Genetic Programming in Data Modelling

Halina Kwasnicka[1] and Ewa Szpunar-Huk[2]

[1] Institute of Applied Informatics, Wroclaw University of Technology, Wyb. Wyspianskiego 27, 50-370 Wroclaw, Poland halina.kwasnicka@pwr.wroc.pl
[2] Institute of Applied Informatics, Wroclaw University of Technology, Wyb. Wyspianskiego 27, 50-370 Wroclaw, Poland ewa.szpunar-huk@pwr.wroc.pl

1 Introduction

Evolutionary Computation paradigm becomes very popular. A number of approaches that exploit the biological evolution ability to adaptation arise. Between them we can distinguish Genetic Algorithms, Evolutionary Strategies and Evolutionary Programming. Genetic Programming (GP) proposed by Koza use the same idea as Genetic Algorithm (GA) and it is perceived as a part of GA. The main difference is that GP use different coding of potential solutions, what forces different genetic operators such as crossover and mutation. The main idea of Evolutionary Computation is to imitate natural selection, that is the mechanism that relates chromosomes with the efficiency of the entity they represent, thus allowing those efficient organisms, which are well-adapted to the environment, to reproduce more often than those which are not.

Genetic Programming is strongly developed in different places – universities, laboratories etc. The web http://www.genetic-programming.org/ is a reach source of knowledge dedicated to GP. The readers can find there useful both theoretical and practical information. Genetic Programming is very popular everywhere, where the algorithm to solve a considered problem is not known, because GP starts from a high-level statement of "what needs to be done". A computer program solving the problem is evolved, what means – it is developed automatically.

The chapter concerns the problem connected with making a great amount of collected in different places (firms, consortia, etc.) data useful. Discovering knowledge from possessed data and making it useful is very important especially for managers.

The chapter, beside Introduction and Summary, consists of the three main sections. Section 2 gives introduction and the example of using Genetic Programming for mathematical modelling task. Section 3 details the Genetic Programming used for classification task, where of course, we can use some rules induction and decision trees generation algorithms, but as we can see, in

same cases Genetic Programming can reveal some advantages. Other important problems are connected with prediction and time series modelling. GP can also be used for these tasks. The third main section (Section 4) presents how GP can be used for such problems. Some conclusions are placed in the last section – Summary (Section 5).

2 Genetic Programming in mathematical modelling

Mathematical modelling can be regarded as a search for equations that provide a good fit to numerical data. It is used in variety of problems like scientific law discovery or prediction tasks in fields where the theory does not give definite answers about the outcome of experiments. The search process however using traditional methods, for example polynomial regression, is often difficult, because in these methods a specific model form must be assumed, what demands strong theoretical knowledge. Usually in case of regression only the complexity of this model can be varied so the search task is often limited to finding a set of coefficients. GP allows searching for a good model in a different, more "intelligent" way, and can be used to solve highly complex, non-linear, probably chaotic or currently not fully or properly understood problems [1].

Induction of mathematical expressions on data using GP is introduced by Koza and is called symbolic regression [2], to emphasise the fact that the object of search is a symbolic description of a model, not just a set of coefficients in a prespecified model. Koza showed that GP could perform well in function identification tasks, because it searches the space of all possible models for both a good model form and the appropriate numeric coefficients for the model. The search space is constrained only by the available model pieces (variables, functions, binary operators, syntax rules). Furthermore, GP is not obligated to include all input variables in the equation, so it can perform a kind of dimensionality reduction.

2.1 Adaptation of GP to mathematical modelling

Genetic programming can be successfully used to find a mathematical data model because it can find the shape of the equation as well as the composition of primitive functions, input variables and values of coefficients at the same time, without any additional information except the performance of the analysed expression on the data. But the key issue to make GP algorithm work is a proper representation of a problem, because GP algorithms directly manipulate the coded representation of equation, which can severely limit the space of possible models. An especially convenient way to create and manipulate the compositions of functions and terminals are the symbolic expressions (S-expressions) of the LISP programming language. In such a language the mathematical notation is not written in standard way, but in prefix notation. It allows building expression trees out of these strings, that can be easily

evaluated then. Some examples of expressions written in algebraic and prefix notation with corresponding trees are shown in Fig. 1.

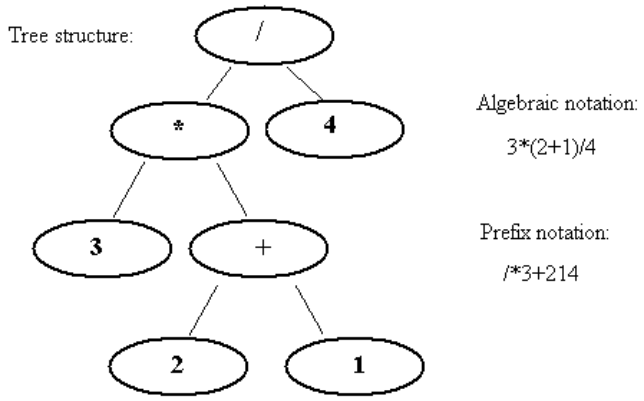

Fig. 1. An examplary tree in GP and the coded expression

In the Genetic Programming paradigm, the individuals in the population are compositions of functions and terminals appropriate to the particular problem's domain. Functions and terminals are adequately nodes and leaves of the trees being evolved by the GP. Therefore, before running GP to search for an equation, it is necessary to define operators dedicated to the considered task. For the mathematical modelling problem the four binary arithmetic functions (+, -, *, /) are usually used, but other specific functions (e.g., trigonometric functions, sqrt, abs) can also be useful. The terminals or leaves are either variables or coefficients. Variables correspond to the data being analysed, while coefficients are defined by numerical ranges.

Using arithmetic functions as operators in GP has some drawbacks. One of them is that the closure property of GP requires, that each of the base functions is able to accept, as its arguments, any value that might possibly be returned by any base function and any value that may possibly be assumed by any terminal. This condition is not satisfied by e.g., division operator. That is why a protected division function $div(a, b)$ is usually defined which returns 1 in two cases: when $a = b$ and when $b = 0$. Otherwise it returns a result of division a/b. Similar protections must be established for many other functions as e.g., $log(a)$ or $sqrt(a)$. The problem with closure property is entirely eliminated in Strongly Type Genetic Programming (STGP) [4], in which it is assumed that each function specifies precisely the data types of its arguments and its returned values, but on the other hand, additional modifications of the searching process are required, like creating more individuals in each generation in order to better explore the search space.

During the evaluation of GP solution some numerical problems may also easily occur, like underflows or overflows caused by extremely small or high values or major precision loss, e.g., determining a trigonometric function such as $sin(a)$ for some value a much larger than 2Π [5]. It is also assumed that the creation of floating-point constants is necessary to do symbolic regression and it seems to be a weak point, because coefficient must be assembled from random values, so destination functions containing constants are usually much more demanding to GP than others. Designing GP algorithm to mathematical modelling it is necessary to consider most of these problems.

2.2 An educational example – the hybrid of Genetic Programming and Genetic Algorithm

In this section we present a detailed description of a simple GP algorithm to mathematical model induction. The work was done as a part of wider study of Szpunar-Huk, described in [24]. The goal of the designed algorithm is to find a function $y = f(x_1, ..., x_n)$, which reflects potentially existing relationship between input and output values. As the input data we have a sample set $S = \{x_1, ..., x_n, y\}$, where $x_1, ..., x_n$ are independent input values and y is an output value (assumed function of x_i). Values $x_1, ..., x_n$ and y are continuous numbers converted to floating point numbers with a specified for an individual experiment precision. The algorithm is tested to examine its ability to do symbolic regression on sample sets of different size and level of noise, generated using predefined mathematical functions, and for different evolution control parameters (e.g., population size, probabilities of crossover and mutation). During experiments generations' numbers with standard deviations needed to find solutions and obtained errors were analysed.

Firstly it is important to define terminals and functions sets used for approximating the test functions. Let us assume:

- The terminal set $T = \{x_1, ..., x_n, R\}$ – the set of possible values of leaves of trees being evolved contains input variables' labels and continuous constants between 1.0 and 100.0.
- The set of available base functions $F = \{+, -, *, /\}$ include just addition $(+)$, subtraction $(-)$, multiplication $(*)$ and division operator $(/)$.

It is important to notice, that in this case the division operator is not protected, so the closure property of GP is not satisfied. This problem is solved by a proper definition of the fitness function. All the base functions have two arguments what significantly simplifies crossover operation.

A fitness function measures how well each individual (expression) in the population performs the considered problem (task). In described experiments two fitness functions were used. The first one was the sum of errors made by an individual on a sample set and could be written as:

$$F = \sum_{i=1}^{N} |ex(x^i) - y^i| \tag{1}$$

where N is a number of samples in the set S, $ex(x^i)$ is a value returned by an expression ex (encoded expression by the considered individual) for the i^{th} sample, y^i is a value of output y for i^{th} sample.

The second function was the average percentage errors made by an individual on a sample set and could be written as:

$$F = \frac{1}{N} * \sum_{i=1}^{N} \frac{100 * |ex(x^i) - y^i|}{ex(x^i)} \tag{2}$$

where as before, N is a number of samples in the set S, $ex(x^i)$ is a value returned by an expression ex for the i^{th} sample, y^i is a value of output y for i^{th} sample.

For both functions, the closer the sums are to zero, the better the expressions (solutions) are. Additionally, when during evaluation of a given expression on any sample from a sample set, division by zero occurs, the fitness function for the individual is set to "undefined". It is also worth mention, that in case of the first function, when the values returned by the sample are close to zero, it is difficult to distinguish between good and bad individuals, what can significantly affect GP performance, but due to the specific of chosen data it was not important in the described experiments.

In the selection phase couples of individuals for crossover are chosen. As a selection method here is proposed the method based on elite selection. Firstly all individuals are ranking according to their fitness (expressions with "undefined" value are placed on the end of the list). Secondly a number of the best different individuals (the exact number is set by an appropriate parameter – usually it is about 10% of population size) are taken to the next generation without any changes. A reason for this is that in GP during crossover phase expressions cannot evolve well because they are frequently destroyed and it is easy to loose the best ones, what can result in poor performance. Selected individuals must be also different to avoid premature convergence of the algorithm. The remaining couples for the crossover phase are generated as follows: the first individual is the next individual from the ranking list (starting from the best one), the second one is randomly chosen from the whole population, not excluding expressions with "undefined" value.

After selection a number of couples (according to a given probability parameter) are copied to the next generation unchanged, the rest individuals undergo the crossover – a random point is chosen in each individual and these nodes (along with their entire sub-trees) are swapped with each other, creating two entirely new individuals, which are placed in the new generation. In the classical GP, each tree could be of potentially any size or shape, but from the implementation point of view, every new tree can not be bigger then the predefined maximum depth, so if too deep tree is generated during crossover operation, it is being cut to the maximum allowed size. Here, in the recombination operation another parameter is also introduced. The parameter specifies probability of choosing a leaf as a place of crossing, because

choosing a random point can result in fast shortening of all individuals in the population.

Mutation is an additional included genetic operator. During mutation operation nodes or leafs are randomly changed with a very little probability, but over a constraint, that the type of node remains unchanged: a terminal is replaced by an another terminal and a function by an another function. It is difficult to estimate the real size of a population mutation since, during reproduction phase – when maximum depth of tree is established – trees are often cut, what also can be viewed as a kind of mutation.

An initial population is randomly generated, but creating a tree with maximum depth is more likely to come than creating a tree with only one or just few nodes.

Experiments

To validate described method, some numerical experiments were conducted. In this case some sample sets were generated according to a given function f, the algorithm was looking for this function, but a sample could contain more variables than it was used by the given function. The algorithm turned out to be a good tool to exactly discover plenty of functions. Table 1 shows some examples of mathematical expressions, the algorithm managed to find in generated data, and the average number of needed generations of the algorithm. During this experiments the number of input variables' labels in the sample set and the number of different input variables in the equation where equal. In a number of experiments the vector of input variables x_i contains

Table 1. Sample of test functions and number of generations needed to find that function by the algorithm

Function	Number of generations
$y = x_1x_1x_1$	3
$y = x_3 + x_2x_1 - x_4$	8
$y = x_1/(x_2 + x_3)$	16
$y = x_1 + x_1x_1 - x_2/x_3$	22
$y = x_1 + x_2 + x_3 + x_4 + x_5 + x_8 + x_7 + x_8$	31
$y = x_1 + x_2/x_3 - x_3/x_1$	45
$y = x_3x_2/(x_5 + x_4) - x_4$	103
$y = x_1 + x_2/x_3 + x_3x_4 - (x_3 + x_3)/x_2$	135
$y = 10x_1 + 5x_2$	137
$y = x_1 + x_2 + x_3x_3 - x_1x_1x_1/(x_2x_2)$	207
$y = x_1 + x_2 + x_3 + x_4 + x_5 + x_8 + x_7 + x_8 + x_2x_3x_4$	375
$y = x_1x_1x_1 + x_2x_2x_2 + x_3x_3x_3 + x_4x_4x_4$	579

a variables that are not taken into account during output calculation, they are additional variables. It means that our algorithm looks for the function

$y = f(x_1, ..., x_{10})$ but the proper dependency is $y = f(x_1, x_2)$ – only two input variables influence the output variable. In such cases the algorithm could also find the relationship, but in longer time. A number of redundant variables we denote by N_{add}, $T_{solution}$ means an average number of generations for finding solution, but δ_{add} – a standard deviation of $T_{solution}$. Exemplary results are shown in table 2. It is important to mention that the algorithm requires

Table 2. The result given by GP+GA for different number of additional variables

N_{add}	$T_{solution}$	δ_{add}
0	8.71	1.9
1	17.4	5.91
2	19.7	5.08
3	20.7	6.75
4	29.7	12.4
5	37.4	14.2
6	42.1	30.6

various control parameters to proper working. Some of them are mentioned above. The most important of them with their presumed values are listed below. They are suggested values, often used in the experiments, but in some tasks their values must be carefully tuned.
A list of control parameters of GP used in the experiments:

- Population size: 300-400
- Maximum depth of the tree: 6-8
- Percent of individuals moved unchanged to the next generation: 10-30%
- Crossover probability: 0.85
- Probability of choosing leaf as a crossing point: 0..1
- Mutation probability: 0.01.
- Number of samples: 100-1000

Tuning of the algorithm

The algorithm in such a form is able to find hidden model in a sample set even with noisy data – the output was generated not exactly from a given function, but outputs were in 5%, 10% or even more noisy. Difficulties however appeared when a tested function included coefficients, especially with real valued coefficients. It is not possible to find relationships, even if it seems to be easy (e.g., $y = 15x_1 + 4x_2 - 23.4x_3/x_2$). Enlarging the initial population size from 500 to 5000 individuals can partially solve this problem, but for plenty of functions it is not sufficient.

To make the algorithm more effective an additional algorithm based on the classic Genetic Algorithm is included. The aim of this algorithm is to find, for a given set of best individuals, after a given number of generations in

GP, as good coefficients as possible. That is why the algorithm is cold Tuning Algorithm (TA).

The tuning algorithm works as follows: for the given individual taken from the GP algorithm a number of coefficients is established. This number defines a length of individuals in the TA, where an individual is a sequence of numbers from the given range and with a given precision. An initial population of the TA contains sequences generated on an expression taken from the GP algorithm and random sequences – one hundred individuals altogether. Generated in such a way population is evolved in the standard way: individuals are selected, crossed over and mutated. The fitness function and the selection operation are the same as in the GP algorithm, but the crossover operation is much simpler: for a given couple of individuals a random point in constants sequence is chosen (the same for both individuals) and a part of one individual is swapped with a part of the second one, what is shown in Fig. 2. Mutation however is much more important here than in GP algorithm. Its aim is to make it possible to tune the coefficients. A special kind of mutation is proposed. Each real number is a set of Arabic numeral, e.g., 13.86 consists of digits 1, 3, 8 and 6. The probabilities of mutation of particular digits are different – the lowest is for the digit placed on the left (the most meaningful) and systematically is increased for the next digits (going to the right). The

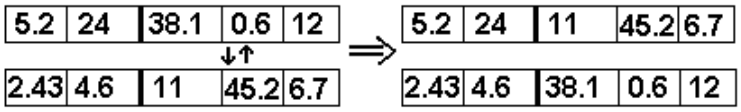

Fig. 2. An example of crossover in TA

scheme of the whole algorithm (GP connected with TA) is shown in Fig. 3. The tuning process is performed every N generations of GP for S different, best individuals from an actual generation of the GP. Incorporating of a GA allows approximation of relationships previously impossible to find. For example, when the dataset was generated using equation 3, the GP has problem with finding the exact coefficients.

$$y = 11.3/x_2 + 10x_3x_3 - 5.2x_2/(x_3 + x_5) + 33.8x_3x_3x_3x_5x_6x_6 + \\ +20(x_4 + x_2)/x_5 + 8x_5x_5x_5 + 12(x_3 + x_5)/x_2 + x_3x_3x_3 \quad (3)$$

Using the tuning algorithm worked every 40 generations of the GP on 15 best individuals after 100 generation of GP, the average percentage error of the best individual on dataset was equal to 1.2% while without tuning only 9.25%.

Input: X, Y
Output: Expression y'$\approx f(\mathbf{X})$
1. Create an initial GP population and set GN (number of generations) to 0
2. Evaluate the fitness of the population members
3. While termination criteria not met do
4. If $GN mod N = 0$ perform tuning
5. Take S individuals for tuning
6. For each of S chosen individuals do
7. Create an initial GA population
8. Evaluate the fitness of the population members
9. While termination criteria not met do
10. Probabilistically apply crossover operator
11. Probabilistically apply mutation operator
12. Go to step 8
13. Probabilistically apply crossover operator
14. Probabilistically apply mutation operator
15. Increase GN
16. Go to step 2
17. End.

Fig. 3. The scheme of the GP in conjunction with GA used as TA algorithms

2.3 General remarks

In the presented example abilities of GP algorithm to discover mathematical model on artificially generated data where shown, but approximation a function using a given finite sample set of values of independent variables and associated values of dependent variables is a very important practical problem. In practice, the observed data may be noisy and there may be no known way to express the relationships involved in a precise way. That is why GP is widely used to solve the problem of discovering empirical relationships from real observed data. To show, how wide variety of domains GP is successfully applied some of experiments with GP on real data are mentioned below.

Koza [3] showed, how the Genetic Programming paradigm can be used to create an econometric model by rediscovering the well-known non-linear econometric "exchange equation" $M = PQ/V$, which relates the money supply M, price level P, gross national product Q, and the velocity of money V in economy. He also showed how Kepler's Third Law can be rediscovered. Sugimoto, Kikuchi and Tomita were able, by applying some improvements to GP, to predict an equation from time course data without any knowledge concerning the equation. They predicted biochemical reactions and the relative square error of predicted and given time-course data were decreased, due to GP, from 25.4% to 0.744% [6]. Langdon and Barrett have used Genetic Programming (GP) to automatically create interpretable predictive models of a small number of very complex biological interactions, which are of great interest to medicinal and computational chemists, who search for new drug

treatments. Particularly, they have found a simple predictive mathematical model of human oral bioavailability [7]. Makarov and Metiu use GP to the analytic solutions of the time-independent Schrodinger equation. They tested their method for a one-dimensional enharmonic well, a double well, and a two-dimensional enharmonic oscillator [8]. Diplock, Openshaw i Turton used GP as a tool for creating new models of complex geographical system. They run parallel GP algorithm on Cray T3D 512 supercomputer to create new types of well performing mathematical model [9].

It is important to notice that in scientific discovery however, obtaining some adequate fit to data is not enough. To fully incorporate some results in the body of scientific work, it is necessary that some form of understanding about the expressions that are induced is achieved. The expressions thus need some further justification before they can be used as models of the phenomenon under study. The resulting equations can then be tested with statistical methods to examine their ability to predict the phenomenon on unseen data.

3 Decision models for classification tasks

Classification task is the one of fundamental and the most studied tasks in the area of data mining. Given a set of predetermined, disjoint target classes $C_1, C_2, ..., C_n$, a set of input attributes $A_1, A_2, ..., A_m$, and a set of training data S with each instance associated with a unique target class label, the objective of this task is to find mutual properties among a set of objects in data set which can be used to determine the target category for new unseen data given its input attributes' values. In the other words, this process finds a kind of decision models – a set of classification rules (or procedures in some cases). Classification rules are commonly expressed in the form of IF-THEN rules, where the IF part of a rule contains a logical combination of conditions on the attributes and the THEN part points to a class label. The popular decision model is also a decision tree, in which the leaves are the class labels while the internal nodes contain attribute-based tests and they have one branch emanating for each possible outcome of the test.

Most rules induction (e.g., CN2) and decision trees generation algorithms (e.g., CART and C4.5) perform local, greedy search to generate classification rules, that are relatively accurate and understandable, but often more complex than necessary. There are several properties of GP, which make them more convenient for application in data mining tasks comparing to other techniques. The reason is that the local, greedy search selects only one attribute at a time, and therefore, the feature space is approximated by a set of hypercubes. In real-world applications, the feature space is often very complex and a large set of such hypercubes might be needed to approximate the class boundaries. GP algorithms perform global search. That is why it copes well with attributes interaction problem, by manipulating and applying genetic operators on the functions. Additionally some data mining algorithms work only with qualita-

tive attributes, so numerical values must be divided into categories. GP can deal with any combination of attribute types. Moreover, GP algorithms have ability to work on large and "'noisy" datasets. They have the high degree of autonomy that makes it possible to discover of knowledge previously unknown by the user.

3.1 Adaptation of GP to clasification task

Classification task is similar to the regression, and regression could also be used to solve some classification problems but generally, building decision models emphasis on discovering high-level, comprehensible classification rules rather than just producing a numeral value by a GP tree. Application of GP in this field is similar, but due to some differences, causes additional problems.

Problems with closure property

The main difference between using GP in classification and regression tasks is that in classification the target class (the goal attribute) is nominal while in regression – continuos one. When data set contains mixture of real-valued and nominal attribute values, different attributes are associated with different functions, so the closure property is usually not satisfied by the standard GP algorithm and the closure problem is much more complicated. One idea to solve the problem is to use logical operators for all inner nodes, while leaves have only boolean values or are represented as functions that return Boolean values. Other approaches to go around the closure requirements are STGP, mentioned in section 2.1 [4, 10] (sometimes called constrained-syntax GP), and its variant called Grammar-Based Genetic Programming (GBGP) [10]. The key idea of STGP is such, that for each function f from a set of functions, the user specifies which terminals/functions can be used as a children node containing function f. In GBGP syntactic and semantic constraints are specified by a grammar [10, 14], so the important preparation step for the GP is to identify a suitable target language, in which to evolve programs. It should be underlined, that the success of evolution is dependent upon the use of a language, that can adequately represent a solution to the problem being solved.

Michigan – Pittsburgh approaches

The next step designing a GP for discovering decision model, after choosing a proper language, is to choose individual representation. There are two main approaches to use GP algorithms to classification rule discovery, based on how rules are encoded in the population of individuals: the Michigan [11] and the Pittsburgh approach [11]. In the Michigan approach each individual encodes a single classification rule, whereas in the Pittsburgh approach each individual encodes a set of classification rules. The choice of individual representation determines the run of GP algorithm. The Michigan is best suitable for binary

classification problem, where there are only two classes to distinguish and one individual is entire solution for the problem. The problem gets more complicated when there are more then two classes in the classification task but this approach is also used in such problems. The most common solution for this in the literature is to run the GP k times, where k is the number of classes [12]. In the i^{th} $(i = 1, ..., k)$ run, the GP discovers rules for the i^{th} class. When GP is searching for rules for a given class, all other classes are merged into a large class. In the Pittsburgh approach GP can work with a population of individuals where each individual corresponds to a rules set, constituting a complete solution for the classification problem, i.e. entire decision tree [11].

Fitness function defining

Discovering useful decision models desiring goal is to obtain rules, which are both accurate and comprehensible. The fitness function evaluates the quality of each rule. Majority of fitness functions used in GP grew out of basic concepts on classification rule evaluation. They usually use TP, FP, TN and FN factors which, for a given rule of the form ***IF condition (A) THEN consequent (B)***, denote respectively:

- TP (True Positives) – a number of examples satisfying both A and B,
- FP (False Positives) – a number of examples not satisfying A but satisfying B,
- FN (False Negatives) – a number of examples not satisfying both A and B,
- TN (True Negativess) – a number of examples satisfying A but not B.

To rely on these factors several formulas used in fitness function have been described in the literature [14], i.e.,

- confidence: $(co = TP/(TP + FP)$
- consistency $(cs = TP/(TP + TN)$
- sensitivity $(Se = TP/(TP + FN))$
- specificity $(Sp = TN/(TN + FP)$

A fitness function usually contains combination of such formulas, e.g., $fitness = Se * Sp$ [19]. Selection of factors appearing in the fitness function clearly depends on the type of the task and desired characteristics of the discovered solution. In the Pittsburgh approach, where one individual is the entire solution, the most popular method to evaluate solutions is accuracy rate, which is simply the ratio of the number of correctly classified test examples over the total number of test examples.

Adjusting properly the fitness function can also addressed the problem of the comprehensibility of the discovered rules. Beside the accuracy of the classification rules on the data set, the function can also take into account a size of the tree, penalising the trees with the high number of nodes. Noda et al. introduced additionally a rule "interestingness" term in the fitness function in order to discover interesting rules [15].

3.2 An example of classification rules discovering using GP

This section contains a description of a successful application of GP to a classification task. As an example we propose research of Ross, Fueten and Yashir [16]. They used GP to solve the problem of identification of microscopic-sized minerals involved in rocks. Mineral identification is a task, that usually done manually by an expert, who uses various visual cues like colour, texture and the interaction of the crystal lattice with different directions of polarised light. It is worth to stress, that proper identification by a human is time-consuming and requires years of experience.

GP algorithm is used by Ross et al. to develop a decision tree from database, which contain 35 colour and texture parameters of mineral specimens' feature. This database was used also to manual mineral identification. These parameters were obtained by multiple-stage process of mineral sample images analysis. Their approach is a Michigan approach, so the algorithm produces one decision tree specialised for identifying a specific mineral. Such a tree, given a sample of data for a grain, can determine, if the grain corresponds to the mineral, the tree is engineered to recognise. Thus decision model consists of several decision trees, one for each mineral species.

3.3 Used algorithm

To represent solution two languages were proposed, so it is a Grammar-Based Genetic Programming algorithm. The Backus-Naur grammar for the languages is as below:

```
L1: Bool    ::=if Param < Param then Bool
                |Param<Param |True|False
    Param   ::=p1|p2|...|p35|<ephemeral const>
L2: Bool    ::=if Bool then Bool else Bool|Bool and Bool
                |Bool or Bool|not Bool|E BoolRel E|True
                |False
    BoolRel ::=<|<=|>|>=|=
    E       ::=E Op E|max(E1,E2)|min(E1,E2)|Param
    Op      :=+|-|/|*
    Param   ::=p1|p2|...|p35<ephemeral const>
```

The first language **L1** consists only of ***If-Then*** statements, a relational operator "$<$" over the 34 parameter values and the logical constants *True* and *False*, while **L2** is more complex and introduces mathematical expressions over parameter values. Ephemeral constants in **L2** are floating-point numbers between 0.0 and 1.0 and relational "$<$" statement work with a degree of error within 10% of the left-hand parameter. For example, in the expression "a=b", if b is within 10% of value of a, the expression returns *True*. Additionally, for **L2** language it was necessary to modify division operator "/" to satisfy closure property. Thus this operator returns "0", if division by zero appears. The

individual coded in both languages return *True*, if the analysed grain parameters conform to the particular mineral, that the tree is designed to identify, and *False* otherwise.

The data set for evolution and experiments contained 11 mineral species and varied number of examples of each mineral: quartz (575), K-spar (369), plag (358), biotite (74), muscovite (36), hornblende (86), CPX (81), olivine (685), garnet (311), calcite (520) and opaque (189). The data set was prepared with participation of geologist. For a given mineral selected for one evolution process, training and testing data are chosen from the entire data set. The training data set consists of a random selection of K samples of mineral being identified and a random selection of K samples for each of the other minerals, so the total number of training examples is N^*K, where N is the number of mineral species.

The fitness function used during evolution for a distinguished mineral M is the following:

$$Fitness = \frac{1}{2}K\left(\frac{Hits}{K} + \frac{Miss}{(N-1)}\right) \tag{4}$$

where $Hits$ is the hits correct number – the number of correctly identified grains of mineral M and $Miss$ is the misses correct number – the number of grains correctly identified as not being mineral M, K – a number of samples of M in the training set.

Experiments were carried out for two kinds of training sets. The first one contained 34 positive examples and 340 negative examples per run, in the second one the number of positives examples were doubled to 70. For experiments of the second type four of the eleven minerals (biotite, muscovite, hornblende and CPX) had to be excluded, because they provided insufficient numbers of positive examples in the grain database to permit accurate training and testing. The testing set was the rest of the database not used for training. The testing formula was similar to the fitness formula:

$$Performance = \frac{1}{2}\left(\frac{Hits}{P} + \frac{Miss}{Q}\right)100\% \tag{5}$$

where P is the total number of grains for mineral M in the testing set, and Q is the total number of other grains. Table 3 shows obtained results of GP runs and, for comparison, the results of testing performance for the same data by neural network (NN) proposed by Thompson et al. [17], trained to identify all the minerals with one single network.

Experiments showed that GP was successfully applied to mineral classification task. The overall performance of the best obtained mineral identifiers ranged from 86% to 98%. The authors noticed that the language **L2**, which is more complex than **L1**, did not lend any great advantages to the quality of solutions but is computationally more expensive due to arithmetic operators. Solutions obtained using more positive grain examples was normally advantageous to the average quality of solutions, but in most of the cases for **L1**, the

Table 3. Obtained result in Minerals classification [16]

		L1				**L2**				**NN**
K		34		70		34		70		74
Mineral		Test	Train	Test	Train	Test	Train	Test	Train	Test
Quartz	Best:	90.2	94.6	92.2	93.8	93.5	97.5	97.8	96.1	96.5
	Av.:	84.7	90.9	85.7	88.9	88.3	93.7	90.1	92.5	
K-spar	Best:	94.2	97.8	93.9	96.0	93.8	99.0	93.4	96.9	88.6
	Av.:	89.7	95.9	91.0	93.5	90.7	97.8	90.8	93.0	
Plag	Best:	86.2	91.8	84.7	93.8	83.3	95.7	95.4	94.3	89.1
	Av.:	75.8	88.0	78.9	88.9	78.6	91.8	80.2	93.0	
Biotite	Best:	98.1	98.1	–	–	96.6	99.4	–	–	97.3
	Av.:	93.7	96.4			92.7	98.0			
Muscovite	Best:	98.3	6.6	–	–	97.6	99.9	–	–	–
	Av.:	74.8	94.1			78.0	98.0			
Homblende	Best:	91.4	97.7	–	–	92.8	98.5	–	–	95.4
	Av.:	86,8	94.7			86.1	95.8			
CPX	Best:	90.3	96.5	–	–	93.8	98.5	–	–	97.5
	Av.:	81.0	88.1			84.9	92.8			
Olivine	Best:	90.0	94.4	89.9	93.8	91.4	96.6	93.2	98.1	92.8
	Av.:	83.0	88.6	85.6	90.0	84.4	92.6	89.4	95.1	
Garnet	Best:	98.2	99.7	99.6	99.6	97.8	100	99.4	99.8	97.4
	Av.:	96.0	98.3	97.6	98.1	95.8	98.9	97.1	98.6	
Calcite	Best:	92.1	97.4	91.8	95.2	92.8	99.0	95.1	98.2	96.0
	Av.:	84.4	93.6	88.5	93.0	86.2	98.3	91.2	95.9	
Opaque	Best:	97.7	99.9	97.7	99.9	98.3	100	98.1	100	95.2
	Av.:	93.6	98.9	96.6	99.4	95.2	99.3	96.5	99.7	

solutions for $K = 70$ were not significantly better than those for $K = 34$ and often suffered from over-training effects. The GP performance is comparable to that obtained with a performance of NN, but obtained models are not a "black-box" and are quite short and possible to understand by a human.

3.4 General remarks

Many companies collect vast amounts of data, however they fail to extract necessary information to support managerial decision-making. That is why classification is an important problem extensively studied in several research areas. Classification systems can find in databases meaningful relationships useful for supporting a decision making process or automatic identifying objects in medicine, astronomy, biology etc., that might take years to find with conventional techniques.

GP was used for deriving classification models from data especially in medical and financial domains. As an example the Gray's and Maxwell work on using GP to classify tumours based on H Nuclear Magnetic Resonance spectra

can be mentioned [12]. They proposed a new constrained-syntax Genetic Programming and reported, that the algorithm can classify such data well. Other example could be one of the newest researches made by Bojarczuk et. al., who proposed new constrained-syntax Genetic Programming algorithm for discovering classification rules [12]. They tested their algorithm on five medical data sets such as: Chest pain, Ljubljana breast cancer, Dermatology, Wisconsin breast cancer and Pediatric Adrenocortical Tumor and obtained good results with respect to predictive accuracy and rule comprehensibility, by comparison with C4.5 algorithm [18]. Another interesting work was presented by Bentley, who used GP to evolve fuzzy logic rules capable of detecting suspicious home insurance claims. He showed that GP is able to attain good accuracy and intelligibility levels for real, home insurance data, capable of classifying home insurance claims into "suspicious" and "non-suspicious" classes [19].

Usually databases considered in data mining (DM) tend to have gigabytes and terabytes of data. The evaluation of individual against the database is the most time consuming operation of the GP algorithm. But thanks to variety of possible modifications and parallel approaches to Genetic Algorithms, scalability of these algorithms can be achieved.

4 GP for prediction task and time series modelling

The amount of data stored in different databases continues to grow fast. This large amount of stored data potentially contains valuable, hidden knowledge. This knowledge could be probably used to improve the decision-making process. For instance, data about previous sales of a company may contain interesting relationships between products and customers, what can be very useful to increase the sales. In medical area, the probability of presence of particular disease in a new patient could be predicted by learning from historical data. In the industry field, reducing fabrication flaws of certain product can be achieved by processing the large quantity of data collected during the fabrication process. That is why analysing data to predict future outcomes is one of the most studied fields in data mining.

In prediction task the database often has a form of time series, since data are collected daily, and intuitively in many areas the order of data is of great importance. Although neural networks (NN) have been very successful in developing decision models, it is difficult to understand such models, since they are normally represented as a 'black box'. It causes that a Genetic Programming becomes popular in prediction task too.

4.1 Adaptation of GP to prediction task

In prediction tasks the goal is to predict the value (the class) of a user-specified goal attribute based on the values of other attributes. Classification models described before are often used as the prediction tool. Classification rules

can be considered as a particular kind of prediction rules, where the rule antecedent contains a combination – typically, a conjunction – of conditions on predicting attribute values, and the rule consequent contains a predicted value for the goal attribute. Therefore GP algorithms for prediction purpose are usually similar to those designed for classification tasks.

GP algorithms become more complicated when learning database contains time series. The simples modification is to allow in terminal set attribute values for not only one evaluated sample, but for some previous samples from a given window size. Achieving good results however often demands further improvements. To deal with specific nature of time series data special functions are added to language or grammar, according to which individuals are generated, to allow past values as input to the evolved program or equation. These can be simple arithmetic operations like an average or maximum value of attribute, which have an implicit time range, more complex like the function: *p(attr,num)*, representing the value of *attr* for the day *num* before the current day being evaluated, or some typical for a given problem including characteristic domain knowledge. GP algorithm to time series modelling can also remain the same as for classification task but only data set can be modified during preprocessing phase, for example by converting to a time series of period-to-period percent change of each attribute.

Desirable property of such a rule is its predicting power in assigning (predicting) the class of the new object. The prediction quality is verified on the test data. Sometimes test data set is given in advance, but when there is only one data set available some techniques for prediction testing are necessary. The most commonly one is *N-fold cross validation*. This method divides the data set to N mutually exclusive data sets of the same size. In each step of total N steps of the algorithm, one sub-data set is used for the testing, and the rest of them for learning process. In the learning part of the algorithm, the goal attribute values are available, while in the testing part these values are used for evaluation of the predicting ability of the rule.

4.2 Adaptation of GP to prediction tasks

Because one of the most interesting area for prediction application is the world of finance, as an example was chosen a system to stock market analysis proposed by Myszkowski [25]. His algorithm GESTIN (GEnetic STrategies of INvestment) was designed to learn a kind of decision model – an effective investment strategy (portfolio management).

The database that was used in experiments came from Warsaw Stock Exchange and consists of 27 major indexes of stock markets. The sample period for the indexes runs from April 14, 1991 until September 7, 2001. The original series are sampled at daily frequency. The sample periods correspond with 2147 observations. Database contains indexes for 223 join-stock companies.

The first modification introduced to GESTIN comparing with the standard GP algorithm is the form of an individual. An individual represents a decision

model, which consists from 2 trees. One tree determines, if given stocks should be bought, and the other, if stocks should be sold. Inner nodes in trees are logical operators: AND, OR and NOT. Outer nodes (terminals) however are not a simple values but complex expressions, which contain indexes' names (names of attributes from database), arithmetical operators and constants of different type. Table 4 shows names of indexes and syntax of contained them outer nodes. Fig. 4 shows an example of a decision tree. If the value of boolean

Table 4. Indexes in outer nodes

Name of index (NoI)	Syntax of leaf
INX, MA8, MA9, MA12, MA15, MA17, MA26, CK	NoI Opeartor Float+
	NoI Operator Float+ $*$ NoI
MACD8, MACD12	NoI Operator Long
WOL, WRB, WRO, WRN, AT	NoI Operator Long
EPS, FE, PE, ROA, ROE, RS, ROC, OBV	NoI Operator Float
EPSR, FER, RSI, MAN	NoI Operator $< -100..100 >$

expression coded in that tree is true, it means (depending on kind of the tree) to sell or to buy stock of a given company.

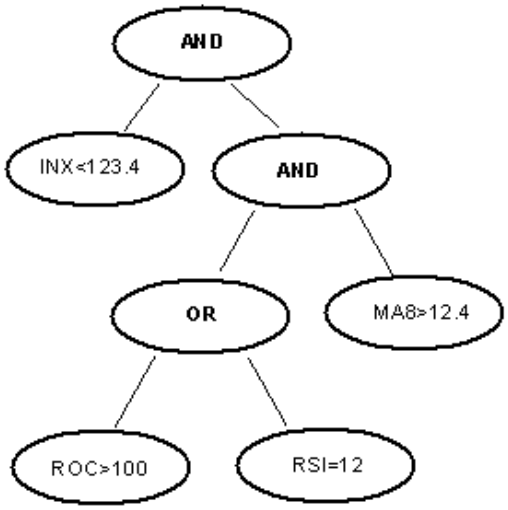

Fig. 4. An exemplary tree in GESTIN algorithm

The course of the algorithm (Fig. 5) is quite typical. Firstly a random population is generated and if termination criteria are not met, evaluation,

```
1.  Create an initial GP population
2.  While termination criteria not met do:
3.  Evaluate the fitness of the population members
4.    For each session from database do (from the oldest to the newest one):
5.      For each join-stock company C do:
6.        For each individual I do:
7.          If I possesses stock of company C and I wants to sell V:
8.            sell V
9.          If I doesn't possess stock of company C and I wants to by V:
10.           buy V
11.         Notice changes in individual's portfolio
12.         Sell all stock from portfolio of individual
13.   Make ranking of individuals
14.   Move N best individuals to the next generation unchanged
15.   Probabilistically apply crossover operator
16.   Probabilistically apply mutation operator
17.   Go to step 2.
18.   End.
```

Fig. 5. The scheme of GESTIN algorithm

selection, crossover and mutation operations on individuals are performed. Characteristic element of the evolution process is the evaluation phase, which needs closer insight. An individual is allowed to speculate on the stock exchange by buying and selling stocks securities sequentially starting with an empty portfolio for data from each session in database from the oldest to the newest session. Finally, all stocks are sold and ranking of individuals is performed according to gain or loss given strategy brought. As the speculation process in reality is very complex, some restriction and simplification are applied during the evaluation process. An individual can't buy stock of company he already possesses. The size of individual's portfolio is set in advance. Additionally, each individual has the same amount of ready cash (rc) at the beginning and maximal number of possible companies (nc) is set, but an individual can spend on stock of one company only rc/nc of cash it has at the beginning, to insert portfolio's diversification. To make speculation more realistic a commission of stockbroker parameter is added, which significantly affect the value of the fitness function.

It is not difficult to create strategy, which is very good for data, it was learn on. It is difficult to create a strategy earning regardless of time period. To check prediction ability of models obtained by GESTIN, some experiments were performed. Firstly 4 different time periods were chosen: one learning – T1 and tree testing – T2, T3 and T4. These periods are characterised by different WIG index values. The first one – the learning period – is a time of bull market (WIG=+13%), the second is a time of bear market (WIG=–28%), the third one is a mixed period of bull market and bear market (WIG=0%), and the last one is a bear market with elements of bull market. Index WIG is

the main index in Warsaw Stock Exchange, and it reflects results potentially obtained using the simplest strategy: "buy and keep" consisting in buying stock at the beginning and selling in the end. Results for using 10 different obtained strategies to the 4 time periods are shown in table 5.

Table 5. Results of 10 different strategies for one learning and three testing time periods of different value of WIG index

	T1	**T2**	**T3**	**T4**
WIG	+13%	–28%	+0%	–8.9%
Strategy 1	+23%	–15%	+0.8%	+6.8%
Strategy 2	+21%	–23%	–4%	+10%
Strategy 3	+21%	–23%	–4%	+4.8%
Strategy 4	+19.4%	–22%	–20.7%	+4.4%
Strategy 5	+25%	–23%	+2.8%	+7.4%
Strategy 6	+18%	–15%	–16%	+3.6%
Strategy 7	+21.6%	–3.2%	+10%	–8.4%
Strategy 8	+21.6%	–21.2%	–4%	+11.6%
Strategy 9	+19.5%	–11%	+9.1%	+2.6%
Strategy 10	+23.8%	–16%	+4.5%	+5.5%
Average	**+21**%	**–19**%	**–2** %	**+5**%

Experimental results turned out to be quite good. Except from the time period T3 average results are better than index WIG. But the strategies are far away from allowing gain during bear market. The best one seems to be the 9-th strategy, which has quite simple and short notation:

```
Selling tree: MACD12<-3.8394
Buying tree : (WOL<-14882 OR RSI<14.43) OR FE>3.3286e+08
```

4.3 Hybrid GP and GA to develope solar cycle's model

The next example shows how the hybrid GP+AG algorithm for mathematical modelling (described in section 2.2) can be used to inferring models based on time series data. The algorithm was used to develope solar cycle's model and experiments concerned both approximating and predicting abilities of the algorithm were conducted.

Sunspots appear as dark spots on the surface of Sun. They typically last for several days, although very large ones may live for several weeks. It is known for over 150 years that sunspots appear in cycles. The average number of visible sunspots varies over time, increasing and decreasing on a regular cycle of between 9.5 to 11 years, on average about 10.8 years. The part of the cycle with low sunspot activity is referred to as "solar minimum" while the portion of the cycle with high activity is known as "solar maximum". The number of sunspots during solar maximum various for cycle to cycle. They

are collected for about 200 years. Changes in the sun activity are shown in Fig. 6. Knowledge of solar activity levels is often required years in advance, e.g., during satellite orbits and space missions planning [20].

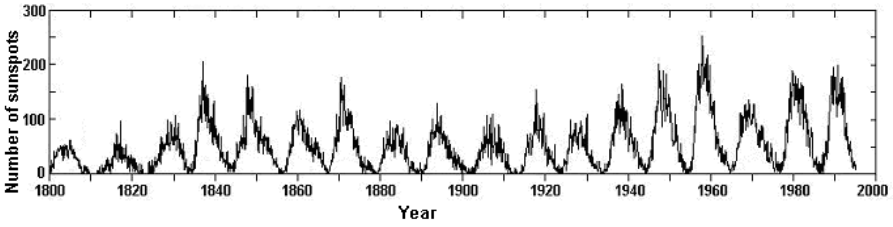

Fig. 6. Changes in Sun activity

To run the algorithm, originally daily collected data had to be specifically transformed into database suited to the GP. General way in which training and test data were generated is presented in Fig. 7. Firstly the learning time period (t_0, t_1), the number of attributes n, and time period length d_1 is determined. Attribute values are sum of sunspots appearing during time period of length d_1. Sunspot number for attribute y is counted for time period length d_2, different or equal to d_1, see Fig. 7a). Additionally distance p, denoted distance between time periods used to value of n^{th} attribute and y, is set. According to this, if $s(t_p, t_k)$ means sunspots number for the time period (t_p, t_k), where t_p is the initial and t_k the final date, for the learning time period (t_0, t_1) attribute values for the first training sample in database is as follows:
$x_1 = s(t_0, (t_0 + d_1))$,
$x_2 = s((t_0 + d_1), (t_0 + 2d_1)), ...,$
$x_n = s((t_0 + (n-2)d_1), (t_0 + (n-1)d_1))$,
$y = s((t_0 + (n-1)d_1 + p), (t_0 + (n-1)d_1 + p + d_2))$
The second training sample is generated similarly, but t_0 gets value $t_0 + d_1$. The last k^{th} sample is the one for which y is set for time period $(t_1, t_1 + d_2)$. Training database is generated in the same way but the initial test sample is the one with first attribute value counted for a time period $(t_1, t_1 + d_2)$, and d_1, d_2, n, p parameters remind unchanged.

In short term tests approximation turned out to be very good. Fig. 8 and 9 show results of two short-time experiments (for different values of parameters) as a comparison between value of an attribute y of given samples from training set and value calculated with obtained model. Evolution was led during 900 generations. In long-term tests approximation differed from one time period to another but was still surprisingly good. The best approximation for the given values of parameters (presented in Fig. 10) was obtained for $d_1 = d_2 = 180$ days, $p = 900$ days and learning time period from 1914 to 1980 year. Average percentage error obtained during this experiment was equal 16.9%.

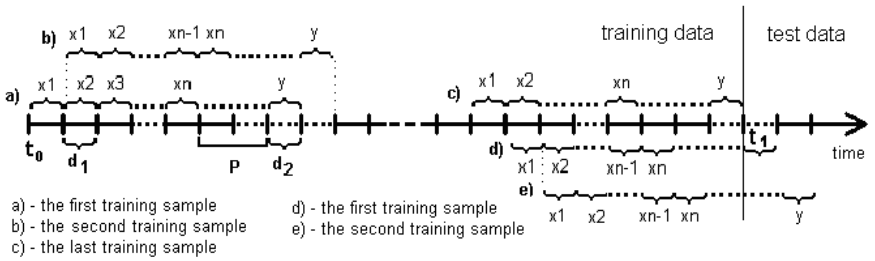

Fig. 7. Data preprocessing

Although derived functions well reflected relationships between data, they turned out to be useless at predicting future values of solar activity both in short and long advance. Increasing number of attributes to 30-40 and lengthening the learning time period could probably improve predicting abilities, but in the case of solar activity, it is not possible due to lack of data collecting in previous centuries.

Described experiments showed that GP algorithm can perform well in real-data, time series modelling. But the prediction task, often much more useful and desired, is however much more difficult and data-dependent, so in this case the GP algorithm failed.

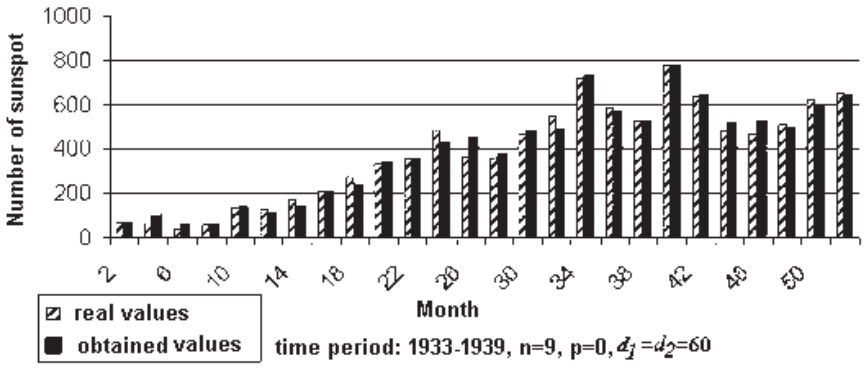

Fig. 8. Sunspot approximation – short-time test

4.4 General remarks

Advances in data collection methods, storage and processing technologies are provided a unique challenge and opportunity for automated data exploration techniques. Enormous amounts of data are being collected periodically: daily,

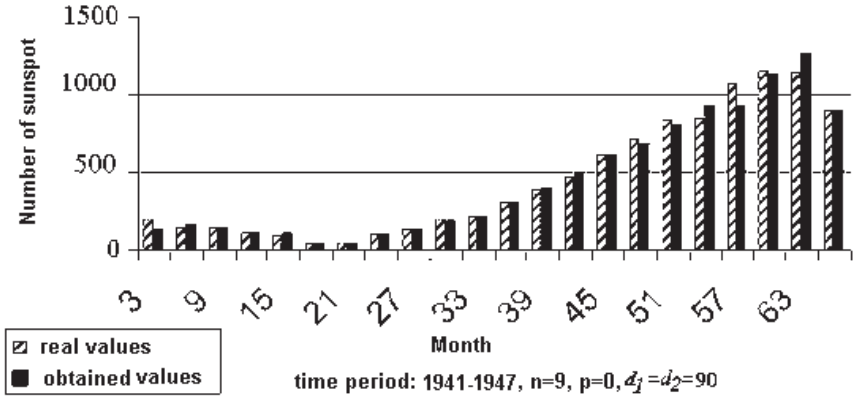

Fig. 9. The best approximation of sunspots – short-time test

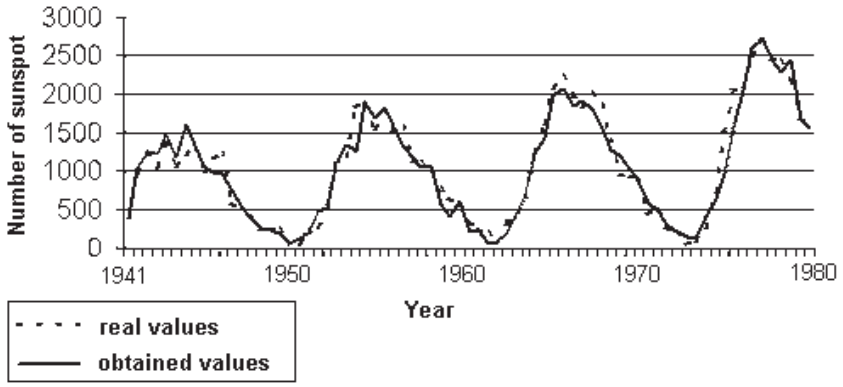

Fig. 10. The best approximation of sunspots – long-time test

weekly, monthly from great scientific projects like the Human Genome Project, the Hubble Space Telescope, from stock trading, from computerised sales and other sources. GP was applied to variety of time series modelling problems for understanding the dynamics of natural systems or/and for decision support systems development.

Each approach demanded introducing of special elements to the standard GP algorithm. As an example could be mentioned Bhattacharyya, Pictet, and Zumbach work, who induce trading decision models from high-frequency foreign exchange FX markets data. They suggested incorporation of domain-related knowledge and semantic restriction to enhance GP search. Their trading model seeks to capture market movement patterns and thereby provides trading recommendations in the form of signals, a " 1" indicating a buy signal, " –1" a sell signal, and "0" indicating to stay out of the market [21].

Another example in financial domain is Doherty's system. Doherty applied GP for induction of useful classification rules from the Standard and Poors COMPUSTAT database, which contains 334 attributes, spanning 50 years of data for nearly 10,000 active (trading on public markets) and 11,000 inactive (non-trading, acquired, or failed) public companies from North America. He transformed and normalised the database for better performance of evolution process [23]. GP was also used in non-financial domains, for example to ecological time series analysis. Whigham's and Recknagel's work on applying the Grammar Based Genetic Programming framework for the discovery of predictive equations and rules for phytoplankton abundance in freshwater lakes from time series data. They have demonstrated that models can be developed for the non-linear dynamics of phytoplankton, both as a set of rules and as mathematical equations [23].

But, what can be noticed analysing existing approaches, it is easy to obtain model exactly (or almost exactly) reflecting relationships hidden in the training database, but in a case of predicting – accuracy of obtained models "good results" are consider to be those, for which prediction on the validation set is even slightly more right than wrong. This shows how difficult prediction task is, and how much further work is needed.

5 Summary

In the chapter we can see usefulness of Genetic Programming for solving selected problems. These problems are connected with data processing – mathematical modelling the numerical data, classification task on the basis of possessed examples, prediction and time series modelling. The problems are not new ones, but still there are not commonly accepted methods for solving them. Genetic Programming is one of valuable approaches for them. In the mathematical modelling, described in section 2, the hybrid method – the Genetic Programming for trees developing and the classical Genetic Algorithm for parameters tuning – work together manifesting their high usefulness. The same method used for time series modelling (section 4) works very well for data approximation but the prediction of Sun spots is too difficult. We do not know, if there exists a solution of this problem, it is possible that this problem cannot be solved on the basis of earlier data or proposed representation of individuals.

Of course, one can find a lot of different applications of GP to data modelling. The authors try to select the interesting ones, showing different problems with using GP. As it is known, the GP is able to solve real problems – see work of Koza, e.g., aerial designed using GP is patented. But, as it is characteristic for all metaheuristics, every using of such approaches must be very carefully tuned for the solved problem. The readers are asked do not use any 'proper' values of parameters and genetic operators, it is necessary to develop the proper ones for the considered problem.

References

1. Keijzer M (2002), Scientific Discovery Using Genetic Programming. PhD Thesis, Danish Technical University, Lyngby
2. Koza JR (1992), Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press
3. Koza JR (1991), A genetic approach to econometric modelling. In: Bourgine P, Walliser B (eds), Economics and Cognitive Science. Pergamon Press, Oxford
4. Montana DJ (1995), Strongly typed genetic programming. Evolutionary Computation 3(2):199-230
5. Raidl GR (1998), A Hybrid GP Approach for Numerically Robust Symbolic Regression. Proc. of the 1998 Genetic Programming Conference, Madison, Wisconsin
6. Sugimoto M, Kikuchi S, Tomita M (2003), Prediction of Biochemical Reactions Using Genetic Programming. Genome Informatics 14: 637-638
7. Langdon WB, Barrett SJ (2004), Genetic Programming in Data Mining for Drug Discovery. In: Ghosh A, Jain LC (eds), Evolutionary Computing in Data Mining. Springer
8. Makarov DE, Metiu H (2000), Using Genetic Programming To Solve the Schrodinger Equation. The Journal of Physical Chemistry A 104 8540-8545
9. Turton I, Openshaw S, Diplock G (1996), Some Geographical Applications of Genetic Programming on the Cray T3D Supercomputer. Proc. of UKPAR'96, Jessope, Shafarenko (eds)
10. Freitas AA (2002), Data Mining and Knowledge Discovery with Evolutionary Algorightms. Springer-Verlag
11. Mallinson H, Bentley P (1999), Evolving Fuzzy Rules for Pattern Classification. Computational Integration for Modelling. Control and Automation '99, Masoud Mohammadian, IOS Press
12. Bojarczuk CE, Lopes HS, Freitas AA (2001), Data mining with constrained-syntax genetic programming: applications in medical data sets. Proc. Intelligent Data Analysis in Medicine and Pharmacology, London
13. Koza JR (1991), Concept formation and decision tree induction using the genetic programming paradigm. In: Schwefel H P, Mnner R (eds), Parallel Problem Solving from Nature. Springer-Verlag, Berlin
14. Freitas AA (2002), A survey of evolutionary algorithms for data mining and knowledge discovery. Ghosh A, Tsutsui S. (eds.) Advances in Evolutionary Computation. Springer-Verlag
15. Noda E, Freitas AA, Lopes HS (1999), Discovering interesting prediction rules with a genetic algorithm. Conference Evolutionary Computation (CEC-99), Washington
16. Ross J, Fueten F, Yashkir DY (2001), Automatic mineral identification using genetic programming. Machine Vision and Applications 13: 61-69, Springer-Verlag
17. Thompson S, Fueten F, Bockus D (2001), Mineral identification using artificial neural networks and the rotating polarizer stage. Comput Geosci, Pergamon Press
18. Gray HF, Maxwell RJ, Martinez-Perez I, Arus C, Cerdan S (1996), Genetic programming for classification of brain tumours from nuclear magnetic resonance biopsy spectra. In: Koza J R, Goldberg D E, Fogel D B, Riolo R L (eds), Proc. of the First Annual Conference, Stanford University, MIT Press

19. Bentley PJ (2000), "Evolutionary, my dear Watson" – investigating committee-based evolution of fuzzy rules for the detection of suspicious insurance claims. Proc. Genetic and Evolutionary Computation Conf. (GECCO-2000), Morgan Kaufmann
20. Kippenhahn R (1994), Discovering the Secrets of the Sun. John Wiley & Sons
21. Bhattacharyya S, Pictet O V, Zumbach G (2002), Knowledge-Intensive Genetic Discovery in Foreign Exchange Markets. IEEE Transactions on Evolutionary Computation
22. Doherty CG (2003), Fundamental Analysis Using Genetic Programming for Classification Rule Induction. Genetic Algorithms and Genetic Programming at Stanford 2003, Stanford Bookstore
23. Whigham PA, Recknagel F (2001), An Inductive Approach to Ecological Time Series Modelling by Evolutionary Computation. Ecological Modelling
24. Szpunar E (2001), Data mining methods in prediction of changes in solar activity (in polish). MA Thesis, Wroclaw University of Technology
25. Myszkowski PB (2002), Data mining methods in stock market analysis (in polish). MA Thesis, Wroclaw University of Technology

Index