GENETIC PROGRAMMING WITH EITHER STOCHASTIC OR DETERMINISTIC CONSTANT EVALUATION

V. Hlavac[*]

Abstract: Constant evaluation is a key problem for symbolic regression, one solved by means of genetic programming. For constant evaluation, other evolutionary methods are often used. Typical examples are some variants of genetic programming or evolutionary systems, all of which are stochastic. The article compares these methods with a deterministic approach using exponentiated gradient descent. All the methods were tested on single sample function to maintain the same conditions and results are presented in graphs. Finally, three different tasks (ten times each) are compared to check the reliability of the methods tested in the article.

Key words: *genetic programming, constant evaluation, gradient descent, symbolic regression*

Received: May 15, 2017 **DOI:** 10.14311/NNW.2018.28.007
Revised and accepted: April 9, 2018

1. Introduction

In real world problems, the role of constants is important. For example, real data can have the form of a harmonic signal. It is very unusual if the period and magnitude of the signal are both either one or some small integer number. Symbolic regression, conducted using genetic programming, will eventually find some combination of available constants, approximating to the required values. However, the resulting function will be very complicated, while the real function is just a sinusoidal function with two multiplications.

Early publications concerning genetic programming [10] reject the use of constants. Such an approach arises from its biological equivalent, in which there are no numbers, and from the idea that programs will write other programs (as a first step in creating artificial life), which would be very complicated with the evaluation of constants. The resulting functions found in this way are rather unreadable, although they were useable in programs.

The first idea was to expand the terminal set with constants. Usually, the symbol set for genetic programming is represented by less than 20 so-called primitive functions, leaving many available symbols for a set of constants. The set of

[*]Vladimir Hlavac; Czech Technical University in Prague, Faculty of Transportation Sciences, Department of Applied Informatics in Transportation, Konviktská 20, 110 00 Prague 1, E-mail: hlavac@fs.cvut.cz

possible constants has to be chosen manually. The program can construct other possible values by combining preset constants using basic arithmetic operations. A typical example is presented in [1], describing how symbolic regression via genetic programming (GP) was used in the optimization of a pharmaceutical zero-order release matrix tablet. In two experiments, constants were generated in the range of 100 with a resolution of 0.01 and in the range of 127 with a resolution of 1, both with a probability of 0.1 (rather low). The experiments used relatively large populations (500-1000-1500 in the first experiment, 12 500 in the second). The authors compared performances with a neural network solution and emphasized that the genetic programming solution provides more understandable results and allows further analysis.

An even older approach was so-called constant variation. Constant variation can be implemented as one of the mutations in the main cycle of genetic programming. Any constant can be modified; or constants from two individuals can be interchanged, or mutually added or multiplied/divided. The modified function can be added to the end of the population, or can be used instead of its source, but only if the new version is better. The biggest advantage of this solution is that no additional method for evaluation of constants is needed; thus, only the main cycle of genetic programming has to be used. The disadvantage is that it works blind, leaving possible improvement only to chance. An example is used in [16]. The author created and published a C-library for genetic programming (at the time of presentation, the source code was available).

The most popular method of an evaluation of the parameters in the genetic programming community is the use of genetic algorithms (GA). During the main genetic programming cycle, any time a new individual is created (by crossover or mutation), its constants are immediately evaluated by means of genetic algorithms. This means creating a new population and complete internal GA evolutions for each of the new GP individuals. On the other hand, there are only a few constants per GP individual to be evaluated, so only a small GA population and a limited number of generations (often 30 to 100) can be used.

Because the internal cycle of GA evolution has to be executed for each new individual of the superset genetic programming population, this method is rather slow. The advantages are that exact values can be detected and that this method is proven to be stable without local optima problems. A typical example of this solution is [11]. A similar method is used in [2] and explained in [3], with the name GPA-ES (ES stands for evolution strategies, representing here GA with so-called intelligent crossover).

In [14], the differential evolution method is used. A different method of evaluating genetic algorithm represented vectors is in [15], described as analytical programming. For the evaluation of constants, the SOMA algorithm is used (related to swarm intelligence).

A more straightforward approach is described in [7], where a gradient method is implemented. Because of a different definition of the problem, the Exponentiated Gradient Descent [9] method was used, alternatively with or without a variable step rate, known from the Simulated Annealing [8].

2. The data

For testing genetic programming with respect to a symbolic regression with dual variable problems, function (1) was used. The advantage is that this function comes from a list of goniometric equivalences; the program was able to find one possible variant

$$f(x, y) = \sin(x + y). \tag{1}$$

Data were modified by adding Gaussian noise with a standard deviation of 0.1. For the testing of genetic programming, the function was modified by two constants. To compare the abilities of different methods of constant evaluation, a third constant was introduced (its value was 1).

The resulting function without added noise is given by (2)

$$f(x, y) = k_1 \sin(k_2 x + k_3 y). \tag{2}$$

Data were generated in the range of $\pm\pi$ for both variables. Constants were set to the values of 3.71, 1.73 and 1. Fig. 1 shows the graph of the function without constants, but with noise.

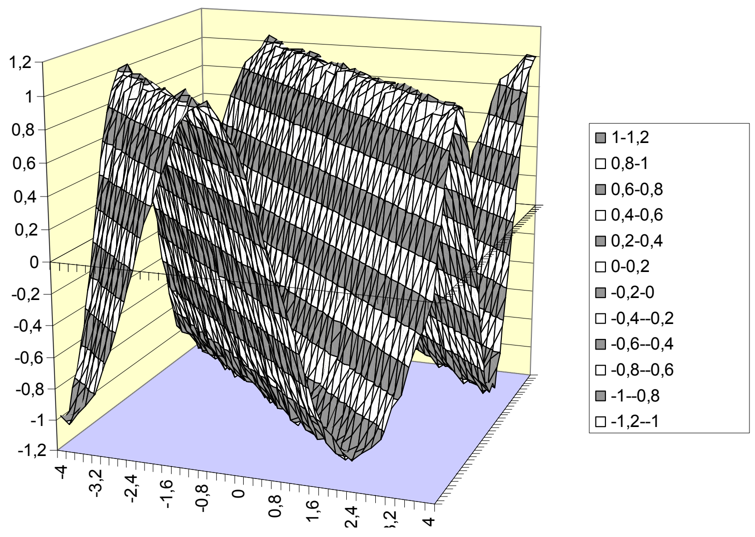

Fig. 1 *Graph of the testing function.*

The fitness function (which genetic programming should minimize) is defined as the sum of squares of the errors in individual points, where the searched function is known [13]:

$$F_k = \sum_{i=1}^{n} (G(x_i, y_i) - f_i(x_i, y_i))^2 = \sum_{i=1}^{n} (z_i - f_i(x_i, y_i))^2. \tag{3}$$

3. The program

All the tested methods were implemented into the program described in [7]. In this program, a slightly different method of constant implementation is used; instead of a leaf object in the tree of the represented function, a point where a multiplicative constant is used is only marked in the tree. The following picture describes the tree, representing (2) in this program, and its translation to standard genetic programming representation.

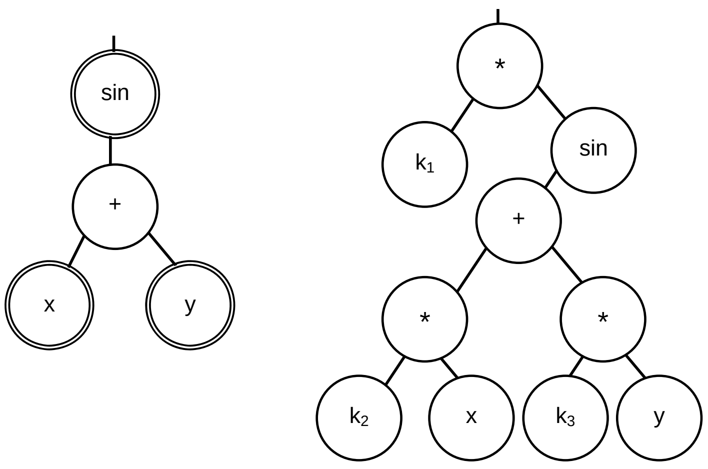

Fig. 2 *A tree representation in the used application using (pre)evaluated constants as a multiplication in marked positions in the tree, storing them in an array in order of use. This can be redrawn using multiplication to create a standard tree representation.*

When comparing the methods, it is important to compare them in the same situation. For this purpose, a direct tree string insertion was implemented. Once a function proposed by the user is given to the application, it evaluates the coefficients by the currently chosen method and presents the given and resulting values in a graph showing the constants and the resulting error value. If the progress of the main genetic programming is being recorded to a log file, then the progress of the chosen method is written there too. It allows graphs, e.g. those presented in this paper, to be generated.

4. Methods

For each method, several constant settings were chosen and tested. Each test was repeated five or seven times. After checking the results, the *median* of the test runs was chosen to represent the method and plotted in graphs. The average cannot be used, because it could be influenced by an individual residual error, and use the best solution as a representation of an individual stochastic method can be biased by a random occasional success.

Note the presence of noise. The best possible value should be 0.28. Sometimes, individual methods return a better error. This is an example of overtraining. In the case of symbolic regression using genetic programming, this is not a problem, because the resulting function is not more complex than the correct one.

The horizontal axis represents the number of evaluations of the fitness function. Because real time depends on the quality of the used program, only this number can be comparable [4]. Fitness function evaluation is most time consuming; because of the tree structure of a genetic programming individual, it is usually achieved by recursion and function interpretation.

4.1 Gradient descent

The gradient descent method is well known, for example, from neural networks, where it creates the main part of back-propagation and other methods developed from this. It has been developed into the hill-climbing method, and both are today nearly synonymous.

For symbolic regression, most of the dependencies sought can be represented as dependencies between physical values. A physical value is basically represented as multiplication as physical unit and a numerical value. So this is a good reason to expect all constants to be multiplicative. For this kind of constant, it is better to use a modified method, exponentiated gradient descent [9]. In this version, instead of adding a value in the direction of the gradient, a multiplication by exponentiated gradient is used

$$k_{t+1}^{(i)} = k_t^{(i)}.e^{-\nabla_\gamma(f(x,y,\gamma_t))\gamma_t^{(i)}}. \tag{4}$$

The method has been tested with a regularly decreasing (variable) step rate. Only the influence of this has been tested. Note, that EGD is a deterministic method, while the others are stochastic. If EGD is repeated, exactly the same values are returned.

The results are shown in the Fig. 3, the vertical axe represents fitness value, while the horizontal represents the number of the fitness function evaluation. The values in the names of the lines in the graph are the teaching coefficient γ. The ratio between the two numbers in the geometric sequence is calculated as $(1 + \gamma)$. If the ratio is too small, a final value cannot be found in reasonable time (line EGD 0.05; number 0.05 in the name of the line denotes the value of the teaching coefficient γ). If it is too big, the step in the gradient descent becomes too small to be able to influence coefficient values. In this case, the graph descent is fast, but stops changing after some number of steps (line EGD 0.50).

The last graph is labelled EGD GPy. This is a native setting of the used application, where the teaching coefficient γ of 0.3 (ratio 1.3) is used, but restarted five times from different points of the geometric sequence. This setting was part of the development of this application [7].

4.2 Evolution strategies

Evolution strategies [6] is an evolutionary computational method, where new individuals are generated as descendants of multiple individuals from previous gen-

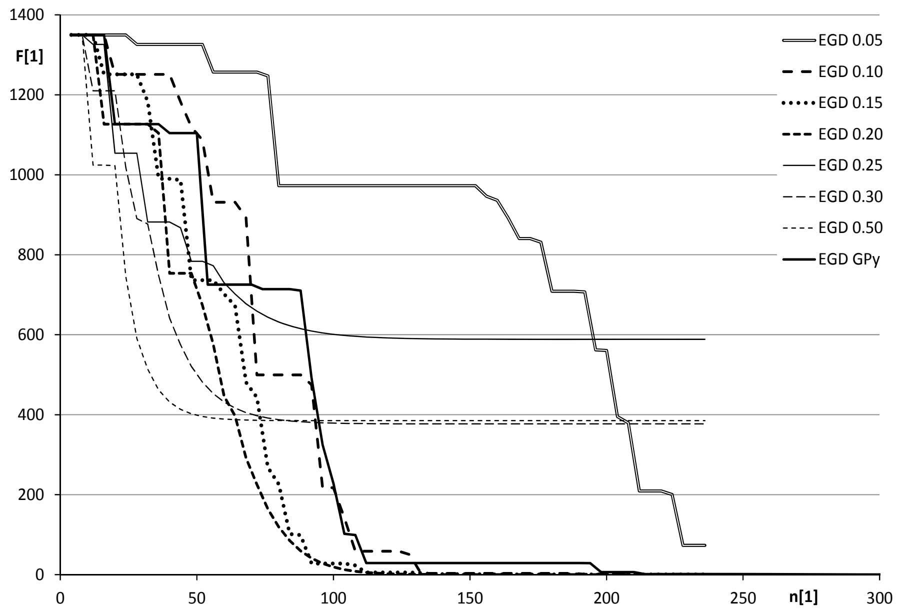

Fig. 3 *Results of exponentiated gradient descent. The method is deterministic: for a given condition it returns the same result. In this graph, a single run is presented, instead of the median (which would be the same).*

erations. The number of individuals in the base population is denoted as μ, the number of parents for a new individual as ρ, and the numbers of newly generated individuals as λ. The biggest difference from the genetic algorithm is that there can be more parents. The resulting child is generated as an arithmetic average of the parents, possibly modified by mutation. Another difference is that there is no evolutionary pressure, as when roulette wheel selection is used.

Two variants were implemented. In evolution strategies, it is possible that $\rho = \mu$, so all of the individuals are used. The average is calculated and from this new point, all the children are generated only by mutation. In evolutionary strategies terminology, this process can be described as ES$(\mu/\rho, \lambda)$, $\mu = \rho$. The comma before λ means that the old generation has been replaced.

The second implemented variant was with more parents, but here limited by $\rho < \mu/2$. The newly generated individuals are compared with the old generation and the best μ individuals from both populations are selected. This method can be described as ES$(\mu/\rho + \lambda)$.

4.2.1 Evolutionary strategies with $\rho = \mu$

The results were not useable. This variant gave no useable results as the value of the fitness function was increased each time, independent of the settings of the ρ from 5 to 30 and the mutations from the 10 % to 30 %. While copying only the average population to the next cycle, it rapidly lost stability. After the method was modified to keep the best individual from the previous generation, it returned this as a constant value. This method proves itself as not suitable for the task of the coefficient evaluation. Graphs are not included here.

4.2.2 Evolutionary strategies with $\rho \ll \mu$

In this case, the evolutionary strategies method resembles genetic algorithms, at least by tending to lose all population diversity and resulting in a frozen population. A smaller number of parents with a higher level of mutation maintains better diversity, so a test run with a population of only 10 is better than one with a population of 20 (in the case of 3 parents). If mutations are implemented at a higher rate, the behavior is better; however, a higher level of mutations prevents this method from finding a combination of constants to obtain lower values of fitness. In the graph in the Fig. 4, the median is presented; using mutation, this method finds a solution in one out of five tests (or in two out of five tests, in the case of 3 parents and 10 individuals in the population, 40 offspring, and 30 mutated; in this case, 50 cycles represent 3570 fitness function evaluations, far outside the range of the Fig. 4). On the graph, the vertical axe represents fitness value, while the horizontal represents the number of the fitness function evaluation.

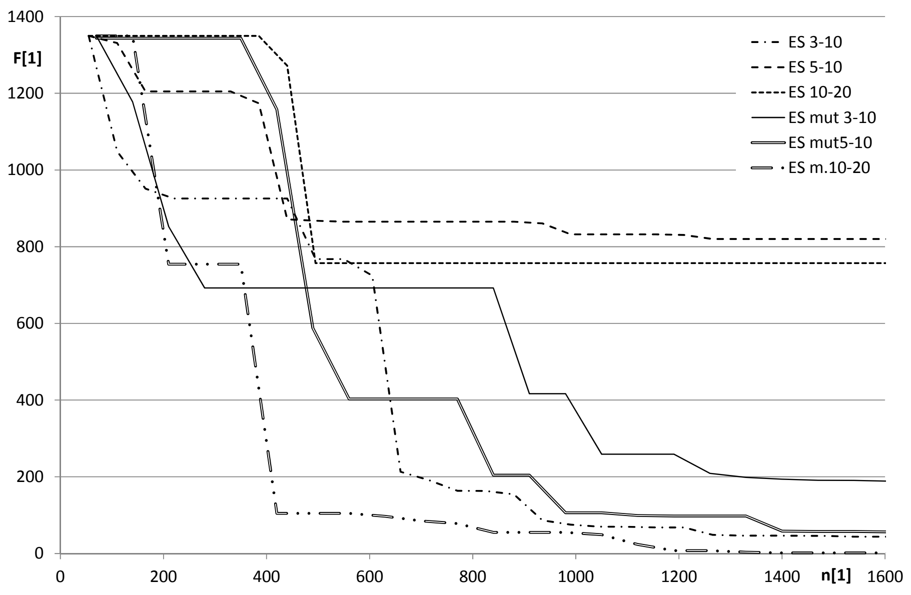

Fig. 4 *Evolutionary strategy ES(μ/ρ+λ) results. If the mutation is active, the probability of a frozen population is lower. Each of the settings was tested seven times, the median run was selected by the error value for the next value after 1000 fitness evaluations and only this run is plotted.*

4.3 Genetic algorithms for real number vectors

This type of genetic algorithms [12] represents individuals as a vector of real numbers. The crossover is implemented as a linear combination of two parents, where the ratio can be random. As inspired by biological genetic crossover, where, for example, a new individual created by the cross-breeding of two breeds of dog of different size is usually medium sized, but can often be bigger or smaller, so-called intelligent crossover is introduced [2]. In this variant, the new individual can be generated on the line connecting these two individuals, but not only inside; it can also have a smaller probability outside these points. If the parents' vector components are marked a_i and b_i and the child c_i, the child will be evaluated as

$$c_i = ka_i + (k-1)b_i, \tag{5}$$

where the ratio (k) is a random value, generated over a wider range than (0; 1); for example, in this application from -0.3 to $+1.3$ in uniform distribution.

At the beginning, the starting population is generated in some chosen range. After a new population is generated, all individuals (including the old generation) are sorted and the new generation is selected.

For the selection of individuals for crossover, roulette wheel selection is used. The probability of parent selection was chosen to decrease from the best to the worst by a geometric series. In this application, the fitness value of the individual is used only for sorting; for the selection, only the resulting order is used (better individuals are chosen with a bigger probability).

Mutations are realized as the multiplication of randomly generated values from the same range, as the generation of a new value for one vector component, or by changing the sign of a component to a negative value, in this case of fixed probabilities of 0.5, 0.25 and 0.25 (if the individual is chosen to be mutated). The best individual is never mutated (elitism).

The results are shown in the Fig. 5. The vertical axe represents fitness value, while the horizontal represents the number of the fitness function evaluation. Four

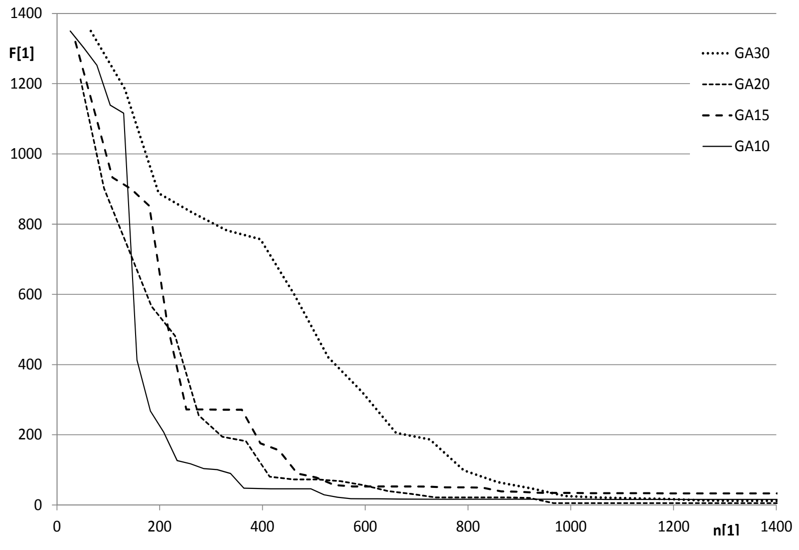

Fig. 5 *Genetic algorithms results (see text)*

settings were tested. Number in the name of the lines in the graph means population size. The number of offspring was twice the population size. The genetic pressure was 1.03 (genetic pressure is the ratio, after sorting, of the probabilities of two consecutive individuals in the population being selected as parents of the new individual). The number in the name of the line in the Fig. 5 is the genetic algorithm population size. The mutation of five randomly chosen individuals (preserving the best one) was constant for all of the tests.

4.4 Genetic algorithms with tournament selection

In this case, the method used in [2] or [5] is implemented. The starting population is generated as in the previous case. Only intelligent crossover is used (no mutation). The new individual is compared only with its parents. Population development and population variability are maintained in this case. If the new individual is better than both parents, then with some probability ($\boldsymbol{r}$) the worse parent is replaced; with the opposite probability ($1 - \boldsymbol{r}$), the better parent is replaced. Usually, if the probability of replacing the better parent is bigger, it gives better results. This agrees with the general rule of evolutionary programming, i.e., that maintaining population diversity is more important than moving to the best solution quickly. The resulting probability of keeping the better parents in the presented test was close to $\boldsymbol{r} = 0.2$.

This method works without mutations and usually without the loss of diversity. The graph in the Fig. 6 presents different population sizes. There are the values of the fitness function on the vertical axe, while the horizontal represents the number of evaluations of the fitness function. The lines in the graph marked "$\boldsymbol{r}$" compare

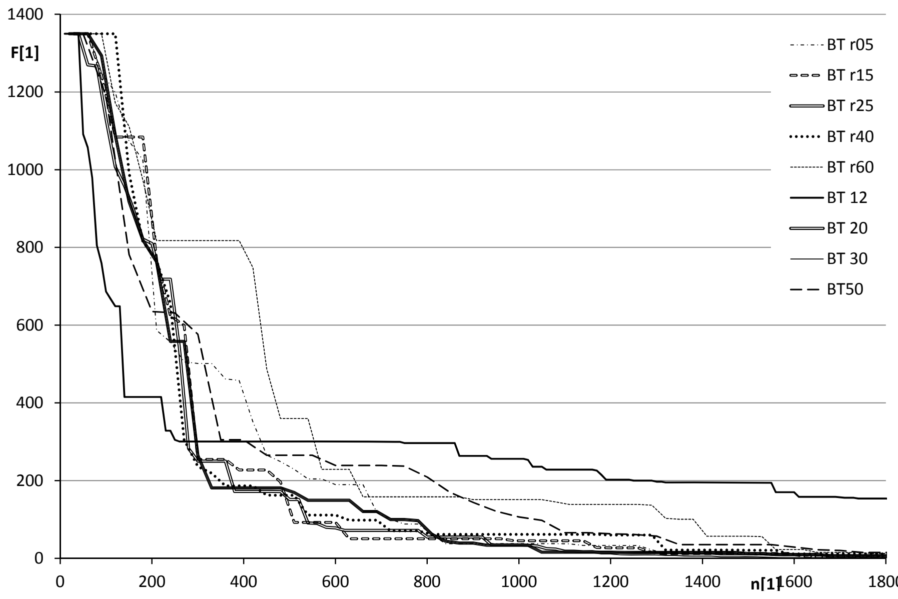

Fig. 6 *[2] uses a variant of genetic algorithms with intelligent crossover.*

the influence of the ratio of how often the worse parent will be replaced and how often the better parent will be replaced, if the child is better than both. $\boldsymbol{r} = 0.4$ represents keeping the better one with a probability of 0.4, on the graph is marked "BT r40". All the "$\boldsymbol{r}$" variants use a population of 30, and all the other probabilities are $\boldsymbol{r} = 0.25$, so two of the graphs have the same values. Only a population with 12 members lost variability in more than half of the tests. The required population size depends on the complexity of the problem.

4.5 Self-Organizing Migrating Algorithm (SOMA)

SOMA [15] is a stochastic optimization algorithm based on the social behavior of cooperating individuals (related to swarm intelligence). In this algorithm, four individuals from the population are selected. The best one is evaluated as the leader and the rest are moved in the direction from their actual coordinates (vector components) to this leader. In the Fig. 7, $\boldsymbol{A}$ represents the leader and $\boldsymbol{B}$, $\boldsymbol{C}$ and $\boldsymbol{D}$, the others. The length of the move is a multiplication of the distance between the leader and the individual, and some constant, marked as $\boldsymbol{v}$ for $\boldsymbol{B}$ in the Fig. 7.

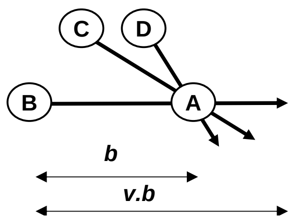

Fig. 7 *SOMA explanation.*

The behavior of the algorithm depends on $\boldsymbol{v}$. The implementation of this algorithm in the used application allows the range in which this coefficient is randomly generated to be set (the same coefficient is used for all three non-leader individuals; they are moved, and then the algorithm repeats the choosing of new four individuals. During the tests, it was shown that to achieve convergence the random multiplication coefficient $\boldsymbol{v}$ should be generated not only in the presumed range from around 1.5 to 2, but below this range, even if this meant attaining a negative value.

The behavior of the method is in the Fig. 8. There are the values of the fitness function on the vertical axe, while the horizontal represents the number of evaluations of the fitness function. The values in the names of the lines in the graph represent the range of the $\boldsymbol{v}$ parameter. Note the negative value of the first line in the legend.

5. Reliability

Constant evaluation is intended for use in the problem of symbolic regression when using genetic programming. For this purpose, the failure of correct constant evaluation is not critical. The main genetic programming evolution cycle generates

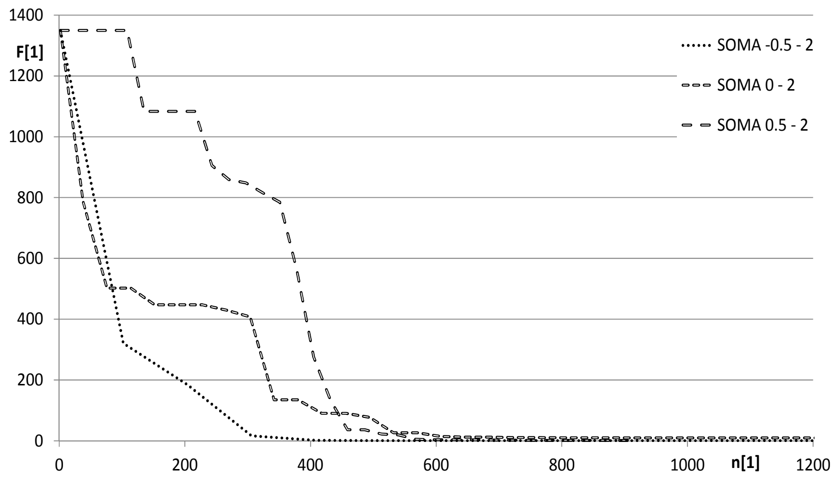

Fig. 8 *SOMA results.*

hundreds of functions regularly and if one combination of primitive functions fails with respect to constant evaluation and the next is successful, the program can still be successful. If the results are represented by the median, then at least 50 % of tests are successful, as presented. This can be enough for use in genetic programming. In many cases, speed can be more critical. Evolutionary computing is very time consuming and this can limit its application. The ordinary user can have a powerful computer, for example a game computer, but usually not a cluster.

To test the reliability, two other functions were used to generate data:

$$f(x,y) = k_1 - \ln(k_2 x + k_3 \sqrt{y}), \tag{6}$$

$$f(x,y) = k_1.\exp(k_2 x + k_3 y^3). \tag{7}$$

For each method and each function, constant evaluation was executed ten times. Tab. I presents how many of them were able to evaluate all of the constants with a precision of at least $\pm 10\,\%$.

Method	$k_1 \sin(k_2 x + k_3 y)$	$k_1 - \ln(k_2 x + k_3 \sqrt{y})$	$k_1.\exp(k_2 x + k_3 y^3)$
EGD 0.20	10	0[1]	10
ES m10/20	7	0	5
GA 10	10	3	8
BT30	10	5	9
SOMA-0.2	10	2	7

Tab. I *Number of successful tests from ten.*

[1]10, if repeated twice.

6. Conclusion

The presented results demonstrate differences among some techniques used in constant evaluation. The results demonstrate the ability of stochastic and deterministic method of constant evaluation to solve proposed problem. In the case of tested functions, the deterministic method of the gradient descent, proposed in [7], was faster (see Fig. 9). It should be noted that the way the gradient descent was implemented included an exponentiated approach and a variable step size. On the other hand, if the gradient of the fitness function lies outside reasonable values, it could fail. The test showed that this deterministic method always returns the same result, while evolutionary methods are stochastic and can still give a usable result with some probability.

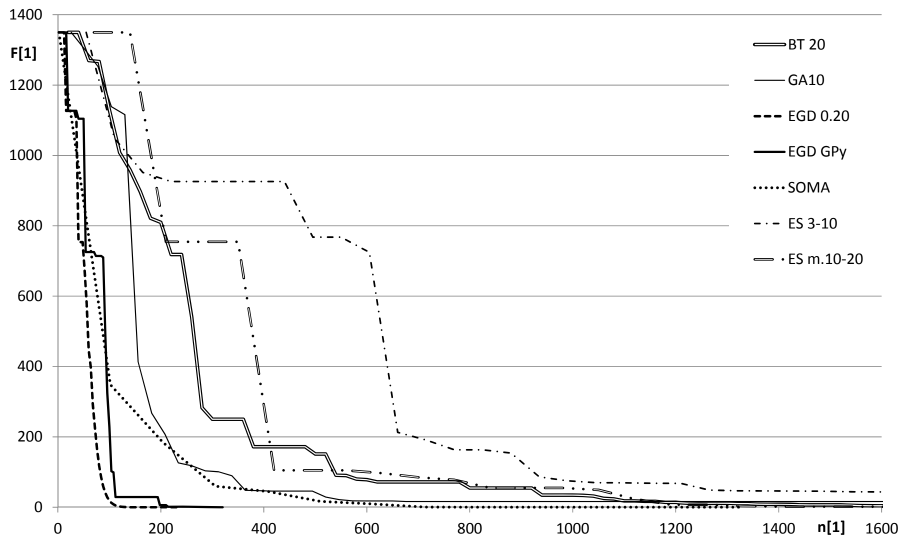

Fig. 9 *The best results from all of the graphs in one. While the deterministic EGD method found combinations with less than 200 fitness function evaluations, stochastic evolutionary methods were slower.*

References

[1] BARMPALEXISA P., KACHRIMANISA K., TSAKONASB A., GEORGARAKIS E. Symbolic regression via genetic programming in the optimization of a controlled release pharmaceutical formulation. In: *Chemometrics and Intelligent Laboratory Systems*, Elsevier. 2011, 107(1), pp. 75–82.

[2] BRANDEJSKY T. Genetic Programming Algorithm with constants pre-optimization of modified candidates of new population. In: *Mendel 2004*, Brno, 2004, pp. 34–38.

[3] BRANDEJSKY T. Evolutionary system to model structure and parameters regression. In: *Neural Network World*, 2012, 22(2), pp. 181–194. ISSN 1210-0552, doi: 10.14311/NNW.2012.22.011

[4] BRANDEJSKY T. Small populations in GPA-ES algorithm. In: *Mendel 2013*, pp. 31–36. ISBN: 978-802144755-4.

[5] BRANDEJSKY T. Influence of (p)RNGs onto GPA-ES behaviors. In: *Neural Network World*, 2017, 27(6), pp. 593–605. ISSN 1210-0552, doi: 10.14311/NNW.2017.27.033

[6] HANSEN N., ARNOLD D.V., AUGER A. Evolution Strategies. In: *Janusz Kacprzyk and Witold Pedrycz (Eds.): Handbook of Computational Intelligence*, Springer, 2015, Chapter 44, pp. 871–898. (Pdf available from: https://www.lri.fr/~{}hansen/es-overview-2015.pdf).

[7] HLAVAC V. A program searching for a functional dependence using genetic programming with coefficient adjustment. In: *2016 Smart Cities Symposium Prague (SCSP)*, 2016.

[8] KIRKPATRICK S., GELATT JR. C.D., VECCHI M.P. Optimization by simulated annealing. *Science*, 1983, 220(4598), pp. 671–680.

[9] KIVINEN J., WARMUTH M.K. Exponentiated gradient versus gradient descent for linear predictors. In: *Information and Computation*, 1997, 132(1), pp. 1–63, Elsevier. Available from: http://www.sciencedirect.com/science/article/pii/S0890540196926127. doi: 10.1006/inco.1996.2612

[10] KOZA J.R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, MA, MIT Press, 1992. ISBN 978-0262111706.

[11] LAHODIUK Y. *Genetic-programming.* GitHub, Inc., Available from: https://github.com/lagodiuk/genetic-programming (published 2014, cited 17.1.2017)

[12] MITCHELL ME. *An Introduction to Genetic Algorithms.* Cambridge, MA: MIT Press, 1996. ISBN 9780585030944

[13] POLI R., LANGDON W.B., MCPHEE N.F., KOZA J. R. *A Field Guide to Genetic Programming.* Lulu.com, 2008.

[14] WEISSER R., OSMERA P., MATOUSEK R. Transplant Evolution with Modified Schema of Differential Evolution: Optimization Structure of Controllers. In: *International Conference on Soft Computing MENDEL.* Brno: MENDEL, 2010.

[15] ZELINKA I. Analytic Programming by Means of Soma Algorithm. In: *Proc. 8th International Conference on Soft Computing Mendel'02*, Brno, Czech Republic, 2002, pp. 93–101. ISBN 80-214-2135-5

[16] ZONGKER D., PUNCH B., RAND B. *Lil-gp Genetic Programming System.* Michigan State University, 1996.