DIFFERENT APPROACHES FOR CONSTANT ESTIMATION IN ANALYTIC PROGRAMMING

Zuzana Kominkova Oplatkova, Adam Viktorin, Roman Senkerik, Tomas Urbanek

Tomas Bata University in Zlin, Faculty of Applied Informatics
Nam T.G. Masaryka 5555, 760 01 Zlin, Czech Republic
{oplatkova, aviktorin, senkerik, turbanek}@fai.utb.cz

KEYWORDS
Analytic programming, Differential evolution, approximation, sextic, quintic.

ABSTRACT

This research deals with different approaches for constant estimation in analytic programming (AP). AP is a tool for symbolic regression tasks which enables to synthesise an analytical solution based on the required behaviour of the system. Some tasks do not need any constant estimation - AP is used in its basic version without any constant estimation handling. Compared to this, cases like data approximation need constants (coefficients) which are essential for the process of precise solution synthesis. This paper offers another strategy to already known and used by the AP from the very beginning and approaches published recently in 2016. This paper compares these procedures and the discussion also includes nonlinear fitting and metaevolutionary approach. As the main evolutionary algorithm, a differential algorithm (de/rand/1/bin) for the main process of AP is used.

INTRODUCTION

Analytic Programming (AP) (Zelinka et al., 2011) is a tool of symbolic regression which uses techniques from the area of evolutionary computation techniques (EVT). The basic case of a regression represents a process in which the measured data is fitted and a suitable mathematical formula is obtained in an analytical way. This process is widely known for mathematicians. They use this process when a need arises for a mathematical model of unknown data, i.e. the relation between input and output values. Classical regression usually requires to select an expected type of model in advance and a suitable method is applied for the coefficient estimation of the proposed model. Compared to that, symbolic regression in the context of EVT means to build a complex formula from basic operators defined by users. The final shape of the expression is managed to breed via evolutionary optimisation algorithms.

Initially, John Koza proposed the idea of symbolic regression done by means of a computer in Genetic Programming (GP) (Back et al., 1997), (Koza, 1998), (Koza, 1999). The other approaches are e.g. Grammatical Evolution (GE) developed by Conor Ryan (O'Neill et al., 2003) and some others included Analytic Programming (Zelinka et al., 2011). The symbolic regression can be used for different tasks: data approximation, design of electronic circuits, optimal trajectory for robots, classical neural networks and pseudo neural networks synthesis (Oplatkova, 2016) and many other applications (Back et al., 1997), (Koza, 1998), (Koza, 1999), (O'Neill et al., 2003), (Zelinka et al., 2011), (Oplatkova, 2009), (Varacha et al., 2006), (Volna et al., 2013). The results and usage depend on the user-defined set of operators and their possible combinations and nesting into themselves.

This paper deals with strategies and their comparison for constants (coefficients) estimation - nonlinear fitting (Zelinka et al., 2011), metaevolutionary approach (Zelinka et al., 2011) and direct encoding in the individuals (extended individual (Viktorin et al, 2016) or a special handling with an individual (Urbanek et al., 2016) and a stance proposed in this paper.

ANALYTIC PROGRAMMING

Basic principles of the AP were developed in 2001 (Zelinka et al., 2005), (Zelinka et al., 2008), (Zelinka et al., 2011).

The core of AP is based on a special set of mathematical objects and operations. The collection of mathematical objects is the set of functions, operators and terminals, which are usually constants or independent variables. Various functions and terminals can be mixed in this set. This set is called general functional set (GFS) due to its variability of the content. The structure of GFS is created by subsets of functions according to the number of their arguments. For example, GFSall is a set of all functions, operators and terminals, GFS3arg is a subset containing functions with only three arguments, GFS0arg represents only terminals, etc. The subset structure presence in GFS is of vital importance 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 (Zelinka et al., 2005), (Zelinka et al., 2008), (Oplatkova, 2009).

The second part of the AP core is a sequence of mathematical operations, which are used for the program synthesis. These operations are used to transform an individual of a population into a suitable program. Mathematically stated, it is a mapping from an individual domain into a program domain. This

Proceedings 31st European Conference on Modelling and
Simulation ©ECMS Zita Zoltay Paprika, Péter Horák, Kata Váradi,
Péter Tamás Zwierczyk, Ágnes Vidovics-Dancs, János Péter Rádics (Editors)
ISBN: 978-0-9932440-4-9/ ISBN: 978-0-9932440-5-6 (CD)

mapping consists of two main parts. The first part is called discrete set handling (DSH) (See Figure 1) (Zelinka et al., 2005), (Lampinen and Zelinka, 1999) and the second one stands for security procedures which do not allow synthesising pathological programs. The method of DSH, when used, allows handling 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 creates the mapping mentioned above which transforms the arbitrary individual into a program.

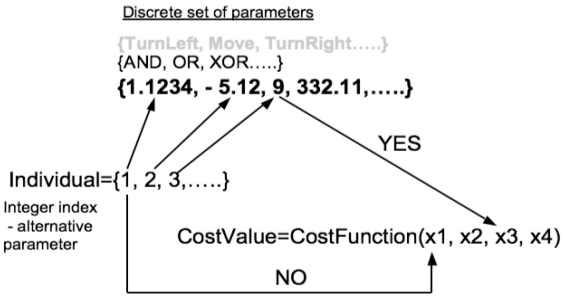

Figure 1: Discrete set handling

AP needs some evolutionary algorithm (Zelinka, 2004) that consists of a population of individuals for its run. Individuals in the population consist of integer parameters, i.e. an individual is an integer index pointing into GFS. The creation of the program can be schematically observed in Fig. 2.

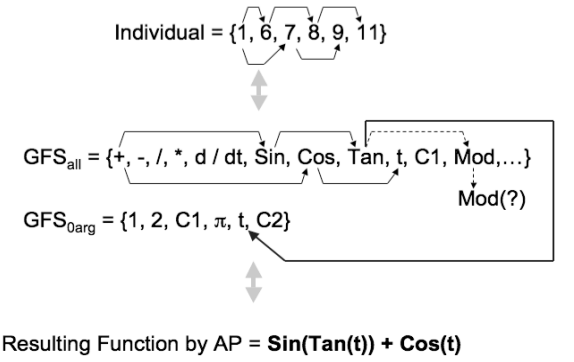

Figure 2: Main principles of AP

An example of the process of the final complex formula synthesis (according to the Fig. 2) follows.
The number 1 in the position of the first parameter means that the operator plus (+) from GFS_{all} is used (the end of the individual is far enough). Because the operator + must have at least two arguments, the next two index pointers 6 (sin from GFS) and 7 (cos from GFS) are dedicated to this operator as its arguments. The two functions, sin and cos, are one-argument functions; therefore the next unused pointers 8 (tan from GFS) and 9 (*t* from GFS) are dedicated to the sin and cos functions. As an argument of cos, the variable *t* is used, and this part of the resulting function is closed (*t* has zero arguments) in its AP development. The one-argument function tan remains, and there is one unused pointer 11, which stands for Mod in GFS_{all}. The modulo operator needs two arguments but the individual in the example has no other indices (pointers, arguments). In this case, it is necessary to employ security procedures and jump to the subset with GFS_{0arg}. The function tan is mapped on *t* from GFS_{0arg} which is in the 11^{th} position, cyclically from the beginning. The detailed description is represented in (Zelinka et al., 2005), (Zelinka et al., 2008), (Oplatkova et al., 2009).

ANALYTIC PROGRAMMING - VERSIONS

The above-described version is the basic one AP_{basic} (Zelinka et al., 2005) - without constant estimation. Such approach is used for tasks like logic circuit design where numerical coefficients are not usually used. It can also be applied to a pre-generated set of numerical values as in genetic programming where e.g. 4000 random numerical values of constants are selected. They are used as standard terminals like variable x.
When a constant estimation is necessary, e.g. in data approximation or pseudo neural network synthesis, etc., firstly general approach is applied which is different from genetic programming technique where all constants (e.g. 4000 random generated values) were part of the nonterminal and terminal sets.
AP uses the constant K (Zelinka et al., 2005) which is indexed during the evolution (1) - (3). The K is a terminal, i.e. GFS_{0arg}. So it is used as a standard terminal, e.g. similar to variable x in the evolutionary process (1). When K is needed, a proper index is assigned – K_1, K_2, ... K_n (2). Numeric values of indexed Ks are estimated (3) via different techniques - AP_{nf} (nonlinear fitting package in Mathematica) (Zelinka et al., 2005), AP_{meta},(metaevolutionary approach with a second/slave evolutionary algorithm) (Zelinka et al., 2005, Oplatkova 2009) and three novel direct approaches AP_{extend} (extended individual - a part of it for AP and the rest of it for constant estimation) (Viktorin et al., 2016), $AP_{direct1}$ (the part behind decimal point determines the K from the selected range) (Urbanek et al., 2016) and $AP_{direct2}$ (new proposed approach in this paper - the whole value determines the K from the selected range). The first 3 versions of AP from its very beginning have been extended by two other approaches in 2016 (Viktorin et al., 2016), (Urbanek et al., 2016).

$$\frac{x^2 + K}{\pi^K} \tag{1}$$

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

$$\frac{x^2 + 3.156}{\pi^{90.78}} \tag{3}$$

AP_{nf} - Nonlinear fitting version

The estimation of constants K has been done via a package for nonlinear fitting in Wolfram Mathematica

environment (www.wolfram.com). The used function was FindFit, which includes different methods. Documentation refers to Conjugate Gradient, Gradient, Levenberg-Marquardt, Newton, NMinimize and Quasi-Newton. When one searches deeper, it can be found that NMinimize includes following techniques Nelder-Mead, Random Search, Simulated Annealing and Differential Evolution. Cost function evaluations for AP_{nf} were not interpreted correctly in previous publications. The package from Mathematica environment contains techniques which belong to a group of iterative algorithms. Thus, each constant estimation of the particularly found model is not only one step evaluation but many iterations are needed. The method is selected automatically in Mathematica. Simple tests showed that mostly around 5000 iterations are necessary in the case of described sextic and quintic problems. The result does not mean necessarily that the found model is perfectly precise. For the suggested model, the nonlinear fitting tries to find the best constants (coefficients). The final cost value comes from the performed nonlinear fitting process - the error between required and actual obtained model.
On that account, authors think that nonlinear fitting case is a specific approach of AP_{meta} (metaevolutionary approach), except the second slave algorithm does not need to be only an evolutionary algorithm. Above mentioned methods might be even faster and more precise for this particular task.

AP_{meta} - metaevolutionary approach

Generally, metaevolution means the evolution of evolution. Several directions as the usage of an evolutionary algorithm for tuning or controlling of another evolutionary technique or the evolutionary design of evolutionary algorithms are discussed for instance in (Diosan, 2009), (Edmons, 2001), (Jones, 2002), (Oplatkova, 2009), (Kordik, 2010), Deugo, 2004), (Eiben, 2007).
In AP_{meta}, the metaevolution means that one evolutionary algorithm drives the main process of symbolic regression and the second is used for the constant estimation. This meta approach of analytic programming is used when the constants are not possible to estimate by AP_{nf} because of the character of the problem. In data approximation tasks, a technique from non-linear fitting package can be used easily because the problem is designed so that the found constants (e.g. coefficients of polynomials) move the basic shape of the curve around the coordinate system. However, it is not possible to employ such a package in the case of the synthesis of more sophisticated problems, for instance, pseudo neural networks synthesis (Kominkova Oplatkova 2016). These applications do not use the found result as a model which could be adjusted to some "measured" values in the sense of interpolation but the obtained solution is used further as a part of the complex technique to find a quality of the solution and cost function estimation.

AP_{meta} is a time-consuming process and the number of cost function evaluations, which is one of the comparable factors, is usually very high. This fact is given by two evolutionary procedures (Fig. 3).

$$EA_{master} \Rightarrow program \Rightarrow K_{indexing} \Rightarrow EA_{slave} \Rightarrow K_{estimation} \Rightarrow final \cdot solution$$

Figure 3: Schema of AP procedures

EA_{master} is the main evolutionary algorithm for AP, EA_{slave} is the second evolutionary algorithm inside AP. Thus, the number of cost function evaluation (CFE) is given by (4).

$$CFE = EA_{master} * EA_{slave} \tag{4}$$

As mentioned in the last paragraph of AP_{nf} section, nonlinear fitting (NF) methods adopted in Mathematica environment are iterative processes. Thus, EA_{slave} in the case of AP_{nf} would be a number of iterations of used NF method.

The following three approaches were developed to find a suitable constant estimation which will decrease the number of cost function evaluations to (5).

$$CFE = EA_{master} \tag{5}$$

$AP_{extended}$ - extended individual

The constant handling technique with an extended individual was introduced in (Viktorin et al, 2016). The individual used in AP has an extended part which is used for the evolution of constant values.
The important task was to determine what the correct size of an extension is (6).

$$k = l - floor\left((l-1)/(\max\_\arg)\right), \tag{6}$$

where k is the maximum number of constants that can appear in the synthesised program (extension) of length l and max_arg is the maximum number of arguments needed by functions in GFS. Also, the *floor()* is a common floor round function. The final individual dimensionality (length) will be $k+l$ and the example might be:

- Program length $l = 10$
- GFS: {+, -, *, /, *sin*, *cos*, *x*, *k*}
- GFS maximum argument $max\_arg = 2$
- Extension size $k = 10 - floor((10-1) / 2) = 6$
- Dimensionality of the extended individual $k+l = 16$

This means, that the EA will work with individuals of length 16, but only first 10 features will be used for indexing into the GFS and the rest will be used as constant values.
It is worthwhile to note that only features which are going to be mapped to GFS are rounded and the rest is omitted (not rounded). An example can be viewed in

Fig. 3. Individual features in bold are the constant values.

$$Individual = \left\{ \begin{matrix} 5.08, 1.64, 6.72, 1.09, 6.20, \\ 1.28, \mathbf{0.07}, \mathbf{3.99}, \mathbf{5.27}, \mathbf{2.64} \end{matrix} \right\}$$
$$Rounded\ individual = \{5, 2, 7, 1, 6, 1\}$$
$$GFS_{all} = \{+, -, *, /, sin, cos, x, k\}$$
$$\mathbf{Program:}\ \mathbf{cos(k1 * (x - k2))}$$
$$\mathbf{Replaced:}\ \mathbf{cos(0.07 * (x - 3.99))}$$

Figure 3: Principles of $AP_{extended}$

$AP_{direct1}$ - direct encoding of K in the individual 1

This constant handling technique was introduced in (Urbanek et al, 2016). It works with a direct encoding in an individual and is based on a part behind a decimal point which as a proportional pointer determines the value from the selected range of K.
The part behind decimal point is obtained from (7).

$$ind_K = \left| ind - ind_f \right|, \tag{7}$$

where $ind = \{x_1, x_2, x_3 .... x_n\}$ and $ind_f = \{floor(x_1), floor(x_2), floor(x_3) .... floor(x_n)\}$
The decimal values in ind_K are in the interval <0,1>. The corresponding K is then computed easily from (8).

$$K = ind_K * \left| rangeK_{\max} - rangeK_{\min} \right| + rangeK_{\min} \tag{8}$$

The mapping is done in the standard procedure as in AP_{basic} and general approach of K indexation. When K is needed, the value in the corresponding position from (8) is directly used.

$AP_{direct2}$ - direct encoding of K in the individual 2

Within a later analysis of $AP_{direct1}$ behaviour, authors found out some problematic issues connected with the neighbourhood of arguments which are responsible for K estimation. Since they are dependent only on the decimal part of the argument regardless the integer part of the value, two points placed on the opposite sides of the coordinate system can be neighbours from ind_K point of view. It does not help the evolutionary optimisation process which expects for a successful performance that two points lie next to each other physically in the coordinate system.
This new approach is based on the previous and above-described $AP_{direct1}$ (Urbanek et al, 2016). The difference is in the different computation of ind_K (9). It takes the value of the not rounded individual as the proportional part in respect of length of all components in GFS_{All}.

$$ind_K = \frac{ind}{Dim(GFS_{All})}, \tag{9}$$

where $ind = \{x_1, x_2, x_3 .... x_n\}$ and $Dim(GFS_{All})$ means the number of all non-terminals and terminals used in AP. For instance, if GFS_{All}={+, -, /, *, x, K}, the $Dim(GFS_{All})$ = 6 and the valid range for arguments in the individual is in the interval <1,6>.

USED EVOLUTIONARY ALGORITHM - DIFFERENTIAL EVOLUTION

As mentioned above, the Analytic Programming needs an evolutionary algorithm for the optimisation - finding the best shape of the complex formula. This research used Differential Evolution (Price, 2005) in its canonical version DE/Rand/1/Bin. Future research expects to use some other strategies as DE/Best/1/Bin or SHADE which was quite promising in (Viktorin et al., 2016).
DE is a population-based optimisation method that works on real-number-coded individuals (Price, 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 randomly selected third 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 a perturbed 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 $\vec{x}_{i,G}$, then $\vec{x}_{i,G}$ is replaced with $\vec{u}_{i,G+1}$; otherwise, $\vec{x}_{i,G}$ remains in the population as $\vec{x}_{i,G+1}$. DE is quite robust, fast, and effective, with global optimisation ability. It does not require the objective function to be differentiable, and it works well even with noisy and time-dependent objective functions. Description of used DERand1Bin mutation strategy is presented in (10). Please refer to (Price and Storn 2001, Price 2005) for the description of all other strategies.

$$u_{i,G+1} = x_{r1,G} + F \bullet \left(x_{r2,G} - x_{r3,G} \right) \tag{10}$$

PROBLEM DESIGN

These above-mentioned strategies of $AP_{extended}$, $AP_{direct1}$ and $AP_{direct2}$ were applied on standard benchmark tests - approximation of polynomial expression - quintic (11) and sextic (12).

$$x^5 - 2x^3 + x \tag{11}$$

$$x^6 - 2x^4 + x^2 \tag{12}$$

RESULTS AND DISCUSSION

The paper will compare $AP_{extended}$, $AP_{direct1}$ and $AP_{direct2}$ strategies with differential evolution DE/Rand/1/Bin. The setting was based on some previous research in this field (Tab. 1.).

Table 1: DE settings

PopSize	50
F	0.5
CR	0.8
Generations	4000
Max. CF Evaluations (CFE)	200 000

The previously published results (Oplatkova, 2009) stated that the cost function evaluations for quintic and sextic problems were in the interval <500, 18 000>. Compared to these already published results with AP_{nf}, it seems that currently, we do not provide any improvement (Tab. 1.) in the sense of convergence speed. As already mentioned, AP_{nf} is a specific case of AP_{meta}. Therefore the correct statement of the cost function evaluations should be <500, 18 000> * cca 5000 iterations which is equal to <2 500 000, 90 000 000>. Thus, our setting in Tab.1. means the significant reduction of computation time.
All simulations were performed 30 times out. The results for the quintic problem are depicted in Fig. 4 - Fig. 6.

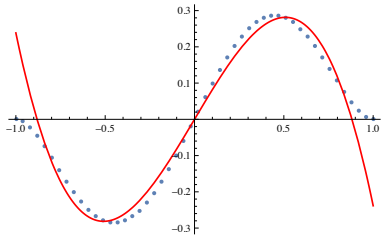

Figure 4: Quintic problem with $AP_{extended}$

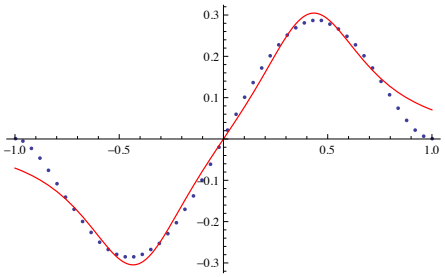

Figure 5: Quintic problem with $AP_{direct1}$

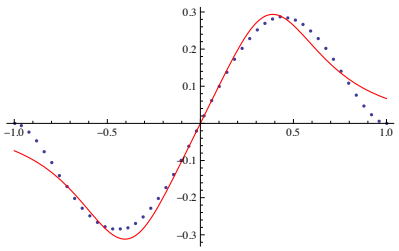

Figure 6: Quintic problem with $AP_{direct2}$

The results for the sextic problem are depicted in Fig. 7. - Fig. 8.

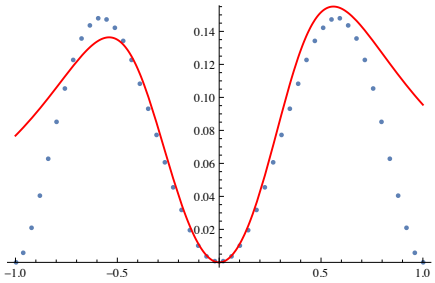

Figure 7: Sextic problem with $AP_{extended}$

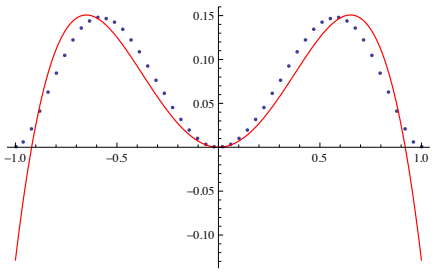

Figure 8: Sextic problem with $AP_{direct1}$

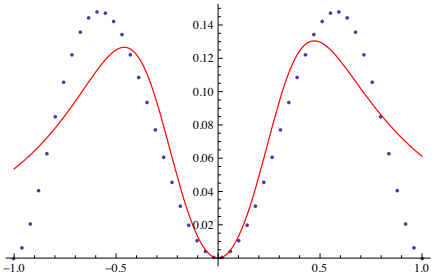

Figure 9: Sextic problem with $AP_{direct2}$

Tab. 2. and Tab. 3. show statistical measures from the performed simulations.

Table 2: Statistical results for quintic

	$AP_{extended}$	$AP_{direct1}$	$AP_{direct2}$
Min	1.84172	1.14635	1.2115
Max	3.05308	2.82948	2.41022
Avg	2.46638	1.90875	1.90549
Median	2.53626	2.07038	1.86441
St.Dev.	0.280835	0.508758	0.394415

Table 3: Statistical results for sextic

	$AP_{extended}$	$AP_{direct1}$	$AP_{direct2}$
Min	1.07682	1.02398	1.02347
Max	2.37171	2.08186	2.41894
Avg	1.86946	1.61992	1.675
Median	1.85952	1.71677	1.7273
St.Dev.	0.377488	0.269501	0.3595

The results showed that $AP_{extended}$, $AP_{direct1}$ and $AP_{direct2}$ are comparable in the achieved results.
The evolution process within $AP_{direct1}$ and $AP_{direct2}$ was carried out longer ($20x200000 = 4x10^6$ CFE) for

possible further analysis. The results for the sextic problem can be found in Tab. 4.

Table 4: Statistical results for sextic - 4 000 000 CFE

	$AP_{direct1}$	$AP_{direct2}$
Min	0.471035	0.094396
Max	1.11581	1.0897
Avg	0.740252	0.76077
Median	0.73997	0.843761
St.Dev.	0.155372	0.2293

The post analysis showed that the process got often stuck in local optima for a long time, e.g., Fig.10 depicts the history of cost function evaluation on one example of one $AP_{direct2}$ run. However, some quality solutions (e.g. Fig. 11) were obtained after circa CFE equal to 2 000 000 which is still significantly less than AP_{nf}.
The results also proved that the assumption of authors which led to $AP_{direct2}$ proposal was wrong. The evolution can work even with the individuals who assume non physical neighbourhood.
The future plans include to leave the evolution in the process when the acceptable error will be reached. The final number of cost function evaluations will be compared.

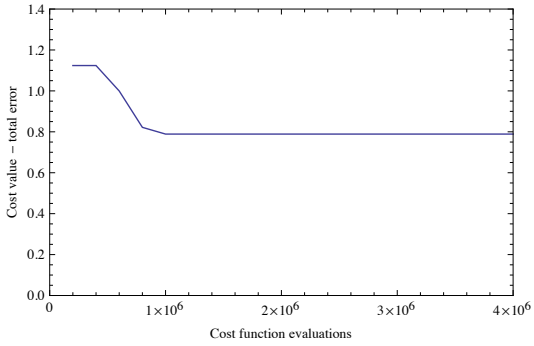

Figure 10: CFE history of one $AP_{direct2}$ run

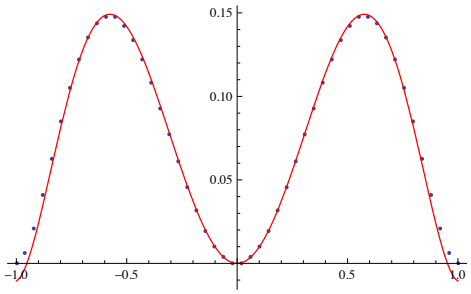
Figure 11: Sextic problem with $AP_{direct2}$ and DE

Since (Viktorin, 2016) presented better results with SHADE strategy of differential evolution for the same setting for population size and the number of generations (Tab. 1), future plans include usage of different strategies of DE and different evolutionary algorithms as Self-organizing migrating algorithm, Particle swarm algorithm and others. The best option for mentioned SHADE strategy was Min = 0.000139781 for the quintic (Fig. 12) and Min= 0.113134 for the sextic which secure very precise fitting.

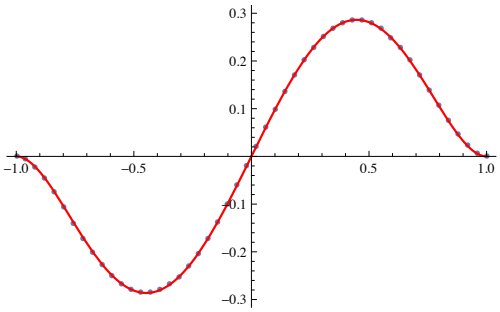
Figure 12: Quintic problem with $AP_{extended}$ and SHADE

CONCLUSION

This paper deals with Analytic programming and compares three novel approaches for constant estimation - $AP_{extended}$, $AP_{direct1}$ and $AP_{direct2}$. All simulations were performed with a DE/Rand/1/Bin strategy of differential evolution algorithm.
The results showed that all three approaches are comparable and use significantly less number of cost function evaluations than AP_{nf} or AP_{meta}.
Future plans include - comparison of cost function evaluations for these three mentioned approaches when an acceptable error is reached. Certainly, other evolutionary techniques as for instance SHADE will be employed for further testing.

ACKNOWLEDGEMENT

This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic within the National Sustainability Programme project No. LO1303 (MSMT-7778/2014) and also by the European Regional Development Fund under the project CEBIA-Tech No. CZ.1.05/2.1.00/03.0089, further it was supported by Grant Agency of the Czech Republic—GACR P103/15/06700S and by Internal Grant Agency of Tomas Bata University in Zlin under the project No. IGA/CebiaTech/2017/004.

REFERENCES

Back T., Fogel D. B., Michalewicz Z., *Handbook of evolutionary algorithms*, Oxford University Press, 1997, ISBN 0750303921

Deugo D., Ferguson D.: Evolution to the xtreme: Evolving evolutionary strategies using a meta-level approach, Proceedings of the 2004 IEEE congress on evolutionary computation, IEEE Press, Portland, Oregon, pp. 31–38, 2004

Dioşan, L., Oltean, M.: Evolutionary design of evolutionary algorithms. Genetic Programming and Evolvable Machines, Vol. 10, Issue 3, p. 263-306, 2009

Edmonds, B.: Meta-genetic programming: Co-evolving the operators of variation, Elektrik, Vol. 9, Issue 1, pp. 13-29, 2001

Eiben A.E., Michalewicz Z., Schoenauer M., Smith J.E.: Parameter control in evolutionary algorithms, pp. 19–46, Springer, 2007

Jones D.F., Mirrazavi S.K., Tamiz M.: Multi-objective meta-heuristics: An overview of the current state-of-the-art, European Journal of Operational Research, Volume 137, Issue 1, 16 February 2002, Pages 1-9, ISSN 0377-2217.

Kominkova Oplatkova Z., Senkerik R. (2013): Evolutionary Synthesis of Complex Structures - Pseudo Neural Networks for the Task of Iris Dataset Classification. In Nostradamus 2013: Prediction, Modeling and Analysis of Complex Systems. Heidelberg : Springer-Verlag Berlin, 2013, p. 211-220. ISSN 2194-5357. ISBN 978-3-319-00541-6.

Kominkova Oplatkova Z., Senkerik R. (2016): Control Law and Pseudo Neural Networks Synthesized by Evolutionary Symbolic Regression Technique, in Al-Begain K., Bargiela A.: Seminal Contributions to Modelling and Simulation - Part of the series Simulation Foundations, Methods and Applications, pp 91-113, doi: 10.1007/978-3-319-33786-9_9, ISBN: 978-3-319-33785-2.

Kordík P., Koutník J., Drchal J., Kovářík O., Čepek M., Šnorek M.: Meta-learning approach to neural network optimization, Neural Networks, Vol. 23, Issue 4, p. 568-582, 2010, ISSN 0893-6080.

Koza J. R. et al., *Genetic Programming III; Darwinian Invention and problem Solving*, Morgan Kaufmann Publisher, 1999, ISBN 1-55860-543-6

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

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

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

Oplatkova Z.: Metaevolution: Synthesis of Optimization Algorithms by means of Symbolic Regression and Evolutionary Algorithms, Lambert Academic Publishing Saarbrücken, 2009, ISBN: 978-3-8383-1808-0

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

Price, K. and Storn, R. (2001), *Differential evolution homepage*, [Online]: http://www.icsi.berkeley.edu/~storn/code.html, [Accessed 29/02/2012].

Urbanek T., Prokopova Z., Silhavy R., Kuncar A.: New Approach of Constant Resolving of Analytical Programming. In *30th European Conference on Modelling and Simulation*, 2016, p. 231-236. ISBN 978-0-9932440-2-5.

Viktorin A., Pluhacek M., Kominkova Oplatkova Z., Senkerik R.: Analytical Programming with Extended Individuals. In *30th European Conference on Modelling and Simulation*, 2016, p. 237-244. ISBN 978-0-9932440-2-5.

Volna, E., Kotyrba, M., & Jarusek, R. (2013). Multi-classifier based on Elliott wave's recognition. Computers & Mathematics with Applications, 66(2), 213-225.

Zelinka et al.: Analytical Programming - a Novel Approach for Evolutionary Synthesis of Symbolic Structures, in Kita E.: Evolutionary Algorithms, InTech 2011, ISBN: 978-953-307-171-8

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., 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.

AUTHOR BIOGRAPHIES

ZUZANA KOMINKOVA OPLATKOVA is an associate professor at Tomas Bata University in Zlin. Her research interests include artificial intelligence, soft computing, evolutionary techniques, symbolic regression, neural networks. She is an author of around 100 papers in journals, book chapters and conference proceedings. Her e-mail address is: oplatkova@fai.utb.cz

ADAM VIKTORIN was born in the Czech Republic, and went to the Faculty of Applied Informatics at Tomas Bata University in Zlín, where he studied Computer and Communication Systems and obtained his MSc degree in 2015. He is studying his Ph.D. at the same university and the field of his studies are: Artificial intelligence, data mining and evolutionary algorithms. His email address is: aviktorin@fai.utb.cz

ROMAN SENKERIK was born in the Czech Republic, and went to the Tomas Bata University in Zlin, where he studied Technical Cybernetics and obtained his MSc degree in 2004, Ph.D. degree in Technical Cybernetics in 2008 and Assoc. prof. in 2013 (Informatics). He is now an Assoc. prof. at the same university (research and courses in: Evolutionary Computation, Applied Informatics, Cryptology, Artificial Intelligence, Mathematical Informatics). His email address is: senkerik@fai.utb.cz

TOMAS URBANEK was born in Zlin in 1987. He received a B.Sc. (2009), M.Sc. (2011) in Information Technology from Faculty of Applied Informatics, Tomas Bata University in Zlin. He is a doctoral student at the Computer and Communication Systems Department. Major research interests are software engineering, effort estimation in software engineering and artificial intelligence.