PERGAMON

Engineering Applications of Artificial Intelligence 14 (2001) 727–736

Engineering Applications of
ARTIFICIAL
INTELLIGENCE

www.elsevier.com/locate/engappai

Genetic programming: principles and applications

S. Sette[a,*], L. Boullart[b]

[a] Department of Textiles, Ghent University, Technologiepark 9, 9052 Gent, Belgium
[b] Department of Automation & Control Engineering, Ghent University, Technologiepark 9, 9052 Zwijnaarde, Belgium

Abstract

Genetic algorithms (GA) has given rise to two new fields of research where (global) optimisation is of crucial importance: '*genetic based machine learning*' (GBML) and '*genetic programming*' (GP). An introduction by the authors to GA and GBML was given in two previous papers (Eng. Appl. Artif. Intell. 9(6) (1996) 681; Eng. Appl. Artif. Intell. 13(4) (2000) 381). In this paper, the last domain (GP) will be introduced, thereby making up a trilogy which gives a general overview of the whole field. In this third part, an overview will be given of the basic concepts of GP as defined by Koza. A first (educational) example of GP is given by solving a simple symbolic regression of a sinus function. Finally, a more complex application is presented in which GP is used to construct the mathematical equations for an industrial process. To this end, the case study 'fibre-to-yarn production process' is introduced. The goal of this study is the automatic development of mathematical equations for the prediction of spinnability and (possible) resulting yarn strength. It is shown that (relatively) simple equations can be obtained which describe accurately 90% of the fibre-to-yarn database.

Keywords: Genetic programming; Genetic algorithms; Textiles; Fibre-to-yarn; Production process

1. Introduction

Genetic algorithms (GAs) are generally used as an optimisation technique to search the global optimum of a function. However, this is not the only possible use for GAs. Other fields of applications where robustness and global optimisation are needed could also benefit greatly from GAs. The two most important alternative domains applying GAs are *genetic based machine learning* (GBML) and *genetic programming* (GP).

GBML is a GA driven implementation of rule based machine learning (RBML). The goal in RBML is to generate a set of rules using an automated learning algorithm. This set of rules should allow the machine to perform optimally in its environment. A well-known GBML architecture is the so-called '*learning classifier system*' (LCS) developed by Holland (1968, 1973) and Holland and Reitman (1978). More recent GBML architectures are for example the XCS developed by Wilson (1994), ELF by Bonnarini (1997) and (F)ECS by Sette and Boullart (2000). An introduction to GBML was given by Boullart and Sette (1998).

GP is basically a GA applied to a population of computer programs (CP). While a GA usually operates on (coded) strings of numbers a GP has to operate on CP. This demands a special *representation* of the operands which can, for example, be implemented by using the programming language LISP. GP allows, in comparison with GA, the optimisation of much more complicated structures and can therefore be applied to a greater diversity of problems. Koza has extensively described GP in his book 'Genetic Programming, on the programming of computers by means of natural selection' (1992).

An introduction to GAs and GBML was given by the authors in Sette et al. (1996) and Sette and Boullart (2000). Both papers illustrated the use of these algorithms by applying them to an industrial production process in textiles: the (cotton) fibre-to-yarn production process. The GA was used to search for a new yarn with optimal strength and elongation characteristics. GBML was used to generate a number of rules for predicting the spinnability of the yarn (based upon fibre blend and spinning machine settings).

This paper is the last paper in this introductory series and will illustrate the concepts of GP (as described more extensively by Koza) and apply it to the same aforementioned real life industrial production problem,

*Corresponding author. Tel.: +32-9264-5413; fax: +32-9264-5846.
E-mail addresses: stefan.sette@rug.ac.be (S. Sette), boullart@autoctrl.rug.ac.be (L. Boullart).

0952-1976/01/$ - see front matter
PII: S0952-1976(02)00013-1

generating equations for the spinnability and the yarn strength. Finally, some conclusions based upon the comparison of the three different (but related) techniques will be presented.

2. Introducing GP

The first part of this introduction will give a schematical overview of the GP algorithm. The operation is very similar to the steps performed in a GA. The second part will focus on the most essential feature of GP: the representation of the solution/operand in order to apply GAs to CP. Due to the obvious restriction in scope, the underlying description cannot be complete, but will concentrate on the important aspects in view of the application at hand.

2.1. General overview of the GP algorithm

A schematical overview is given in Fig. 1.
The following steps can be distinguished:

1. Generation of a random population of CP.
2. Evaluation of the fitness of all CPs in the population. Moreover if a certain criterion is reached (for example, a certain fitness threshold), the algorithm is terminated and the CP with the highest fitness is selected as the final result.
3. Replacing the current population by a new population by means of applying genetic operators (reproduction, crossover and mutation) probabilistically.
4. Return to step 2

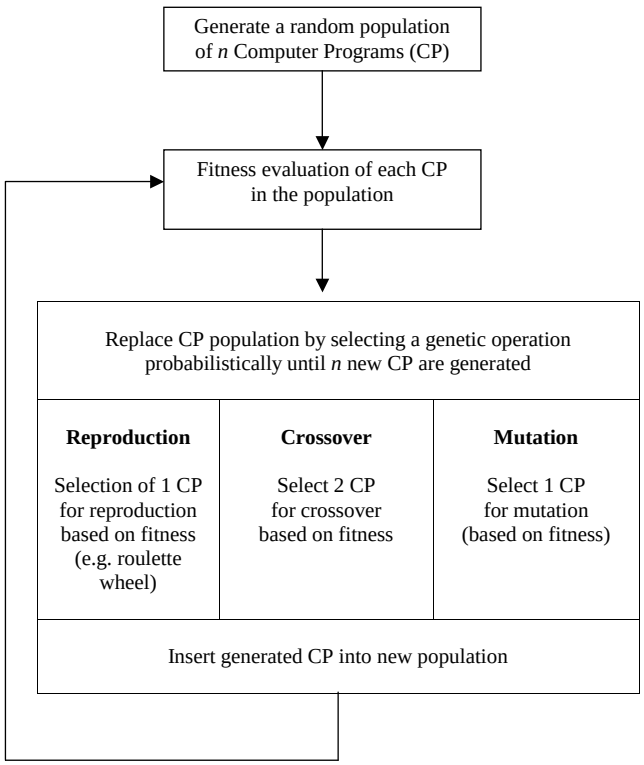

Fig. 1. Schematical overview of GP.

It is clear that this procedure is nearly identical to the one followed in a 'classic' GA. The major difference will be the representation and the corresponding fitness evaluation of the CP and this will be discussed in the next paragraph.

2.2. Representation of the CP

In a GA, a population member is a (often binary) coded representation of a number. However, a population member in GP is a hierarchically structured (computer) program consisting of *functions* and *terminals*. The functions and terminals are selected from a set of functions and a set of terminals. For example, a function set F could contain the basic arithmetic operations: $F = \{+, -, *, /\}$. However, the function set may also include other mathematical functions, Boolean operators, conditional operators or any user-defined operators. For each of the operators, the number of arguments ('arity') has to be defined. For F, the corresponding number of operators is given by $\{2, 2, 2, 2\}$. The terminal set T contains the arguments for the functions. For example $T = \{x, y\}$ with x and y being two independent variables.

A CP can now be depicted as a rooted, point-labelled tree with ordered branches, using operations (internal points of the tree) from the function set and arguments (leaves of the tree) from the terminal set.

An example of such a tree using the aforementioned F and T is given in Fig. 2.

The aforementioned tree representation can be implemented in almost any programming language

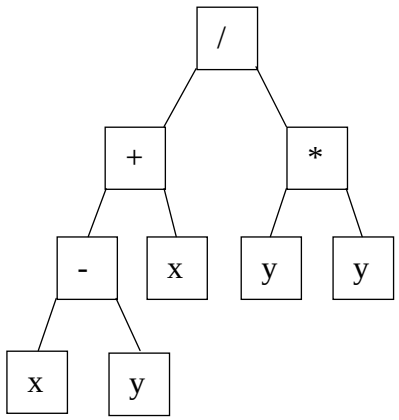

Fig. 2. A tree representation of a CP corresponding with the equation $(x - y) + x/y^2$.

(C++, Pascal, …), but probably the most convenient is LISP as it has the following advantages:

- Programs and data have the same form (the so-called S-expressions): in other words a CP can be treated as if it were data (allowing easily GA operations) *and* immediately evaluated to assess the results.
- The S-expression is equivalent to a parse tree of a CP. But, contrarily to LISP, most programming languages produce (when compiling) such a parse tree, only in an intermediate stage without being accessable for manipulation.
- LISP allows and supports easily dynamic changing programming structures.

An elementary introduction to LISP is given by Steele (1984).

Using a LISP S-expression, the aforementioned example can be written as

$$(/(+(-x\,y)x)(*y\,y)).$$

This expression, using a prefix notation, is read from left to right applying recursively each function to the next two arguments or sub-S-expressions.

Furthermore, to obtain a useful representation with regard to solving a problem, the function set and terminal set must fulfill two important requirements:

1. *Closure property* of the function set and terminal set: each function of the function set must be able to process *all* possible argument values generated by other functions or the terminal set. For example: mathematical functions should be 'protected' versus division by zero, negative logarithms, negative square root, etc. A possible solution can be implemented by defining (new) protected ('pseudo') functions in which the argument is tested and a specific result is returned when the function could not process the argument in a native way.
2. *Sufficiency property* of the function and terminal set: the problem must be solvable using the (proposed) selection of functions and terminals. In other words: the human operator/programmer must choose a set of functions and terminals which are relevant (and could usefully be applied by the GP) to the problem. It must be possible to describe the final solution adequately by the selected functions/terminals. This requires a certain knowledge or insight into the problem itself. Depending on the problem, the selection of a set of sufficient functions/terminals could range from obvious to almost impossible. Also the complexity of the final solution is largely influenced by the chosen function/terminal set (see example below).

Finally, the *fitness* of each CP can easily be determined by evaluating the corresponding S-expression and comparing it to a target value (possible several target values selected from the entire domain space). The exact implementation is problem dependant. Several fitness possibilities are suggested by Koza: raw fitness, standardised fitness, adjusted fitness and normalised fitness.

2.3. Genetic operations on CP

The genetic operations applied in GP are the basic GA operators: reproduction, crossover and mutation.

Reproduction can have several different implementations:

- Fitness-proportionate reproduction: the probability that a certain CP is selected is proportionate to its fitness (roulette wheel algorithm).
- Rank selections: similar to fitness-proportionate but the numerical fitness value is replaced by a ranking.
- Tournament selection: Two CPs are chosen at random and the CP with the highest fitness is selected.

Crossover is determined by choosing two CPs based on fitness (see reproduction algorithm) and generating for each CP the crossover point (node) at random. For example: consider the following CPs (Fig. 3) with crossover points 2 and 3. The sub-tree of CP 1 starting from crossover point 2 will be swapped with the sub-tree of CP 2 at crossover point 3.

Resulting in (Fig. 4)

Or written as S-expressions (swappables are in boldface):

$$\text{CP1}: \quad (/(+(-\mathbf{x}\,\mathbf{y})\mathbf{x})(*y\,y)),$$
$$\text{CP2}: \quad (*x(*\mathbf{x}\,\mathbf{y})).$$

Resulting after crossover in

$$\text{CP1}': \quad (/(*\mathbf{x}\,\mathbf{y})(*y\,y)),$$
$$\text{CP2}': \quad (*x(+(-\mathbf{x}\,\mathbf{y})\mathbf{x})).$$

As can be seen from the previous example, a crossover between S-expressions consists simply of swapping the (at random selected) sub-S-expressions.

Mutation can be implemented in two different ways:

- A random terminal or function is selected and replaced by another (at random selected) terminal or function.
- An at random selected sub-tree is replaced by an at random generated sub-tree.

However, in general, mutation only plays a minor role in GP and can be omitted in most cases (Koza, 1992).

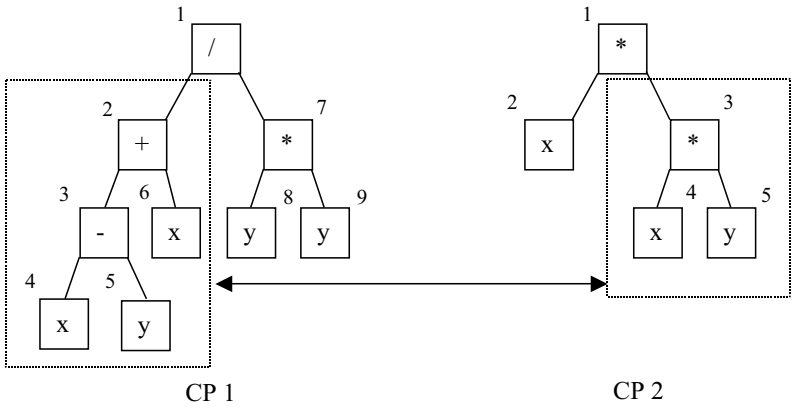

Fig. 3. Example crossover (start).

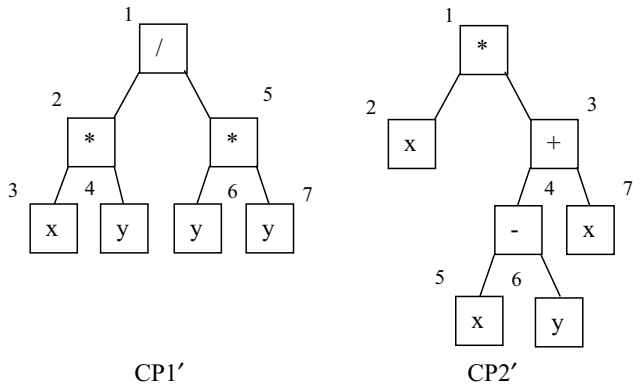

Fig. 4. Example crossover (result).

2.4. Some advanced operations on CP

Permutation operates on a single CP and allows for the permutation of the k arguments of an (internal) function of the corresponding S-expression, selecting at random one permutation of the $k!$ possibilities. Not only its effect is limited to (non-commutative) functions, but it is even unclear if this operator is useful at all (Koza, 1992).

The *Editor operator* allows the simplification of S-expressions. Two cases can be distinguished:

1. Domain independent: if a function only has constant arguments, the function is evaluated and replaced (in the S-expression) by the result.
2. Domain dependant: in this case a number of domain specific rules are applied to simplify the S-expression. For example, the subtraction of two identical sub-S-expressions can be replaced by 0.

The advantages are a simpler, shorter S-expression, which becomes more 'readable' to the user and possibly improve overall performance of the GP.

The *encapsulation operator* searches for the useful sub-S-expressions (which can be considered as the building blocks of a GP) and includes them into a newly defined function/terminal set. To this end, the operator uses the following steps:

- Select a S-expression proportionate with fitness.
- Select at random an internal function.
- Replace the internal function and all his arguments with a reference function. The function set is augmented to include this new encapsulated function. No arguments can be given to this encapsulated function, as the arguments are already fixed for this function. The encapsulated function is indivisible and has become a new terminal of the tree.

This operator is also known as the 'automatically defined function' (ADF) operator.

Decimation allows for a quick (probabilistic) elimination of low-fitness individuals within a population. Decimation will reduce the population (at a specified time or generation) to a certain percentage based upon a probabilistic selection proportionate with the fitness of each individual. It may speed up the overall computational process.

2.5. Applications of GP

Koza has shown that GP can be applied to a great diversity of problems. He especially illustrated the use of GP with regard to optimal control problems (where he obtained considerable better results on well-known problems), robotic planning (artificial ant in search of food problem), symbolic regressions and the well-known Boolean 11-multiplexer problem.

The applications, which will be considered in this paper, belong to the domain of symbolic regression. First, a very simple example will serve as an introduction. Finally, GP will be used to generate mathematical

equations for an existing database of an industrial production process.

3. Simple symbolic regression application

This (educational) experiment has the (limited) goal of finding a mathematical expression for a $\sin(x)$, $0 \leqslant x \leqslant 180$, using only the function set $\{+, -, *, /\}$ and the terminal $T = \{x, [0...9]\}$.

To this end, the software package GPCPP4 (freeware) from Adam Fraser has been used to demonstrate the use of GP for modelling the sinus function. The software was only slightly modified with regard to the function (using the symbolic regression module) and the corresponding fitness calculation. The function, which was to be approximated by symbolic regression, is

$$g(x) = 25\sin(x/3.0)$$

with x expressed in radials from 0 to 9 ($x/3.0$ effectively corresponding with a range in degrees from 0 to 172). The fitness f is based on the sum of the differences between the GP generated programs and the real sinus values for $10x$ values ($x = 0, 1, \ldots, 9$)

$$f = 200 - \sum_{i=0}^{9} |g(x_i) - \mathrm{GP}(x_i)|.$$

Each difference has a maximum value of 20. In other words, the fitness will always be situated between 0 and 200. It turns out that when using different alternative fitness functions (such as sum of squares) almost similar results are produced (thereby proving the robustness of the underlying method).

The population size is initialised at 100, evolving for 30 generations.

The following (best of generation) S-expressions were generated during these 30 generations (spaces between characters have been removed to allow for a one line S-expression):

E1 : (−(+(*(% 6X)(*(*(+X9)5)(%(% X5)4)))
× (*X7))(*XX)),

E2 : (−(+(*(% 6X)(*(*(+X(−X1))5)(%(% X5)4)))
× (*X7))(*XX)),

E3 : (−(+(*X(% 94))(*X7))(*XX)),

E4 : (−(+(*X(% 94))(+(% X(% 94))(*X 7)))(*X X)).

With % the protected division operator defined as

If (denominator = 0 then result = 1) else (result = numerator/denominator)

An overview of the results is given in Table 1. The complexity herein is defined as the number of terminals and functions used in each equation.

Table 1
Overview of some results

	Generation	Fitness	Complexity
E1	1	131	23
E2	2	185	25
E3	11	182	13
E4	20	190	19

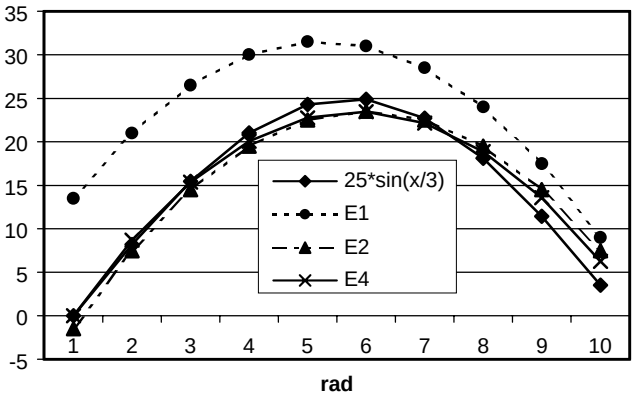

Fig. 5. Results for the symbolic regression of $g(x)$.

Written in a more conventional way, the S-expressions are

$$\mathrm{E1} = -x^2 + \tfrac{17}{2}x + \tfrac{27}{2}, \quad \mathrm{E3} = -x^2 + \tfrac{37}{4}x,$$

$$\mathrm{E2} = -x^2 + 10x - \tfrac{3}{2}, \quad \mathrm{E4} = -x^2 + \tfrac{349}{36}x.$$

E1, E2 and E4 are visualised in Fig. 5 together with the function $g(x)$.

The result E4 is very close to $g(x)$ with a maximum fitness of 190 (200 being identical to $g(x)$ in the 10 selected x-values). As can be seen from E1 to E4 the sub-S-expressions (*X7) and (*XX) are high fitness building blocks, which are kept over generations to construct higher fitness program trees. At generation 11 (E3) this sub-S-expression is combined with (*X(% 94)), finally resulting in the optimal solution E4.

The importance of a suitable function set (sufficiency property) is demonstrated by implementing the symbolic regression for $\sin(x)$ with three different function sets. Table 2 shows results, where all functions have run until a minimum fitness has been reached.

The results for F_3 are the least accurate (highest error) and need the largest number of generations (resulting in the largest computing time). The aforementioned example shows that the inclusion of a suitable function in the function set is extremely important for obtaining good results (within an acceptable time frame).

Table 2
Results for different function sets

Function sets	Result	Generation	Error (final)
F_1: {+, −, *, /, sin}	sin(x)	0	0.00
F_2: {+, −, *, /, cos}	cos($x+4.66$)*	12	0.40
F_3: {+, −, *, /}	$-x^2+9.7x$	29	1.36

*$4.66-\Pi=1.519$, which is almost equal to $\Pi/2$ (3.3% error).

4. Industrial application: constructing mathematical equations for the fibre-to-yarn process by means of GP

One of the important production processes in the textile industry is the spinning process. Starting from cotton fibres, yarns are (usually) created on a rotor-spinning machine. The spinnability of a fibre and the yarn characteristics (if spinnable) are dependant on the fibre quality and on the machine settings of the spinning machine. It would be a great benefit to be able to predict the spinnability and yarn strength departing from a certain quality and from machine settings. To this end, two totally different modelling approaches can be considered: the so-called 'white' modelling and the so-called 'black box' modelling.

In white modelling, the process is described by mathematical equations, which are based upon (theoretical) physical knowledge of the process. Extensive physical information about the process is in this case available through physical, chemical or mechanical equations giving the user a thorough insight into the operation of the process. However, due to the large input (and output) dimensions of the fibre-to-yarn process and their complex interactions, no exact mathematical model of a spinning machine is known to exist nor is it likely that such a model will ever be constructed.

A black box model will, in contrast to white modelling, simply connect input parameters to the output without giving or containing any substantial physical justification to the process itself. Black box models have been successfully constructed by Pynckels et al. to predict the spinnability (1995) and the characteristics (1997) of the yarn using neural networks with a backpropagation learning rule.

Sette (1998) introduced the so-called 'grey modelling' of the fibre-to-yarn process, by constructing a new GBML algorithm: Fuzzy efficiency based classifier system (FECS). FECS generates a rule set by means of an automated learning mechanism. This method established not only a relationship between input and output parameters, but also generated physically intelligible information (represented as a rule set) and a degree of the applicability of each rule.

In all cases, the grey models reached a minimum accuracy of 90%, whereas the black box models have a slight advantage (92% a 93%) on accuracy but lack any additional physical information. GP can be seen as a 'grey modelling' algorithm, where physical interpretable equations are generated, departing from (meaningless) random initialised equations.

In the underlying paragraph GP will be used to generate mathematical equations for the spinnability of a yarn and also for the resulting yarn strength (the most important yarn characteristic).

4.1. Experimental set-up

The data set for the fibre-to-yarn process was selected from the database described by Sette et al. (1996) on which also the spinnability models of Pynckels et al. (1995, 1997) are based. This database was originally build within the framework of a BRITE/EURAM project (1990–1993) and consisted of 20 different cotton types for which five different spinning machine settings were systematically changed. The data set used for modelling is a subset of the aforementioned database and was constructed as follows:

- five machine parameters m_i (yarn count, twist, navel, breaker and rotor) with several possible *discrete* settings were used to set up the machine;
- five different *continuous* fibre characteristics f_i were selected (length, uniformity, strength, elongation and micronaire).

The whole fibre-to-yarn database consisted of 2160 data samples of which 1240 are spinnable. All parameters are scaled between -1 and 1, with the exception of spinnability where 0 corresponds with not spinnable and 1 with spinnable.

The LISP code of the GP is a modified version of the code presented by Koza. It now supports multiple dimensional input parameters, reading learning data from an ASCII file and more functions in the GP function set. The function set consisted of:

- the basic functions '+, −, *, /,
- a maximum/minimum function 'max, min' selecting maximum/minimum of two parameters,
- a step function S which rounds a parameter to 1.0 if the parameter is larger then 0.5 or 0.0 otherwise,
- a (protected: only positive values) square root function,
- cos, sin functions.

Two experiments were conducted:

1. Prediction of spinnability: a learn file is at random constructed containing 500 spinnable and 500 unspinnable samples. The whole file is presented for testing. The population for the GP consisted of 10 000 members evolving for 51 generations.

2. Prediction of yarn tenacity: a learn file is constructed by selecting 5 times at random 30 samples (for a total of 150 learn samples) within a yarn strength range 0.4 (obtaining an equal distribution of the learn samples within the ranges [−1,−0.6,−0.2,0.2,0.6,1]). The whole file (1240 samples) is presented for testing. The population of the GP consisted of 5000 members evolving for 51 generations.

4.2. Results

For spinnability Sp an accuracy of 90.1% towards the whole database using the following generated equation (Eq. (1)):

$$\mathrm{Sp} = S(\mathrm{Max}(\mathrm{Max}(\mathrm{Max}(m_1,\ m_3 + m_2), S(m_1)) \times (-4.6/m_5)), m_3 - m_5). \quad (1)$$

The following conclusions can be made from this equation:

Spinnability is only *dependant on the machine parameters* m_i: 90% of the database is correctly described without the use of the fibre parameters f_i. A similar result with regard to the importance of the machine parameters on the spinnability of a yarn was obtained when using FECS. About 77% of the samples can be correctly predicted using only the sub-expression $|m_3—m_5|$, indicating the crucial importance of these two machine parameters.

For tenacity, Ten a relative procentual accuracy of 90.5% is obtained towards the learn file and 90.2% towards the whole database using the following generated equation (Eq. (2)):

$$\mathrm{Ten} = \sin(f_3 * \sqrt{|\sin(\cos(f_3 - 6.0 - \sin(m_1)|)}. \quad (2)$$

This equation shows that a reasonable good accuracy for predicting the tenacity can be reached using only *two* parameters: fibre strength f_3 and yarn count m_1.

Fig. 6 shows the general behaviour of the aforementioned GP generated yarn tenacity equation.

From this figure it is clear that:

- The curve for yarn count = 1 (m_1) gives the highest yarn tenacity for all possible different fibre strengths. Even a low fibre strength still results in an 'acceptable' yarn tenacity (yarn tenacity > -0.4).
- The second curve for yarn count = −0.18 shows an almost linear behaviour between fibre strength and yarn tenacity in the region fibre strength [−1, 0.25]. For higher fibre strengths, the yarn tenacity significantly drops (obtaining a final value < 0.4).
- The last curve (corresponding with a yarn count = −1) shows for lower fibre strengths the same behaviour as for yarn count = −0.18. However, for higher fibre strengths [0.25–0.5] an almost constant yarn tenacity is obtained followed by a sharp increase in yarn tenacity (fibre strengths [0.5–1.0]).

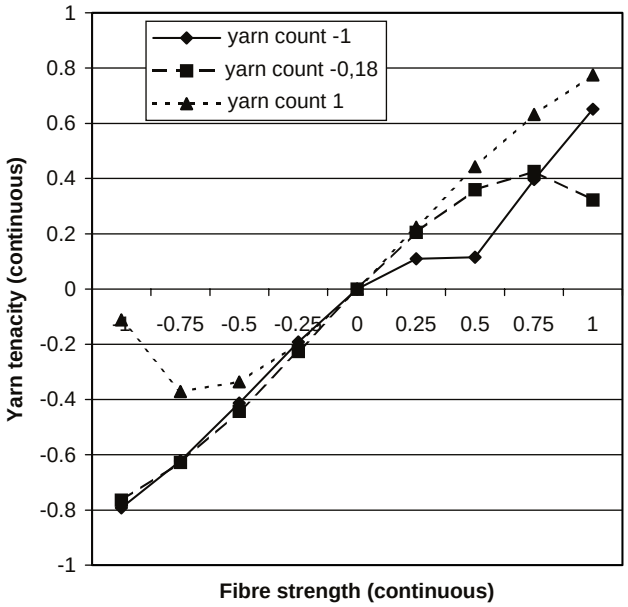

Fig. 6. GP equation for yarn tenacity.

A three-dimensional surface can be constructed if the yarn count is considered continuous. This surface is presented in Fig. 7.

Although this surface is rather artificial (as the yarn count now becomes continuous) it still could provide interesting information for experiments with other (discrete) yarn count values. A strange feature in this surface is the steep valley for higher fibre strengths. This is a result of the 'combination' between the two curves of Fig. 1 with yarn count −0.18 and −1. The combination of a high fibre strength with a low yarn count can therefore result in an 'unpredictable' yarn strength as this section of the figure contains a lot of variation.

4.3. Comparing the results from GBML (FECS) and GP

The rule format for FECS consisted of the same five continuous (fuzzy) fibre characteristics f_1, f_2, f_3, f_4 and f_5 (length, uniformity, strength, elongation and micronaire), five (discrete) machine parameters m_1, m_2, m_3, m_4 and m_5 (yarn count, twist, navel breaker and rotor speed) and one output parameter: spinnability (yes/no) or yarn strength (represented in three fuzzy classes low, medium and high). These rules were all coded using a ternary alphabet {0, 1, #} with the '#' symbol representing either a 0 or 1. A detailed discussion of this GBML implementation is given by Sette and Boullart (2000).

Tables 3 and 4 give the ten most important rules generated by FECS for predicting spinnability and yarn strength (ordered according to importance: rule 1 being the least important and rule 10 the most important). An

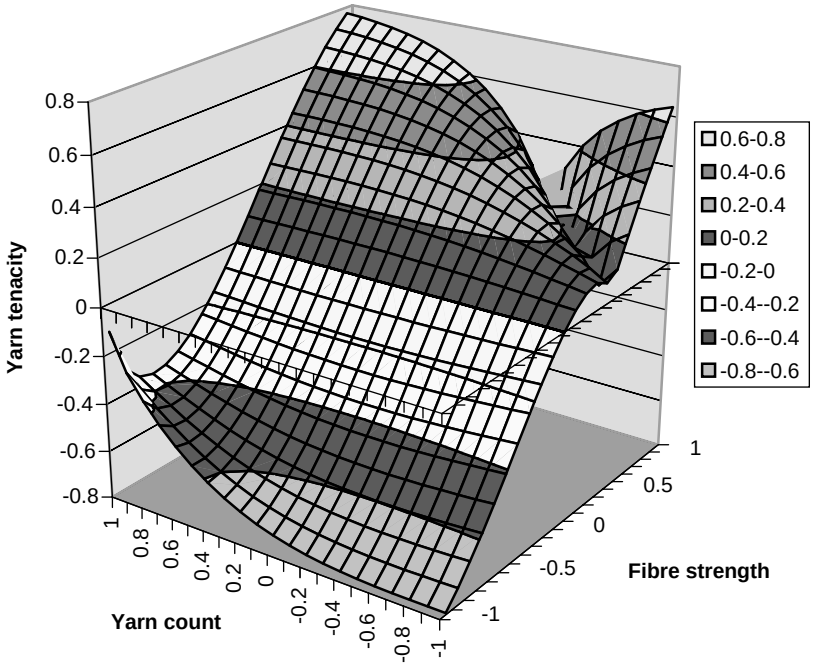

Fig. 7. GP equation for yarn tenacity (three dimensional).

Table 3
Ten most important rules for predicting spinnability

Input										Output	Rule nr.	Reward
m_1	m_2	m_3	m_4	m_5	f_1	f_2	f_3	f_4	f_5			
#0	01	0	##	1	0	0	#	#	0	1	1	44
##	0#	#	10	0	0	#	1	#	#	1	2	45
0#	#0	1	##	1	#	#	#	0	#	0	3	46
00	01	1	##	1	#	#	#	#	#	0	4	51
0#	10	1	0#	0	#	#	#	#	#	1	5	80
0#	00	1	##	1	#	#	#	#	#	0	6	86
##	01	1	0#	0	#	#	#	#	#	1	7	115
01	01	0	##	#	#	#	#	#	#	1	8	117
#1	10	0	##	#	#	#	#	#	#	1	9	118
##	##	1	##	1	#	0	0	#	#	0	10	214

Table 4
Ten most important rules for predicting the yarn strength

Input										Output	Rule nr.	Reward
m_1	m_2	m_3	m_4	m_5	f_1	f_2	f_3	f_4	f_5			
1#	#0	#	00	0	#	1	1	#	#	2	1	14.8
01	01	0	0#	#	#	1	1	#	#	2	2	15.4
01	00	#	00	0	#	#	1	#	#	2	3	16.8
10	##	#	0#	1	#	#	#	0	#	1	4	16.9
#1	#0	1	0#	0	0	#	0	#	#	0	5	17.9
1#	00	0	##	#	#	#	1	#	#	2	6	20.5
0#	00	0	0#	#	0	#	0	0	#	0	7	21.9
0#	00	0	01	1	0	#	0	#	#	0	8	23.8
10	#1	#	##	#	#	#	1	#	#	2	9	48.2
00	0#	1	0#	0	0	#	0	#	#	0	10	117.2

indication of the importance of each input parameter is given by considering all his meaningful selections (at least one 0 or 1). A summary of the number of meaningful selections (limited to the ten most important rules) is given in Table 5.

Comparing the results from this table with the equation for spinnability it is clear that the important parameters for spinnability (m_1, m_2, m_3 and m_5) are effectively the only ones used in the GP generated equation for spinnability. The same can be said for the prediction of yarn strength, although it is less pronounced (some other parameters not included in the GP equation also have considerable values).

The equation for tenacity can be reduced (with loss of accuracy) to the following set of three rules (Table 6):

Table 5
Meaningful selections for each input parameter

	m_1	m_2	m_3	m_4	m_5	f_1	f_2	f_3	f_4	f_5
Spinnability	**7**	**9**	**9**	3	**8**	2	2	2	1	1
Strength	**10**	9	6	8	6	4	2	**9**	2	0

All FECS rules, except for rule 4, have an immediate equivalent in the above GP equations derived rules.

The equation for spinnability can be rewritten as

$$\mathrm{Sp} = S(\mathrm{Max}(\mathrm{Max}(\mathrm{Max}(m_1,\ m_3 + m_2), S(m_1)) \times (-4.6/m_5)), m_3 - m_5).$$

From which follows (Table 7):

Table 6
Rule reduction of tenacity equation

f_3	M_1	Tenacity	FECS rules nr.
0	0#	0	5, 7, 8 and 10
0	10	1	(4)
1	##	2	1, 2, 3, 6 and 9

Table 7
Rule reduction of spinnability equation

m_1	m_2	m_3	m_5	Spinnability	FECS rules nr.
##	##	1	1	0	3, 4, 6, 10
##	##	1	0	1	5, 7, (2)
#1 or 1#	##	0	#	1	1, 8, 9, (2)

Again all FECS rules have a corresponding GP-derived rule. The GP-derived rules are however more general and therefore even better to convert in 'rules of thumb'. For example:

The first rule for yarn strength {0 0#: 0} reads 'If the fibre strength is low and the yarn count is low or medium then the resulting yarn strength will be low.'

The first rule for spinnability {## ## 1 1: 0} reads 'If the navel and rotor speed are high, yarn count and twist do not matter, the yarn is not spinnable.'

4.4. Conclusions

The following conclusions can be made with regard to this experiment:

- GP is somewhat less accurate (spinnability: 2%, yarn strength: 4%) when compared to the results obtained with a neural network (backpropagation learning rule).
- GP has about the same accuracy as FECS (within 1%).
- Although supplied with 10 input parameters, GP selected only a few parameters (four for spinnability and two for yarn strength) to reach good accuracy. This in contrast to the neural network which used (a minimum of) 17 input parameters.
- GP selected mostly discrete functions to predict spinnability (a discrete output value) while GP uses continuous functions when applied to the prediction of yarn strength (a continuous value).
- The selected parameters correspond to the FECS evaluation of the most important input parameters.
- From the GP equations, rules can be deducted which correspond with the rule sets generated by genetic based machine learning. These rules are however more general but also provide (compact) physical interpretable information.

5. General conclusions

This paper presented the third and last part of an overview/introduction to the domain of evolutionary computing and its major algorithms. While the first paper (Sette et al., 1996) introduced the general concepts of genetic algorithms, the second paper (Sette and Boullart (2000)) illustrated the use of genetic based machine learning. In this final paper, an introduction was given to the second field of study derived from genetic algorithms: Genetic programming as defined by Koza.

Genetic programming is basically a genetic algorithm applied to CP instead of simple numerical variables. The essential difference with genetic programming is therefore the representation of the individuals (computer programs) of a population. A suitable computer program representation is the so-called S-expressions defined in the programming language LISP. All genetic operators have been redefined in function of this new representation.

A simple symbolic regression served as a first (educational) problem case. Finally, genetic programming was applied to a much more complex problem: the search for mathematical equations within a database corresponding with an industrial production process. The generated mathematical equations allowed a correct overall prediction in all cases (prediction of spinnability and yarn strength) of at least 90%. Also, the resulting equations have a limited complexity (eliminating several input parameters) and give the user an insight into the importance of the database parameters. This corresponded closely with the results obtained using genetic based machine learning. Moreover, the GP equations could be converted into similar (but more general) rules as those generated by FECS.

References

Bonnarini, A., 1997. Evolutionary learning of fuzzy rules: competition and cooperation. In: Pedrycz, W. (Ed.), Fuzzy Modelling: Paradigms and Practice. Kluwer Academic Press, Norwell, MA.

Boullart, L., Sette, S., 1998. High performance learning classifier systems; proceedings EIS (Engineering of Intelligent Systems), Vol 1, Fuzzy Logic/Genetic Algorithms, pp. 249–256.

BRITE/EURAM project BREU 00052-TT, 1990–1993. Research for a mathematical and rule based system which allows to optimise a cotton mixture, based on the interdependence of significant fibre properties, process parameters, yarn properties and spinning machinery performances.

Holland, J.H., 1968. Hierarchical description of universal spaces and adaptive systems, Technical Report ORA Projects 01252 and 0826, Ann abor, University of Michigan, Department of Computer Science & Comm. Science.

Holland, J.H., 1973. Genetic algorithms and the optimal allocation of trials. SIAM Journal of Computing 2 (2), 88–105.

Holland, J.H., Reitman, J.S., 1978. Cognitive systems based on adaptive algorithms. In: Waterman, D.A., Hayes-Roth, F. (Eds.),

Pattern Directed Inference Systems. Academic Press, New York, pp. 313–329.

Koza, J.R., 1992. Genetic Programming, On the Programming of Computers by Means of Natural Selection; MIT Press, Cambridge, MA, ISBN 0-262-11170-5.

Pynckels, F., Sette, S., Van Langenhove, L., Kiekens, P., Impe, K., 1995. Use of neural nets for determining the spinnability of fibres. Journal of the Textile Institute 86(3) 95, 425–437.

Pynckels, F., Sette, S., Van Langenhove, L., Kiekens, P., Impe, K., 1997. Use of neural nets to simulate the spinning process. The Journal of The Textile Institute 88(Part 1) (4), 440–447.

Sette, S., 1998. Lerende systemen door middel van evolutionaire algoritmen (Learning systems by means of evolutionary algorithms). Ph.D. Thesis (in Dutch), University Ghent, Faculty of Applied Sciences, Department of Textiles.

Sette, S., Boullart, L., 2000. An implementation of genetic algorithms for rule based machine learning. Engineering Applications of Artificial Intelligence 13 (4), 381–390.

Sette, S., Boullart, L., Van Langenhove, L., 1996. Optimising a production process by a neural network/genetic algorithm approach. Engineering Applications of Artificial Intelligence 9 (6), 681–689.

Sette, S., Boullart, L., Van Langenhove, L., 2000. Building a rule set for the fibre-to-yarn production process by means of soft computing techniques. Textile Research Journal 70 (5), 375–386.

Steele, G.L., 1984. Common LISP. Digital Press; ISBN 0-932376-41-X.

Wilson, S.W., 1994. ZCS: a zeroth order classifier system. Evolutionary Computation 2, 1–18.

Stefan Sette is a doctoral scientific researcher, and has been working at the Department of Textiles, Ghent University, since 1990. He holds university degrees in theoretical physics (1987), computer science (1989) and received his Ph.D. in 1998. His research concentrated first on (conventional) image analysis applied to textile problems. During the last six years this has evolved into a study of neural networks, genetic algorithms en learning classifier systems, supported by the Department of Control Engineering and Automation. He is currently researching implementations of these models for a paper mill industrial process.

Luc Boullart graduated in Electromechanical Engineering at the Ghent University (Belgium) in 1970, where he also received his Ph.D. in 1975. Currently, he is full professor at the Control Engineering and Automation department of the same university, where he teaches computer science, systems simulation and computer systems in process control. His research interest is in the field of artificial intelligence, more especially in neural networks and genetic algorithms. He is also honorary president of the Belgian Institute for Automatic Control (BIRA), and is chairman of IFAC's committee on Computer Aided Control System Design. Furthermore, he is director of the Institute for Permanent Education of the engineering department of the Ghent University. He is a board member of the Ghent University Senate.