SANTA FE TRAIL FOR ARTIFICIAL ANT BY MEANS OF ANALYTIC PROGRAMMING AND EVOLUTIONARY COMPUTATION

ZUZANA OPLATKOVA, IVAN ZELINKA, ROMAN SENKERIK

Faculty of Applied Informatics
Tomas Bata University in Zlin
Nad Stranemi 4511, 760 05 Zlin
Czech Republic
email: {oplatkova, zelinka, senkerik}@fai.utb.cz

Abstract: This paper deals with the usage of an alternative tool for symbolic regression – analytic programming which is able to solve various problems from the symbolic domain as well as genetic programming and gramatical evolution. The main tasks of analytic programming in this paper, is setting an optimal trajectory for a robot (artificial ant on Santa Fe trail). In this contribution main principles of analytic programming are described and explained. In the second part of the article is in detail described how analytic programming was used for setting an optimal trajectory for an artificial ant according the user requirements. An ability to create so called programs, as well as genetic programming or grammatical evolution do, is shown in that part. In this contribution three evolutionary algorithms were used – Self Organizing Migrating Algorithm, Differential Evolution and Simulated Annealing. The total number of simulations was 150 and results show that the first two used algorithms were more successful than not so robust Simulated Annealing.

Keywords: Analytic Programming, symbolic regression, optimization, evolutionary algorithms

1. INTRODUCTION

The term "symbolic regression" represents a process during which measured data is fitted and a suitable mathematical formula is obtained in an analytical way. This process is well known for mathematicians. It is used when a mathematical model of unknown data is needed. For long time symbolic regression was a domain of humans but in few last decades computers have gone to foreground of interest in this field. Firstly, the idea of symbolic regression done by means of computer has been proposed in Genetic Programming (GP) by John Koza [Koza 1998], [Koza 1999], [www.genetic-programming.org]. The other two approaches are Grammatical Evolution (GE) developed by Conor Ryan [O'Neill, Ryan 2003], [O'Sullivan, Ryan 2002], [www.grammatical-evolution.org] and here described Analytic Programming (AP) designed in [Zelinka 2002], [Zelinka, Oplatkova 2003], [Zelinka, Oplatkova, Nolle 2004], [Zelinka, Oplatkova, Nolle 2005].

Genetic programming was the first tool for symbolic synthesis of so called programs done by means of computer instead of humans. The main idea comes from genetic algorithms (GA) [Davis 1996] which John Koza uses in his GP. The ability to solve very difficult problems was proved many times, and hence, GP today can be applied, e.g. to synthesize highly sophisticated electronic circuits, robot trajectory, biochemistry problems, etc. [Koza 1999].

The other tool is GE which was developed in last decade of 20th century by Conor Ryan. Gramatical evolution has one advantage compared to GP and this is ability to use arbitrary programming language not only LISP as is in the case of the cannonical version of GP. In contrast to other evolutionary algorithms, GE was used with a few search strategies with a binary representation of the populations [O'Sullivan, Ryan 2002], as well as with other algorithms like [O'Neill M., Brabazon A. 2006], [O'Neill, Leahy, Brabazon, 2006]. Other interesting investigations using symbolic regression were carried out by Johnson [Johnson 2003] working on Artificial Immune Systems and Probabilistic Incremental Program Evolution (PIPE) [Salustowicz, Schmidhuber, 1997] generates functional programs from an adaptive probability distribution over all possible programs. To Grammatical Evolution foreruns GADS which solves the approach to grammar [Paterson, 2003], [Paterson, Livesey, 1996]. Other example is artificial immune system programming which was evolved for symbolic regression from an evolutionary algorithm called artificial immune systems [Johnson, 2003]. Approaches which differ in representation and grammar are described in gene expression programming [Ferreira, 2006], multiexpression programming [Oltean, Grosan, 2003], meta-modelling by symbolic regression and pareto Simulated Annealing [Stinstra, Rennen, Teeuwen, 2006]. To the group of hybrid approaches, belongs mainly numerical methods connected with evolutionary systems, e.g. [Davidson, Savic, Walters, 2003].

This contribution demonstrates use of a method which is independent on computer platform, programming language and can use any evolutionary algorithm (as demonstrated by [Zelinka 2002], [Zelinka, Oplatkova 2003], [Zelinka, Oplatkova, Nolle 2004], [Zelinka, Oplatkova, Nolle 2005]) to find an optimal solution of required task.

2. ANALYTIC PROGRAMMING

2.1. Description

Basic principles of the AP were developed in the 2001. Until that time mainly GP and GE existed. GP uses genetic algorithms while AP can be used with any evolutionary algorithm, independently on individual representation. To avoid any confusion based on the use of names according to the used algorithm, name - Analytic Programming was chosen, because AP stands for synthesis of analytical solution by means of evolutionary algorithms [Zelinka 2002], [Zelinka, Oplatkova 2003], [Zelinka, Oplatkova, Nolle 2004], [Zelinka, Oplatkova, Nolle 2005].

AP was inspired, in general, by numerical methods in Hilbert spaces and by GP. Principles of AP [Zelinka, Oplatkova, Nolle 2005] are somewhere between these two philosophies. From GP an idea of evolutionary creation of symbolic solutions is taken into AP while from Hilbert spaces an idea of synthesis of more complicated functions from elementary functions is adopted into AP. Analytic programming is based as well as GP on the set of functions, operators and so-called terminals, which are usually constants or independent variables like for example:

- functions: sin, tan, And, Or …
- operators: +, -, *, /, dt,…
- terminals: 2.73, 3.14, t,…

All these "mathematical" objects create a set which AP tries to synthesize the appropriate solution from. The set of mathematical objects are functions, operators and so-called terminals (usually constants or independent variables). All these objects are mixed together as is shown in Fig. 1 and consists of functions with different number of arguments. The set is called for article purposes "general functional set" – GFS because the content of this set is very variable. The structure of GFS is furcated i.e. it is created by subsets of functions according to the number of their arguments. The content of GFS is dependent only on a 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 only three arguments, GFS_{0arg} represents only terminals, etc.

GFS_{all} = {LerchPhi, BetaRegularized, +, -, /, *, cos, tan, sin, logΓ , x, t, π}

GFS_{2arg} = {+, -, /, *, cos, tan, sin, logΓ , x, t, π}
GFS_{1arg} = {cos, tan, sin, logΓ , x, t, π}
GFS_{0arg} = {x, t, π}

Fig. 1: General Function Set – GFS

This furcated structure is necessary the main principle of AP to be able to work without any difficulties. The core of AP is based on discrete set handling, proposed in [Zelinka 2004a], [Lampinen, Zelinka 1999], see Fig. 2. Discrete set handling (DSH) shows itself as universal interface between EA and symbolically solved problem. That is why AP can be used almost by any evolutionary algorithm.

Briefly said, DSH works with integer indexes which represent numerical or non-numerical expressions (operators, functions,…) in a discrete set. This index then serves like a pointer into a discrete set. Based on that, appropriate objects are chosen for cost function evaluation [Lampinen, Zelinka 1999]. During an evolutionary process only indexes are used for all evolutionary operations. Objects from the discrete set are used (by means of integer index) only in cost function, where is according to integer index a symbolic structure is synthesized and consequently evaluated.

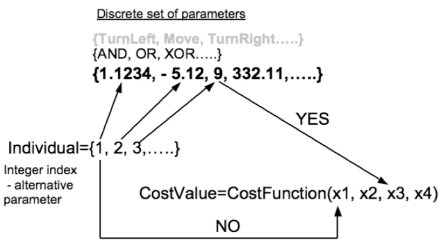
Discrete set of parameters
{TurnLeft, Move, TurnRight.....}
{AND, OR, XOR.....}
{1.1234, - 5.12, 9, 332.11,.....}
YES
Individual={1, 2, 3,.....}
Integer index - alternative parameter
CostValue=CostFunction(x1, x2, x3, x4)
NO

Fig. 2: Discrete set handling

2.2. Mapping method in AP

The nested 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 experiencies with the problem.
The important part of the 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 said, it is mapping from an individual domain into a program domain. This mapping consists of two main parts. The first part is called discrete set handling (DSH) and the second one are security procedures which do not allow to synthesize pathological programs.

Discrete set handling proposed in [Zelinka 2004a], [Lampinen, Zelinka 1999] is used for creating integer indeces which are used as values for the evolutionary process. 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 above mentioned mapping which transforms arbitrary individual into a program. Individuals in the population consist of integer parameters, i.e. an individual is an integer index pointing into GFS.

Analytic programming is basically a series of function mapping. In Fig. 3 an artificial example is demonstrated how a final function is created from an integer individual.

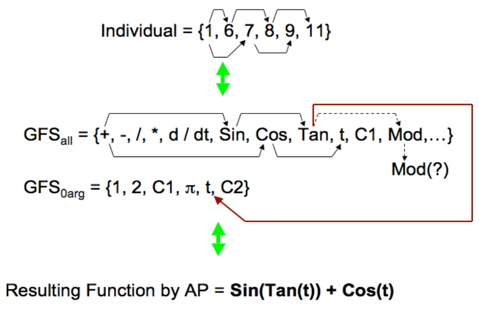
Individual = {1, 6, 7, 8, 9, 11}
GFS$_{all}$ = {+, -, /, *, d / dt, Sin, Cos, Tan, t, C1, Mod,…}
Mod(?)
GFS$_{0arg}$ = {1, 2, C1, π, t, C2}
Resulting Function by AP = Sin(Tan(t)) + Cos(t)

Fig. 3: Main principles of AP

Number 1 in the individual first position means that the operator "+" from GFS$_{all}$ is used. Because the operator "+" must have at least two arguments next two index pointers 6 (sin from GFS$_{all}$) and 7 (cos from GFS$_{all}$) 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$_{all}$) and 9 (t from GFS$_{all}$) are dedicated to sin and cos function. Because variable t is used as an argument of cos, this part of resulting function is closed (t is zero-argument) in its AP development. One-argument function tan remains and because there is one unused pointer 11 tan should be mapped on "Mod (modulo)" which is on 11th position in GFS$_{all}$. But Modulo needs other arguments and individual has no other indexes. Therefore subroutines are used which are described better in next paragraph and instead of modulo is used variable "t" from GFS$_{0arg}$.

To avoid synthesis of pathological functions a few security "tricks" is used in AP. The first one is that GFS consists of subsets containing functions with the same number of arguments. Existence of this furcated structure is used in the special security subroutine which is measuring how far the end of individual is and according to these objects from different subsets are selected to avoid pathological function synthesis. Precisely if more arguments are desired than 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 that the last argument for one argument function will not be a terminal (zero-argument function). If pointer is bigger than length of subset, i.e. pointer is 5 and is used GFS$_0$, then element is selected according to *element = pointer_value mod number_of_elements_in_GFS$_0$*. In this example case selected element would be variable *t* (see GFS$_0$ in Fig. 1).

GFS need not to be constructed only from clear mathematical functions as is demonstrated but also other user-defined functions such as logical functions, functions which represent elements of electrical circuits or robot movement commands can be used.

2.3. Crossover, Mutations and Other Evolutionary Operations

During evolution of a population a 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 evolutionary algorithm. Analytic programming does not contain them in any point of view of its internal structure. Analytic programming is created like a superstructure of EAs for symbolic regression independent on their algorithmic structure. Operations used in evolutionary algorithms are not influenced by AP and vice versa. For example if Differential Evolution (DE) is used for symbolic regression in AP then all evolutionary operations are done according to the DE rules.

2.4. *Versions of AP*

Today, AP exists in three versions. In all three versions the same sets of functions, terminals, etc. as Koza use in GP Koza [Koza 1998], [Koza 1999], [www.genetic-programming.org] are necessary for the program synthesis. The second version (AP_{meta}, lets call the first version AP_{basic}) is modified in the sense of constant estimation. For example, when Koza uses in so called sextic problem [Koza 1999] randomly generated constants, AP uses only one, called "K", which is inserted into the formula at various places by evolutionary processing. The function can look like as in (1).

$$\frac{x-K}{\pi^{K}} \tag{1}$$

When the program is synthesized, then all "K" are indexed so that K_1, K_2, …, K_n, are obtained in formula (2), and then all K_n are estimated by the second evolutionary algorithm and result is in (3).

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

$$\frac{x-1.289}{\pi^{-112}} \tag{3}$$

Because EA "works under" EA (i.e. EA_{master}► program► K indexing ►EA_{slave} ►estimation of K_n), this version is called AP with metaevolution - AP_{meta}. As this version was quite time consuming, another modification of AP_{meta} was done extending the second version by estimation of K. It is done by suitable methods of nonlinear fitting (AP_{nf}). This method has shown the most promising performance when unknown constants are present.

2.5. *Security procedures*

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

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

2.6. *Similarities and Differences*

Because AP was partly inspired by GP, then between AP, GP and GE are some differences as well as some logical similarities. A few of the most important ones are:

Similarity
- Synthesized programs: AP as well as GP and GE is able to do symbolic regression in a general point of view. It means that output of AP is according to all important simulations [Zelinka 2002], [Zelinka, Oplatkova 2003], [Zelinka, Oplatkova, Nolle 2004], [Zelinka, Oplatkova, Nolle 2005] similar to programs from GP and GE (see www.fai.utb.cz/people/zelinka/ap).
- Functional set: AP_{basic} operates in principle on the same set of terminals and functions as GP or GE.
-

Differences
- AP_{meta} or AP_{nf} use universal constant K (difference) which is indexed after program synthesis.
- Individual coding: coding of an individual is different. Analytic programming uses an integer index instead of direct representation as in canonical GP. Grammatical evolution uses binary representation of an individual, which is consequently converted into integers for mapping into programs by means of BNF [O'Neill, Ryan 2003].
- Individual mapping: AP uses discrete set handling, [Zelinka, Oplatkova, Nolle 2005] while GP in its fundamental form uses direct representation in Lisp [Koza 1998] and GE uses grammar - Backus-Naur form (BNF) [O'Neill, Ryan 2003].
- Constant handling: GP uses a randomly generated subset of numbers – constants, GE utilizes user determined constants and AP uses only one constant K for AP_{meta} and AP_{nf}, which is estimated by other EA or by nonlinear fitting.
- Security procedures: 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 of the end of the individual. If a pathological function is synthesized in GP, then synthesis is repeated. In the case of GE, when the end of an individual is reached, then mapping continues from the individual beginning, which is not the case of AP. It is designed so that a non-pathological program is syntesized before the end of the individual is reached (maximally when the end is reached).

2.7. Selected Solved Problems

During AP development and research simulations, a lot of various kind of programs has been synthesized. In (3) a mathematical formula is shown to demonstrate complexity of synthesized formulas, which were randomly generated amongst 1000 formulas to check if the final structure is free of pathologies – i.e. if all functions has the right number of arguments, etc. In this case no attention was paid on mathematical reasonability of the following test programs based on clear mathematical functions. In the previous text, a different approach to the symbolic regression called analytic programming was described. Based on its results and structure, it can be stated that AP seems to be a universal candidate for symbolic regression by means of different search strategies. Problems on which AP was utilized were selected from test and theory problems domain as well as from real life problems. The selected issues are following:

- Random synthesis of function from GFS, 1000 times repeated. The aim of this simulation was to check if pathological function can be generated by AP. In this simulation randomly generated individuals were created and consequently transformed into programs and checked for their internal structure. No pathological program was identified [Zelinka 2002].
- sin(t) approximation was repeated 100 times. Here AP was used to synthesize the program function sin(x) fitting [Zelinka 2002].
- $\left|\left|\cos(t)\right| + \sin(t)\right|$ approximation was repeated 100 times, the same like in the previous example. Main aim was again fitting of dataset generated by a given formula [Zelinka 2002].
- Solving of ordinary differential equations (ODE): u''(t) = cos(t), u(0) = 1, u(π) = -1, u'(0) = 0, u'(π) = 0, was repeated 100 times, in that case AP was looking for suitable function, which would solve this case of ODE [Zelinka 2002].
- Solving of ODE: ((4 + x)u''(x))'' + 600u(x) = 5000(x-x2), u(0)=0, u(1)=0, u''(0)=0, u''(1)=0, was repeated 5 times (due to longer time of simulation in the Mathematica® environment). Again as in the previous case, AP was used to synthesize a suitable function – solution of this kind of ODE. This ODE was used from and represents a civil engineering problem in reality [Zelinka 2002].
- Boolean even and symmetry problems according to the [Koza 1999] for comparative reasons [Zelinka, Oplatkova 2003], [Zelinka, Oplatkova, Nolle 2004].
- Sextic and Quintic problems [Zelinka, Oplatkova 2003].
- Simple neural network synthesis by means of AP – a simple few layered NN synthesis was tested by AP [Zelinka, Varacha, Oplatkova, 2006]
- Optimal robot trajectory estimation [Oplatkova 2005], [Oplatkova, Zelinka 2006].

Such elementary objects are usually simple mathematical operators (+, -, *, …), simple functions (sin, cos, And, Nor, …), user-defined functions (simple commands for robots – MoveLeft, TurnRight, …), etc. Output of symbolic regression is a more complex "object" (formula, function, command,…), solving a given problem like data fitting of so called Sextic and Quintic problem described by Eq. (4), [Koza 1999], [Zelinka, Oplatkova 2003], randomly synthesized function Eq. (5) [Zelinka, Oplatkova 2003], Boolean problems of parity and symmetry solution (basically logical circuits synthesis) Eq. (6) [Koza 1999], [Zelinka, Oplatkova, Nolle 2005] or synthesis of quite complex robot control command Eq. (7) [Koza 1998], [Oplatkova 2005]. Equations (4) - (7) mentioned here are just only a few samples numerous repeated experiments done by AP and are used to demonstrate how complex structures can be produced by symbolic regression in general sense for different problems.

$$x\left(K_1 + \frac{\left(x^2 K_3\right)}{K_4\left(K_5 + K_6\right)}\right) * \left(-1 + K_2 + 2x\left(-x - K_7\right)\right) \qquad (4)$$

$$\sqrt{t}\left(\frac{1}{\log(t)}\right)^{\sec^{-1}(1.28)} \log^{\sec^{-1}(1.28)}\left(\sinh\left(\sec\left(\cos(1)\right)\right)\right) \qquad (5)$$

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]] (6)

Prog2[Prog3[Move, Right, IfFoodAhead[Left, Right]], IfFoodAhead[IfFoodAhead[Left, Right], Prog2[IfFoodAhead[IfFoodAhead[IfFoodAhead[Left, Right], Right], Right], IfFoodAhead[Prog2[Move, Move], Right]]]] (7)

Remaining text in this article is investigation on evolutionary synthesis of robot commands, which is well known in genetic programming like Santa Fe trail of artificial ant.

3. PROBLEM DESIGN

3.1. *Santa Fe Description*

The Santa Fe trail, demonstrated in Fig. 4, was chosen from [Koza 1999] to make a comparative study with the same problem which was solved by Koza in Genetic Programming.

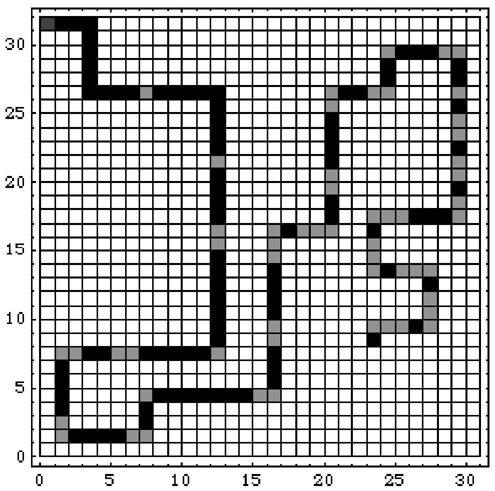

Fig. 4: Santa Fe trail

The aim of the task is that an artificial ant should go through defined trail and eat all food what is there.

The SantaFe trail is defined as a 32 x 31 fields where food is set out. In Fig. 4 a black field is food for the ant. The grey one is basically the same as a white field but for clearness was used the grey colour. The grey fields represent obstacles (fields without food on the road) for the ant. If there would not be these holes the ant could go directly through the way. It would be enough to go and see before ant if there is food. If yes ant would go straight and eat the bait. If not it would turn around and see where is food and the cycle would repeat till the ant would eat the last bait.

In the real world robots have obstacles in their moving. Therefore also in this case such approach was chosen. The first problem which ant has to overcome is the simple hole (position {8, 27} in [Koza 1999]). Second one is two holes in the line (positions {13,16} and {13,17}), or three holes ({17,15}, {17,16}, {17,17}). Next problem is holes in the corners – one (position {13,8}, two ({1,8},{2,8}) and three holes ({17,15}, {17,16}, {17,17}).

3.2. *Set of functions*

The set of functions used for movements of the ant is following. As a set of variables GFS_0, i. e. in the case of this article functions, which provide moving of an ant, without any argument which could be add during the process of evolution.
The set consist of

- GFS_0 = {Left, Right, Move},

where
GFS_0 – a set of variables and terminals, zero argumented functions GFS_{0arg}
Left – function for turning around in the anticlockwise direction
Right – function for turning around in the clockwise direction
Move – function for moving straight and if a bait is in the field where the ant is moved, it is eaten.

This set of functions is not enough to make successfully a desired task. More functions are necessary. Then a GFS_2 and GFS_3 were set up.

- GFS_2 = {IfFoodAhead, Prog2}
- GFS_3 = {Prog3}

Where the number in GFS means the arity of the functions inside, i.e. number of arguments which are needed to be evaluated correctly. Arguments are added to those functions during evolution process see above – Description of AP.
IfFoodAhead is a decision function – the ant controls the field in front of it and if there is food, the function in the field for truth argument is executed; otherwise function in false position is performed.
Prog2 and **Prog3** are the same function in the principle. They do 2 or 3 functions in the same time. These two functions were originally defined also in

Koza's approach but in AP it is necessary because of structure of generating the program.

3.3. Fitness function

The aim of the ant is to eat all food on the way. There are 89 baits. This is so called raw fitness. And the value of cost function (8) is calculated as a difference between raw fitness and a number of baits eaten by an ant [Koza 1999] which went through the grid according to just generated way.

$$CV = 89 - Number\_of\_Food \quad (8)$$

Number_of_Food - number of eaten baits by an ant according to synthesized way

The aim is to find such formula whose cost value is equal zero. To obtain an appropriate solution two constraints should be set up into a cost function. One is a limitation concerned to number of steps. It is not desired ant to go field by field in the grid. A requirement to the fastest way and the most effective is desired. Then a limit of steps was equal to 600. According to original assignment 400 steps should be sufficient. But as [Marik et al. 2004] says Koza's optimal solution was as in (9). But as simple solution showed 545 steps are necessary for an ant to eat all food in the Santa Fe trail.

```
IfFoodAhead[Move, Prog3[Left,
Prog2[IfFoodAhead[Move, Right],              (9)
Prog2[Right, Prog2[Left, Right]]],
Prog2[IfFoodAhead[Move, Left], Move]]]
```

Functionality of the equation (9) can be described by following. If bait is in front of the ant it moves on the field and eats the food. If there is nothing it does following 3 commands. If food is in front of the ant it moves and eats the food, if not it turns twice right. Next Prog2(Left, Right) is not necessary there, this is the reason why all program takes 545 steps instead of 404 in the case of no Prog2(Left,Right). Then next control of food in front of ant is again, if yes ant moves and eats the food. If not it turns left to the original direction as it was at the beginning of the program. If the cycle is somewhere interrupt, e.g. in the case of truth in the first function IfFoodAhead, the cycle is repeated still from the beginning until all food is not eaten or constrained steps are not reached.

The second constraint could be concerned to the length of the list of commands for an ant. The more longer list could cause the more steps to reach all food is eaten. In this preliminary study this constraint was not set up. But in further studies a penalization concerned to this constraint will be surely used.

4. USED EVOLUTIONARY ALGORITHMS

In this paper SelfOrganizing Migrating Algorithm (SOMA), Differential Evolution (DE) and Simulated Annealing (SA) were used as an evolutionary algorithm. For detailed information see – [Zelinka 2004a], [Price, Storn, Lampinen 2005], [Kirkpatrick, Gelatt, Vecchi 1983].

4.1. Differential Evolution (DE)

Differential Evolution is a population-based optimization method that works on real-number coded individuals [Price, Storn, Lampinen 2005]. For each individual $\vec{x}_{i,G}$ in the current generation G, DE generates a new trial individual $\vec{x}'_{i,G}$ by adding the weighted difference between two randomly selected individuals $\vec{x}_{r1,G}$ and $\vec{x}_{r2,G}$ to a third randomly selected individual $\vec{x}_{r3,G}$. The resulting individual $\vec{x}'_{i,G}$ is crossed-over with the original individual $\vec{x}_{i,G}$. The fitness of the resulting individual, referred to as perturbated vector $\vec{u}_{i,G+1}$, is then compared with the fitness of $\vec{x}_{i,G}$. If the fitness of $\vec{u}_{i,G+1}$ is greater than the fitness of $\vec{x}_{i,G}$, $\vec{x}_{i,G}$ is replaced with $\vec{u}_{i,G+1}$, otherwise $\vec{x}_{i,G}$ remains in the population as $\vec{x}_{i,G+1}$.

Differential Evolution is robust, fast, and effective with global optimization ability. It does not require that the objective function is differentiable , and it works with noisy, epistatic and time-dependent objective functions.

4.2. Self-Organizing Migrating Algorithm (SOMA)

SOMA is a stochastic optimization algorithm that is modelled on the social behaviour of cooperating individuals [Zelinka 2004a] It was chosen because it has been proven that the algorithm has the ability to converge towards the global optimum [Zelinka 2004a]. SOMA works on a population of candidate solutions in loops called *migration loops*. The population is initialized randomly distributed over the search space at the beginning of the search. In each loop, the population is evaluated and the solution with the highest fitness becomes the leader *L*. Apart from the leader, in one migration loop, all individuals will traverse the input space in the direction of the leader. Mutation, the random perturbation of individuals, is an important operation for evolutionary strategies (ES). It ensures the diversity amongst the individuals and it also

provides the means to restore lost information in a population. Mutation is different in SOMA compared with other ES strategies. SOMA uses a parameter called PRT to achieve perturbation. This parameter has the same effect for SOMA as mutation has for GA.
The novelty of this approach is that the PRT Vector is created before an individual starts its journey over the search space. The PRT Vector defines the final movement of an active individual in search space.

The randomly generated binary perturbation vector controls the allowed dimensions for an individual. If an element of the perturbation vector is set to zero, then the individual is not allowed to change its position in the corresponding dimension.

An individual will travel a certain distance (called the PathLength) towards the leader in *n* steps of defined length. If the PathLength is chosen to be greater than one, then the individual will overshoot the leader. This path is perturbed randomly.

4.3. Simulated Annealing (SA)

Simulated Annealing is one of older algorithms compared to SOMA and DE. It was introduced by Kirkpatrick et al. for the first time [Kirkpatrick, Gelatt, Vecchi 1983]. An inspiration for developing this algorithm was annealing of metal. In the process metal is heated up to temperature near the melting point and then it is cooled very slowly. The purpose is to eliminate unstable particles. In other words, particles are moved towards an optimum energy state. Metal is then in more uniform crystalline structure.

This approach was used in the case of simulated annealing including terms. It start off from a randomly selected point. Then a certain (depends on user) number of points is generated in the neighbourhood. The point with the best cost value is selected to be the middle of new neighbourhood (start point for a new loop). But it is possible to accept also worse value of cost function. The acceptance is based on a probability which decreases with number of iterations. In the case that the best cost value is in the start point this one is chosen for the next loop.

5. EXPERIMENTAL RESULTS

The main idea is to show that SOMA, DE and SA are able to solve such problems of symbolic regression – setting a trajectory - under Analytic Programming.

50 simulations were carried out for each algorithm, i.e. 150 simulations in total. SOMA and DE have almost all simulations with positive results; only one case in both algorithms did not reach the extreme. SA was not so successful, only 14 positive results. To show that AP is able to work with arbitrary evolutionary algorithms we suppose to carry simulations out with Genetic Algorithms (GA) and other algorithms and also parallel computing is intended in this field. Data from all simulations were processed and vizualized in [Oplatkova 2005], [Oplatkova, Zelinka 2006].

In simulations done for purposes of this article following setting was used to run SOMA, DE and SA according to Tab. 1 - Tab. 3.

Tab. 1: Setting of SOMA

Parameter	Value
PathLength	3
Step	0.22
PRT	0.21
PopSize	200
Migrations	50
MinDiv	-0.1
Individual Length	50

Tab. 2: Setting of DE

Parameter	Value
NP	200
F	0.8
CR	0.2
Generations	700
Individual Length	50

Tab. 3: Setting of SA

Parameter	Value
T	10 000
T_{min}	0.000 01
α	0.986
MaxIter	1 500
MaxIterTemp	93
Individual Length	50

Firstly, the results show values of cost function evaluations. This parameter shows good performance of Analytic Programming. As you can see in Tab. 4 the lowest number of cost function evaluations equals to 2 697 for SA and 3396 for SOMA. DE was also not so far with its 4030 cost function evaluations.

The Fig. 5 uses data from Tab. 4 and shows it in a graphical way where the diamond means the average value. As can be seen, SA had the lowest average value. But this might be caused by only 14 cases which were included in the chart while SOMA and DE had 49 positive cases.

Tab. 4: Cost function evaluation for SOMA, DE and SA

	Cost Function Evaluation		
	SOMA	DE	SA
Minimum	3 396	4 030	2 697
Maximum	134 114	136 011	98 241
Average	61 966	66 620	50 142

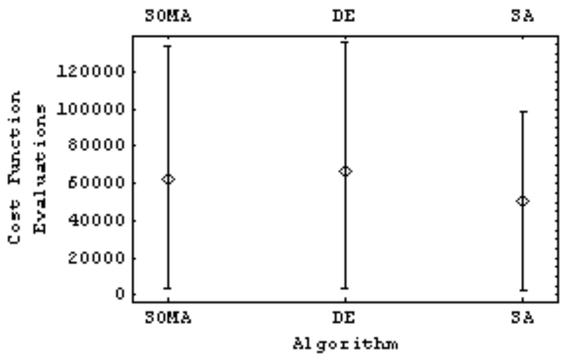

Fig. 5: Graphical representation of minimal, maximal and average values of cost function evaluation for SOMA, DE and SA

Second indicator depicts histogram of successful hits and the number of cost function evaluations for each hit – Fig. 6 - Fig. 8. Negative results are not included.

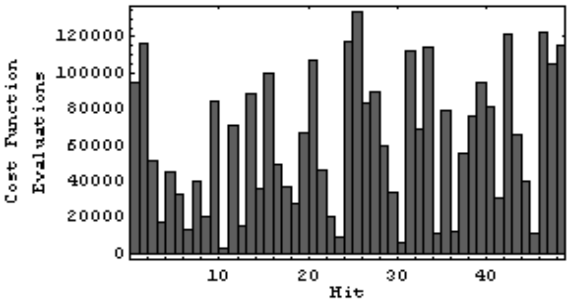

Fig. 6: Histogram of SOMA algorithm

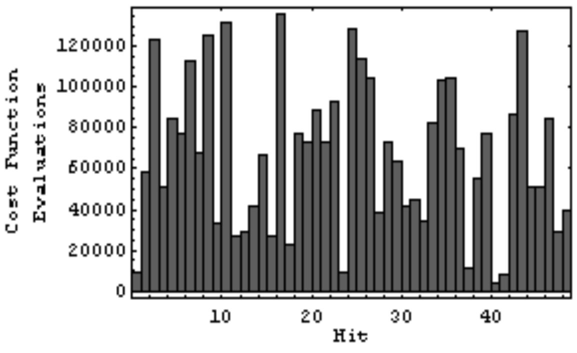

Fig. 7: Histogram of DE algorithm

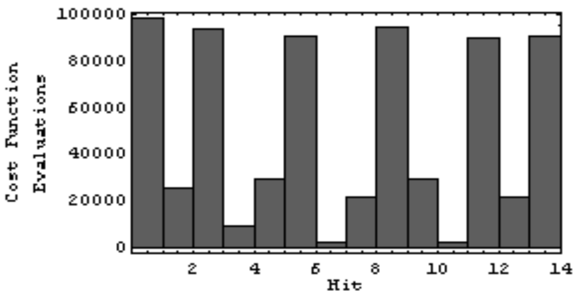

Fig. 8: Histogram of SA algorithm

Another creation of histograms can be made from the point of view of number of cases (axe y) which appeared in some interval of cost function values (axe x). This approach can be seen in Fig. 9 - Fig. 11.

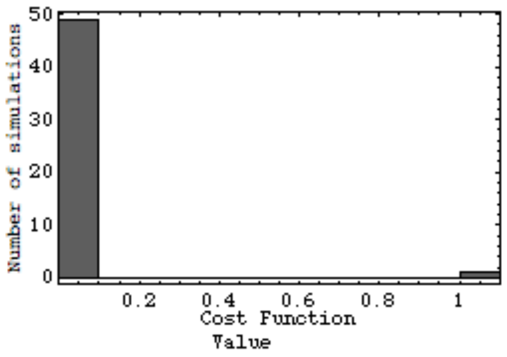

Fig. 9: Histogram of SOMA algorithm – number of cases in specific intervals of cost function values

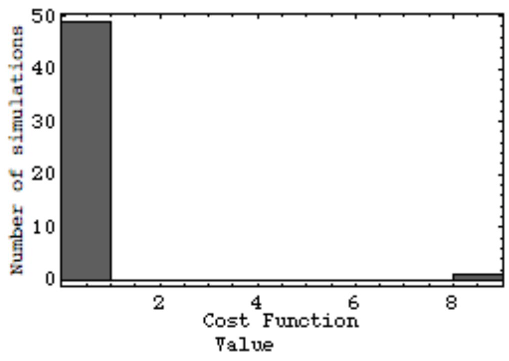

Fig. 10: Histogram of DE algorithm – number of cases in specific intervals of cost function values

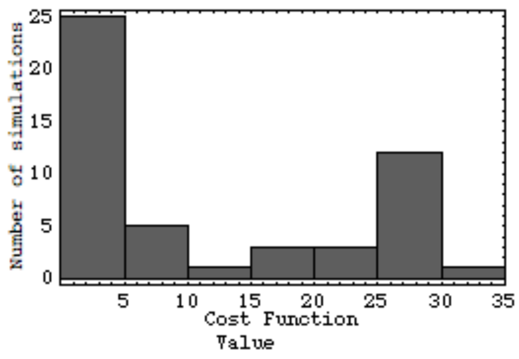

Fig. 11: Histogram of SA algorithm – number of cases in specific intervals of cost function values

Next point, which we were interested in, was a number of commands for the ant and number of steps required to eat all baits (Tab. 5, Tab. 6). In the Tab. 6 you can see that DE found a route which is overcome in the least number of steps. Sorted lists of pairs – commands and steps – are seen for all 3 algorithms in Tab. 5. As it is shown, it can be stated that the smallest number of commands does not have to cause the smallest number of steps. And vice versa, that small number of steps does not mean the small set of commands.

Tab. 5: Number of commands

	Number of leaves (commands)		
	SOMA	DE	SA
Minimum	11	11	15
Maximum	50	50	50
Average	32	32	26

Tab. 6: Number of steps

	Number of steps		
	SOMA	DE	SA
Minimum	396	367	406
Maximum	606	604	605
Average	547	540	535

The following picture depicts that the ant went through all the fields (Fig. 12); the white "X" shows fields which were attended by the ant. The notation (10) contains set of rules for the ant how to go successfully through the trail. In the notation (11) the whole description of the route can be seen where Ea, So, We, No means east, south, west and north (which cardinal point the ant is turned into). The numbers in brackets are positions on the grid.

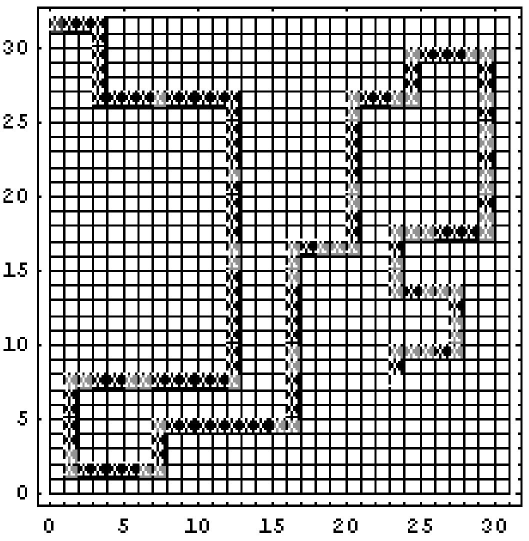

Fig. 12: Santa Fe Trail overcome by ant found by DE

IfFoodAhead[Move, IfFoodAhead[
Move, Prog2[Prog2[Right,
IfFoodAhead[Prog2[IfFoodAhead[
IfFoodAhead[Move, Move], Move],
Move], Prog3[IfFoodAhead[Move,
IfFoodAhead[Prog3[Right, Right, Prog2[
Left, Prog2[IfFoodAhead[Prog2[Prog2[(10)
Left, Move], Right], IfFoodAhead[Move,
Left]], Prog2[IfFoodAhead[Move,
Move], Prog2[IfFoodAhead[Move,
Right], Right]]]]], Left]], Left,
IfFoodAhead[Move, Right]]]]], Move]]]

{{32, 1}, {32, 2}, {32, 3}, {32, 4}, {So}, {31, 4}, {30, 4}, {29, 4}, {28, 4}, {27, 4}, {We}, {So}, {Ea}, {27, 5}, {27, 6}, {27, 7}, {So}, { Ea}, {No}, {Ea}, {27, 8}, {27, 9}, {27, 10}, {27, 11}, {27, 12}, {27, 13}, {So}, {26, 13}, {25, 13}, {24, 13}, {23, 13}, {We}, {So}, {Ea}, {So}, {22, 13}, {21, 13}, {20, 13}, {19, 13}, {18, 13}, {We}, {So}, {Ea}, {So}, {17, 13}, { We}, {So}, {Ea}, {So}, {16, 13}, {15, 13}, {14, 13}, {13, 13}, {12, 13}, {11, 13}, {10, 13}, {9, 13}, {We}, {So}, {Ea}, { So}, {8, 13}, {We}, {8, 12}, {8, 11}, {8,10}, {8, 9}, {8, 8}, {No}, {We}, {So}, {We}, {8, 7}, {No}, {We}, { So}, {We}, {8, 6}, {8, 5}, {8, 4}, {No}, {We}, {So}, {We}, {8, 3}, {No}, {We}, {So}, {We}, {8, 2}, {No}, {We}, {So}, {7, 2}, {6, 2}, {5, 2}, {4, 2}, {We}, {So}, {Ea}, {So}, {3, 2}, {We}, {So}, { Ea}, {So}, {2, 2}, {We}, {So}, {Ea}, {2, 3}, {2, 4}, {2, 5}, { 2, 6}, {So}, {Ea}, {No}, {Ea}, {2, 7}, {So}, {Ea}, {No}, {Ea}, {2, 8}, {So}, {Ea}, {No}, {3, 8}, {4, 8}, {Ea}, {No}, {We}, {No}, { 5, 8}, {Ea}, {5, 9}, {5, 10}, {5,11}, {5, 12}, {5, 13}, {5, 14}, {5, 15}, {So}, {Ea}, {No}, {Ea}, { 5, 16}, {So}, {Ea}, {No}, {Ea}, {5, 17}, {So}, {Ea}, {No}, {6, 17}, {7, 17}, {8, 17}, {Ea}, {No}, {We}, {No}, {9, 17}, {Ea}, {No}, {We}, {No}, {10, 17}, {11, 17}, {12, 17}, {13, 17}, {14, 17}, {Ea}, {No}, {We}, {No}, {15,17}, {Ea}, {No}, {We}, {No}, {16, 17}, {Ea}, {No}, {We}, {No}, {17, 17}, {Ea}, {17, 18}, {17, 19}, {17, 20}, {So}, {Ea}, {No}, {Ea}, { 17, 21}, {So}, {Ea}, {No}, {18, 21}, {19, 21}, {Ea}, {No}, {We}, {No}, {20, 21}, {Ea}, {No}, {We}, {No}, {21, 21}, {22, 21}, {23, 21}, {24, 21}, {25, 21}, {Ea}, {No}, {We}, {No}, {26, 21}, {Ea}, {No}, {We}, {No}, {27, 21}, {Ea}, {27, 22}, {27, 23}, {27, 24}, {So}, {Ea}, {No}, {Ea}, {27, 25}, {So}, {Ea}, {No}, {28, 25}, {29, 25}, {Ea}, {No}, {We}, {No}, { 30, 25}, {Ea}, {30, 26}, {30, 27}, {30, 28}, {So}, {Ea}, {No}, {Ea}, {30, 29}, {So}, {Ea}, {No}, {Ea}, {30, 30}, {So}, {29, 30}, {28, 30}, {27, 30}, {26, 30}, {We}, {So}, {Ea}, {So}, {25, 30}, {We}, {So}, {Ea}, {So}, {24, 30}, {23, 30}, {We}, {So}, {Ea}, {So}, {22, 30}, {We}, { So}, {Ea}, {So}, {21, 30}, {20, 30}, {We}, {So}, {Ea}, {So}, {19, 30},{We}, {So}, {Ea}, {So}, {18, 30}, {We}, {18, 29}, {18, 28}, {18, 27}, {No}, {We}, {So}, {We}, {18, 26}, {No}, {We}, {So}, {We}, {18, 25}, {No}, {We}, {So}, {We}, {18, 24}, {No}, {We}, {So}, {17, 24}, {16, 24}, {We}, {So}, {Ea}, {So}, {15, 24}, {We}, {So}, {Ea}, {So}, {14, 24}, {We}, {So}, {Ea}, {14, 25}, {14, 26}, {So}, {Ea}, {No}, {Ea}, {14, 27}, {So}, {Ea}, { No}, {Ea}, {14, 28}, {So}, {13, 28}, {12, 28}, {11, 28}, {We}, {So}, {Ea}, {So}, {10, 28}, {We}, {10, 27}, {10, 26}, {10, 25}, {No}, {We}, {So}, {We}, {10, 24}, {No}, {We}, {So}, {9, 24}, {8, 24}} (11)

6. CONCLUSIONS

This contribution deals with an alternative algorithm for symbolic regression. This study shows that this algorithm is suitable not only for mathematical regression but also for setting of optimal trajectory for artificial ant which can be replaced by robots in real world, in industry.

To compare with standard GP it can be stated on the basic results above that AP can solve this kind of problems in shorter times as cost function evaluations are counted.

The aim of this study was not to show that AP is better or worse than GP (or GE when compared) but that AP is also a powerful tool for symbolic regression with support of different evolutionary algorithms.

The main object of the paper was to show that symbolic regression done by AP is able to solve also cases where linguistic terms as for example commands for movement of artificial ant or robots in real world are. Here simulations for 3 algorithms – SOMA, DE and SA were carried out. As figures showed SOMA and DE were more successful in positive results than SA was. This proved that a good performance of AP depends on a choice of suitable robust and powerful evolutionary algorithms.

During simulations carried in this problem were reached following results:

- 50 simulations for each algorithm means 150 in total for all 3 algorithms.
- Positive results
  - 49 from 50 simulations for SOMA
  - 49 from 50 for DE
  - and 14 from 50 for SA

which accomplished the required tasks thus Analytic Programming is able to solve such kind of problems in symbolic regression. This result also says that Simulated Annealing is not so powerful tool as other two evolutionary algorithms are. It is supposed that the cost function is very complicated with quite a lot of local optima and therefore the Simulated Annealing was not so successful as SOMA or DE were.

- Solutions which fulfil conditions which were laid down by John Koza [Koza 1998] concerned to number of steps, were found (2 by SOMA and 3 by DE). It means 5 solutions were successful under the 400 steps. Moreover, 17 (SOMA) + 20 (DE) + 6 (SA), in total 43 from 150 were successful under the 545 steps which was introduced by John Koza [Koza 1998], [Marik et al., 2004] as an optimal one.

Future research is key activity in this field. The following steps are to finished simulations with GA and other evolutionary algorithms and to try some other class of problems to show that Analytic Programming is powerful tool as Genetic Programming or Grammatical Evolution are.

7. ACKNOWLEDGEMENTS

This work was supported by the grant NO. MSM 7088352101 of the Ministry of Education of the Czech Republic and by grants of Grant Agency of Czech Republic GACR 102/06/1132.

8. REFERENCES

Davidson J.W., Savic D.A., Walters G.A., 2003, Symbolic and numerical regression: experiments and applications, Informatics and Computer Science – An international Journal, Vol. 150, Elsevier, USA, 2003, pages 95 – 117, ISSN:0020-0255

Davis L. 1996, *Handbook of Genetic Algorithms* International Thomson Computer Press, ISBN 1850328250, 1996

Ferreira C., 2006, Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence, Springer, 2006, ISBN: 3540327967

Johnson C. G., 2003, Artificial Immune System Programming for Symbolic Regression, Lecture notes in Computer Sciences series, Springer, Volume 2610/2003, 2003, ISSN 0302-9743.

Johnson Colin G. 2003, Artificial immune systems programming for symbolic regression, In C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, and E. Costa, editors, Genetic Programming: *6th European Conference*, LNCS 2610, p. 345-353, 2003

Kirkpatrick S., Gelatt C. D., Vecchi M. P. 1983, Optimization by Simulated Annealing, *Science*, 13 May 1983, *220* (4598), p. 671 – 680

Koza J.R. 1998: *Genetic Programming* MIT Press, ISBN 0-262-11189-6, 1998

Koza J.R., Bennet F. H., Andre D., Keane M. 1999: *Genetic Programming III* Morgan Kaufnamm pub., ISBN 1-55860-543-6, 1999

Lampinen J., Zelinka I. 1999, *New Ideas in Optimization* – Mechanical Engineering Design Optimization by Differential Evolution, Vol. 1. London: McGraw-Hill, 1999, ISBN 007-709506-5

Mařík V. a kol. 2004, *Artificial Intelligence IV.* 2004, Academia, Praha, Czech edition

O'Neill M., Ryan C. 2003: *Grammatical Evolution. Evolutionary Automatic Programming in an Arbitrary Language* Kluwer Academic Publishers, 2003, ISBN 1402074441

Oltean M., Grosan C., 2003, Evolving Evolutionary Algorithms using Multi Expression Programming, The 7th European Conference on Artificial Life, September 14-17, 2003, Dortmund, Edited by W. Banzhaf (et al), LNAI 2801, pp. 651-658, Springer-Verlag, Berlin, 2003

O'Neill M., Brabazon A. 2006. Grammatical Differential Evolution. In Proc. International Conference on Artificial Intelligence (ICAI'06), pp.231-236, CSEA Press.

O'Neill M., Leahy F., Brabazon A. 2006. Grammatical Swarm: A variable-length Particle Swarm Algorithm. Swarm Intelligent Systems, pp.59-74, Springer.

Oplatkova Z. 2005, Optimal Trajectory of Robots Using Symbolic Regression, *In: CD-ROM of Proc. 56th International Astronautical Congress 2005*, Fukuoka, Japan, 2005, paper nr. IAC-05-C1.4.07

Oplatkova Z., Zelinka I., Investigation on Artificial Ant using Analytic Programming, GECCO 2006, Seattle, Washington, USA, 8 – 12 July 2006, ISBN 1-59593-186-4

O'Sullivan J., Ryan C., 2002, An Investigation into the Use of Different Search Strategies with Grammatical Evolution, *Proceedings of the 5th European Conference on Genetic Programming*, p.268 - 277, 2002, Springer-Verlag London, UK, ISBN:3-540-43378-3

Paterson N., 2003 Genetic Programming with context sensitive grammars, doctoral thesis, University of St. Andrews, 2003

Paterson N., Livesey M.,1996, Distinguishing genotype and phenotype in genetic programming, In Koza, Goldberg, Fogel & Riolo, eds. Late Breaking Papers at GP 1996, MIT Press, 1996, ISBN 0-18-201-031-7

Price K., Storn R. M., Lampinen J. A. 2005: Differential Evolution : *A Practical Approach to Global Optimization* (Natural Computing Series) Springer; 1 edition , 2005, ISBN: 3540209506

Salustowicz R. P., Schmidhuber J. 1997: Probabilistic Incremental Program Evolution, *Evolutionary Computation*, vol. 5, nr. 2, 1997, pages 123 – 141, MIT Press, ISSN 1063-6560

Stinstra E., Rennen G., Teeuwen G., 2006, Meta-modelling by symbolic regression and Pareto Simulated Annealing, Tilburg University, Netherlands, nr. 2006-15, ISSN 0924-7815

www.genetic-programming.org

www.grammatical-evolution.org

Zelinka I. 2002: Analytic programming by Means of Soma Algorithm. Mendel '02, *In: Proc. 8th International Conference on Soft Computing Mendel'02*, Brno, Czech Republic, 2002, 93-101., ISBN 80-214-2135-5

Zelinka I. 2004a, „SOMA – Self Organizing Migrating Algorithm", In: B.V. Babu, G. Onwubolu (eds),), *New Optimization Techniques in Engineering* Springer-Verlag, 2004, ISBN 3-540-20167X

Zelinka I., Oplatkova Z. 2003: Analytic programming – Comparative Study. *CIRAS'03, The second International Conference on Computational Intelligence, Robotics, and Autonomous Systems*, Singapore, 2003, ISSN 0219-6131

Zelinka I., Varacha P., Oplatkova Z., *Evolutionary Synthesis of Neural Network*, Mendel 2006 – 12th International Conference on Softcomputing, Brno, Czech Republic, 31 May – 2 June 2006, pages 25 – 31, ISBN 80-214-3195-4

Zelinka I.,Oplatkova Z, Nolle L. 2004b: Boolean Symmetry Function Synthesis by Means of Arbitrary Evolutionary Algorithms-Comparative Study, ESM '2004, I*n: Proc. 18th European Simulation Multiconference*, Magdeburg, Germany 2004

Zelinka I.,Oplatkova Z, Nolle L. 2005: Boolean Symmetry Function Synthesis by Means of Arbitrary Evolutionary Algorithms-Comparative Study, *International Journal of Simulation Systems, Science and Technology, Volume 6*, Number 9, August 2005, pages 44 - 56, ISSN: 1473-8031, online http://ducati.doc.ntu.ac.uk/uksim/journal/Vol-6/No.9/cover.htm, ISSN: 1473-804x

9. AUTHORBIOGRAPHIES

ZUZANA OPLATKOVA was born in Czech Republic, and went to the Tomas Bata University in Zlin, where she studied technical cybernetics and obtained her MSc. degree in 2003. Ph.D. she defended in March 2008. She is a senior lecturer (artificial intelligence) at the same university. Her e-mail address is: oplatkova@fai.utb.cz

IVAN ZELINKA was born in Czech Republic, and went to the Technical University of Brno, where he studied technical cybernetics and obtained his degree in 1995. He obtained Ph.D. degree in technical cybernetics in 2001 at Tomas Bata University in Zlin. Now he is Associate Professor (artificial intelligence, theory of information), head of department and Vice dean for Science and International Relations. His e-mail address is: zelinka@fai.utb.cz and his Web-page can be found at http://www.fai.utb.cz/people/zelinka/hp

Tab. 7: Sorted numbers of steps and commands for all algorithms

SOMA				DE				SA			
sorted by steps		sorted by commands		sorted by steps		sorted by steps		sorted by commands		sorted by steps	
396	49	594	11	367	49	599	11	406	25	577	15
399	36	596	11	387	49	592	12	406	25	592	16
409	21	568	14	390	50	564	13	409	23	605	16
409	22	594	14	409	18	542	14	503	22	592	17
409	23	594	14	409	18	568	14	503	22	537	19
421	37	577	15	409	50	577	14	537	19	503	22
456	50	544	16	421	50	581	14	577	15	503	22
489	17	590	16	475	16	581	14	577	49	409	23
521	50	594	16	496	50	583	15	592	16	406	25
532	50	606	16	509	21	594	15	592	17	406	25
533	20	489	17	516	46	475	16	592	50	594	34
533	27	544	17	517	49	533	16	594	34	594	34
537	34	583	17	519	49	409	18	594	34	577	49
540	27	576	18	525	38	409	18	605	16	592	50
542	27	533	20	533	16	533	18				

544	16	550	20	533	18	568	18				
544	17	409	21	533	20	584	19				
548	30	589	21	533	32	604	19				
548	50	409	22	541	49	533	20				
550	20	409	23	542	14	550	20				
551	43	559	24	550	20	509	21				
551	50	584	24	551	50	581	22				
559	24	583	26	557	31	596	23				
562	50	533	27	562	29	562	29				
568	14	540	27	564	13	557	31				
572	34	542	27	568	14	533	32				
574	27	574	27	568	18	525	38				
576	18	548	30	572	50	599	42				
577	15	537	34	573	49	516	46				
581	49	572	34	577	14	581	47				
581	50	399	36	581	14	367	49				
583	17	421	37	581	14	387	49				
583	26	551	43	581	22	517	49				
584	24	603	47	581	47	519	49				
589	21	396	49	583	15	541	49				
590	16	581	49	584	19	573	49				
592	50	596	49	588	50	589	49				
594	11	604	49	589	49	591	49				
594	14	606	49	591	49	595	49				
594	14	456	50	592	12	597	49				
594	16	521	50	594	15	601	49				
594	50	532	50	595	49	390	50				
596	11	548	50	595	50	409	50				
596	49	551	50	596	23	421	50				
601	50	562	50	597	49	496	50				
603	47	581	50	599	11	551	50				
604	49	592	50	599	42	572	50				
606	16	594	50	601	49	588	50				
606	49	601	50	604	19	595	50				