9

Analytical Programming - a Novel Approach for Evolutionary Synthesis of Symbolic Structures

Ivan Zelinka[1], Donald Davendra[2], Roman Senkerik[3], Roman Jasek[4] and Zuzana Oplatkova[5]

[1,2,3,4,5]*Tomas Bata University in Zlin, Faculty of Applied Informatics, Nad Stranemi 4511, Zlin, 76001*

[1]*Department of Computer Science, Faculty of Electrical Engineering and Computer Science VSB-TUO, 17. listopadu 15, 708 33 Ostrava-Poruba Czech Republic*

1. Introduction

This chapter discusses an alternative approach for symbolic structures and solutions synthesis and demonstrates a comparison with other methods, for example Genetic Programming (GP) or Grammatical Evolution (GE). Generally, there are two well known methods, which can be used for symbolic structures synthesis by means of computers. The first one is called GP and the other is GE. Another interesting research was carried out by Artificial Immune Systems (AIS) or/and systems, which do not use tree structures like linear GP and other similar algorithm like Multi Expression Programming (MEP), etc. In this chapter, a different method called Analytic Programming (AP), is presented. AP is a grammar free algorithmic superstructure, which can be used by any programming language and also by any arbitrary Evolutionary Algorithm (EA) or another class of numerical optimization method. This chapter describes not only theoretical principles of AP, but also its comparative study with selected well known case examples from GP as well as applications on synthesis of: controller, systems of deterministic chaos, electronics circuits, etc. For simulation purposes, AP has been co-joined with EA's like Differential Evolution (DE), Self-Organising Migrating Algorithm (SOMA), Genetic Algorithms (GA) and Simulated Annealing (SA). All case studies has been carefully prepared and repeated in order to get valid statistical data for proper conclusions.

The term *symbolic regression* represents a process during which measured data sets are fitted, thereby a corresponding mathematical formula is obtained in an analytical way. An output of the symbolic expression could be, for example, $\sqrt[N]{x^2 + \frac{y^3}{k}}$, and the like. For a long time, symbolic regression was a domain of human calculations but in the last few decades it involves computers for symbolic computation as well.

The initial idea of symbolic regression by means of a computer program was proposed in GP (Koza, 1990; 1998). The other approach of GE was developed in Ryan et al. (1998) and AP in Zelinka et al. (2005a). Another interesting investigation using symbolic regression were carried out in Johnson (2004) on AIS and Probabilistic Incremental Program Evolution (PIPE), which generates functional programs from an adaptive probability distribution over

all possible programs. Yet another new technique is the so called *Transplant Evolution*, see Weisser & Osmera (2010b), Weisser & Osmera (2010a) and Weisser et al. (2010) which is closely associated with the conceptual paradigm of AP, and modified for GE. GE was also extended to include DE by O'Neill & Brabazon (2006). Symbolic regression is schematically depicted in Figure 1. Generally speaking, it is a process which combines, evaluates and creates more complex structures based on some elementary and noncomplex objects, in an evolutionary way. Such elementary objects are usually simple mathematical operators $(+, -, \times, ...)$, simple functions (*sin*, *cos*, *And*, *Not*, ...), user-defined functions (simple commands for robots – MoveLeft, TurnRight, ...), etc. An output of symbolic regression is a more complex "object" (formula, function, command,...), solving a given problem like data fitting of the so-called Sextic and Quintic problem described by Equation (1) (Koza et al., 1999; Zelinka & Oplatkova, 2003a), randomly synthesized function by Equation (2) (Zelinka & Oplatkova, 2003a), Boolean problems of parity and symmetry solution (basically logical circuits synthesis) by Equation (3) (Koza et al., 2003; Zelinka et al., 2005a), or synthesis of quite complex robot control command by Equation (4) (Koza, 1998; Oplatkova & Zelinka, 2006). Equations (1)–(4) mentioned here are just a few samples from numerous repeated experiments done by AP, which are used to demonstrate how complex structures can be produced by symbolic regression in general for different problems.

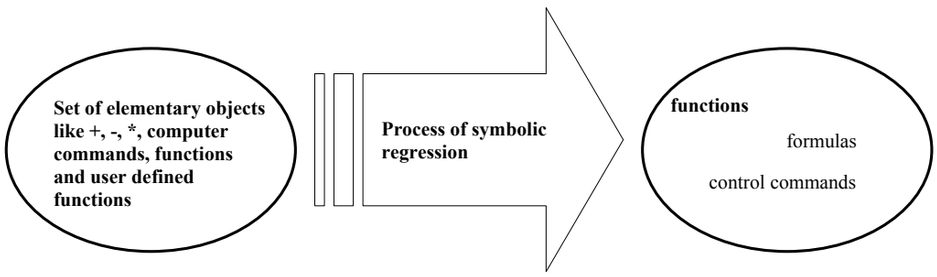


Fig. 1. Symbolic regression - schematic view

$$x\left(K_1+\frac{\left(x^2K_3\right)}{K_4\left(K_5+K_6\right)}\right)*\left(-1+K_2+2x\left(-x-K_7\right)\right) \tag{1}$$

$$\sqrt{t}\left(\frac{1}{\log\left(t\right)}\right)^{\sec^{-1}(1.28)}\log^{\sec^{-1}(1.28)}\left(\sinh\left(\sec\left(\cos\left(1\right)\right)\right)\right) \tag{2}$$

$$\begin{array}{l} Nor[(Nand[Nand[B||B, B\&\&A], B])\&\&C\&\&A\&\&B, \\ Nor[(!C\&\&B\&\&A||!A\&\&C\&\&B||!C\&\&!B\&\&!A)\&\& \\ (!C\&\&B\&\&A||!A\&\&C\&\&B||!C\&\&!B\&\&!A)|| \\ A\&\&(!C\&\&B\&\&A||!A\&\&C\&\&B||!C\&\&!B\&\&!A), \\ (C||!C\&\&B\&\&A||!A\&\&C\&\&B||!C\&\&!B\&\&!A)\&\&A]] \end{array} \tag{3}$$

$$\begin{array}{l} \text{Prog2[Prog3[Move, Right, IfFoodAhead[Left, Right]],} \\ \text{IfFoodAhead[IfFoodAhead[Left, Right], Prog2[IfFoodAhead[} \\ \text{IfFoodAhead[IfFoodAhead[Left, Right], Right], Right],} \\ \text{IfFoodAhead[Prog2[Move, Move], Right]]]]} \end{array} \tag{4}$$

1.1 Genetic programming

GP was the first tool for symbolic regression carried out by means of computers instead of humans. The main idea comes from GA, which was used in GP (Koza, 1990; 1998). Its ability to solve very difficult problems is well proven; for example, GP performs so well that it can be applied to synthesize highly sophisticated electronic circuits (Koza et al., 2003).

The main principle of GP is based on GA, which is working with populations of individuals represented in the LISP programming language. Individuals in a canonical form of GP are not binary strings, different from GA, but consist of LISP symbolic objects (commands, functions, ...), etc. These objects come from LISP, or they are simply user-defined functions. Symbolic objects are usually divided into two classes: functions and terminals. Functions were previously explained and terminals represent a set of independent variables like x, y, and constants like π , 3.56, etc.

The main principle of GP is usually demonstrated by means of the so-called trees (basically graphs with nodes and edges, as shown in Figure 2 and Figure 3, representing individuals in LISP symbolic syntax). Individuals in the shape of a tree, or formula like $0.234Z + X - 0.789$, are called programs. Because GP is based on GA, evolutionary steps (mutation, crossover, ...) in GP are in principle the same as GA. As an example, GP can serve two artificial parents – trees on Figure 2 and Figure 3, representing programs $0.234Z + X - 0.789$ and $ZY(Y + 0.314Z)$. When crossover is applied, for example, subsets of trees are exchanged. Resulting offsprings of this example are shown on Figure 3.

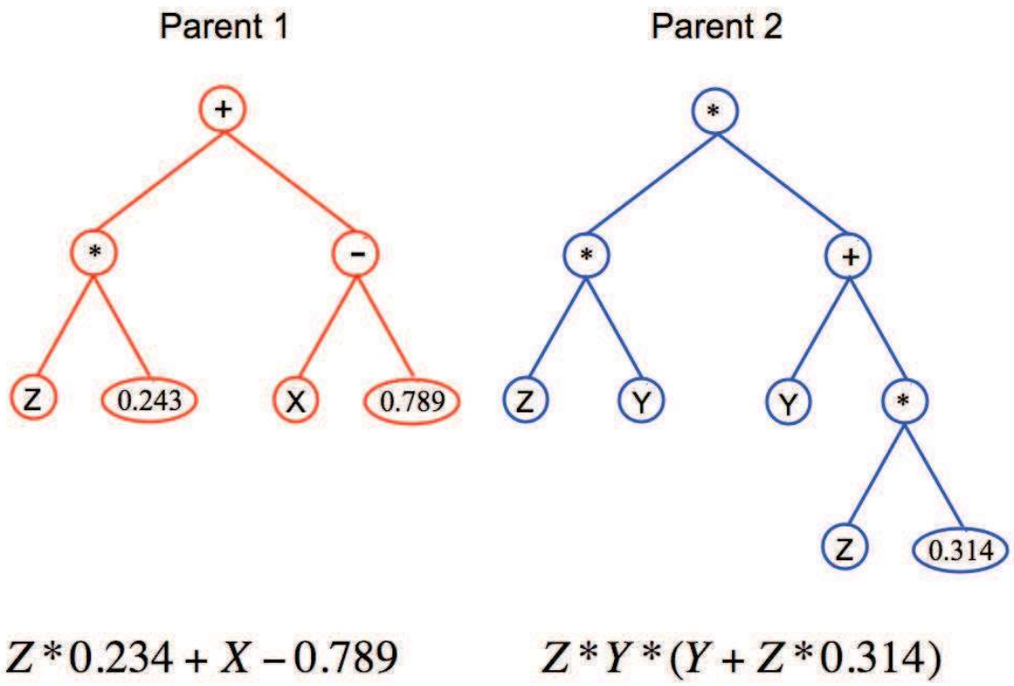


Fig. 2. Parental trees

Subsequently, the offspring fitness is calculated, such that the behavior of the just-synthesized and evaluated individual-tree should be as similar as possible to the desired behavior. The desired behavior can be regarded as a measured data set from some process (a program that

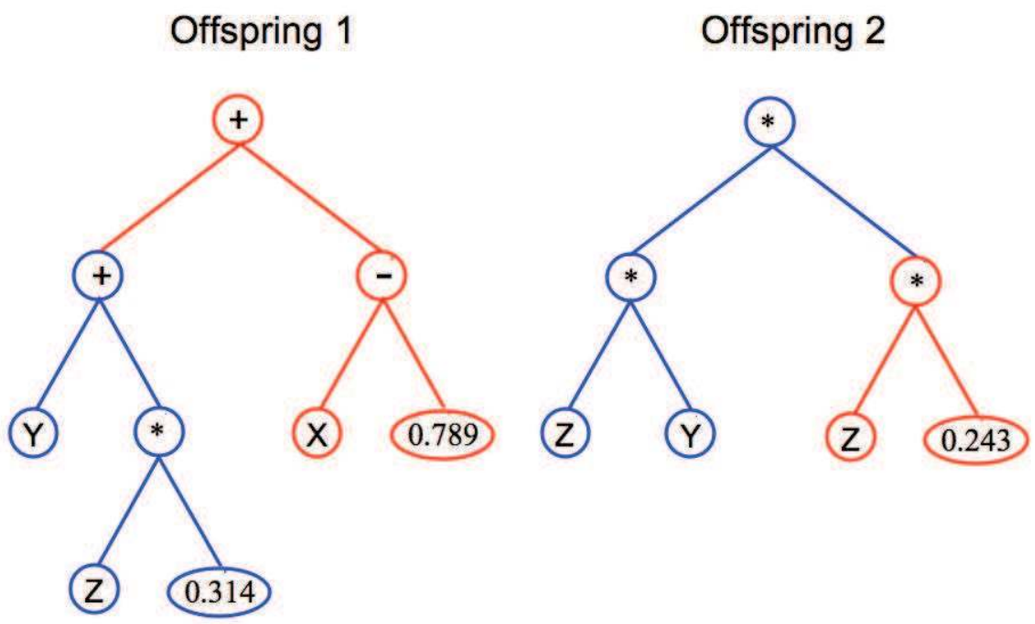


Fig. 3. Offsprings

should fit them as well as possible) or like an optimal robot trajectory, i.e., when the program is evaluating a sequence of robot commands (TurnLeft, Stop, MoveForward,...) leading as close as possible to the final position. This is basically the same for GE.
For detailed description of GP, see Koza (1998), Koza et al. (1999).

1.2 Grammatical evolution

GE is another program developed in O'Neill & Ryan (2003), which performs a similar task to that of GP. GE has one advantage over GP, and this is the ability to use any arbitrary programming language, not only LISP as in the case of the canonical version of GP. In contrast to other EA's, GE was used only with a few search strategies, and with a binary representation of the populations (O'Neill & Ryan, 2003). The last successful experiment with DE applied on GE was reported in O'Neill & Brabazon (2006). GE in its canonical form is based on GA, thanks to a few important changes it has in comparison with GP. The main difference is in the individual coding.
While GP manipulates in LISP symbolic expressions, GE uses individuals based on binary strings. These are transformed into integer sequences and then mapped into a final program in the Backus-Naur Form (BNF) (O'Neill & Ryan, 2003), as explained by the following artificial example. Let $T = \{+, -, \times, /, x, y\}$ be a set of operators and terminals and let F = {epr, op, var} be the so-called nonterminals. In this case, the grammar used for final program synthesis is given in Table 1. The rule used for individuals transforming into a program is based on Equation (5) below. GE is based on binary chromosome with a variable length, divided into the so-called codons (range of integer values, 0-255), which is then transformed into an integer domain according to Table 2.

Nonterminals		Unfolding	Index
expr	::=	op expr expr	0
		var	1
op	::=	+	0′
		-	1′
		*	2′
		/	3′
var	::	X	0″
		Y	(1″)

Table 1. Grammatical evolution - rules

Chromozone	Binary	Integer	BNF index
Codon 1	101000	40	0
Codon 2	11000011	162	2′
Codon 3	1100	67	1
Codon 4	10100010	12	0″
Codon 5	1111101	125	1
Codon 6	11100111	231	1″
Codon 7	10010010	146	Unused
Codon 8	10001011	139	Unused

Table 2. Grammatical evolution - codon

$$\begin{aligned} &\text{unfolding} \;=\; \text{codon mod rules} \\ &\text{where rules is number of rules for given nonterminal} \end{aligned} \tag{5}$$

Synthesis of an actual program can be described by the following. Start with a nonterminal object expr. Because the integer value of Codon 1 (see Table 2) is 40, according to Equation (5), one has an unfolding of *expr = op expr expr* (40 mod 2, 2 rules for *expr*, i.e., 0 and 1). Consequently, Codon 2 is used for the unfolding of *op* by * (162 mod 4), which is the terminal and thus the unfolding for this part of program is closed. Then, it continues in unfolding of the remaining nonterminals (*expr expr*) till the final program is fully closed by terminals. If the program is closed before the end of the chromosome is reached, then the remaining codons are ignored; otherwise, it continues again from the beginning of the chromosome. The final program based on the just-described example is in this case $x \cdot y$ (see Figure 4). For a fully detailed description of GE principles, see O'Neill & Ryan (2003) or consult [http://www.grammaticalevolution.org/].

1.3 Analytic programming

The final method described here and used for experiments in this chapter is called AP, which has been compared to GP with very good results (see, for example, Zelinka & Oplatkova (2003a), Oplatkova (2005), Zelinka et al. (2005a), Oplatkova & Zelinka (2006), Zelinka et al. (2008)) or visit website *www.ivanzelinka.eu*.

The basic principles of AP were developed in 2001 and first published in Zelinka (2001) and Zelinka (2002a). AP is also based on the set of functions, operators and terminals, which are usually constants or independent variables alike, for example:

- **functions**: *sin*, *tan*, *tanh*, *And*, *Or*,...

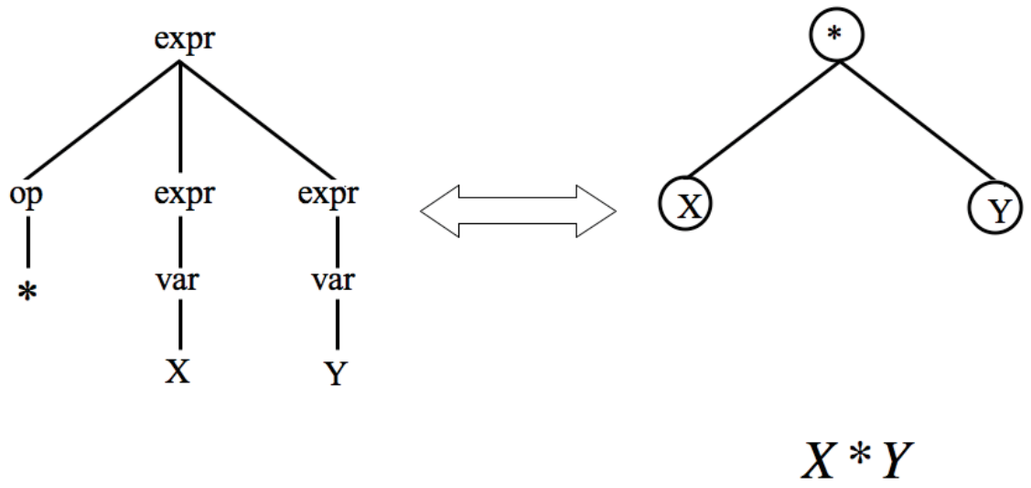


Fig. 4. Final program by GE

- **operators**: +, -, ×, /, dt,...
- **terminals**: 2.73, 3.14, t,...

All these objects create a set, from which AP tries to synthesize an appropriate solution. Because of the variability of the content of this set, it is called a general functional set (GFS). The structure of GFS is nested, i.e., it is created by subsets of functions according to the number of their arguments (Figure 5). The content of GFS is dependent only on the user. Various functions and terminals can be mixed together. For example, GFS_{all} is a set of all functions, operators and terminals, GFS_{3arg} is a subset containing functions with maximally three arguments, GFS_{0arg} represents only terminals, etc. (Figure 5).
AP, as further described later, is a mapping from a set of individuals into a set of possible programs. Individuals in population and used by AP consist of non-numerical expressions (operators, functions,...), as described above, which are in the evolutionary process represented by their integer position indexes (Figure 6, Figure 7, see also Chapter 2). This index then serves as a pointer into the set of expressions and AP uses it to synthesize the resulting function-program for cost function evaluation.
Figure 7 demonstrates an artificial example as to how a final function is created from an integer individual via Discrete Set Handling (DSH). Number 1 in the position of the first parameter means that the operator + from GFS_{all} is used (the end of the individual is far enough). Because the operator + must have at least two arguments, the next two index pointers 6 (*sin* from GFS) and 7 (*cos* from GFS) are dedicated to this operator as its arguments. The two functions, *sin* and *cos*, are one-argument functions, so the next unused pointers 8 (*tan* from GFS) and 9 (*t* from GFS) are dedicated to the *sin* and *cos* functions. As an argument of *cos*, the variable *t* is used, so this part of the resulting function is closed (*t* is zero-argument) in its AP development. The one-argument function *tan* remains, and because there is one unused pointer 9, *tan* is mapped on *t* which is on the 9th position in GFS.
To avoid synthesis of pathological functions, a few security *tricks* are used in AP. The first one is that GFS consists of subsets containing functions with the same or a smaller number of arguments. The nested structure (see also Figure 5) is used in the special security subroutine, which measures how far the end of an individual is and, according to this, mathematical

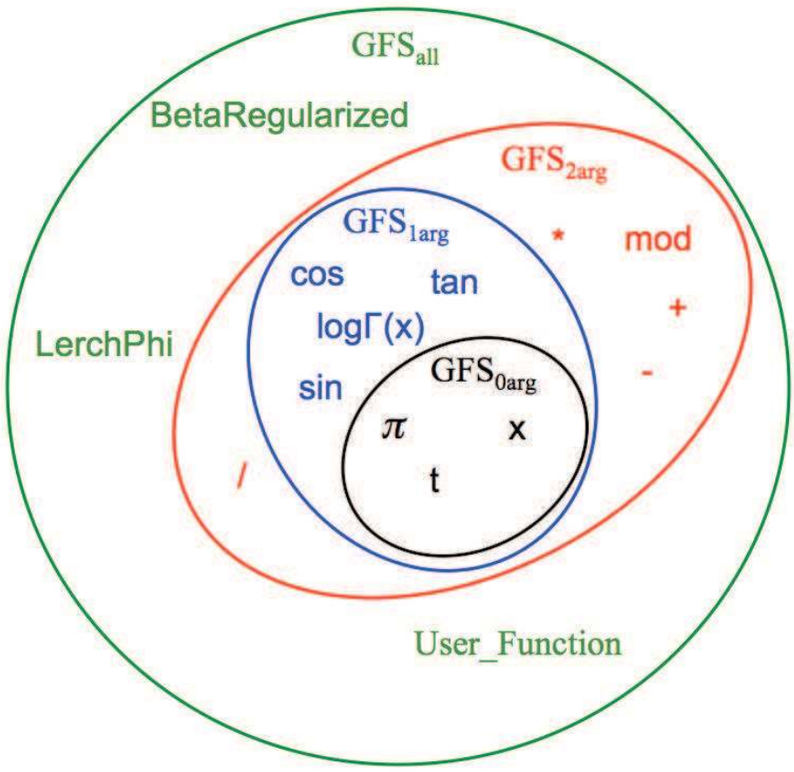


Fig. 5. Hierarchy in GFS

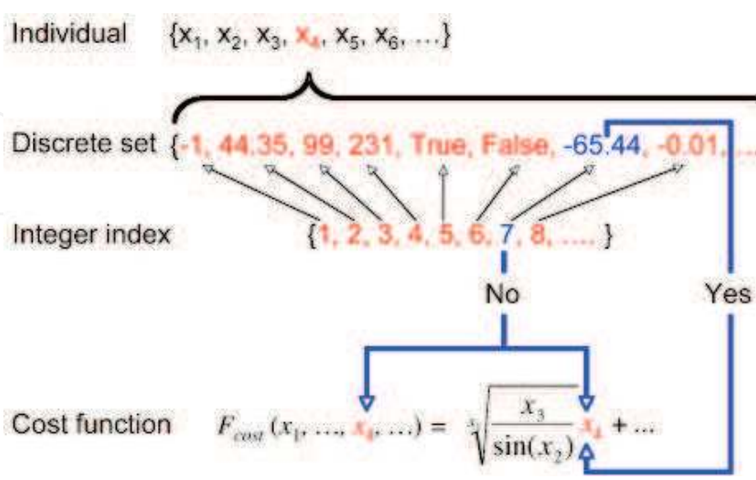


Fig. 6. DSH-Integer index

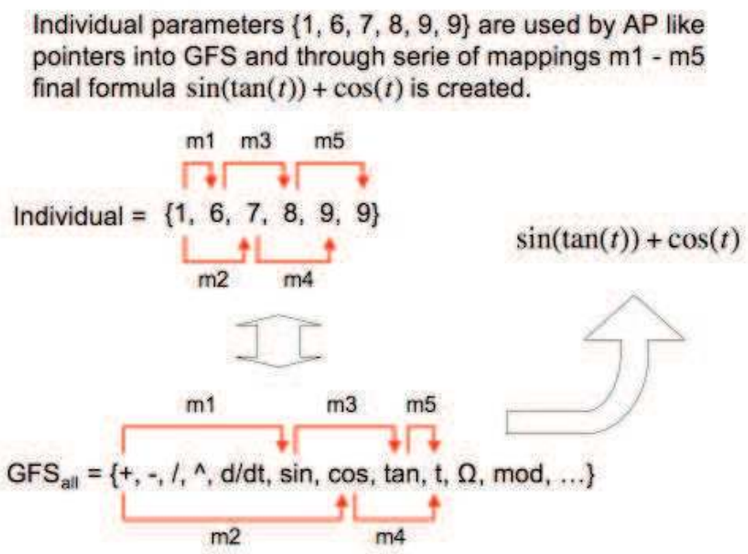


Fig. 7. Principle of mapping from GFS to programs

elements from different subsets are selected to avoid pathological functions synthesis. More precisely, if more arguments are desired then a possible function (the end of the individual is near) will be replaced by another function with the same index pointer from the subset with a smaller number of arguments. For example, it may happen that the last argument for one function will not be a terminal (zero-argument function). If the pointer is longer than the length of subset, e.g., a pointer is 5 and is used GFS_0, then the element is selected according to the rule: element = pointer_value mod number_of_elements_in_GFS_0. In this example, the selected element would be the variable t (see GFS_0 in Figure 5).
GFS need not be constructed only from clear mathematical functions as demonstrated above, but may also be constructed from other user-defined functions, e.g., logical functions, functions which represent elements of electrical circuits or robot movement commands, linguistic terms, etc.

1.3.1 Versions

AP was evaluated in three versions. All three versions utilize the same set of functions for program synthesis, terminals, etc., as in GP (Koza, 1998; Koza et al., 1999)). The second version labelled as AP_{meta} (the first version, AP_{basic}) is modified in the sense of constant estimation. For example, the so-called sextic problem was used in Koza (1998) to randomly generate constants, whereas AP uses only one, called K, which is inserted into the formula (6) below at various places by the evolutionary process. When a program is synthesized, all K's are indexed as K_1, K_2, ..., K_n to obtain (7) the formula, and then all K_n are estimated by using a second EA, the result of which can be, for example, (8). Because EA (slave) "works under" EA (master), i.e., $EA_{master} \rightarrow$ program $\rightarrow$ K indexing $\rightarrow EA_{slave} \rightarrow$ estimation of K_n, this version is called AP with metaevolution, denoted as AP_{meta}.

$$\frac{x^2 + K}{\pi^K} \tag{6}$$

$$\frac{x^2 + K_1}{\pi^{K_2}} \tag{7}$$

$$\frac{x^2 + 3.56}{\pi^{-229}} \tag{8}$$

Due to this version being quite time-consuming, AP_{meta} was further modified to the third version, which differs from the second one in the estimation of K. This is accomplished by using a suitable method for nonlinear fitting (denoted AP_{nf}). This method has shown the most promising performance when unknown constants are present. Results of some comparative simulations can be found in Zelinka & Oplatkova (2003b), Zelinka et al. (2005b) and Oplatkova et al. (2008). AP_{nf} was the method chosen for the simulations described in this chapter.

1.3.2 Data set structure and mapping method

The subset structure presence in GFS is vitally important for AP. It is used to avoid synthesis of pathological programs, i.e. programs containing functions without arguments, etc. Performance of AP is, of course, improved if functions of GFS are expertly chosen based on experiences with solved problem.

An important part of AP is a sequence of mathematical operations which are used for program synthesis. These operations are used to transform an individual of a population into a suitable program. Mathematically stated, it is mapping from an individual domain into a program domain. This mapping consists of two main parts. The first part is DHS and the second one are security procedures which do not allow synthesizing of pathological programs. DHS proposed in Lampinen & Zelinka (1999), Zelinka (2004) is used to create an integer index, which is used in the evolutionary process like an alternate individual handled in EA by the method of integer handling. The method of DSH, when used, allows to handle arbitrary objects including nonnumerical objects like linguistic terms hot, cold, dark, ..., logic terms (True, False) or other user defined functions. In the AP, DSH is used to map an individual into GFS and together with Security Procedures (SP) creates the above mentioned mapping, which transforms the arbitrary individual into a program. Individuals in the population consist of integer parameters, i.e. an individual is an integer index pointing into GFS.

AP is basically a series of function mapping. In Figure 7, an artificial example is given as to how a final function is created from an integer individual. Number 1 in the position of the first parameter means that the operator + from GFS_{all} is used (the end of the individual is far enough). Because the operator + has to have at least two arguments, the next two index pointers 6 (*sin* from GFS) and 7 (*cos* from GFS) are dedicated to this operator as its arguments. Both functions, sin and cos, are one-argument functions so the next unused pointers 8 (*tan* from GFS) and 9 (t from GFS) are assigned to *sin* and *cos* function. Because *cos* has used variable t as an argument, this part of resulting function is closed (t is zero-argument) in its AP development. Only one-argument function remains and since there is one unused pointer 9 *tan*, it is mapped on t which is on the 9th position in GFS.

To avoid synthesis of pathological functions, a few security tricks are used in AP. The first one is that GFS consists subsets containing functions with the same number of arguments. Existence of this nested structure is used in the special security subroutine, which is measuring how far the end of individual is and according to this objects from different subsets are selected to avoid pathological function synthesis. Precisely, if more arguments are desired than the possible (the end of the individual is near) function will be replaced by another

function with the same index pointer from subset with lower number of arguments. For example, it may happen if the last argument for one argument function will not be a terminal (zero-argument function) as demonstrated in Figure 7.
GFS doesn't need to be constructed from only clear mathematical functions as is demonstrated, but also from other user-defined functions, which can be used, e.g. logical functions, functions which represent elements of electrical circuits or robot movement commands.

1.3.3 Crossover, mutations and other evolutionary operations

During evolution of a population, a number of different operators are used, such as crossover and mutation. In comparison with GP or GE, evolutionary operators like mutation, crossover, tournament, selection are fully in the competence of used EA. AP does not contain them in any point of view of its internal structure. AP is created like a superstructure of EAs for symbolic regression independent on their algorithmical structure. Operations used in EA's are not influenced by AP and vice versa. For example if DE is used for symbolic regression in AP then all evolutionary operations are done according to the DE rules. AP just transforms individuals into formulas.

1.3.4 Reinforced evolution

During evolution, more or less appropriate individuals are synthesized. Some of these individuals are used to reinforce the evolution towards a better solution synthesis. The main idea of reinforcement is based on the addition of the just-synthesized and partly successful program into an initial set of terminals. Reinforcement is based on a user-defined criterion used in decision as to which individual will be used as an addition into the initial set of terminals. A criterion for the decision is in fact a threshold, i.e., by a user-defined cost value, under which conditions are synthesized solutions being added into GFS.
For example, if the threshold is set to 5, and if the fitness of all individuals (programs in the population) is bigger than 5, then evolution is running on the basic, i.e., initially defined, GFS. When the best individual in the actual population is less than 5, then it is entirely added into the initial GFS and is marked as a terminal. Since this moment, evolution is running on the enriched GFS containing a partially successful program. Thanks to this advantage, evolution is able to synthesize the final solutions much faster than the AP without reinforcement. This fact has been repeatedly verified by simulations on different problems. When the program is added into GFS, the threshold is also set to its fitness. If furthermore, an individual with better fitness than the just-reset threshold is synthesized, then the old one is rewritten by the better one, and the threshold is rewritten by a new fitness value.
It is quite similar to Automatically Defined Functions (ADF) for GP; however, the set of functions and terminals in GP can contain more than one ADF, (which of course at least theoretically increases the complexity of the search space to the order of $n!$), including properly defined arguments of these ADF and critical situation checking (selfcalling,...). This is not a problem of AP reinforcement, because adding a program into the initial GFS is regarded as a terminal (or a terminal structure), i.e., no function, no arguments, no selfcalling, etc., and the cardinality of the initial GFS set increases only by one.

```
Start of simulation No. 5    Time: 22:7 :54.351481

Appended suboptimal solution with CV = 4    Time: 22:7 :54.547317

Appended suboptimal solution with CV = 3    Time: 22:7 :54.577788

Appended suboptimal solution with CV = 2    Time: 22:8 :41.529214

Appended suboptimal solution with CV = 1    Time: 22:10 :14.346728

Number of cost function evaluations: 36270
Cost value: 0
The best individual:
 {8, 5, 8, 3, 1, 9, 4, 2, 3, 9, 7, 4, 4, 10, 1, 10, 5, 8, 5, 7, 10, 6, 5, 10, 9, 9, 7, 2, 6, 3}
Solution: (((B ⋁ (((A ⋀ C ⋀ ¬ B) ⋁ (B ⋀ C ⋀ ¬ A) ⋁ (¬ A ⋀ ¬ B ⋀ ¬ C)) ⋀ C)) ⊼ (B ⊼ (C ⋀ A))) ⋀
    A ⋀ ((A ⊽ ((A ⋀ C ⋀ ¬ B) ⋁ (B ⋀ C ⋀ ¬ A) ⋁ (¬ A ⋀ ¬ B ⋀ ¬ C))) ⊼
      ((A ⋀ C ⋀ ¬ B) ⋁ (B ⋀ C ⋀ ¬ A) ⋁ (¬ A ⋀ ¬ B ⋀ ¬ C)))) ⊽
  (A ⊽ ((A ⊽ ((A ⋀ C ⋀ ¬ B) ⋁ (B ⋀ C ⋀ ¬ A) ⋁ (¬ A ⋀ ¬ B ⋀ ¬ C))) ⊽
      (((A ⋀ C ⋀ ¬ B) ⋁ (B ⋀ C ⋀ ¬ A) ⋁ (¬ A ⋀ ¬ B ⋀ ¬ C)) ⊽ ((A ⋀ C ⋀ ¬ B) ⋁ (B ⋀ C ⋀ ¬ A) ⋁ (¬ A ⋀ ¬ B ⋀ ¬ C)))))

    {2005, 11, 28, 22, 14, 45.450393}
```

Fig. 8. Effect of reinforced evolution

1.3.5 Security procedures

Security Procedures (SP) are in AP as well as in GP, used to avoid various critical situations. In the case of AP, security procedures were not developed for AP purposes after all, but they are mostly an integrated part of AP. However, sometimes they have to be defined as a part of cost function, based on specific situations (for example situation 2, 3 and 4, shown below). Critical situations are like:

1. pathological function (without arguments, self-looped...)
2. functions with imaginary or real part (if not expected))
3. infinity in functions (dividing by 0, ...)
4. *frozen* functions (an extremely long time to get a cost value - hrs...)
5. etc.

Put simply, an SP can be regarded as a mapping from an integer individual to the program, which is checked as to how far the end of the individual is and based on this information, a sequence of mapping is redirected into a subset with a lower number of arguments. This satisfies the constraint that no pathological function will be generated. Other activities of SP are integrated as part of the cost function to satisfy items 2-4, etc.

1.3.6 Similarities and differences

Similarities and Differences Because AP was partly inspired by GP, then between AP, GP and GE some differences as well as some similarities logically exist. A few of these are:

1. Synthesized programs (similarity): AP as well as GP and GE is able to do symbolic regression in a general point of view. It means that the output of AP is according to simulations (Zelinka, 2002b;c; Zelinka & Oplatkova, 2004; Zelinka et al., 2004), similar to programs from GP and GE.
2. Functional set (similarity): AP_{basic} operates in principle on the same set of terminals and functions as GP or GE, while AP_{meta} or AP_{nf} use a universal constant K (difference), which is indexed after program synthesis.

(a) Subfigure 1 (b) Subfigure 2

Fig. 9. Security

3. Individual coding (difference): coding of an individual is different. AP uses an integer index instead of direct representation as in canonical GP. GE uses the binary representation of an individual, which is consequently converted into integers for mapping into programs by means of BNF (O'Neill & Ryan, 2003).
4. Individual mapping (difference): AP uses DSH, while GP in its canonical form uses direct representation in LISP (Koza, 1998) and GE uses BNF.
5. Constant handling (difference): GP uses a randomly generated subset of numbers - constants (Koza, 1998), GE utilizes user determined constants and AP uses only one constant K for AP_{meta} and AP_{nf}, which is estimated by another EA or by nonlinear fitting.
6. Security procedures (difference): to guarantee synthesis of non-pathological functions, procedures are used in AP which redirect the flow of mapping into subsets of a whole set of functions and terminals according to the distance to the end of the individual. If pathological function is synthesized in GP, then synthesis is repeated. In the case of GE, when the end of an individual is reached, the mapping continues from the individual beginning, which is not the case in AP. It is designed so that a non-pathological program is synthesized before the end of the individual is reached (maximally when the end is reached).

2. Selected applications

This section briefly describes some selected applications of AP, which has been conducted during the past few years and cover a comparative study with GP techniques published by J. R. Koza as well as other different applications. In each subsection, the main idea of the AP application is described, results alongside references to publications, cumulating in the full report of proposed application.

2.1 Randomly generated solutions

This part discuss the first and very simple experiment (see Zelinka (2002b)) which has been done with AP. It was focused on the verification as to whether AP as it was programmed, is able to produce reasonable structures (programs, formulas, etc...) which are complete, i.e. there are not missing arguments, division by 0 etc. In this simulation, randomly selected functions from GFS were selected, i.e. randomly generated individuals has been transformed to programs by AP - no evolution has been taken into consideration in this experiment. The terminal set contains only one variable t and a few constants. The nonterminal set consisted of various and just randomly (by user) collected mathematical functions. Final solutions were synthesized from both sets. A few examples of synthesized formulas are represented by Equations (9) - (13). Selected formulas were also visualized to show interesting behavior of the synthesized programs. They are depicted in Figure 10 - 15. It is important to remember that there was no another deeper mathematical reason to synthesize such, on the first look wild, functions. There was only one aim - to check whether AP is able to synthesize structurally acceptable solutions. Instead of used mathematical functions, user defined functions can also be used. There was 1000 randomly generated individuals, converted into programs (formulas) and verified. No pathology in their structures has been observed.

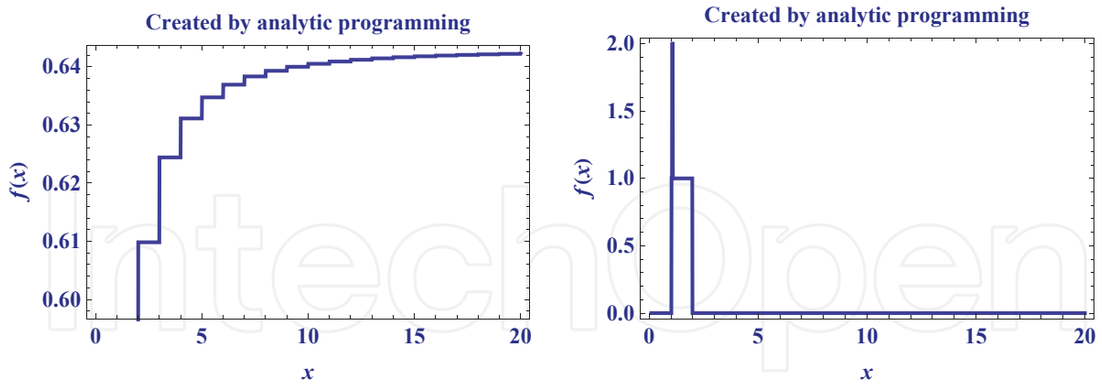
$$\cos^{-1}\left(\sec^{-1}\left(e^{-i\coth(1-it)}\right)^{\operatorname{sech}^{-1}\left(\left(\frac{\cosh^{-1}(K)}{\phi}\right)^{K}\right)}\right) \tag{9}$$

$$\cos^{-1}\left(\sec^{-1}\left(\frac{\log\left(\frac{\phi}{t}\right)}{\log(\cot(\gamma))}\right)^{\operatorname{sech}^{-1}\left(\operatorname{csch}^{-1}(t)^{-K}\right)}\right) \tag{10}$$

$$\tanh\left(\cot^{-1}\left(\frac{\phi}{\gamma}\right)^{\sinh^{-1}\left(e^{\phi-t}\right)}\right) \tag{11}$$

$$\frac{\sqrt{1-\pi^2\sin^2\left(\cos\left(e^{\operatorname{sech}(\cosh(\cosh(\sin(t))))}\right)\right)}\csc\left(\cos\left(e^{\operatorname{sech}(\cosh(\cosh(\sin(t))))}\right)\right)}{\pi} \tag{12}$$

$$\sqrt{\operatorname{sech}^{-1}\left(\operatorname{sech}^{-1}\left(\exp\left(\sinh^{-1}\left(\frac{t^{\cos^{-1}(\gamma)}\phi^{\cos^{-1}(\gamma)}}{\cot^{-1}(A)}\right)\right)\right)\right)} \tag{13}$$



(a) (b)

Fig. 10. Visualization of randomly synthesized individuals - functions

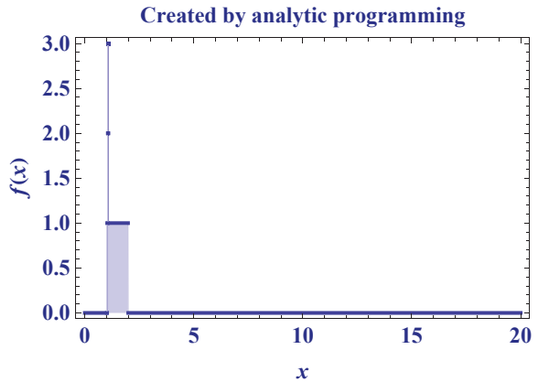
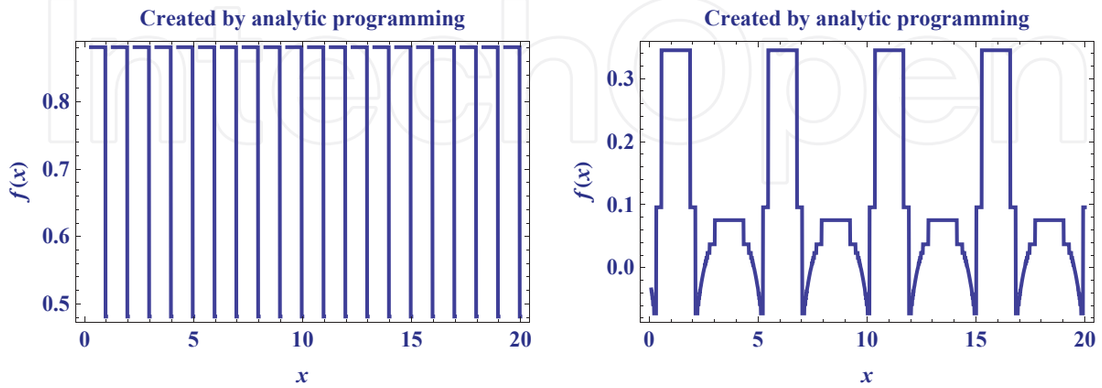
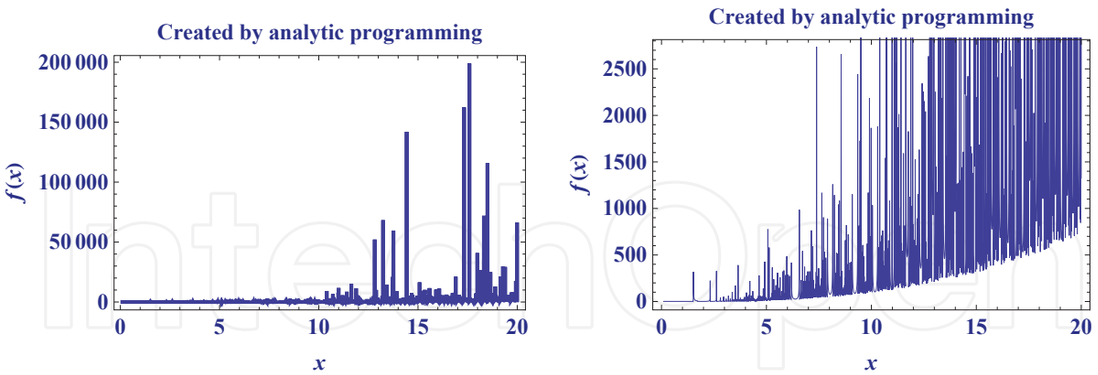


Fig. 11. Another view of the function from Figure 10b



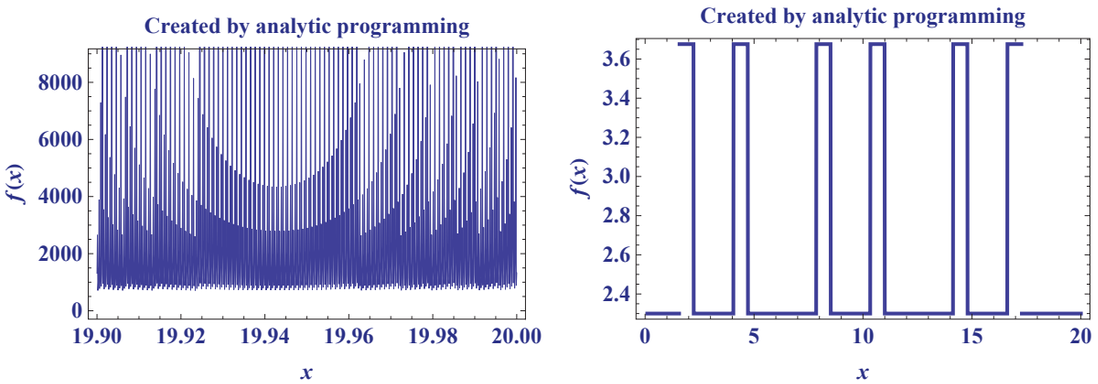
(a) (b)

Fig. 12. Visualization of randomly synthesized individuals - functions



(a) (b)

Fig. 13. Visualization of randomly synthesized individuals - functions, b) detailed view



(a) (b)

Fig. 14. a) More detailed view of Figure 13b and b) behavior of different function

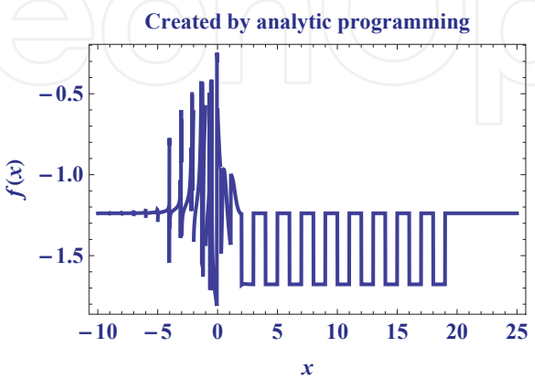


Fig. 15. Another interesting function

2.2 Comparison with selected GP examples

To verify more properly the functionality of AP, a set of comparative simulations based on selected examples from Koza's GP, have been done. Two algorithms were used for comparison of AP with GP - DE and SOMA. Simulations were focused on selected examples from Koza (1998) and Koza et al. (1999), especially:

- Sextic problem - $x^6 - x^4 + x^2$
- Quintic problem - $x^5 - 2x^3 + x$
- Boolean even-k-parity problem - synthesis of logical function in a few versions containing 3, 4, 5 and 6 input variables
- Boolean symmetry problem - synthesis of logical function in a few versions containing 3, 4, 5 and 6 input variables

Based on the studies in Koza (1998) andKoza et al. (1999), the above mentioned problems have been selected for comparative study. The first two are focused on data fitting. Data are generated by means of polynomials $x^6 - x^4 + x^2$ and $x^5 - 2x^3 + x$. Equations 14 - 17 are typical example of synthesized solutions, especially Equations 14 and 16 are solutions with general constants K and Equations 15 and 17 are their fitted versions. In Figure 16, fitted data-dots and fitting by synthesized programs (black lines) is depicted. Another study - Booleans even-3-parity and symmetry problems were selected for comparative study and are fully reported in Zelinka et al. (2004) and Zelinka & Oplatkova (2004). In general, Boolean even-k-parity problems means that if the number of logical inputs of value True is even, then the output is True. If number of logical inputs of value True is not even, then the output is False. Number of all possible inputs (combinations) from $2^3 = 8$ for 3-parity problem to $2^6 = 64$ for a 6-parity problem. Truth table for 3-parity problem is given in Table 3. Symmetry problem has been investigated in the same way. Output of this logical function is True whenever True and False values are symmetrically distributed on inputs, see Koza (1998) andKoza et al. (1999). A typical example of synthesized solution is Equation 18. The full report of this comparative study is in Zelinka et al. (2004) and Zelinka & Oplatkova (2004).

$$\frac{xK[[5]]K[[6]](x(-K[[18]])-K[[19]]+2x)}{\frac{K[[23]](K[[24]]+x)}{x^2}+\frac{x(x-K[[20]])}{K[[21]]+K[[22]]}}+\left(\frac{K[[9]](xK[[10]]+x)(K[[25]]+x)}{K[[7]]K[[8]]}+K[[2]]-x\right)$$
$$(xK[[11]]-K[[12]]+x)-\frac{xK[[4]]\left(-K[[13]]+K[[14]]+K[[17]]+\frac{K[[15]]}{K[[16]]}-x\right)}{-K[[3]]-x}-K[[1]] \tag{14}$$

$$\frac{0.00621529x(0.793939-1.x)}{-1.x-0.934705}+(0.465773(x+2.82445)x-1.x-7.45208)(0.181218-0.749217x)+$$
$$\frac{2.9596(0.432881x-3.70673)x}{\frac{0.21213(x+13.054)}{x^2}+0.456758(x-0.562963)x}+1.27265 \tag{15}$$

$$x(x^2(x(K[[7]]+x)-K[[2]])+x(K[[4]]-K[[5]])+xK[[6]]+K[[1]]-K[[3]]-1) \tag{16}$$

$$x\left(1.-2.193908007555499^{-16}x+x^2(-2.+x(4.66960974116765^{-16}+x))\right) \tag{17}$$

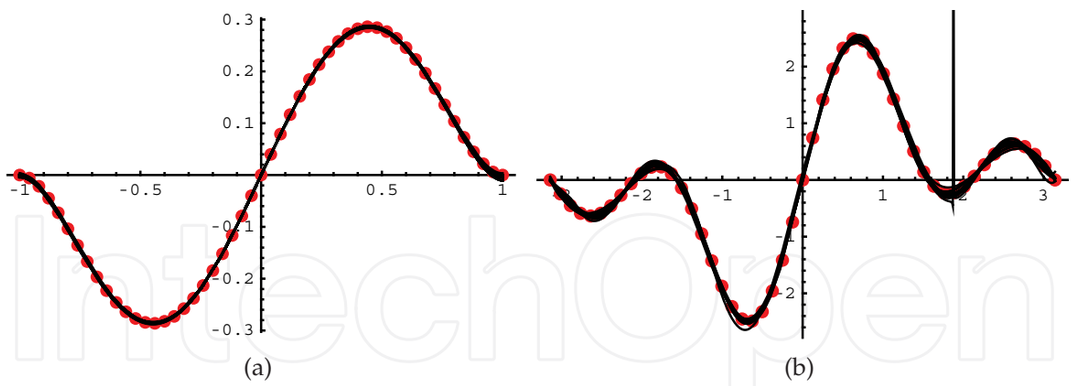


Fig. 16. Quintic (a) and Sinus 3 (b), all 50 successful simulations (black lines) very well fit the measured data (red dots)

Input 1	Input 2	Input 3	Output
True	True	True	False
True	True	False	True
True	False	True	True
False	True	True	True
True	False	False	False
False	True	False	False
False	False	True	False
False	False	False	True

Table 3. Truth table for Boolean even-3-parity problem according to Koza (1998)

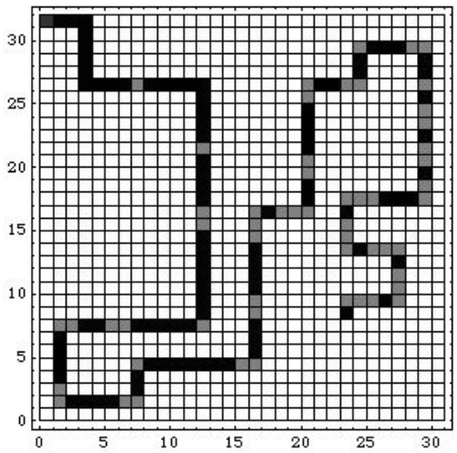
$$
\begin{aligned}
&((A \wedge (((((B \wedge A) \vee (C \wedge A) \vee (\neg C \wedge B) \vee (\neg C \wedge \neg A)) \,\overline{\wedge}\, ((B \wedge A) \vee (C \wedge A) \\
&\vee (\neg C \wedge B) \vee (\neg C \wedge \neg A))) \vee (B \bar{\vee} A)) \,\overline{\wedge}\, ((A \vee (B \wedge A) \vee (C \wedge A) \\
&\vee (\neg C \wedge B) \vee (\neg C \wedge \neg A)) \wedge B \wedge ((B \wedge A) \vee (C \wedge A) \vee (\neg C \wedge B) \vee (\neg C \wedge \neg A))))) \\
&\overline{\wedge} C) \,\overline{\wedge}\, (C \vee (C \bar{\vee} (A \,\overline{\wedge}\, (C \,\overline{\wedge}\, ((B \wedge A) \vee (C \wedge A) \vee (\neg C \wedge B) \\
&\vee (\neg C \wedge \neg A))))))
\end{aligned}
\tag{18}
$$

2.3 Santa Fe trail and Artificial ant

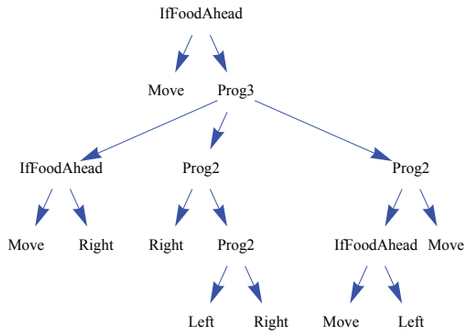
Another test of AP has been done on the setting of an optimal trajectory of an artificial ant on the so-called SantaFe Trail. In this experiment, SOMA (Zelinka, 2004) and DE (Price, 1999) were selected as the two EA's for simulation. In space and other comparative industries, the number of robots used for specific tasks are increasing daily. Subsequently, precise tasks such as optimal trajectory setting of the robot is a very desirable attribute. The problem description for this preliminary study is taken from Koza (1998). The aim of this experiment is that a robot, in this case, an artificial ant, should go through a defined trail and eat all food which is there. One of the possible ways for the ant to transverse is the so-called SantaFe trail, which is demonstrated in Figure 19. The SantaFe trail is defined as a 32 x 31 field, where food is set out. In Figure 19, a black field is food for the ant. The grey one is basically the same as a white field but for clarity, the grey color was used. The grey fields represent obstacles (fields without food on the road) for the ant. If obstacles do not exist, the ant would have a clear run. Ideally,

it would be the best to precede the ant and ascertain if any food is present in the current path. If food is present, then the ant would follow and eat it. If no food is found, the ant would change trajectory. This cycle would repeat till all the food is devoured.
However, in the real world, robots face obstacles during their movement. Therefore, this analogy is also applied in this research. The first obstacle which the ant has to overcome is the simple hole (position [8,27] in Figure 19. The second obstacle are the two holes in the line (positions [13,16] and [13,17]), or three holes ([17,15], [17,16], [17,17]). Next obstacle are the holes in the corners - one (position [13,8]], two ([1,8],[2,8]) and three holes ([17,15], [17,16], [17,17]). In the case of AP's application, all simulations (i.e. DE as well as SOMA) have shown very good performances and results were fully comparable with techniques like GP. For full report see Oplatkova & Zelinka (2006).

$$IfFoodAhead[Move, Prog3[IfFoodAhead[Move, Right], Prog2[Right, Prog2[Left, Right]], Prog2[IfFoodAhead[Move, Left], Move]]] \quad (19)$$



(a)



(b) Tree representation of the solution from the Equation 19

Fig. 17. SantaFe trail (a) and its solution (b)

2.4 Deterministic chaos synthesis

Zelinka et al. (2008) introduces the notion of chaos synthesis by means of EA's and develops a new method for chaotic systems synthesis. This method is similar to GP and GE and is applied with three EA's: DE, SOMA, and GA. The aim of this research is to synthesize new and "simple" chaotic systems based on some elements contained in a pre-chosen existing chaotic system and a properly defined cost function. The research consists of 11 case studies: the aforementioned three EA's in 11 versions. For all algorithms, 100 simulations of chaos synthesis were repeated and then averaged to guarantee the reliability and robustness of the proposed method. The most significant results are carefully selected, visualized and reported in this section.
Methods used in generating new chaotic systems from physical systems or from "manipulations" (e.g., control and parameter estimation (Grebogi & Lai, 1999; Hu et al., 1999) are based on deterministic mathematical analysis. Along with these classical methods, there

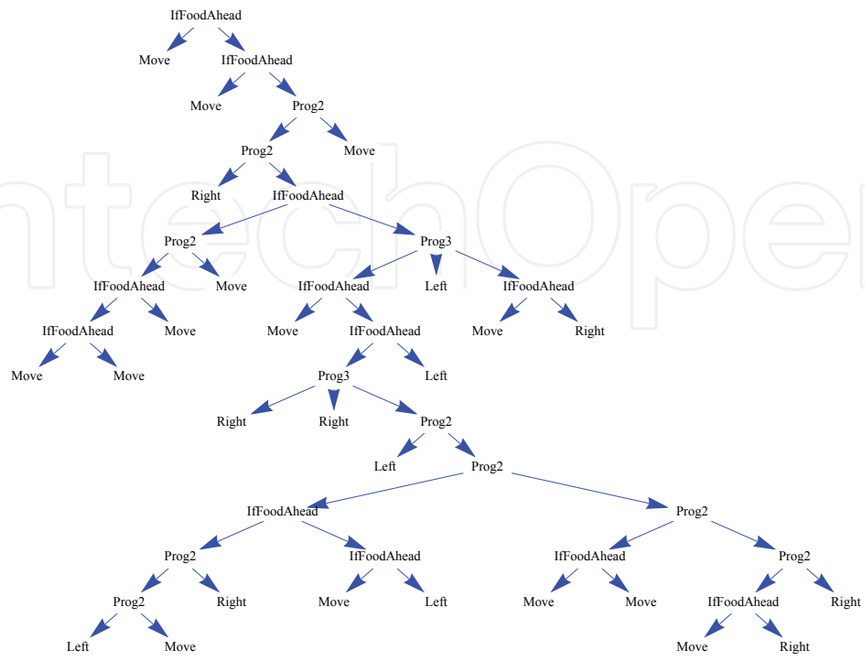


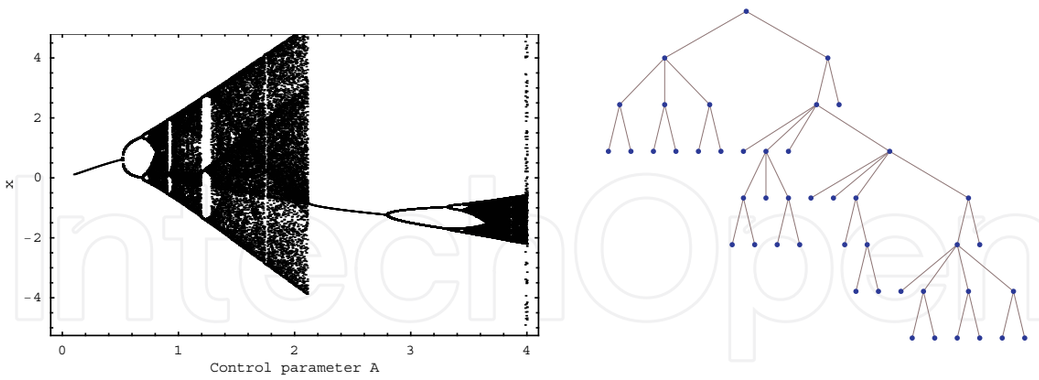
Fig. 18. Another more complex solution of the SanteFe Trail

are also numerical methods based partly on deterministic and partly on stochastic methods, called EA's (Back et al., 1997). EA's were used in searching solutions in many computationally hard problems including classes of P and NP problems (Garey & Johnson, 1979). In chaos studies, they were also used for chaos control as in Zelinka (2005), Richter (2002) or Zelinka (2006), amongst others.

The aim of the this research is to show that EA-based on GP-like techniques is capable of synthesizing chaotic behavior in the sense that the mathematical descriptions of chaotic systems are synthesized symbolically by means of EA's. The ability of EAs to successfully solve this kind of black-box problems has a proven track record (see, for example, Zelinka & Nolle (2005)), and is reinforced in this chapter.

Based on statistically robust simulations, a lot of interesting chaotic systems has been synthesized. In Zelinka et al. (2008), a number of chaotic systems in mathematical description as well as its bifurcation diagrams were reported. As an example Equation21 and Figure 19 can be used. The extended case study (for more EAs and continuous systems) is reported in the book chapter (Zelinka et al., 2010) where the synthesis of the continuous systems are taken into consideration.

$$\frac{Ax - A - x}{\frac{A}{2x} - \frac{A(A-x)}{Ax+2A-x+1} + A + x} \tag{20}$$



(a) (b)

Fig. 19. Bifurcation diagram of the synthesized system (a) and solution (see Equation 21) in the tree representation (b)

2.5 Control law synthesis

Another application of AP (Oplatkova, Senkerik, Belaskova & Zelinka, 2010; Oplatkova, Senkerik, Zelinka & Holoska, 2010a;b) is focused on the synthesis of control law for discrete chaotic system. The interest in the control of chaotic systems has been an active area of research during the past decade. One of the first and important initial studies, of EA for control use was reported in Zelinka, Senkerik & Navratil (2009), Zelinka et al. (2006b) and Zelinka et al. (2006a), where the control law was based on the Pyragas method: Extended delay feedback control - ETDAS (Pyragas, 1995). Those papers were focused on the tuning of several parameters inside the control technique for a chaotic system. Compared to that, a presented paper Oplatkova, Senkerik, Belaskova & Zelinka (2010); Oplatkova, Senkerik, Zelinka & Holoska (2010a;b) shows a possibility as to how to generate the whole control law (not only to optimize several parameters) for the purpose of stabilization of a chaotic system. The synthesis of control is inspired by the PyragasâĂŹs delayed feedback control technique (Just, 1999; Pyragas, 1990). Unlike the original OGY control method (Ott et al., 1990), it can be simply considered as a targeting and stabilizing algorithm together in one package. Another big advantage of the Pyragas method is the amount of accessible control parameters. Instead of EA utilization, AP was used. Control law from the proposed system can be viewed as a symbolic structure, which can be created according the requirements for the stabilization of chaotic system. The advantage is that it is not necessary to have some “preliminary” control law and only to estimate its parameters. This system will generate the structure of the law also with suitable parameter values. The articles Oplatkova, Senkerik, Zelinka & Holoska (2010a), Oplatkova, Senkerik, Belaskova & Zelinka (2010) and Oplatkova, Senkerik, Zelinka & Holoska (2010b) contain 12 simulations with selected EAs applied with AP in order to synthesize suitable control law. Interested readers are recommended to read these articles.

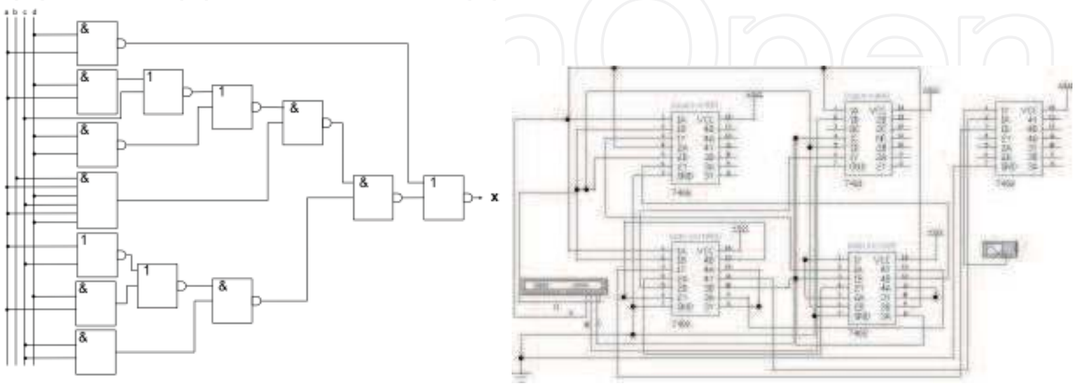
2.6 Algorithm synthesis

Our personal experiences have led to the hypothesis that a new algorithm (in this case evolutionary) can be created by AP. The main idea was that subroutines (operators) of selected EAs has been taken into consideration as symbolic objects (functions, ...), i.e. like members of the nonterminal set. Terminal set consisted of individuals (i.e. integer vector). The aim

of this simulation was to use EA's in such a way that with AP and a defined nonterminal and terminal sets, other versions of EA's were created. In Oplatkova (2009), the progress from the first study of the synthesis of a new EA to the simulations with more operators and higher dimensional systems is described. At the onset, DE was taken and its operators were separated into modules which were able to work independently. These operators were set up as simple functions for successful evaluations of AP. During the repeated simulations of AP, successful solutions as well as the original DE and other successful solutions (DE synthesis) were found. The next step continued with more operators from other evolutionary and stochastic algorithms such as SOMA, Hill Climbing and SA (Oplatkova, 2009). In this case, a new design of the cost function was utilized. With respect to the order of obtained cost values, the measurement was changed to minimize the difference between found extreme and the global. Penalization concerned to cost function evaluations was also applied. Simulations were performed in 2 dimensional space. This led to the third step, to use high dimensional benchmark functions as criterion in AP. The obtained results from higher dimensional test functions were then applied on 16 benchmark function in 2, 20 and 100 dimensional space for 4 found algorithms. Altogether, 192 simulations were carried out in 100 times repetition, equating to nearly 4×10^9 cost function evaluations. Results are depicted in tables and graphs in the Appendix of Oplatkova (2009). From results obtained, it can be stated that AP synthesized algorithms are able to optimize multimodal functions. Future research is open to add more operators, to tune parameters of found algorithms or to try to synthesize a new evolutionary operator itself.

2.7 Electronic circuits synthesis

In the diploma thesis of Strakos (2005), three electronic circuits were experimentally synthesized. The main point of this AP application was to confirm, that EA's with AP are possible to successfully design electronics circuits. In the first part of Strakos (2005) the general theory (GP, GE and AP) is outlined, while in the experimental part the synthesis of three electronic circuits (traffic light control, heat control and train station control) is described. All three control systems has been successfully designed by AP. Each winning solution was visualized as a circuit and hardware implementation (see for example Figure 20). In all three experiments (50 times repeated) AP had been observed to be capable of electronic circuit synthesis.



(a) Logical circuit designed by AP (b) HW reprezentation

Fig. 20. Circuit designed by AP (a) and its hardware implementation (b)

2.8 Nonlinear dynamical system identification

Synthesis, identification and control of complex dynamical systems are usually extremely complicated. When classics methods are used, some simplifications are required, which tends to lead to idealized solutions that are far from reality. In contrast, the class of methods based on evolutionary principles is successfully used to solve this kind of problems with a high level of precision. In this section, an alternative method of EA's, which has been successfully proven in many experiments like chaotic systems synthesis, neural network synthesis or electrical circuit synthesis. Zelinka, Senkerik, Oplatkova & Davendra (2009) discusses the possibility of using EA's for the identification of chaotic systems. The main aim of this work is to show that EA's are capable of the identification of chaotic systems without any partial knowledge of it's internal structure, i.e. based only on measured data. Two different EA's are presented and tested in a total of 10 versions. Systems selected for numerical experiments is the well-known logistic equation. For each algorithm and its version, repeated simulations were done, amounting to 50 simulations. Typical example of evolutionary identification is in Equation (21), (22) and visualization in Figure 21. According to obtained results, it can be stated that evolutionary identification is an alternative and promising way as to how to identify chaotic systems. Extended case study is also reported in Zelinka et al. (2010).

$$x\left(A+\frac{\left(-1-A+x-Ax+x^2-\frac{-A+x}{A}\right)\left(A+A(x+Ax)\right)}{2A^2}\right) \tag{21}$$

$$(1-x)x^2(3A+x-3Ax+Ax^2) \tag{22}$$

2.9 Neural network synthesis

Based on the case studies in Zelinka (2002c), Zelinka & Oplatkova (2003b), Zelinka & Oplatkova (2004) and Zelinka & Volna (2005), synthesis of Neural Networks (NN) was used for this study. Two problems, solved by AP were chosen : linearly and nonlinearly separable (XOR) problems. Concerning to previous simulations (Maniezzo, 1993), in this case of NN synthesis, simple elementary objects like $+, -, /, *, Exp, x_1, x_2, K$ were used. During evolution, more complicated structures from these simple objects of NN nature were created. As a learning algorithm, evolutionary fitting of weights was used, because the use of algorithms like back-propagation would be complicated on final neural structures. Figure 22 can be used as an example, where two examples of AP synthesized Artificial NN (ANN) is depicted. AP has been successfully used for different kind of problems. Positive results has shoved that AP can be used in this way. In the future more complex study on NN synthesis are going to be done by means of other EA's.

2.10 PDE solution synthesis

Zelinka (2001) outlines the use of AP on mathematical problems of civil engineering and problems of mathematical physics. In this paper, a few evolutionary simulations with AP has been done in order to get solutions of selected problems. This set of simulations was focused on ODE solving (see Figure 23 and Equation (23)). In Rektorys (1999), it is solved by the means of Ritz and Galerkin method on apriori selected functional base, which was orthogonal. Here it was solved with AP without any apriori demand.

This simulation was focused on finding the solution of a quite complicated ODE problem originating from mechanical engineering. Original solution obtained by means of Ritz

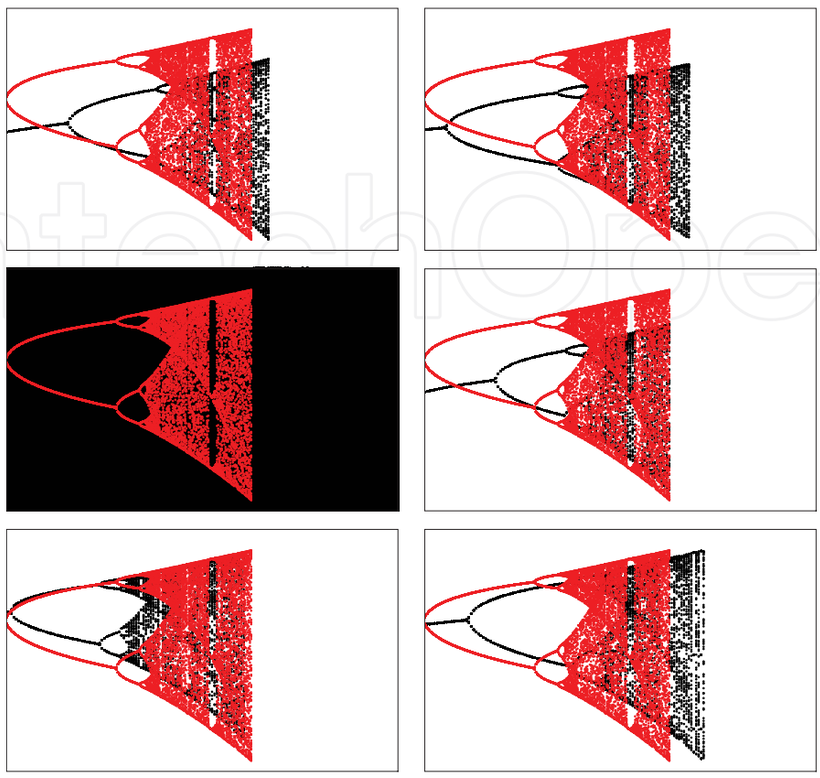
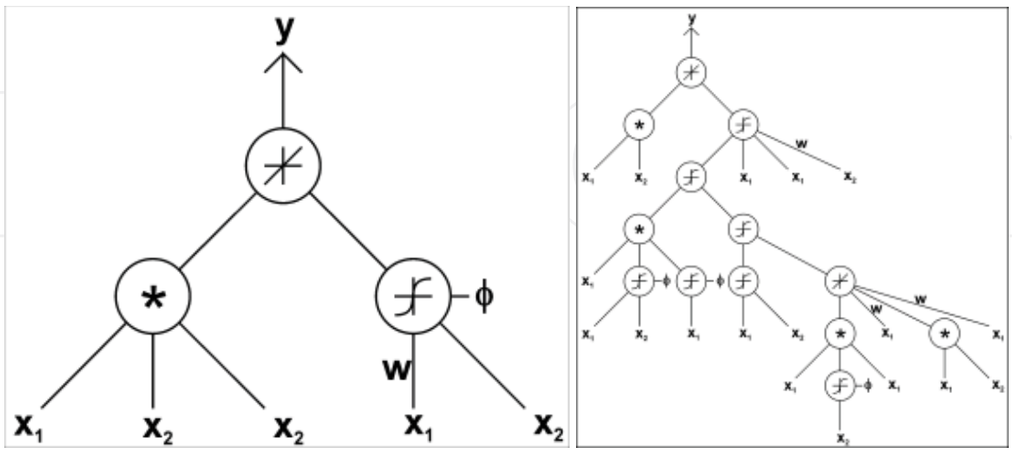


Fig. 21. Original (black fat points) and identified (red thin points) behavior



(a) ANN for linear separable problem (b) ANN for XOR problem

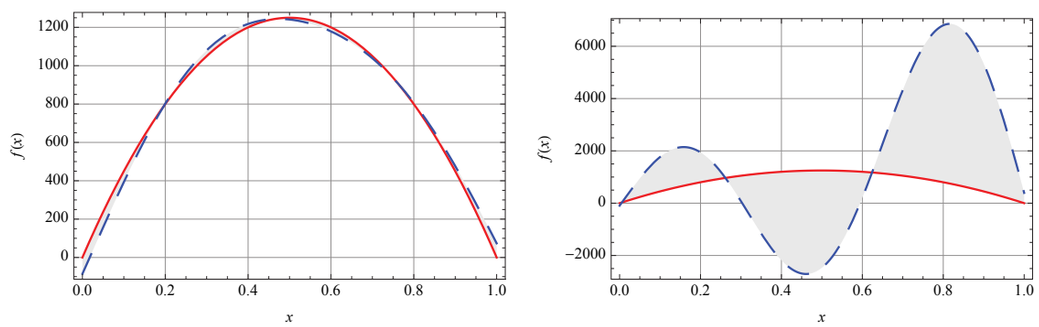
Fig. 22. Synthesized ANN by AP a) for linearly separable problems, b) for XOR problem

and Galerkin method (Rektorys, 1999) is depicted in Figure 23. Base in Hilbert space consisted of sinus functions and final solution of unknown u(x) was $u(x) = 1.243\sin(\pi x) + 0.0116\sin(2\pi x) + 0.00154\sin(3\pi x)$.
The original solution obtained by means of Ritz and Galerkin method and AP embedded with SOMA was used in two ways. In the first one, SOMA was used to estimate only parameters a, b, c of founded $u(x)$, i.e. $u(x) = a\sin(\pi x) + b\sin(2\pi x) + c\sin(3\pi x)$.. In the original solution all three coefficients were calculated by means of quite complicated Ritz and Galerkin methods. SOMA was able to find all three coefficients as is depicted in Figure 23. This problem was basically a classical optimization because functions were a priori known. This simulation was 100 times repeated and in all cases has lead to the same results. The second use of SOMA here was not focused only on parameter estimation. SOMA was used with AP on a complex set of functions $(sin, cos, ...)$ operators : $+, -, /, Power, \times$ and constants a, b, c to find their best combination i.e. to build up function fitting function $5000(x - x^2)$ as closely as possible, see Equations 24 and 25.

$$((4 + x)u(x)'')'' = 5000\left(x - x^2\right) \tag{23}$$

$$u = 1.243\sin(\pi x) + 0.0116\sin(2\pi x) + 0.00154\sin(3\pi x) \tag{24}$$

$$u = 1.243\sin(\pi x) - .3\sin(2\pi x) + .1\sin(3\pi x) \tag{25}$$



(a) The best solution (Equation 24) of the Equation 23

(b) Bad solution, see (Equation 25)

Fig. 23. Synthesized PDE solutions by AP a) the best solution, b) typical but not suitable solution observable at the beginning of the evolution. Original solution given by numerical Galerkin/Ritz method in the Hilbert space is represented by solid red line, while evolutionary synthesized solution is in dashed blue line.

3. Conclusion

Based on various applications of AP, it can be stated that AP seems to be powerful algorithmical equivalent of such methods like GP or GE. The AP method has been carefully tested during the last 9 years on various examples including selected examples from GP for comparative study. Results from all of experiments, partially reported here, confirm the fact that AP is possible to use for the same class of problems like another algorithms (GP, GE, etc.).

4. Acknowledgement

The following two grants are acknowledged for the financial support provided for this research: Grant Agency of the Czech Republic - GACR102/09/1680 and Grant of the Czech Ministry of Education - MSM7088352101.

5. References

Back, T., Fogel, D. & Michalewicz, Z. (1997). *Handbook of Evolutionary Computation*, Institute of Physics.

Garey, M. & Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, New York.

Grebogi, C. & Lai, Y. (1999). Controlling chaos, *in* H. Schuster (ed.), *Handbook of Chaos Control*, Wiley-VCH, New York.

Hu, G., Xie, F., Xiao, J., Yang, J. & Qu, Z. (1999). Control of patterns and spatiotemporal chaos and its application, *in* H. Schuster (ed.), *Handbook of Chaos Control*, Wiley-VCH, New York.

Johnson, C. (2004). Artificial immune systems programming for symbolic regression, *in* C. Ryan, T. Soule, M. Keijzer, E. Tsang & R. P. E. Costa (eds), *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 345–353.

Just, W. (1999). Principles of time delayed feedback control, *in* H. Schuster (ed.), *Handbook of Chaos Control*, Wiley-VCH, New York.

Koza, J. (1990). Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems., *Stanford University, Computer Science Department, Techincal Report* STAN-CS-90-1314.

Koza, J. (1998). Genetic programming, *MIT Press* .

Koza, J., Bennet, F., Andre, D. & Keane, M. (1999). *Genetic Programming III*, Morgan Kaufnamm, New York.

Koza, J., Keane, M. & Streeter, M. (2003). Evolving inventions, *Scientific American* pp. 40–47.

Lampinen, J. & Zelinka, I. (1999). Mechanical engineering design optimisation by differential evolution, *in* D. Corne, M. Dorigo & F. Glover (eds), *New Ideas in Optimization*, McGraw-Hill, London, UK, pp. 127–146.

Maniezzo, V. (1993). Searching among search spaces: Hastening the genetic evolution of feedforward neural networks, *in* C. R. R. R. F. Albrecht & N. C. Steele (eds), *Artificial Neural Nets and Genetic Algorithms*, Springer-Verlag.

O'Neill, M. & Brabazon, A. (2006). Grammatical differential evolution, *Proceedings of International Conference on Artificial Intelligence*, CSEA Press, pp. 231–236.

O'Neill, M. & Ryan, C. (2003). *Grammatical Evolution, Evolutionary Automatic Programming in an Arbitrary Language*, Springer-Verlag, New York.

Oplatkova, Z. (2005). Optimal trajectory of robots using symbolic regression, *Proceedings of 56th International Astronautics Congress*, Fukuoka, Japan.

Oplatkova, Z. (2009). *Metaevolution: Synthesis of Optimization Algorithms by means of Symbolic Regression and Evolutionary Algorithms*, Lambert Academic Publishing, New York.

Oplatkova, Z., Senkerik, R., Belaskova, S. & Zelinka, I. (2010). Synthesis of control rule for synthesized chaotic system by means of evolutionary techniques, *MEndel 2010*, Technical universitu of Brno, Brno, Czech Republic, pp. 91–98.

Oplatkova, Z., Senkerik, R., Zelinka, I. & Holoska, J. (2010a). Synthesis of control law for chaotic henon system - preliminary study, *ECMS 2010*, ECMS, Kuala Lumpur, Malaysia, pp. 277–282.

Oplatkova, Z., Senkerik, R., Zelinka, I. & Holoska, J. (2010b). Synthesis of control law for chaotic logistic equation - preliminary study, *AMS 2010*, ASM, Kota Kinabalu, Borneo, Malaysia.

Oplatkova, Z. & Zelinka, I. (2006). Investigation on artificial ant using analytic programming, *Proceedings of Genetic and Evolutionary Computation Conference*, Seattle, WA, pp. 949–950.

Oplatkova, Z., Zelinka, I. & Senkerik, R. (2008). Santa fe trail for artificial ant by means of analytic programming and evolutionary computation, *International Journal of Simulation, Systems, Science and Technology* Vol.9(No.3): 20–33.

Ott, E., Grebogi, C. & Yorke, J. (1990). Controlling chaos, *Phys. Rev. Lett.* Vol. 64: 1196 – 1199.

Price, K. (1999). An introduction to differential evolution, *in* D. Corne, M. Dorigo & F. Glover (eds), *New Ideas in Optimization*, McGraw-Hill, London, UK, pp. 79–108.

Pyragas, K. (1990). Continuous control of chaos by self-controlling feedback, *Phys. Lett. A*, Vol. 170: 421–428.

Pyragas, K. (1995). Control of chaos via extended delay feedback, *Physics Letters A* 206(206): 323–330.

Rektorys, K. (1999). *Variational methods in Engineering Problems and Problems of Mathematical Physics (Czech Ed.)*, Academia.

Richter, H. (2002). An evolutionary algorithm for controlling chaos: The use of multi-objective fitness functions, *in* M. Guervós, J. Panagiotis, A. Beyer, F. Villacanas & H. Schwefel (eds), *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 308–317.

Ryan, C., Collins, J. & O'Neill, M. (1998). Grammatical evolution: Evolving programs for an arbitrary language, *Lecture Notes in Computer Science*, First European Workshop on Genetic Programming.

Strakos, R. (2005). Design electronics circuits by means of evolution method, *Diploma thesis*, UTB Zlín, Zlin, Czech Republic.

Weisser, R. & Osmera, P. (2010a). Two-level tranpslant evolution, *17th Zittau Fuzzy Colloquium*, Zittau, Germany.

Weisser, R. & Osmera, P. (2010b). Two-level transplant evolution for optimization of general controllers, *New Trends in Technologies*, Sciyo.

Weisser, R., Osmera, P. & Matousek, R. (2010). Transplant evolution with modified schema of differential evolution : Optimization structure of controllers, *International Conference on Soft Computing MENDEL.*, Brno, Czech Republic.

Zelinka, I. (2001). Analytic programming by means of new evolutionary algorithms, *Proceedings of 1st International Conference on New Trends in Physics'01*, Brno, Czech Republic, pp. 210–214.

Zelinka, I. (2002a). Analytic programming by means of soma algorithm, *Proceedings of First International Conference on Intelligent Computing and Information Systems*, Cairo, Egypt, pp. 148–154.

Zelinka, I. (2002b). Analytic programming by means of soma algorithm, *Proc. 8th International Conference on Soft Computing*, VUT Brno, Mendel'02 Czech Republic, pp. 93–101.

Zelinka, I. (2002c). Analytic programming by means of soma algorithm, *First International Conference on Intelligent Computing and Information Systems*, Ain Shams University, ICICIS'02 Egypt.

Zelinka, I. (2004). Soma - self organizing migrating algorithm, *in* B. Babu & G. Onwubolu (eds), *New Optimization Techniques in Engineering*, Springer-Verlag, New York, pp. 167–218.

Zelinka, I. (2005). Investigation on evolutionary deterministic chaos control, *Proceedings of IFAC*, Prague, Czech Republic.

Zelinka, I. (2006). Investigation on realtime deterministic chaos control by means of evolutionary algorithms, *Proceedings of First IFAC Conference on Analysis and Control of Chaotic Systems*, Reims, France, pp. 211–217.

Zelinka, I., Chen, G. & Celikovsky, S. (2010). Chaos synthesis by evolutionary algorithms, *in* G. C. I. Zelinka, H. Richter & S. Celikovsky (eds), *Evolutionary Algorithms and Chaotic Systems*, Springer, Germany, pp. 357–398.

Zelinka, I., Guanrong, C. & Celikovsky, S. (2008). Chaos synthesis by means of evolutionary algorithms, *International Journal of Bifurcation and Chaos* Vol.18(No.4): 911–942.

Zelinka, I. & Nolle, L. (2005). Plasma reactor optimizing using differential evolution, *in* K. Price, J. Lampinen & R. Storn (eds), *Differential Evolution: A Practical Approach to Global Optimization*, Springer-Verlag, New York, pp. 499–512.

Zelinka, I. & Oplatkova, Z. (2003a). Analytic programming – comparative study, *Proceedings of Second International Conference on Computational Intelligence, Robotics, and Autonomous Systems*, Singapore.

Zelinka, I. & Oplatkova, Z. (2003b). Analytic programming - comparative study, *Proceedings of The second International Conference on Computational Intelligence, Robotics, and Autonomous Systems*, CIRAS'03 Singapore, p. paper ID PS040204.

Zelinka, I. & Oplatkova, Z. (2004). Boolean parity function synthesis by means of arbitrarry evolutionary algorithms - comparative study, *8th World Multiconference on Systemics, Cybernetics and Informatics*, SCI 2004 Orlando, USA.

Zelinka, I., Oplatkova, Z. & Nolle, L. (2004). Boolean symmetry function synthesis by means of arbitrarry evolutionary algorithms - comparative study, *18th European Simulation Multiconference*, Gruner Druck, GmbH, ESM 2004 Magdeburg, Germany, pp. 143–148.

Zelinka, I., Oplatkova, Z. & Nolle, L. (2005a). Analytic programming – symbolic regression by means of arbitrary evolutionary algorithms, *Int. J. of Simulation, Systems, Science and Technology* Vol. 6(No. 9): 44–56.

Zelinka, I., Oplatkova, Z. & Nolle, L. (2005b). Analytic programming - symbolic regression by means of arbitrary evolutionary algorithms, *In: Special Issue on Inteligent Systems of International Journal of Simulation, Systems, Science and Technology* Vol.6(No.9): 44–55.

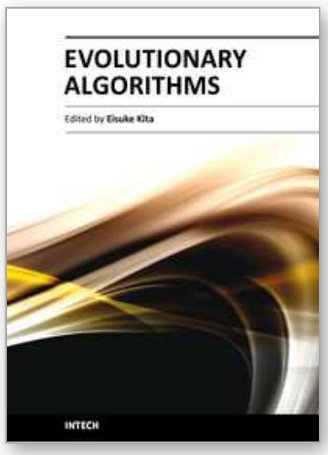
Zelinka, I., Senkerik, R. & Navratil, E. (2006a). Investigation on real time deterministic chaos control by means of evolutionary algorithms, *1st IFAC Conference on analysis and control of chaotic systems*, Université de Reims, CHAOS'06 Reims France, pp. 211–217.

Zelinka, I., Senkerik, R. & Navratil, E. (2006b). Optimization of feedback control of chaos by evolutionary algorithms, *1st IFAC Conference on analysis and control of chaotic systems*, Université de Reims, CHAOS'06 Reims France, pp. 97–102.

Zelinka, I., Senkerik, R. & Navratil, E. (2009). Investigation on evolutionary optimitazion of chaos control, *CHAOS, SOLITONS and FRACTALS* 40(1): 111–129. doi:10.1016/j.chaos.2007.07.045.

Zelinka, I., Senkerik, R., Oplatkova, Z. & Davendra, D. (2009). Evolutionary identification of chaotic system, *2nd IFAC Conference on analysis and control of chaotic systems*, Queen Mary, University of London, CHAOS'09 London UK.

Zelinka, I. & Volna, E. (2005). Neural network synthesis by means of analytic programming - preliminary results, *Proc. 11th International Conference on Soft Computing Mendel'05*, Technical universoty of Brno, Mendel'05 Brno Czech Republic, p. paper 504.



Evolutionary Algorithms

Edited by Prof. Eisuke Kita

ISBN 978-953-307-171-8
Hard cover, 584 pages
Publisher InTech
Published online 26, April, 2011
Published in print edition April, 2011

Evolutionary algorithms are successively applied to wide optimization problems in the engineering, marketing, operations research, and social science, such as include scheduling, genetics, material selection, structural design and so on. Apart from mathematical optimization problems, evolutionary algorithms have also been used as an experimental framework within biological evolution and natural selection in the field of artificial life.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Ivan Zelinka, Donald Davendra, Roman Senkerik, Roman Jasek and Zuzana Oplatkova (2011). Analytical Programming - a Novel Approach for Evolutionary Synthesis of Symbolic Structures, Evolutionary Algorithms, Prof. Eisuke Kita (Ed.), ISBN: 978-953-307-171-8, InTech, Available from: http://www.intechopen.com/books/evolutionary-algorithms/analytical-programming-a-novel-approach-for-evolutionary-synthesis-of-symbolic-structures

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen