CREATING EVOLUTIONARY ALGORITHMS BY MEANS OF ANALYTIC PROGRAMMING – DESIGN OF NEW COST FUNCTION

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

Evolutionary algorithms, symbolic regression, Analytic Programming, Selforganizing Migrating Algorithm, Hill-Climbing, Differential Evolution.

ABSTRACT

This contribution deals with a new idea of how to create evolutionary algorithms by means of symbolic regression and Analytic Programming. The motivation was not only to tune some existing algorithms to their better performance, but also to find a new robust evolutionary algorithm. In this study operators of Differential Evolution (DE), SelfOrganizing Migrating Algortithm (SOMA), Hill Climbing were used during a process of Analytic Programming. The results showed that AP was able to find the originally defined DE, but also new structure which has a similar behaviour but slower convergence in multimodal function than DE. This, in further work, leads to including the conditions of convergence to CostFunction. Results produced in 100 repeated simulations are displayed in graphical and tabular form.

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 widly known for mathematicians. They use this process when a need arises for mathematical model of unknown data. For a long time, symbolic regression was a domain of humans but in the few past decades computers have gone to forefront of interest in this field. Initially, the idea of symbolic regression done by means of a 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) developed in [Zelinka 2002, 2003, 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, for e.g. to synthesize highly sophisticated electronic circuits [Koza 1999].

The other tool is GE which was developed in the last decade of the 20th century by Conor Ryan. GE has one advantage over GP and this is its ability to use arbitrary programming languages, and 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 the use of methods which is independent of computer platform (as author of AP suggests), programming language and can use any evolutionary algorithm (as demonstrated by [Zelinka 2002, 2003, 2005]) to find an optimal solution of the required task.

ANALYTIC PROGRAMMING – PRINCIPLES AND DATA STRUCTURES

Basic principles of the AP were developed in 2001. Until that time only GP and GE had 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, the name - Analytic Programming was chosen, since AP represents synthesis of analytical solution by means of evolutionary algorithms.

The core of AP is based on a special set of mathematical objects and operations. The set of mathematical objects is set of functions, operators and so-called terminals (as well as in GP), which are usually constants or independent variables. This set of variables is usually mixed together as shown in Fig. 1 and consists of functions with different number of arguments. Because of a variability of the content of this set, it is called here "general functional set" – GFS. The structure of GFS is created by subsets of functions according to the number of their arguments. 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. 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. The content of GFS is dependent only on the user. Various

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

functions and terminals can be mixed together [Zelinka 2002, 2003, 2005].
The second part of the AP core 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 the 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) [Lampinen 1999, Zelinka 2002, 2003, 2005] and the second one stands for security procedures which do not allow synthesizing pathological programs. 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 arbitrary individual into a program. Individuals in the population consist of integer parameters, i.e. an individual is an integer index pointing into GFS.

GFS_{all} = {+, -, /, ^, d/dt, Sin, Cos, Tan, t, C1, Mod, ...}
GFS_{3arg}= {BetaRegularized, ...}
GFS_{2arg}= {+, -, /, ^, Log, Mod, GammaRegularized...}
GFS_{1arg}= {Sin, Cos, Tan, Abs, Re, Im, ...}
GFS_{0arg}= {t, x, y, z, C1, C2, Kinchin, ...}

Fig. 1 Structure of GFS (this example related to the environment of Mathematica®)

The creation of the program can be schematically observed in Fig. 2. The individual contains numbers which are indices into GFS. The detailed description is represented in [Zelinka, 2004].

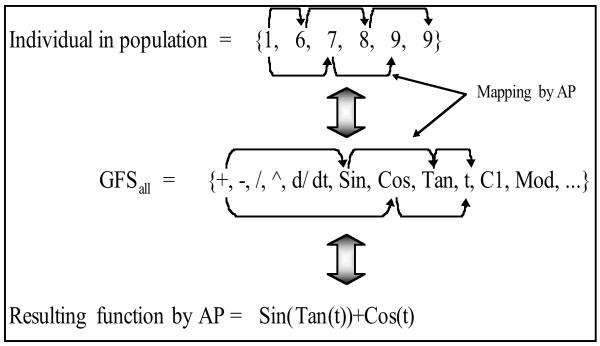

Fig. 2: Main principles of AP

PROBLEM DESIGN

Operators of evolutionary algorithms

For our purpose to create evolutionary algorithms by means of Analytic Programming, we extended algorithms from Differential Evolution [Oplatkova, 2006], SelfOrganizing Migrating Algorithm and Hill Climbing algorithm. Details of these algorithms can be found in [Price 2005, Zelinka 2004 and Russel 1995]. It was necessary to separate its operators like mutation, crossover and selection of parents. The following operators were put inside GFS sets according to the number of arguments.

GFS_{0arg}= {SelectDE, SelectLeaderSOMA, SelectSOMARandLeader, SelectHillClimb}

GFS_{1arg}= { MutateDERand1, CrossDEExp, CrossDEBin, MutateDEBest2, MutateDERand2, MutateDECurrentToBest, MutateDEBest1, SOMAATOWithPRT, SOMAATOWithoutPRT, SOMAATORandWithPRT, SOMAATORandWithoutPRT, HillClimbing }

SelectDE – this is the operator which selects individuals from population for other instructions. In this case, the output will be 4 individuals – one active individual and 3 randomly chosen.
MutateDERand1- here mutation is produced as follows: one of the randomly chosen parents is subtracted from the second parent and a so called differential vector is produced. This vector is multiplied by a mutable constant and the result of this operation is a weighted differential vector. The third parent plus the weighted differential vector produces a noisy vector. This noisy vector is the output of the MutateDERand1.
MutateDEBest2, MutateDEBest1, MutateDERand2, MutateDECurrentToBest are mutation functions of other version of Differential Evolution.
CrossDEBin – the active individual gives some arguments and the input individual to CrossDEBin gives some other arguments and the trial vector is created. This is given by crossover constant Cr. If random number from interval <0,1> is less than Cr the arguments from active individual is taken, otherwise it is from the individual which is input of CrossDEBin.
CrossDEExp is similar crossover to *CrossDEBin*. The difference is in the choice of arguments into the trial vector. Until first case of random number from interval <0,1> is less than Cr, arguments from active individual are taken, then the rest from the input individual of CrossDEExp.
SelectLeaderSOMA – choses the best individual in the population (with the minimal value of cost function).
SelectSOMARandLeader – choses the random individual from population.
SOMAATOWithPRT – is the operator which create a table of new individuals which are in the direction from active individual to Leader in Steps and the best individual is selected as an output individual.
SOMAATOWithoutPRT, SOMAATORandWithPRT, SOMAATORandWithoutPRT – are similar as the previous one, the only difference is in the use of PRTVector and best individual as Leader or random individual as Leader.
SelectHillClimb – choses random point in the Cost Function.

HillClimbing – is a process of Hill Climbing algorithm. If the randomly chosen point from the neighboroughood has less cost value, it is chosen as a new startpoint, otherwise the current startpoint is used again.
All above described operators work as modules with some input and some output. The functionality is related to one active individual. Therefore for application for all individuals in the population *FinalAlgorithm* is therefore set up as well.

Original Differential Evolution of DERand1Bin version can be written as the equation (1)

CrossDEBin(MutateRand1(SelectDE)) (1)

Original SOMA in version All To One is then used as equation (2).

SOMAATOWithPRT (SelectLeaderSOMA) (2)

Hill Climbing has similar notation (equation 3)

HillClimbing (SelectHillClimb) (3)

Cost Function design

When Analytic Programming creates a complex formula, it is necessary to assign some value that represents if the individual is suitable and its quality. In the case of creating new evolutionary algorithms trying on some benchmark functions is logical. In this study we applied newly generated algorithm on two test functions – to obscure how closed the minimum value the algorithm reached. Two cost functions were DeJong1st as example of unimodal function and Schwefel as example of multimodal [Zelinka, 2004].
DeJong1st and Schwefel functions are described in an analytical way as shown in equations (4) and (5).

$$\sum_{i=1}^{Dim} x_i^2 \quad (4)$$

$$\sum_{i=1}^{Dim} -x_i \sin(\sqrt{|x_i|}) \quad (5)$$

These two functions can be seen as graphs in the following figures – Fig. 3 and Fig. 4 for DeJong1st, Fig. 5 and Fig. 6 for Schwefel function.

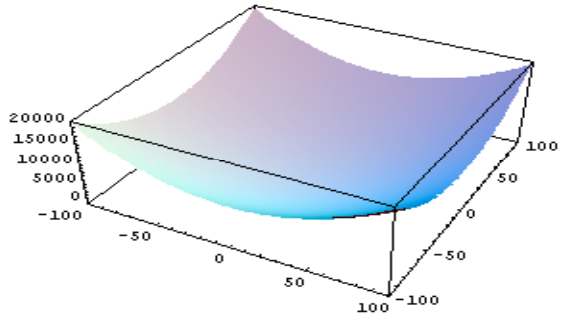

Fig. 3: DeJong1st – 3D exapmle of unimodal function

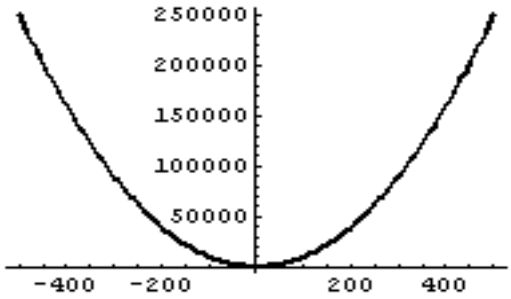

Fig. 4: DeJong1st – 2D exapmle of unimodal function

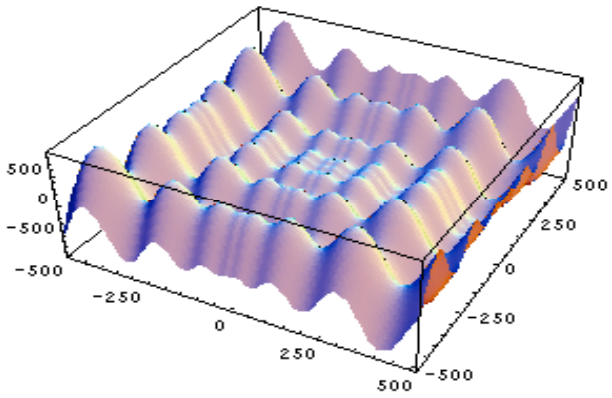

Fig. 5: Schwefel – 3D exapmle of multimodal function

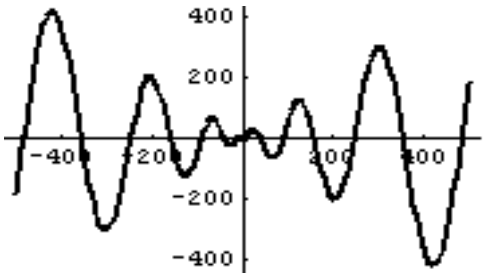

Fig. 6: Schwefel – 2D exapmle of multimodal function

The value of Cost Function was designed so that initially the generated algorithm is tried to observe if it is able to find the minimum value on the easy unimodal function DeJong1st. Better said, it is testing the difference between global extreme and the extreme approached by a new generated algorithm. If the difference under 10^{-7} is reached, then the Schwefel function is tested similarly.
If the algorithm is successful on both functions, the value is set as seen equation (6) in the case that number of cost function evaluations were less than the average.

|CFESchwefel - avgCFESchwefel| / SchwefelValue (6)

where
CFESchwefel is number of costfunction evaluations used to reach the SchwefelValue by the generated program
avgCFESchwefel is the average value of the number of cost function evaluation reached by SOMA and DE in 100 times repeated simulations [Oplatkova, 2006].

SchwefelValue is the value of reached extreme

If the number of cost functions were higher, the value is behaving according to equation (7).

SchwefelValue |CFESchwefel - avgCFESchwefel| (7)

In the case the algorithm was not successful in the Schwefel function but was successful in DeJong1st function, the rules are similar as in the case of Schwefel function, as seen in equations (8) and (9).

|CFEDeJong – avgCFEDeJong| / DeJongValue (8)

DeJongValue (| CFEDeJong - avgCFEDeJong | + |CFEDeJong – avgCFEDeJong|) (9)

where
CFEDeJong is the number of costfunction evaluations used to reach the DeJongValue by the generated program:
avgCFEDeJong is the average value of the number of cost function evaluation reached by SOMA and DE in 100 times repeated simulations [Oplatkova, 2006].
DeJongValue is value of the reached extreme.

In the the case generated algorithm was not successful at all, the final equation is used (10).

DeJongValue |CFEDeJong | (10)

This is not the only one way as to how to design a suitable cost function. This one differs from the previous one not only in including the number of cost function evaluations inside the CostFunction but also in the approach to the value of the extreme itself [Oplatkova, 2006]. In previous cases, we used the original value of the extreme, but more suitable is to find the difference from the global extreme. Then we are close to zero value and it is more predictive.

RESULTS

This section compares DE and SOMA with a new developed algorithms. The following figures are histories of behaviour of the best individual in the population - 100 times repeted for DE and SOMA algorithms – DeJong1st (Fig. 7 and Fig. 8) and Schwefel (Fig. 9 and Fig. 10).

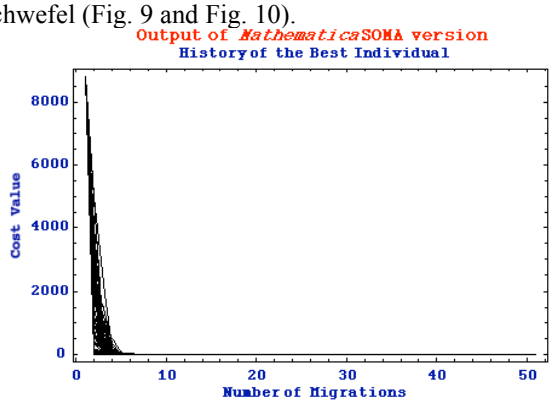

Fig. 7: DeJong1st – 100times repeated for SOMA

Following table show values of extremes for DeJong and Schwefel which were found by DE and SOMA.

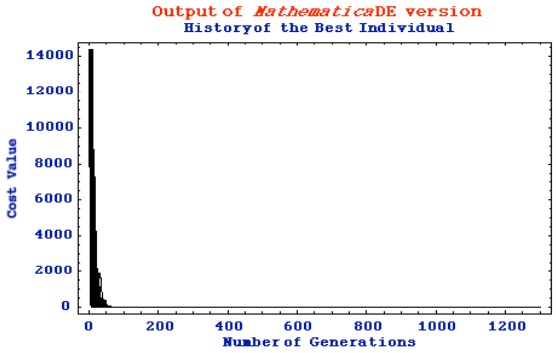

Fig. 8: DeJong1st – 100 times repeated for DE

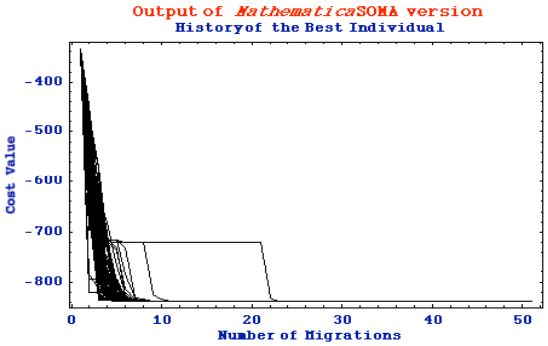

Fig. 9: Schwefel – 100 times repeated for SOMA

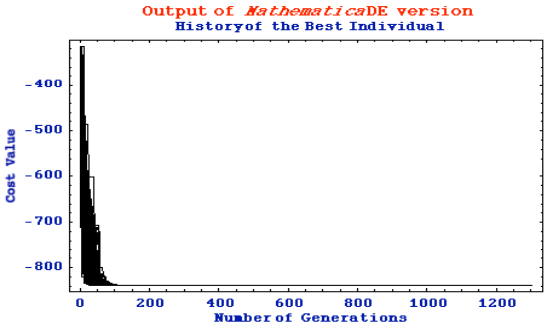

Fig. 10: Schwefel – 100 times repeated for DE

Table 1 Values of extremes found by DE and SOMA

	Original DE		Original SOMA	
	DeJong	**Schwefel**	**DeJong**	**Schwefel**
Minimum	2.04492 x 10^{-8}	-837.966	2.39949 x 10^{-16}	-837.966
Maximum	6.61369 x 10^{-6}	-837.966	1.45227 x 10^{-14}	-837.966
Average	9.29224 x 10^{-7}	-837.966	3.6897 x 10^{-15}	-837.966

During our simulation we found successful and also non successful solutions.
Following equations (11 – 14) belong to non successful solutions.

SelectDE (10)

SelectLeaderSOMA (11)
CrossDEBin(SelectDE) (12)
HillClimbing(SelectDE) (13)

The successful solution we can divide into two groups – which found subsolutions with requested diversity but the number of cost function evaluations were high and the final solution therefore was not so good (equations 15 and 16). The second group contains solution which were successful in all conditions including original algorithms of SOMA and DE (equations 17 - 19).

SOMAATOWithPRT(SOMAATORandWithPRT(SOMAATOWithoutPRT(MutateDERand1(SelectSOMALeader)))) (14)

SOMAATOWithPRT(MutateDEBest1(MutateDERand1(MutateDECurrentToBest(MutateDEBest1(MutateDECurrentToBest(SelectSOMARandLeader)))))) (15)

CrossDEBin(MutateRand1(SelectDE)) (16)

SOMAATORandWithPRT (SelectDE) (17)

MutateDEBest1(MutateDERand1(SelectSOMARandLeader)) (18)

Following Figures 11 and 12 show graphs for 100times repeated simulations of algorithm with notation in (15).

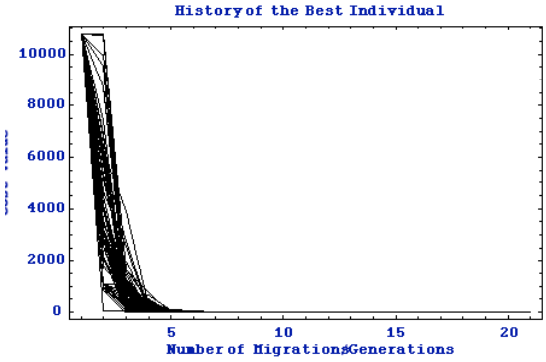

Fig. 11: DeJong – 100times repeated for new algorithm

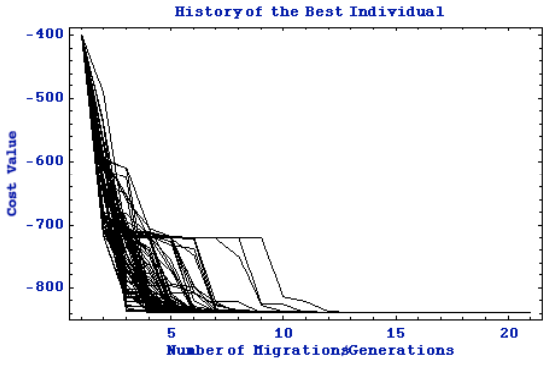

Fig. 12: Schwefel – 100times repeated for new algorithm

Table 2 shows values of extremes which were found by two new generated algorithms.

Table 2 Values of extremes for DeJong and Schwefel found by new generated algorithms

	Generated algorithm (15)		Generated algorithm (16)	
	DeJong	**Schwefel**	**DeJong**	**Schwefel**
Minimum	5.86771x 10^{-10}	-837.966	7.05752 x 10^{-10}	-837.966
Maximum	1.53905 x 10^{-4}	-800.053	5.90596 x 10^{-4}	-799.892
Average	9.06618 x 10^{-6}	-835.993	3.24827x 10^{-5}	-835.871

The number of generations or migrations in new algorithms in the graph might be a little confusing. Number of cost function evaluations (CFE) in one loop for SOMA, DE and two new evolutionary algorithms are in equations (20 - 23). It means that 150 generations in DE means 3000 CFE if number of individuals is 20. Similar CFE (3109) in SOMA is for Migrations = 6. In new algorithms, 5 loops means 8282 and 3227 CFE for (15) and (16).

(PopSize – 1) Migrations (PathLength / Step) (19)

NP Generations (20)

NP Generations (3 (PathLength / Step) + 1) (21)

NP Generations ((PathLength / Step) + 5) (22)

As can be seen, the generated programs were able to find minimum values, along with DE and SOMA. But not in all cases as table 2 shows even if CFE is higher than in SOMA and DE. On the other hand the connection of several evolutionary operators show the promising approach, and its advantage which might occur in higher dimensional problems.

CONCLUSION

This contribution was concerned to the method of how to create new evolutionary algorithms by means of symbolic regression using Analytic Programming. In this study design of cost function was changed. Some other improvements are proposed to be done in the future, mainly to try to use on more arguments test functions – to prove the robustness of the generated algorithms. We also suppose that more complicated structure which uses more cost function evaluations will show their advantage and probably more velocity in convergence to the global extreme. Also elementary functions will be replenished with other operators from other evolutionary algorithms.

On the basis of reached results it can be stated that:

- Analytic Programming can be used as a tool for creating new evolutionary algorithms
- All was found during first randomly generated population. It is hoped to find more complex algorithms with better behaviour during evolution which should be applied on the first population, mainly in higher dimensional cost functions.
- All operators should have the same structure concerned to inputs and outputs to be simply mixed together.

The results are promising for further research.

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

Babu, B.V., "Evolutionary Computation - At a Glance", NEXUS, Annual Magazine of Engineering Technology Association, BITS, Pilani, pp. 3-7, 2001, also online http://discovery.bits-pilani.ac.in/discipline/chemical/BVb/publications.html

Johnson C. 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

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

Lampinen J., Zelinka I.,1999,New Ideas in Optimization – Mechanical Engineering Design Optimization by Differential Devolution, Volume 1. London: McGraw-hill, 1999, 20 p., ISBN 007-709506-5

Price K., Storn R. M., Lampinen J. A.. 2005: Differential Evolution : A Practical Approach to Global Optimization (Natural Computing Series) Springer; 1 edition (December 22, 2005) ISBN: 3540209506

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

Oplatkova Z., Zelinka I., 2006, Creating Evolutionary Algorithms by means of Analytic Programming – Preliminary Study, Mendel 2006 – 12th International Conference on Softcomputing, Brno, Czech Republic, 31 May – 2 June 2006, pages 19 – 24, ISBN 80-214-3195-4

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

Russel S. J., Norvig P., 1995, Artificial Intelligence: Modern Approach, Prentice Hall, 1995, ISBN: 0131038052

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

www.ft.utb.cz/people/zelinka/ap

www.genetic-programming.com

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., 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., 2004: „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., 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 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