on the algorithmic language ALGOL 60, Commun. ACM 6 (1) (1963) 1–17.
[23] P.S. Ngan, M.L. Wong, W. Lam, K.S. Leung, J.C.Y. Cheng, Medical data mining with evolutionary computation, Artificial Intell. Med. 16 (1999) 73–96.
[24] W. Pedrycz, Generalized fuzzy Petri nets as pattern classifiers, Pattern Recogn. Lett. 20 (1999) 1489–1498.
[25] L. Prechelt, Proben1—A set of neural network benchmark problems and benchmarking rules, Tech.Rep. 21/94, University of Karlsruhe, Karlsruhe, Germany, 1994.
[26] J.R. Quinlan, Induction of decision trees, Mach. Learning 1 (1986) 81–106.
[27] A. Ratle, M. Sebag, Genetic programming and domain knowledge: beyond the limitations of grammar-guided machine discovery, in: M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E.

Lutton, J.J. Merelo, H.P. Schwefel (Eds.), Parallel Problem Solving from Nature-PPSN VI 6th International Conference, Springer-Verlag, Paris, 2000, pp. 211–220.
[29] C. Ryan, J.J. Collins, M. O'Neil, Grammatical evolution: evolving programs for an arbitrary language, in: W. Banzhaf, R. Poli, M. Schoenauer, T.C. Fogarty (Eds.), Genetic Programming, Lecture Notes in Computer Science, Springer, 1998.
[30] A. Singleton, Genetic Programming with C++, BYTE Magazine, 1994.
[31] A. Tsakonas, G. Dounias, Hierarchical classification trees using type-constrained genetic programming, in: Proceedings of the 1st International IEEE Symposium in Intelligent Systems, Varna, 2002.
[32] A. Tsakonas, G. Dounias, H. Axer, D.G. von Keyserlingk, Data classification using fuzzy rule-based systems represented as genetic programming type-constrained trees, in: Proceedings of the UKCI-01, Edinbourgh, 2001, pp. 162–168.
[33] A.Tsakonas, G. Dounias, A scheme for the evolution of feedforward neural networks using bnf-grammar driven genetic programming, in: Proceedings of the Eunite-02, Algarve, 2002.
[34] P. Whigham, Search bias, language bias and genetic programming, in: Genetic Programming 1996, MIT Press, 1996, pp. 230–237.
[35] M.L. Wong, A flexible knowledge discovery system using genetic programming and logic grammars, Decision Support Syst. 31 (2001) 405–428.
[36] T. Yu, P. Bentley, Methods to evolve legal phenotypes, in: Lecture Notes in Computer Science 1498, Proceedings of the Parallel Problem Solving from Nature V, 1998, pp. 280–291.
[37] A.E. Eiben, J.E. Smith, Introduction to Evolutionary Computing, Springer, 2003, ISBN 3-540-40184-9.
[38] T. Baeck, D.B. Fogel, Z. Michalewicz (Eds.), Handbook of Evolutionary Computation, IOP Publishing, 1997, ISBN 0-750-30895-8.
[39] W. Banzhaf, P. Nordin, R.E. Keller, F.D. Francone, Genetic Programming—an Introduction; on the Automatic Evolution of Computer Programs and Its Applications, Morgan Kaufmann, 1988, ISBN 3-920993-58-6.
[40] W.H. Wolberg, W.N. Street, O.L. Mangasarian, Machine learning techniques to diagnose breast cancer from fine-needle aspirates, Cancer Lett. 77 (1994) 163–171.
[41] W.H. Wolberg, W.N. Street, O.L. Mangasarian, Image analysis and machine learning applied to breast cancer diagnosis and prognosis, Anal. Quantitative Cytol. Histol. 17 (2) (1995) 77–87.
[42] Statlog, Results on the Pima Indians diabetes domain, University of Jena. Available from: <http://www.minet.uni-jena.de/www/fakultaet/schukat/ME/SS04/data/statlog/diabetes.txt> [Date of last access: January 20, 2005].
[43] J.W. Smith, J.E. Everhart, W.C. Dickson, W.C. Knowler, R.S. Johannes, Using the ADAP learning algorithm to forecast the onset of diabetes mellitus, in: Proceedings of the Symposium on Computer Applications and Medical Care, IEEE Computer Society Press, 1988, pp. 261–265.
[44] J. Zhang, Selecting typical instances in instance-based learning, in: Proceedings of the Ninth International Machine Learning Conference, Morgan Kaufmann, Aberdeen, Scotland, 1992, pp. 470–479.
[45] W.H. Wolberg, O.L. Mangasarian, Multisurface method of pattern separation for medical diagnosis applied to breast cytology, Proc. Natl. Acad. Sci. 87 (1990) 9193–9196.
[46] P.A. Whigham, Grammatically based genetic programming, in: J.P. Rosca (Ed.), Proceedings of the Workshop on Genetic Programming: From Theory to Real World Applications, Tahoe City, CA, USA, pp. 33–41.
[47] N.L. Cramer, A representation for the adaptive generation of simple sequential programs, in: J.J. Grefenstette (Ed.), Proceedings of an International Conference on Genetic Algrorithms and the Applications, Carnegie Mellon University, Pittsburgh, PA, USA, pp. 183–187.

[48] Y. Shan, R.I. McKay, C.J. Lokan, D.L. Essam, Software project effort estimation using genetic programming, International Conference on Communication Circuits and Systems, UESTC Press, Chengdu, China, 2002, pp. 1108–1112.
[49] M. Schoenauer, M. Sebag, F. Jouve, B. Lamy, H. Maitournam, Evolutionary identification of macro-mechanical models, in: P.J. Angeline, K.E. Kinnear Jr (Eds.), Advances in Genetic Programming II, MIT Press, Cambridge, MA, 1996.
[50] Y. Shan, R.I. McKay, R. Baxter, H. Abbass, D. Essam, H.X. Nguyen, Grammar model-based program evolution, in: Proceedings of the Congress on Evolutionary Computation 2004, Portland, US, 2004, pp. 478–485.
[51] N.X. Hoai, R.I. McKay, H.A. Abbass, Bias in tree adjoining grammars, in: Proceedings of the European Conference on Genetic Programming (EUROGP), Lecture Notes in Computer Science LNCS-2610, Springer-Verlag, Berlin, 2003.
[52] T.G. Dietterich, An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization Machine Learning, Kluwer Academic Publishers, Boston, 1999.
[53] M.J. Pazzani, Searching for dependencies in Bayesian classifiers, in: D. Fisher, H.J. Lenz (Eds.), Learning from Data: AI and Statistics V, Springer-Verlag, 1996, pp. 239–248.
[54] T.G. Dietterich, G. Bakiri, Solving multiclass learning problems via error-correcting output codes, J. Artificial Intell. Res. 2 (1995) 263–286.
[55] R. Detrano, A. Janosi, W. Steinbrunn, M. Pfisterer, J. Schmid, S. Sandhu, K. Guppy, S. Lee, V. Froelicher, International application of a new probability algorithm for the diagnosis of coronary artery disease, Amer. J. Cardiol. 64 (1989) 310.
[56] J.H. Gennari, P. Langley, D. Fisher, Models of incremental concept formation, Artificial Intell. 40 (1989) 11–61.
[57] D. Aha, D. Kibler, M. Albert, Instance-based learning algorithms, Mach. Learning 6 (1991) 37–66.
[58] W. Schiffmann, M. Joost, R. Werner, Comparison of optimized backpropagation algorithms, in: Proceedings of the European Symposium on Artificial Neural Networks, ESANN'93, Brussels, 1993, pp. 97–104.