# New Approach of Constant Resolving of Analytical Programming

Tomas Urbanek
Zdenka Prokopova
Radek Silhavy
Ales Kuncar
Department of Computer and Communication Systems
Tomas Bata University in Zlin
Nad Stranemi 4511
Email: turbanek@fai.utb.cz

**KEYWORDS**

Analytical Programming; Constant Creation; Differential Evolution; Symbolic Regression

**ABSTRACT**

This papers' aim is to provide the Artificial Intelligence community with a better tool for symbolic regression. In this paper, the method of analytical programming and constant resolving is revisited and extended. Nowadays, analytical programming mainly uses two methods for constant resolving. The first method is meta-evolution, in which the second evolutionary algorithm is used for constant resolving. The second method uses non-linear fitting algorithm. This paper reveals the third method, which use the basic mathematics to generate constants. The findings of this study have a number of important implications for future practice.

## INTRODUCTION

A lot of application in symbolic regression field requires some kind of algorithm for constant generation (Koza 1992), (O'Neill, Brabazon & Ryan 2002). Therefore the effective numeric constant resolving algorithm is very important. In this paper, we use analytical programming as a method for symbolic regression (Zelinka, Davendra, Senkerik, Jasek & Oplatkova 2011). This method based on existence of any kind of evolutionary algorithm which generate a pointers to function tables. Analytical programming (AP) from these pointers maps and assemble a final regression function. In the analytical programming algorithm, two main approaches can be selected for constant resolving. The first approach is meta-evolution, in which the second or slave evolutionary algorithm is used for constant resolving. The second approach is to use of non-linear fitting algorithm. Both approaches added to analytical programming a lot of complexity. This study introduces a new approach for constant resolving in analytical programming algorithm. This approach is founded on basic mathematical calculation.

Section II is devoted to the original algorithm of analytical programming. Section III presents the new approach for constant resolving. Section IV presents the methods used for this study. Section V summaries the results of this research. Finally, Section VI presents the conclusions of this study.

### *Differential Evolution*

Differential Evolution is an optimization algorithm introduced by Storn and Price in 1995, (Storn & Price 1995). This optimization method is an evolutionary algorithm based on population, mutation and recombination. Differential Evolution is easy to implement and has only four parameters which need to be set. The parameters are: Generations, NP, F and Cr. The Generations Parameter determines the number of generations; the NP Parameter is the population size; the F Parameter is the Weighting Factor; and the Cr Parameter is the Crossover Probability, (Storn 1996). In this research, the differential evolution is used as an analytical programming engine.

### *Analytical Programming*

Analytical Programming, is a symbolic regression method. The core of analytical programming is a set of functions and operands. These mathematical objects are used for the synthesis of a new function. Every function in the analytical programming set core has its own varying number of parameters. The functions are sorted according to these parameters into General Function Sets (GFS). For example, $GFS_{1par}$ contains functions that have only one parameter e.g. $sin()$, $cos()$, or other functions. AP must be used with any evolutionary algorithm that consists of a population of individuals for its run (Oplatkova, Senkerik, Zelinka & Pluhacek 2013).

The function of analytical programming can be seen in Figure 1. In this case, Evolutionary Algorithm is Differential Evolution. The initial population is generated using Differential Evolution. This population, which must consist of natural numbers, is used for analytical programming purposes. The analytical programming then constructs the function on the basis of this population. This function is evaluated by its Cost Function. If the termination condition is met, then the algorithm ends. If the condition is not met, then Differential Evolution creates a new population through the Mutation and Recombination processes. The whole process continues with the new population. At the end of the analytical programming process, it is assumed that one has a function that is the optimal solution for the given task.

Proceedings 30th European Conference on Modelling and Simulation ©ECMS Thorsten Claus, Frank Herrmann, Michael Manitz, Oliver Rose (Editors)
ISBN: 978-0-9932440-2-5 / ISBN: 978-0-9932440-3-2 (CD)

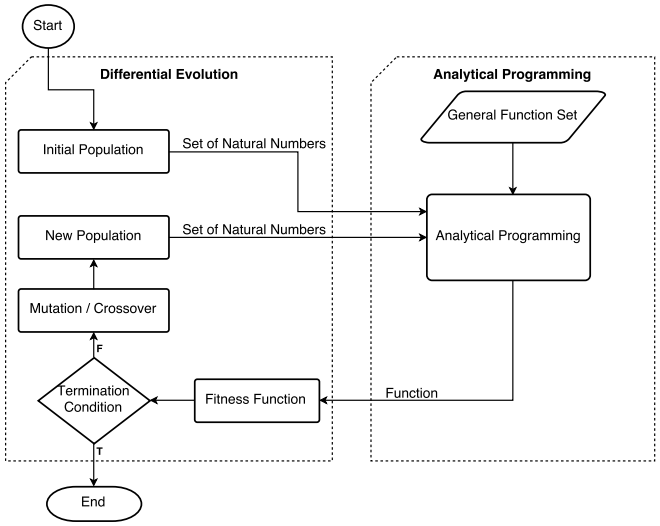


Fig. 1. Scheme of Analytical Programming with Differential Evolution algorithm

## ORIGINAL ALGORITHM

Let's have the individual of the $n$ length

$$\mathbf{ind} = (x_1, x_2, \ldots, x_n)$$

where $x_i \in \mathbb{R}^+$.

This individual is then rounded

$$\mathbf{ind_r} = (\|x_1\|, \|x_2\|, \ldots, \|x_n\|)$$

where $\|.\|$ is nearest integer function.

Let's have a set called $GFS_{all}$ which consists of $m$ functions

$$GFS_{all} = \{\{f_1, fp_1\}, \{f_2, fp_2\}, \ldots, \{f_m, fp_m\}\}$$

where $f_m$ is function and $fp_m$ is number of parameters of function $f_m$.

Then these functions are sorted to 4 sets : $GFS_0$, $GFS_{1,0}$, $GFS_{2,1,0}$ and $GFS_{all}$.

TABLE I. $GFS$S BY PARAMETERS

| |
|---|
| $GFS_0 = \{\{f_1, 0\}, \{f_2, 0\}, \ldots, \{f_n, 0\}\} \subset GFS_{all}$ |
| $GFS_1 = \{\{f_1, 1\}, \{f_2, 1\}, \ldots, \{f_n, 1\}\} \subset GFS_{all}$ |
| $GFS_2 = \{\{f_1, 2\}, \{f_2, 2\}, \ldots, \{f_n, 2\}\} \subset GFS_{all}$ |
| $GFS_{1,0} = GFS_1 \cup GFS_0$ |
| $GFS_{2,1,0} = GFS_2 \cup GFS_1 \cup GFS_0$ |

The sets from table I are expanded to the maximum value from the individual because we expected that the value of each number in individual could be higher then the length of $GFS$.

TABLE II. EXAMPLE OF $GFS$S, WHEN MAXIMUM VALUE OF INDIVIDUAL IS 6

| |
|---|
| $GFS_0 = \{\{K, 0\}, \{x, 0\}, \{K, 0\}, \{x, 0\}, \{K, 0\}, \{x, 0\}\}$ |
| $GFS_{1,0} = \{\{Sin, 1\}, \{Cos, 1\}\{K, 0\}, \{x, 0\}\{Sin, 1\}, \{Cos, 1\}\}$ |
| $GFS_{2,1,0} = \{\{Plus, 2\}, \{Minus, 2\}, \{Sin, 1\}, \{Cos, 1\}, \{K, 0\}, \{x, 0\}\}$ |
| $GFS_{all} = \{\{Plus, 2\}, \{Minus, 2\}, \{Sin, 1\}, \{Cos, 1\}, \{K, 0\}, \{x, 0\}\}$ |

Then we need to construct a matrix with functions mapped by individual.

After that we have

$$\mathbf{function} = ((f_1, fp_1), (f_2, fp_2), \ldots, (f_n, fp_n))$$

where $f_n$ is function and $fp_n$ is number of parameters of function $f_n$.

Then we need to choose which function is applied to which function. The values are pointers to the functions in $GFS$s. After that, we have constructed function; however, there is a constant $K$, which have to be resolved. Now we have two possibilities

- Meta-evolution e.g. Differential Evolution
- Non-linear least square fitting for example Levenberg-Marquardt

## NEW APPROACH

Let's have the individual of the $n$ length

$$\mathbf{ind} = (x_1, x_2, \ldots, x_n)$$

where $x_i \in \mathbb{R}^+$.

This individual is then rounded

$$\mathbf{ind_f} = (\lfloor x_1 \rfloor, \lfloor x_2 \rfloor, \ldots, \lfloor x_n \rfloor)$$

where $\lfloor . \rfloor$ is round to floor.

Now we can construct a difference between $\mathbf{ind}$ and $\mathbf{ind_f}$.

$$\mathbf{ind_c} = \mathbf{ind} - \mathbf{ind_f}$$

In vector $\mathbf{ind_f}$ are pointers to $GFS$s and $\mathbf{ind_c}$ are corresponding constants.

The decimal numbers in $\mathbf{ind_c}$ are in range $< 0, 1 >$. These numbers could be easily converted to constants into the chosen range.

Let's have a set called $GFS_{all}$ which consists of $m$ functions

$$GFS_{all} = \{\{f_1, fp_1\}, \{f_2, fp_2\}, \ldots, \{f_m, fp_m\}\}$$

where $f_i$ is function and $fp_i$ is number of parameters of function $f_i$.

Then this functions are sorted to 4 sets : $GFS_0$, $GFS_{1,0}$, $GFS_{2,1,0}$ and $GFS_{all}$.

The next key change in analytical programming algorithm is function selection. In this new approach, the $GFS$s are not expanded to the maximum value of the individual. The selection of function is controlled by modulo function. On the position where the constant $K$ will be mapped; we can read a constant number from the $\mathbf{ind_c}$ vector. The original mapping algorithm is the same. Now we have constructed function with resolved constants.

## PROBLEM STATEMENT

The overall research question to be answered within the study is whether there is a possibility to outperformed the original analytical programming method. This section presents the design of the research question. We performed experiments to get an insight in the constant resolving of analytical programming. The research question of our study can be outlined as follows:

RQ: Analysing the impact of new approach on the calculation duration and minimization performance of analytical programming.

The research question (RQ) aims to get an insight on the new approach of constant resolving of analytical programming and understand the actual effectiveness of this technique. For this reason, we use 3 different methods for constant resolving. Analytical programming with differential evolution and two new versions of proposed algorithm. Then, we try to outperformed the original constant resolving algorithm of analytical programming. To asses the performance of fitness function, we used descriptive statistics.

## METHOD

New constant resolving algorithm for analytical programming was tested for searching regression functions. Results were compared by descriptive statistics.

Following functions have been tested

- $f(x) = 45.5$
- $f(x) = 3x + 0.65$
- $f(x) = 2.3x^2 - 20x - 5.6$
- $f(x) = 3.65 * sin(2x)$

These functions were selected with emphasis on constant resolving. Functions such as constant, linear, quadratic and harmonic were tested. There was generated 20 points for each function. And the task for analytical programming was to fit these points. Three methods of constant resolving were tested.

- Analytical programming with differential evolution further referred to as AP+DE
- New analytical programming version with constant range $< -1000, 1000 >$ further referred to as AP2(-1000,1000)
- New analytical programming version with constant range $< 0, 10 >$ further referred to as AP2(0,10)

TABLE III. SYSTEM CONFIGURATION

| Parameter | Value |
|---|---|
| CPU | AMD Phenom II X2 3GHz |
| RAM | 8 GB |
| Operation system | Windows 7 Professional 64 bit |
| Programming language | LUA 5.2 |

Table III shows the system configuration for performing tests.

Table IV shows the analytical programming set-up. The number of leafs (functions built by analytical programming can be seen as trees) was set to 16. This value was sufficient for the purpose of this paper.

TABLE IV. SET-UP OF ANALYTICAL PROGRAMMING

| Parameter | Value |
|---|---|
| Number of leafs | 16 |
| GFS - functions | plus, minus, multiply, divide, power, log, log10, exp, sqrt, floor, ceil, abs, sin, cos |
| GFS - constants | $x$, $K$ |

TABLE V. SET-UP OF DIFFERENTIAL EVOLUTION

| Parameter | Value |
|---|---|
| NP | 45 |
| Generations | 300 |
| F | 0.2 |
| Cr | 0.9 |

Table V shows the set-up of differential evolution. The best set-up of differential evolution is the subject of further research.

### A. *Fitness function*

Fitness function used for this task is as following:

$$LAD = \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{1}$$

where $y_i$ is actual value and $\hat{y}_i$ is predicted value.

## RESULTS

For each function, 100 equations were calculated by the aforementioned constant resolving approaches for statistical evaluation. There were collected information about duration and least absolute deviation error.

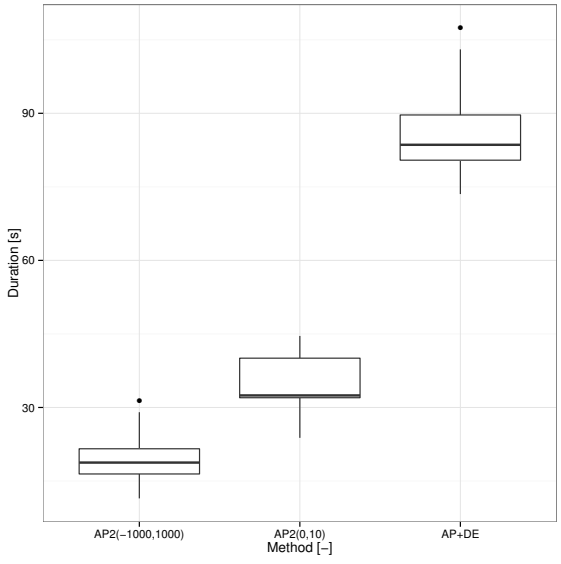


Fig. 2. Comparison of time duration of each method for function $f(x) = 45.5$

Figure 2 depicts the box plot comparison of time duration for each constant resolving method. As can be seen,

the new approach for constant resolving with constant range (-1000,1000) was nearly three times faster than analytical programming with differential evolution. AP2(0,10) performed slightly worse than AP2(-1000,1000).

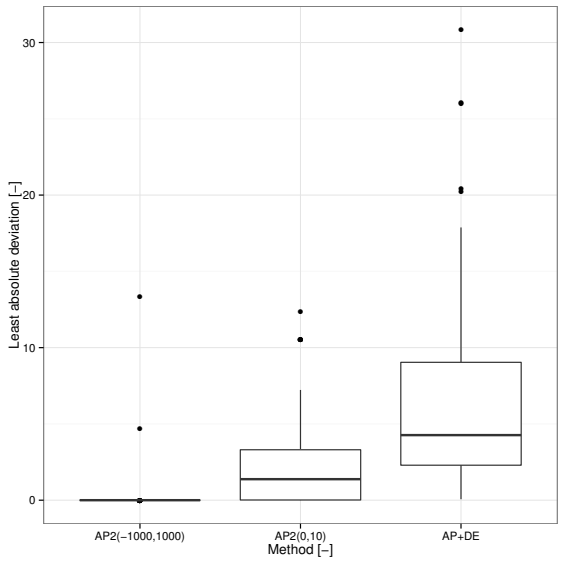


Fig. 3. Comparison of LAD error of each method for function $f(x) = 45.5$

Figure 3 depicts the box plot comparison of LAD error for each constant resolving method. As can be seen, AP2(-1000,1000) find the minimum nearly in each of 100 equations. All presented approaches find the minimum. AP2(0,10) method perform as the second best.

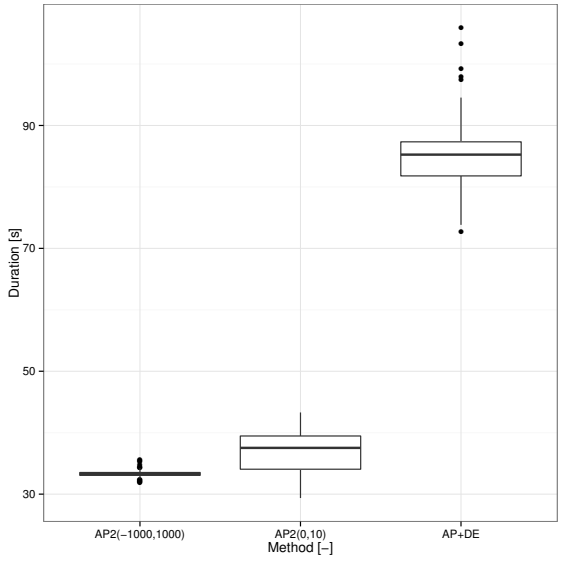


Fig. 4. Comparison of time duration of each method for function $f(x) = 3x + 0.65$

Figure 4 depicts the box plot comparison of time duration for each constant resolving method. As can be seen, AP2(-1000,1000) and AP2(0,10) performed nearly three times faster than analytical programming with differential evolution constant resolving. AP2(0,10) performed slightly worse than AP2(-1000,1000).

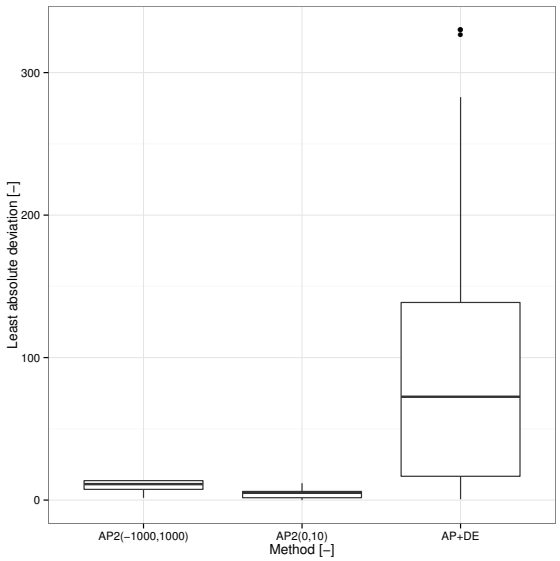


Fig. 5. Comparison of LAD error of each method for function $f(x) = 3x + 0.65$

Figure 5 depicts the box plot comparison of LAD error for each constant resolving method. As can be seen, AP2 generated lower error than 25% of AP+DE approach. There also can be seen a very low variance in AP2 method.

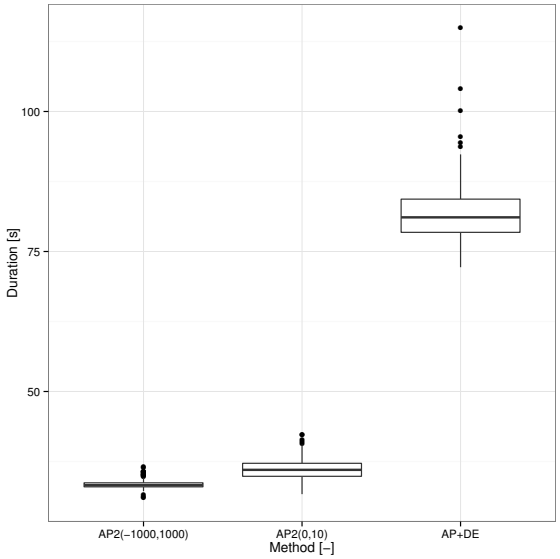


Fig. 6. Comparison of time duration of each method for function $f(x) = 2.3x^2 - 20x - 5.6$

Figure 6 depicts the box plot comparison of time duration for each constant resolving method. As can be seen, AP2(-1000,1000) and AP2(0,10) performed nearly two times faster than analytical programming with differential evolution constant resolving. AP2(-1000,1000) performed slightly worse than AP2(0,10).

Figure 7 depicts the box plot comparison of LAD error for each constant resolving method. None of the presented approaches find the minimum value. AP2(0,10) performed

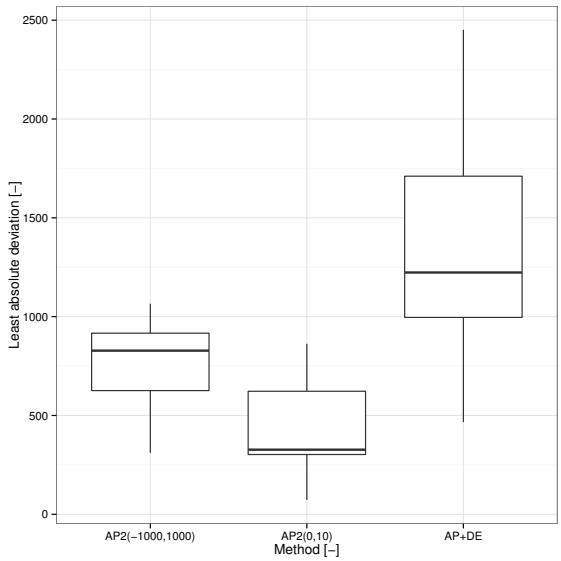


Fig. 7. Comparison of LAD error of each method for function $f(x) = 2.3x^2 - 20x - 5.6$

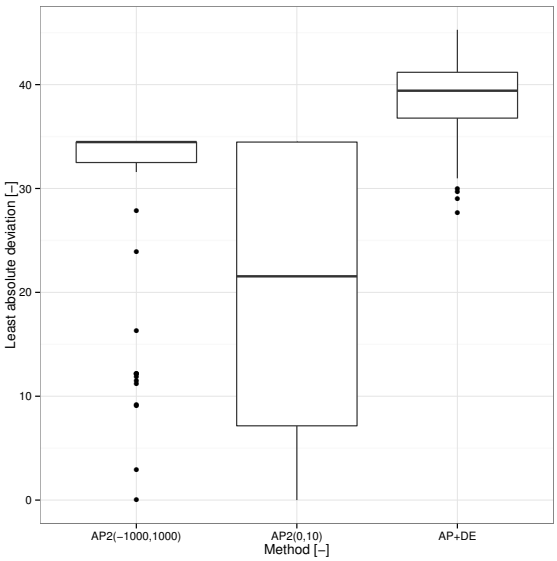


Fig. 9. Comparison of LAD error of each method for function $f(x) = 3.65 * sin(2x)$

better than other presented approaches. As can be seen, AP2 approach had lower variance than AP+DE approach.

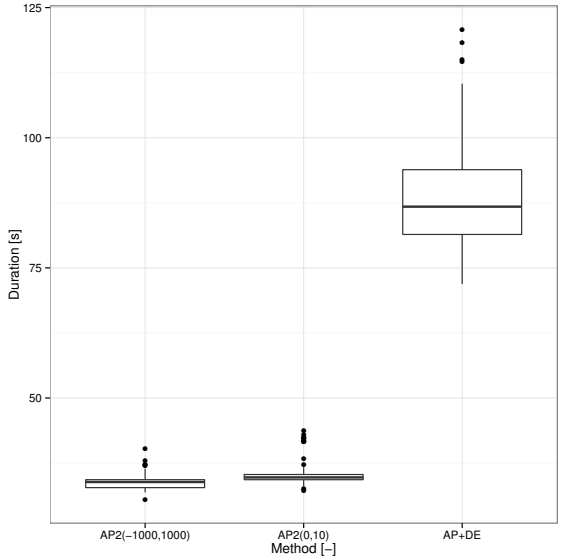


Fig. 8. Comparison of time duration of each method for function $f(x) = 3.65 * sin(2x)$

Figure 8 depicts the box plot comparison of time duration for each constant resolving method. As can be seen, AP2(-1000,1000) and AP2(0,10) performed nearly three times faster than analytical programming with differential evolution constant resolving. AP2(-1000,1000) and AP2(0,10) performed nearly identical.

Figure 9 depicts the box plot comparison of LAD error for each constant resolving method. Only AP2(0,10) find the minimum value. AP+DE and AP2(-1000,1000) perform notably worse than AP2(0,10).

Table VI summarises the statistics for each method and equation. As can be seen, the hardest equation to minimize was equation E3 where the minimum value was 71,84 for AP2(0,10). The new approach also has lower values for means and medians than meta-evolution approach AP+DE.

## DISCUSSION

The study started out with a goal to answer the question of whether the new constant resolving technique outperforms the standard constant resolving solution in analytical programming algorithm. This question is answered in the result section.

There is question (RQ), which must be answered. For answering this question, we need to study figures in result section. As could be seen in figures 2, 4, 6 and 8, the new approach could achieve up to 3 times lower calculation duration than standard approaches. These results were expected, because we remove time complexity of constant resolving using another differential evolution. The most surprising aspect of the results is in the minimization performance. The figures 3, 5, 7 and 9 depicted that the minimization performance are more stable and the new approach finds more accurate minimum value; however, this could be caused by the setting of slave differential evolution.

## THREATS OF VALIDITY

It is widely recognised that several factors can bias the validity of simulation studies. Therefore, our results are not devoid of validity threats.

### *External validity*

External validity questions whether the results can be generalized outside the specifications of a study (Milicic & Wohlin 2004). The first validity issue to mention is that either analytical programming nor differential evolution has been exhausted via fine-tuning. Therefore, future work is required to exhaust all the parameters of these methods to

TABLE VI. STATISTICAL COMPARISON OF TESTED METHOD

| Equation Method | Minimum | 1st Qu. | Median | Mean | 3rd Qu. | Maximum |
|---|---|---|---|---|---|---|
| E1 : AP2(-1000,1000) | 0,00 | 0,00 | 0,00 | 0,18 | 0,00 | 13,36 |
| E1 : AP2(0,10) | 0,00 | 0,01 | 1,38 | 2,31 | 3,31 | 12,37 |
| E1 : AP+DE | 0,07 | 2,29 | 4,27 | 6,17 | 9,04 | 30,83 |
| E2 : AP2(-1000,1000) | 1,58 | 7,52 | 11,20 | 10,28 | 13,65 | 13,65 |
| E2 : AP2(0,10) | 0,00 | 1,66 | 5,12 | 4,38 | 6,07 | 11,87 |
| E2 : AP+DE | 0,62 | 16,70 | 72,52 | 93,85 | 138,68 | 330,06 |
| E3 : AP2(-1000,1000) | 310,80 | 625,90 | 827,90 | 798,00 | 916,30 | 1065,30 |
| E3 : AP2(0,10) | 71,84 | 302,40 | 326,83 | 433,71 | 622,45 | 862,60 |
| E3 : AP+DE | 466,10 | 996,20 | 1223,30 | 1345,80 | 1710,70 | 2451,50 |
| E4 : AP2(-1000,1000) | 0,01 | 32,50 | 34,47 | 31,17 | 34,47 | 34,47 |
| E4 : AP2(0,10) | 0,00 | 7,15 | 21,55 | 20,42 | 34,47 | 34,47 |
| E4 : AP+DE | 27,65 | 36,78 | 39,42 | 38,78 | 41,19 | 45,28 |

use their best versions. Threat to external validity could be also the implementation of the analytical programming and differential evolution algorithms. Although we used standard implementations, there is considerable amount of code, which could be the threat to validity.

## CONCLUSIONS

In this paper, the new approach of constant resolving in analytical programming algorithm was presented. The presented approach is founded on basic mathematical calculations. The main benefit of this solution is that, there are no another calculation of evolutionary algorithm or non-linear fitting algorithm for constant resolving. There is also a benefit that there is no need to set a slave meta-evolution algorithm. Nevertheless, the range for constants must be set. Future research should therefore concentrate on the investigation of a proper set-up for analytical programming constant range.

## ACKNOWLEDGEMENT

This study was supported by the internal grant of Tomas Bata University in Zlin No. IGA/FAI/2016/035 funded from the resources of specific university research.

## REFERENCES

### REFERENCES

Koza, J. (1992), *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press.

Milicic, D. & Wohlin, C. (2004), Distribution patterns of effort estimations, *IEEE Conference Proceedings of Euromicro 2004, Track on Software Process and Product Improvement* pp. 422–429. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1333398

O'Neill, M., Brabazon, A. & Ryan, C. (2002), *Genetic Algorithms and Genetic Programming in Computational Finance*, Springer US, Boston, MA, chapter Forecasting Market Indices Using Evolutionary Automatic Programming, pp. 175–195. http://dx.doi.org/10.1007/978-1-4615-0835-9_8

Oplatkova, Z. K., Senkerik, R., Zelinka, I. & Pluhacek, M. (2013), Analytic programming in the task of evolutionary synthesis of a controller for high order oscillations stabilization of discrete chaotic systems, *Computers & Mathematics with Applications* 66(2), 177–189. http://www.sciencedirect.com/science/article/pii/S0898122113001004

Storn, R. (1996), On the usage of differential evolution for function optimization, *Proceedings of North American Fuzzy Information Processing* pp. 519–523. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=534789

Storn, R. & Price, K. (1995), *Differential Evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces*, Vol. 11, Technical Report TR-95-012.

Zelinka, I., Davendra, D., Senkerik, R., Jasek, R. & Oplatkova, Z. (2011), *Analytical programming-a novel approach for evolutionary synthesis of symbolic structures*, InTech, Rijeka.

**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

**ZDENKA PROKOPOVA** was born in Rimavska Sobota, Slovak Republic in 1965. She graduated from the Slovak Technical University in 1988, with a Masters degree in Automatic Control Theory. She received her Technical Cybernetics Doctoral degree in 1993 from the same university. She worked as an Assistant at the Slovak Technical University from 1988 to 1993. During 1993-1995, she worked as a programmer of database systems in the Datalock business firm. From 1995 to 2000, she worked as a Lecturer at Brno University of Technology. Since 2001, she has been at Tomas Bata University in Zlin, in the Faculty of Applied Informatics. She presently holds the position of Associate Professor at the Department of Computer and Communication Systems. Her research activities include programming and applications of database systems, mathematical modelling, computer simulation and the control of technological systems.

**RADEK SILHAVY** was born in Vsetin in 1980. He received a B.Sc. (2004), M.Sc. (2006), and Ph.D. (2009) in Engineering Informatics from the Faculty of Applied Informatics, Tomas Bata University in Zlin. He is a Senior Lecturer and researcher in the Computer and Communication Systems Department. His Ph.D. research was on The Verification of the Distributed Schema for the Electronic Voting System. His major research interests are software engineering, empirical software engineering and system engineering.

**ALES KUNCAR** was born in Prerov in 1989. He received a B.Sc. (2012), M.Sc. (2014) in Computer and Communication Systems 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 navigation systems, MEMS sensors and mathematical modelling.