SANTA FE TRAIL FOR ARTIFICIAL ANT WITH SIMULATING ANNEALING – PRELIMINARY STUDY

Zuzana Oplatková and Ivan Zelinka
Faculty of Applied Informatics
Tomas Bata University
Nad Stranemi 4511, Zlin, 762 72, Czech Republic
E-mail: {oplatkova,zelinka} @ fai.utb.cz

KEYWORDS

Santa Fe trail, symbolic regression, Analytic Programming, Simulated Annealing

ABSTRACT

The paper deals with a novelty tool for symbolic regression – Analytic Programming (AP) which is able to solve various problems from the symbolic regression domain. One of tasks for it can be setting an optimal trajectory for artificial ant on Santa Fe trail which is main application of Analytic Programming in this paper. In this contribution main principles of AP are described and explained. In second part of the article how AP was used for setting an optimal trajectory for artificial ant according the user requirements is in detail described. An ability to create so called programms, as well as Genetic Programming (GP) or Grammatical Evolution (GE) do, is shown in that part. AP is a superstructure of evolutionary algorithms wich are necessary to run AP. In this contribution Simulated Annealing as an evolutionary algorithm was used to carry preliminary simulations out.

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. They used this process when they need a mathematical model of unknown data. 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 was proposed in Genetic Programming (GP) by John Koza [Koza 1998, 1999, www.genetic-programming.com]. The other two approaches are Grammatical Evolution (GE) developed by Conor Ryan [O'Neill 2003, O'Sullivan 2002, www.grammatical-evolution.org] and here described Analytic Programming (AP) designed in [Zelinka 2002, 2003, 2004b, 2005].

GP was the first tool for symbolic regression 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 [Koza 1999].

The other tool is GE which was developed in last decade of 20th century by Conor Ryan. GE has one advantage compared to GP and this is ability to use arbitrary programming language not only LISP as is in the case of GP. In contrast to other evolutionary algorithms, GE was used only with a few search strategies, with a binary representation of the populations [O'Sullivan 2002]. Other 2 interesting investigations using symbolic regression was carried out by Johnson [Johnson 2003] working on Artificial Immune Systems and Probabilistic Incremental Program Evolution (PIPE) [Salustowicz 1997] generates functional programs from an adaptive probability distribution over all possible programs.

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

ANALYTIC PROGRAMMING

Description

Basic principles of the AP were developed in the 2001. The name was given according to simple pattern. Until that time only GP and GE have been existed. GP uses genetic algorithms while AP can be used with any evolutionary algorithm, independently on individual representation. To avoid any confusion, based on use of names according to the used algorithm, name - Analytic Programming was chosen, because of AP stands for synthesis of analytical solution by means of evolutionary algorithms.

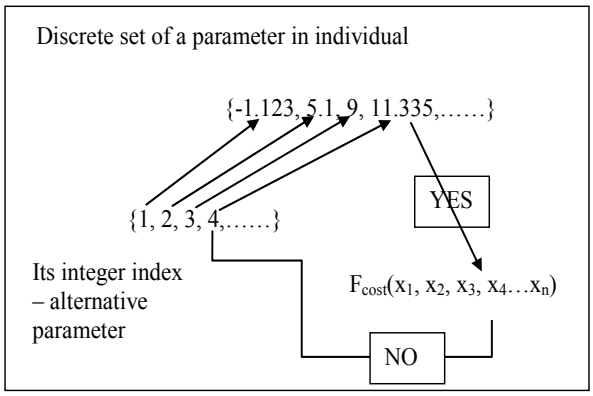
According to authors of AP [Zelinka 2005], AP was inspired by numerical methods in Hilbert spaces (space with mutually orthogonal functions) and by GP. Principles of AP [Zelinka 2005] are somewhere between these two philosophies. From GP an idea of evolutionary creation of symbolic solutions is taken into AP while from Hilbert spaces an idea of functional spaces and building of resulting function by means of searching process usually done by numerical methods like Ritz or Galerkin is adopted into AP. AP 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, Tanh, And, Or
- operators: +, -, *, /, dt,…

Proceedings 20th European Conference on Modelling and Simulation
Wolfgang Borutzky, Alessandra Orsoni, Richard Zobel © ECMS, 2006
ISBN 0-9553018-0-7 / ISBN 0-9553018-1-5 (CD)

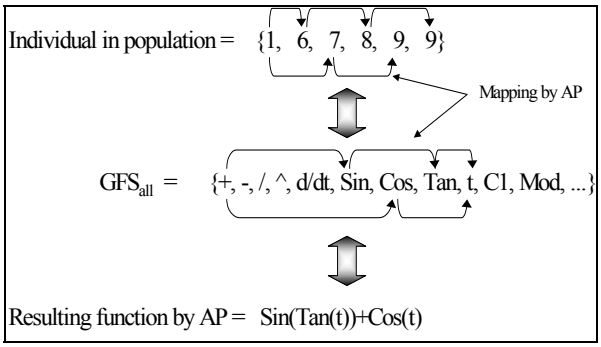
• terminals: 2.73, 3.14, t,…
All these "mathematical" objects create a set which AP tries to synthesize the appropriate solution from. Main principle (core) of AP is based on discrete set handling, proposed in [Zelinka 2002], see Fig. 1 and Fig. 2.
Discrete set handling shows itself as universal interface between EA and symbolically solved problem. That is why AP can be used almost by any evolutionary algorithm. Analytic programming was used e.g. in:
- sextic, quintic, 3sine, 4sine problem [Zelinka 2003], algorithms Simulated Annealing (SA), Genetic Algorithms (GA), Differential Evolution (DE) [Storn, 1995, Price, 2005] and SelfOrganizing Migrating Algorithm (SOMA) [Zelinka, 2004a] were used
- Boolean symmetry and parity problems [Zelinka 2004b, 2005], again with SA, GA, DE and SOMA
- Solving of ordinary differential equations (ODE): u''(t) = cos(t), u(0) = 1, u(π) = -1, u'(0) = 0, u'(π) = 0, 100 times repeated, in that case AP was looking for suitable function, which would solve this case of ODE, by DE and SOMA in [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, 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 the reality, in [Zelinka 2002]

Briefly said, in AP individuals consist of non-numerical expressions (operators, functions,…) as described above, which are in evolutionary process represented by their integer indexes (Fig. 1 & 2). This index then serves like a pointer into this set of expressions and AP uses it to synthesize resulting function-program for cost function evaluation [Zelinka 2002].
This description was shown on mathematical operators and objects as functions, variables etc. for simplicity. But in the case of this contribution in the set of functions are linguistic terms which represent for example commands for mobility of robot.



Figures 1: Discrete set handling

To find a final formula, use of evolutionary algorithm is necessary as was mentioned in the introduction. The evolutionary algorithm works as an optimization tool which finds the best solution according to value of cost function [Zelinka 2004a, Lampinen 1999].



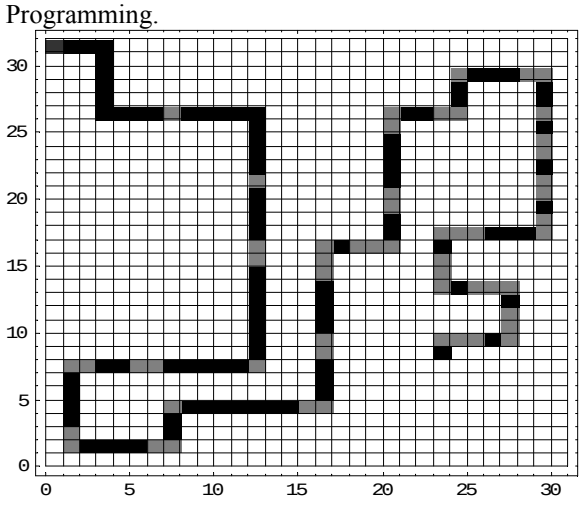
Figures 2: Main principles of AP

AP exists today in three versions - AP_{basic}, AP_{meta} and AP_{nf} [Zelinka 2005]. For our purposes we used AP_{basic}.
This approach is completely different than in GP. Its main principle is based on genetic algorithm which is working with population of individuals represented in LISP programming language. Individuals in canonical form of GP are not binary strings as is usual for GA, but consist of LISP symbolic objects like sin(), +, Exp(), MyFunction, etc. Origin of these objects comes from LISP or they are simply user defined functions. Symbolic objects are usually divided into two classes: functions and terminals. Functions were just explained and terminals here represents a set of independent variables like x, y, and constants like π, 3.56, etc [Koza 1998, 1999]. On these individuals are applied operations of crossover and mutation as is in the standard GA.
Another approach to GP is way of enforcing dimensional constraints through formal grammar. It restricts GP search space to dimensionally admissible laws [Ratle 2000]. Other investigation which adjusts GP to achieve improved predictive performance and reliability of the induced expressions was presented in [Keijzer 2003].

PROBLEM DESIGN

Santa Fe Description

The Santa Fe trail, demonstrated in Fig. 3, was chosen from [Koza 1998] to make a comparative study with the same problem which was solved by Koza in Genetic Programming.



Figures 3: 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. 3 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.
But in 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 Fig. 3). 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]).

Set of functions

The set of functions used for movements of the ant is following. As a set of variables GFS0 [Zelinka 2003], 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
GFS0 = {Left, Right, Move},
where
GFS0 – a set of variables and terminals [Zelinka 2003]
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 GFS2 and GFS3 were set up.
GFS2 = {IfFoodAhead, Prog2}
GFS3 = {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.
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.

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 (1) is calculated as a difference between raw fitness and a number of baits eaten by an ant [Koza 1998] which went through the grid according to just generated way.

$$CV = 89 - NumberFood \quad (1)$$

NumberFood - 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 [Mařík 2004] says Koza's optimal solution was as in (2). But as simple solution showed 545 steps are necessary for an ant to eat all food in the Santa Fe trail.

```
IfFoodAhead[Move,
 Prog3[IfFoodAhead[Move, Right],
  Prog2[Right, Prog2[Left, Right]],
  Prog2[IfFoodAhead[Move, Left], Move]]] (2)
```

Functionality of the equation (2) 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 length of 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.

Used evolutionary algorithm

In this preliminary comparative study Simulated Annealing was used as an evolutionary algorithm.
Simulated Annealing was introduced by [Kirkpatrick 1987] for the first time. 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. Simulated annealing is a better variation of Hill-Climbing algorithm [Rich 1991]. Both start off from a randomly selected point. Then a certain (depends on user) number of points is generated in the neighbourhood (MaxIterTemp). In the case of hill-climbing algorithm, the point with the best cost value is selected to be the middle of new neighbourhood (start point for a new loop). In the case that the best cost value is in the start point this one is chosen for the next loop. This subroutine is repeated several (depends on user) times (MaxIter). Hill – climbing mostly ends in a local optimum. Compared to this, simulated annealing offers a little bit different approach which is described in next paragraph. It means that there is a chance to find a global optimum, not only a local one.
The principle of acceptation solution during run of Simulated Annealing is following. If the new cost value is better than the old one new one is accepted immediately. It means that the difference between these two cost values is negative. If the difference is positive (the new cost value is worse than the old one) a number from interval <0, 1> is generated. If it is lower than the probability according to equation (3) the new point is accepted, otherwise the old one continues in the process.

$$p(T) = e^{-\frac{\Delta E}{T}} \qquad (3)$$

where
p(T) - probability of transition for temperature T
ΔE - difference between cost values of previous and current solution
T - current temperature – control parameter for cooling schedule

The algorithm starts with high temperature T, which is decreased in steps. Equation (4) shows standard cooling function [Kirkpatrick 1987].

$$T_{n+1} = \alpha\, T_n \qquad (4)$$

where
T_{n+1} - temperature in the next step
T_n - temperature in the current step
α - cooling coefficient from interval <0, 1>

If alpha is close to zero, the solution can converge to a local optimum, which is not desirable, because cooling is too fast. On the other hand, if alpha is very close to one cooling is very slow and it takes a lot of time to find optimum.
Simulated Annealing offers to find a global optimum better than Hill-Climbing which goes from a start point in the direction of the biggest gradient because probability causes that also a worse solution than the previous can be accepted, which can mean finding a global optimum in the end.

EXPERIMENTAL RESULTS

This article continues with study which was presented in IAF 2005 Fukuoka, Japan for SelfOrganizing Migrating Algorithm and GECCO 2006, USA for Differential Evolution (now in review process). The main idea is to show that also Simulated Annealing is able to solve such kind of problems under Analytic Programming.
As a reason of time consuming simulations until the printing this article were done 9 simulations with 4 positive results for SA. Simulations are still in process. Genetic Algorithms (GA) are supposed to be carried out as well as parallel computing is intended in this field.
The time consuming means that one simulation can hypothetically take 1 – 3 days on the computer with Athlon XP 1800+ processor and 256 MB RAM memory. It depends on number of cost function evaluations. All simulations were carried out in Mathematica environment (www.wolfram.com). One cost function evaluation took there 1 to 6 second.
In simulations done for purposes of this article following setting was used to run Simulated Annealing succesfully (Table 1).

Table 1: Setting of Simulated Annealing

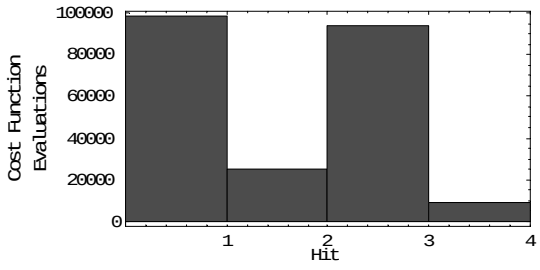
Parameter	Value
T	10 000
T_{min}	0.000 01
α	0.986
MaxIter	1 500
MaxIterTemp	93
Individual Length	50

Firstly the results are concerned to cost function evaluation. As you can see in Table 2 the lowest number of cost function evaluations equals to 9286.

Table 2: Cost function evaluation for SA

	Cost Function Evaluation
Minimum	9 286
Maximum	98 241
Average	56 622

Second indicator depicts histogram of successful hits and the number of cost function evaluations for each hit – Fig. 4. There are not included 5 negative results.



Figures 4: Histogram of SA algorithm

Next point which we were interested in was a number of commands for the ant and number of steps required to eat all baits (Table 3).
Table is divided into two parts. Left side is sorted up according to number of commands and the right side according to number of Steps. 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.

Table 3: Numbers of commands and steps for SA

Number of commands	Number of steps	Number of steps	Number of commands
15	577	577	15
17	592	577	49
49	577	592	17
50	592	592	50

The minimal value of steps was 577 for SA from all 9 simulations which were carried out for this study. The program has 49 or 15 leaves of simple functions which was also the lowest number of commands in all simulations. All is showed in following Table 4 where minimal, maximal and average value of commands (leaves) and steps are.

Table 4: Number of commands and steps for SA

	Number of leaves (commands)	Number of steps
Minimum	15	577
Maximum	50	592
Average	33	585

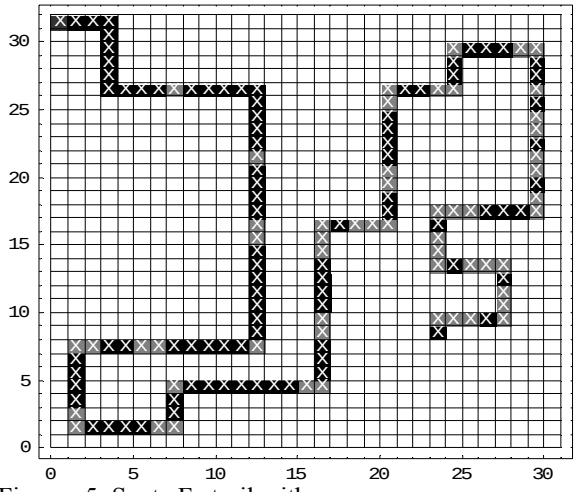
In following equation the fastest way of the ant can be seen from number of steps point of view and also from number of commands point of view from these 4 simulations which were successful.

```
Prog2[Prog2[
  IfFoodAhead[Prog2[Move, Move], Right],
  Right], Prog2[Prog2[
   IfFoodAhead[Move, Left], Move], Left]]   (4)
```

Proof of the shortest way can be seen in Fig. 5. This is the same Santa Fe trail as in Fig. 3. The white "X" shows fields which were attended by the ant. It went through all way and ate all food.

CONCLUSION

This contribution deals with a tool for symbolic regression. This study shows that this tool 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 time could be also decreased by parallelization of the process, which is one of further plans, as Koza did in GP. He uses in GP activity computer-cluster consisting of hundreds PCs [Koza 2003]. But in our case we used only **1** computer.



Figures 5: Santa Fe trail with success pass

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 9 simulations with Simulated Annealing were done. It follows up the previous 2 studies where SOMA and DE were used. All statistic data were arranged in tables and figures to show performance of AP with SA.
Reached results - 4 from 9 simulations were with positive result, i. e. only 4 simulations found solution 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 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.
Future research is key activity in this field. The following steps are to finished simulations with GA and to try some other class of problems to show that Analytic Programming is powerful tool as Genetic Programming or Grammatical Evolution are.

ACKNOWLEDGMENTS

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 and GACR 102/05/0271

REFERENCES

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

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

Keijzer M. 2003: Improving Symbolic Regression with Interval Arithmetic and Linear Scaling, Lectures in Computer Science, Volume 2610, EuroGP, Springer, 2003, pages 70-82, ISSN: 0302-9743

Kirkpatrick S., Gelatt C. D., Vecchi M. P. 1983, Optimization by Simulated Annealing, Science, 13 May 1983, Volume 220, Number 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

Koza J. R., Keane M. A., Streeter M. J. 2003: Evolving Inventions, ScientificAmerican, February 2003, p. 40-47, ISSN 0036-8733, (online www.sciam.com)

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

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

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

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

Ratle A., Sebag M. 2000: Genetic Programming and Domain Knowledge: Beyond the Limitations of Grammar-Guided Machine Discovery, Parallel Problem Solving from Nature - PPSN VI, 6th International Conference, Paris, France, Proceedings. Lecture Notes in Computer Science 1917 Springer 2000, pages 211-220, ISBN 3-540-41056-2

Rich K., Knight K. 1991: Artificial Intelligence, 2nd edition, MacGraw Hill, 1991, ISBN 0-07-052263-4

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

Storn R., Price K. 1995: Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, ICSI, March 1995. (Available via ftp from ftp.icsi.berkeley.edu/pub/techreports/1995/tr-95-012.pdf

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., 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. 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, Nolle L. 2004b: Boolean Symmetry Function Synthesis by Means of Arbitrary Evolutionary Algorithms-Comparative Study, ESM '2004, In: 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

AUTHOR BIOGRAPHIES

ZUZANA OPLATKOVA was born in Czech Republic, and went to the Tomas Bata University in Zlin, where she studied technical cybernetics and obtained her degree in 2003. She is now Ph.D. student and 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 associated professor (artificial intelligence, theory of information) and head of department. His e-mail address is: zelinka@fai.utb.cz and his Web-page can be found at http://www.ft.utb.cz/people/zelinka/hp