Effects of Constant Optimization by Nonlinear Least Squares Minimization in Symbolic Regression

Michael Kommenda, Gabriel Kronberger,
Stephan Winkler, Michael Affenzeller, and Stefan Wagner
Heuristic and Evolutionary Algorithms Laboratory
University of Applied Sciences Upper Austria
Softwarepark 11, 4232 Hagenberg, AUSTRIA
{michael.kommenda, gabriel.kronberger,
stephan.winkler, michael.affenzeller, stefan.wagner}
@fh-hagenberg.at

ABSTRACT

In this publication a constant optimization approach for symbolic regression is introduced to separate the task of finding the correct model structure from the necessity to evolve the correct numerical constants. A gradient-based nonlinear least squares optimization algorithm, the Levenberg-Marquardt (LM) algorithm, is used for adjusting constant values in symbolic expression trees during their evolution. The LM algorithm depends on gradient information consisting of partial derivations of the trees, which are obtained by automatic differentiation.

The presented constant optimization approach is tested on several benchmark problems and compared to a standard genetic programming algorithm to show its effectiveness. Although the constant optimization involves an overhead regarding the execution time, the achieved accuracy increases significantly as well as the ability of genetic programming to learn from provided data. As an example, the Pagie-1 problem could be solved in 37 out of 50 test runs, whereas without constant optimization it was solved in only 10 runs. Furthermore, different configurations of the constant optimization approach (number of iterations, probability of applying constant optimization) are evaluated and their impact is detailed in the results section.

Categories and Subject Descriptors

I.2.2 [**Automatic Programming**]: Program synthesis; I.2.8 [**Problem Solving, Control Methods, and Search**]: Heuristic Methods

General Terms

Algorithms, Experimentation

GECCO'13 Companion, July 6–10, 2013, Amsterdam, The Netherlands.

Keywords

Symbolic Regression, Constant Optimization, Nonlinear Least Squares Optimization, Automatic Differentation

1. INTRODUCTION

Symbolic regression is the task of learning a model in form of a mathematical formula which describes the relation between a dependent variable y and several independent variables x and the according weights w given as $y = f(x, w) + \epsilon$. Contrary to other regression tasks the structure of the model as well as the variables used in the final model are not predetermined, but rather evolved during the optimization process. Hence, symbolic regression consists of three correlated subproblems:

- Detecting the best-suited model structure
- Selecting the appropriate subset of variables (feature selection)
- Determining optimal numerical constants and variable weights

Tree-based genetic programming (GP) [9] is commonly used to solve symbolic regression problems and to find the appropriate model structure. In addition, it incorporates an efficient feature selection step [20] to detect relevant variables. However, determining optimal variable weights and numerical constants in evolved formulas is problematic in GP [15]. This publication focuses on how numerical constants are obtained for a given model and how this can be further improved by incorporating a local optimization step.

In tree-based GP for symbolic regression every symbolic expression tree encodes a mathematical formula, where every inner node represents a function (e.g., addition, sine, ...) and every terminal node either a variable or a numerical constant. Numerical constants are randomly initialized during the tree creation, where concrete values are chosen from a predetermined distribution (e.g., uniform $[-20, 20]$). This implementation of constants in symbolic regression is defined as ephemeral random constants (ERC). A drawback of ERC is that constants are not modified during the whole GP run and therefore the only way of creating new constant values is by combining existing constants from the initial population via crossover.

A way to obtain new numerical values is the inclusion of mutation operators in GP which add random noise from a Gaussian distribution, either additive $\mathcal{N}(0, \sigma)$ or multiplicative $\mathcal{N}(1, \sigma)$, to constants [17]. As both ways of obtaining new constant values (mutation or combination of existing constants) are rather undirected, the progress of the GP algorithm can be unnecessarily hampered by misleading constant values; even if the correct model structure is found, its fitness value is small as long as the constant values are not set correctly.

In addition to mutation operators several other ways to modify and optimize numerical constants have been introduced within the last several years. In [7] a hybridization between GP and a genetic algorithm called GA-P is proposed, to separate the task of finding the corrected model structure from obtaining correct constant values. In GA-P an individual consists of the symbolic expression tree and an array of referenced constants, which is optimized by the genetic algorithm. Other hybrid approaches following a similar methodology combine GP with simulated annealing [18], evolution strategies [25, 2], or differential evolution [26, 12].

Apart from hybridizing GP with another metaheuristic, nonlinear optimization techniques using quadratic approximations [22] or gradient descent [21] have shown promising results. However, the local gradient search introduced by Topchy and Punch [21] was outperformed by symbolic regression with linear scaling [8]. Nevertheless, we pursue the idea that optimization techniques for obtaining better constant values in combination with linear scaling further improve the evolutionary search process of GP to solve symbolic regression problems.

The subsequent parts of the publication are organized as follows: the details of the nonlinear optimization technique for tuning numerical constants are described in Section 2. The experimental setup for demonstrating the benefits and limitations of the approach is described in Section 3 and the obtained results are discussed in Section 4. Finally, Section 5 concludes the paper and gives an overview of possible improvements and future research directions.

2. METHODS

Optimizing constants of symbolic expression trees is a nonlinear and multidimensional problem. It can also be regarded as a model fitting problem, where the model's constants are treated as parameters, which should be optimized to provide the best possible fit to given target values. As these models encode mathematical formulas they can be differentiated, provided that the functions occurring are differentiable, and it is possible to calculate their gradient. The gradient of a model supplies a concrete search direction, whose information could be used to accelerate the optimization process. Therefore, a gradient-based optimization technique, the Levenberg-Marquardt algorithm, was chosen to tune the constants as opposed to derivative-free algorithms or heuristics.

Calculation of Partial Derivatives of Symbolic Expression Trees

For the calculation of the partial derivations forming the gradient $\bigtriangledown f$ (Equation 1) of a symbolic expression tree, all constant values have to be extracted and replaced by an according parameter β_i.

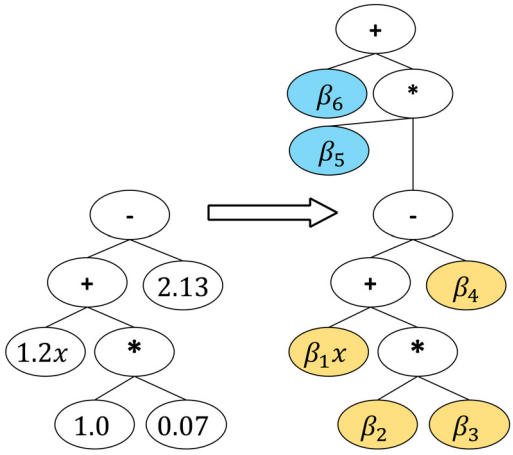

Figure 1: Transformation of a symbolic expression tree for constant optimization.

$$\bigtriangledown f = \left(\frac{\partial f}{\partial \beta_1}, \frac{\partial f}{\partial \beta_2}, ..., \frac{\partial f}{\partial \beta_n} \right) \quad (1)$$

Additionally, artificial nodes are inserted in the symbolic expression tree to account for the linear scaling terms. Figure 1 illustrates the process of transforming symbolic expression trees as a preparatory step before the constant optimization and gradient calculation takes place. The colored nodes indicate model parameters, for which the partial derivative must be calculated and the color distinguishes between actual tree nodes (yellow) and artificially introduced scaling nodes (blue).

For the calculation of all partial derivation according to the parameter vector β automatic differentiation [14] is used. The advantage of automatic differentiation is that it provides fast and accurate results compared to symbolic or numerical differentiation and is therefore especially suited for gradient-based optimization.

An overview of how automatic differentiation for symbolic expression trees works is given in [21]. In the present publication automatic differentiation was not reimplemented, but rather an implementation provided by AutoDiff[1] [19], a C# library that provides fast, accurate and automatic differentiation (computes derivative / gradient) of mathematical functions, is used.

Optimization of Constants

The Levenberg-Marquard (LM) algorithm [11] is especially suited for optimizing model parameters (in this case constant values of the model), because it minimizes the sum of squared errors $Q(\beta)$ between a model's predictions $f(x, \beta)$ and observed target values y with respect to the parameter vector β (Equation 2).

$$Q(\beta) = \sum_{i=0}^{m} (y_i - f(x_i, \beta))^2 \quad (2)$$

The LM algorithm is an iterative procedure which gradually improves the model quality starting from initial parameters using the calculated gradient. Internally, the gradient

[1] `http://autodiff.codeplex.com`

is used to calculate the Jacobian matrix consisting of all numerical values for the partial derivatives of β over all data points. The Jacobian matrix is afterwards used to update the parameter vector and the whole iteration starts anew until a specified stopping criterion is reached (number of iterations, achieved precision, or minimal changes in the gradient). The implementation of the LM algorithm used for constant optimization is provided by ALGLIB[2] [3].

The start values for the LM algorithm are the previously extracted constant values in the specific tree and 0.0 respectively 1.0 for the additive and multiplicative scaling terms and the algorithm is stopped after a certain number of iterations. After the LM algorithm has finished the constants in the specific symbolic expression tree are updated with the tuned values. Another possibility would be to calculate the quality of individual with the optimized constants, but the tree does not get updated. A disadvantage of this method could be that without updating the constants, the optimization starts at the same starting values multiple times and performs the same optimization steps.

Nevertheless, both methods are supported in the presented approach by introducing a boolean flag, which determines whether the optimized constants are written back to the original tree. The pseudo code describing the necessary steps for optimization the constants of a symbolic expression tree is stated in Algorithm 1.

Algorithm 1 Constant optimization of symbolic expression trees.

```
Add scaling tree nodes
Extract initial constant values
Transform the tree for constant optimization
Start the LM algorithm
while Stopping criterion not reached do
  Calculate the gradient with automatic differentiation
  Perform LM iteration
end while
Calculate quality with optimized constants
if UpdateConstants then
  Write back optimized constants
end if
```

3. EXPERIMENTAL SETUP

We performed several experiments on selected benchmark problems to test the performance of the constant optimization approach. The problems on which the experiments were conducted were recently published as recommended symbolic regression benchmarks [24] and additionally the Poly-10 [13] and Friedman-2 [4] problem were included. A detailed listing of the problems and the according numbers of training and test samples is given in Table 1.

Genetic Programming

A modification to the standard GP algorithm is used by including strict offspring selection [1], which means that child individuals are only accepted in the next generation if their quality surpasses the quality of the better parent. Therefore, the number of evaluated individuals per generation is not constant anymore, but depends on the state of the evolutionary run (how easy it is to generate better children).

A characteristic of offspring selection GP (OSGP) is that the algorithm stops automatically if it gets too hard to fill the next generation with better individuals. This is stated by the selection pressure, which is calculated as the number of created child individuals (accepted and discarded) divided by the population size and which is used as stopping criterion for the algorithm.

The fitness function for the GP algorithm is the coefficient of determination R^2 (Equation 3).

$$R^2(x,y) = \frac{\mathrm{Cov}(x,y)^2}{\mathrm{Var}(x) * \mathrm{Var}(y)} \tag{3}$$

As a consequence individuals can be linearly transformed without worsening their fitness, but have to be scaled before other quality values (e.g., mean square error, average relative error, ...) are calculated. The result of a GP run is the best model found on the training partition during the algorithm execution.

Further details about the algorithm and parameter settings are listed in Table 2. Regarding the parameter settings, the Nguyen-7 problem has a slightly modified function set, as it includes the unary operations log and exp, which are required to solve the problem exactly.

Table 2: Genetic programming algorithm settings.

Parameter	Value
Tree initialization	PTC2 [10]
Maximum tree length	50 nodes
Maximum tree depth	12
Population size	500 individuals
Elites	1 individual
Selection	Gender specific selection [1] (random & proportional)
Offspring selection	Strict offspring selection
Crossover probability	100%
Crossover operator	Subtree crossover
Mutation probability	25%
Mutation operator	Change symbol Single point mutation Remove branch Replace branch
Fitness function	Maximize R^2
Termination criterion	Max. selection pressure of 100
Terminal symbols	*constant*, $weight * variable$
Function symbols	binary functions$(+, -, \times, /)$

Optimization Framework

All the benchmark problems, algorithms and the constant optimization approach presented in this publication are implemented in HeuristicLab[3] [23], an open source framework for heuristic optimization.

In addition to the experimental results in this publication, all the raw data of the conducted experiments is available online at `http://dev.heuristiclab.com/AdditionalMaterial` and can be inspected with a recent version of HeuristicLab.

[2] `http://www.alglib.net`

[3] `http://dev.heuristiclab.com`

Table 1: Definition of benchmark problems and training and test ranges.

Name	Function	Training	Test
Nguyen-7	$f(x) = log(x+1) + log(x^2+1)$	20 samples	500 samples
Keijzer-6	$f(x,y,z) = \frac{30xz}{(x-10)y^2}$	20 samples	120 samples
Vladislavleva-14	$f(x_1,...,x_5) = \frac{10}{5+\sum_{i=1}^{5}(x_i-3)^2}$	1024 samples	5000 samples
Pagie-1	$f(x,y) = \frac{1}{1+x^{-4}} + \frac{1}{1+y^{-4}}$	676 samples	1000 samples
Poly-10	$f(x_1,...,x_{10}) = x_1x_2 + x_3x_4 + x_5x_6 + x_1x_7x_9 + x_3x_6x_{10}$	250 samples	250 samples
Friedman-2	$f(x_1,...,x_{10}) = 10\sin(\pi x_1 x_2) + 20(x_3 - 1/2)^2 + 10x_4 + 5x_5 + Noise$	500 samples	5000 samples
Tower	Real world data	3136 samples	1863 samples

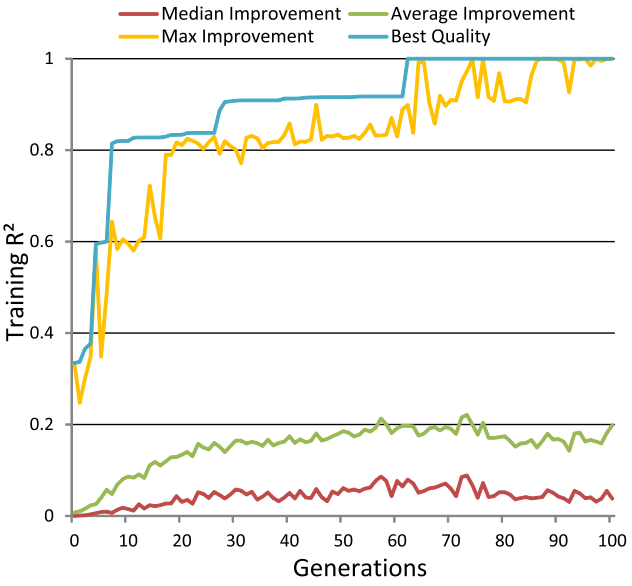

Figure 2: Constant optimization improvement and best quality of an exemplary GP run solving the Poly-10 problem.

4. RESULTS

The quality improvement for individuals obtained by constant optimization is outlined by an exemplary algorithm run solving the Poly-10 problem. The improvement is measured as the quality difference (coefficient of determination, R^2) after and before constant optimization is performed.

Figure 2 displays the best quality (after constant optimization) achieved so far and the maximum, average, and median constant optimization improvement. It can be seen, that the average and median improvement are rather low, but increase over time, whereas the maximum improvement in each generation reaches almost the current best quality. An explanation for this might be, that child individuals created by crossover and mutation from high quality parents achieve a rather low quality, which can be drastically increased by tuning their numerical constants.

Quality

All the results presented in this section are an aggregation of 50 repetitions of each algorithm configuration to take stochastic effects into account. The achieved training and test qualities, as well as the success rate for noise-free problems, are shown in Table 3, where all algorithm configurations are tested on every problem. For an easier interpretation the success rate is defined as the relative number of runs that achieved a test $R^2 \geq 0.99$, which means that at least 99% of the variation of the target function must be explained by the obtained model.

The first three columns in Table 3 represent OSGP runs with varying population sizes (500, 1000, and 5000) without constant optimization. In the last three columns results for different constant optimization probabilities (25%, 50% and 100% for 10 iterations) with a population size of 500 are shown. As the maximum selection pressure is used as termination criterion, the number of evaluated individuals need not be the same across the different algorithm configurations and it could be suspected the constant optimization achieves better results by prolonging the runtime of the algorithm. However, the results have been controlled and no statistically significant difference regarding the number of evaluated solutions has been detected (data not shown).

Generally, it can be noted that OSGP runs with a population size of 5000 outperform runs using a smaller population size on all problems except the Keijzer-6 problem. Additionally, all algorithm configurations with constant optimization (regardless of the probability) achieved better results as the standard OSGP configurations (except the Nguyen-7 problem). However, constant optimization provides little additional value for runs on the Nguyen-7 and Keijzer-6 problem, because they are already solved with a high probability by standard OSGP runs.

An interesting fact is that constant optimization runs with the same population size as standard OSGP runs are able to solve hard symbolic regression problems more regularly (e.g., the success rate on the Pagie-1 increases from 0.08 to above 0.70). Hence, it can be deduced that the search behavior of GP with and without constant optimization differs. A likely explanation is that GP is able to evolve the necessary model structure, but fails to find the right numerical values without constant optimization.

It can also be concluded that the higher the constant optimization probability, the better the achieved qualities, but the increase in quality and success rate between 50% and 100% constant optimization probability is rather small. Therefore, a probability of 50% seems to be sufficient to achieve high quality results on the tested benchmark problems. Furthermore, it is especially outlined in the GP benchmark publication [24] that the Vladislavleva-14 and Pagie-1 are regarded difficult for symbolic regression systems, but with constant optimization (50% or 100%) they are solved by approximately 80% of the GP runs.

Table 3: Quality of the best training solution of the GP algorithm with different population sizes (OSGP) and constant optimization configurations (CoOp). For every problem the first line shows the training quality $R^2(\mu \pm \sigma)$, the second line the test quality and the third line the success rate for problems without noise.

Problem	OSGP 500	OSGP 1000	OSGP 5000	CoOp 25%	CoOp 50%	CoOp 100%
Nguyen-7	1.000 ± 0.000 1.000 ± 0.000 1.00	1.000 ± 0.000 1.000 ± 0.000 1.00	1.000 ± 0.000 1.000 ± 0.000 1.00	1.000 ± 0.000 0.961 ± 0.191 0.96	1.000 ± 0.000 0.980 ± 0.111 0.94	1.000 ± 0.000 1.000 ± 0.000 1.00
Keijzer-6	1.000 ± 0.000 0.961 ± 0.150 0.82	1.000 ± 0.000 0.898 ± 0.276 0.82	1.000 ± 0.000 0.876 ± 0.274 0.74	1.000 ± 0.000 0.977 ± 0.072 0.90	1.000 ± 0.000 0.981 ± 0.104 0.94	1.000 ± 0.000 0.979 ± 0.122 0.96
Vladislavleva-14	0.861 ± 0.098 0.723 ± 0.280 0.12	0.903 ± 0.073 0.775 ± 0.259 0.18	0.979 ± 0.037 0.905 ± 0.229 0.48	0.985 ± 0.024 0.965 ± 0.087 0.54	0.996 ± 0.012 0.993 ± 0.020 0.86	0.995 ± 0.017 0.992 ± 0.027 0.84
Pagie-1	0.955 ± 0.027 0.755 ± 0.309 0.08	0.965 ± 0.022 0.811 ± 0.270 0.06	0.984 ± 0.011 0.893 ± 0.222 0.20	0.995 ± 0.006 0.597 ± 0.463 0.36	0.999 ± 0.003 0.857 ± 0.316 0.74	0.999 ± 0.002 0.875 ± 0.292 0.78
Poly-10	0.823 ± 0.116 0.748 ± 0.200 0.02	0.879 ± 0.085 0.833 ± 0.108 0.04	0.971 ± 0.043 0.965 ± 0.054 0.62	0.981 ± 0.035 0.968 ± 0.063 0.72	0.996 ± 0.015 0.992 ± 0.031 0.94	0.996 ± 0.015 0.991 ± 0.035 0.94
Friedman-2	0.836 ± 0.027 0.768 ± 0.172	0.857 ± 0.036 0.831 ± 0.102	0.908 ± 0.035 0.836 ± 0.191	0.959 ± 0.001 0.871 ± 0.151	0.967 ± 0.000 0.920 ± 0.086	0.964 ± 0.000 0.864 ± 0.142
Tower	0.877 ± 0.007 0.876 ± 0.012	0.880 ± 0.006 0.877 ± 0.024	0.892 ± 0.006 0.890 ± 0.008	0.919 ± 0.006 0.916 ± 0.007	0.925 ± 0.005 0.921 ± 0.006	0.932 ± 0.005 0.927 ± 0.005

Table 4: Success rate and median execution time (hh:mm:ss) of the GP algorithm with different population sizes (OSGP) and variable number of iterations for constant optimization (CoOp). For the last two problems the test $R^2(\mu \pm \sigma)$ is stated as the success rate cannot be calculated.

Problem	OSGP 500	OSGP 1000	OSGP 5000	CoOp 50% 3x	CoOp 50% 5x	CoOp 50% 10x
Nguyen-7	1.00 00:03:22	1.00 00:06:45	1.00 01:09:21	0.92 00:11:32	0.92 00:14:43	0.94 00:13:53
Keijzer-6	0.82 00:03:08	0.82 00:10:48	0.74 00:57:21	0.92 00:08:10	0.88 00:12:25	0.94 00:23:37
Vladislavleva-14	0.12 00:27:24	0.18 01:13:01	0.48 09:08:52	0.56 03:44:48	0.82 05:18:25	0.94 05:58:53
Pagie-1	0.08 00:11:46	0.06 00:32:44	0.20 02:31:18	0.26 02:52:48	0.52 02:52:00	0.74 03:53:04
Poly-10	0.02 00:05:50	0.04 00:17:58	0.62 02:38:27	0.78 00:53:17	0.88 00:47:00	0.94 02:34:44
Friedman-2	0.768 ± 0.172 00:09:31	0.831 ± 0.102 00:21:39	0.836 ± 0.191 02:52:05	0.946 ± 0.046 01:38:17	0.943 ± 0.076 01:53:14	0.920 ± 0.086 03:24:23
Tower	0.876 ± 0.012 00:59:01	0.877 ± 0.024 02:38:53	0.890 ± 0.008 14:51:47	0.902 ± 0.010 13:27:36	0.912 ± 0.008 14:06:21	0.921 ± 0.006 33:10:23

Table 5: Results

Problem	OSGP Pop 500	CoOp 50% 5x
Feature-10	0.925 ± 0.009	0.960 ± 0.003
	0.858 ± 0.017	0.669 ± 0.248
Feature-25	0.879 ± 0.030	0.966 ± 0.008
	0.709 ± 0.227	0.708 ± 0.147

Execution Time

So far the overhead in terms of execution time was neglected in favor of the achieved quality on benchmark problems. In Table 4 the median execution time and the success rate (test R^2 for Friedman-2 and Tower) are displayed to make the results comparable regarding the speed of execution. However, it must be noted that the algorithm parameters are not tuned in terms of execution speed; i.e. it could be the case that a maximum selection pressure of 50 would be sufficient to achieve comparable results as stated here. The important point of Table 4 is that the relative differences between the algorithm configurations can be seen, as well as the effect of the number of iterations for constant optimization.

The OSGP with a population size of 500 is clearly the fastest algorithm, but fails to solve the more difficult problems. The algorithm configurations with 50% constant optimization applied take about the same execution time to finish as the OSGP runs with a population size of 5000, but achieve a higher quality. Therefore, the best tradeoff between quality and execution time, turns out to be the configuration that applies constant optimization for 5 iterations (second last column of Table 4).

Feature Selection Problems

As previously mentioned the GP algorithm with constant optimization enabled learns better from presented training data and as a consequence should be more likely to overfit; although it has rarely been observed on the benchmark problems (except for very few cases of models learned on the Nguyen-1 and the Keijzer-6 problem). This hypothesis is tested on two artificially generated problems, which contain 100 variables (normally distributed $\mathcal{N}(0,1)$) of which either 10 respectively 25 (Feature-10, Feature-25) are linearly combined (the weights of the variables are uniformly distributed $[0, 10]$) and a noise term contributing to 10% of the variance is added. For these two problems the training partition consists of 120 and the test partition of 500 samples and the highest possible R^2 on the test partition is 0.9 due to the noise term. As the size of the training partition (120) is only slightly larger than the available features (100), it is rather hard to find the right combination of features without producing overfit models.

Results regarding 50 repetitions of two algorithm configurations, OSGP and constant optimization with a probability of 50% for 5 iterations (CoOp 50% 5x), with a population size of 500 are stated in Table 5. Whereas both algorithm configurations fail to produce well generalizing models on the Feature-25 problem, the configuration with constant optimization enabled obviously produces overfit models for the Feature-10 problem. A reason for this might be the small amount of available training samples and measures to counter overfitting should be included if it is detected (e.g., using a validation partition).

5. CONCLUSION AND FUTURE WORK

In this publication a nonlinear least squares optimization approach for tuning numerical constants of symbolic expression trees evolved for solving symbolic regression problems is presented. The developed approach uses the Levenberg-Marquardt algorithm to improve the constants and the necessary gradient is calculated by an automatic differentiation procedure. To test the effectiveness several well-known problems for symbolic regression were used as benchmarks and the obtained results compared to standard algorithm configurations. Furthermore, the influence of different parameters, namely the probability to apply constant optimization and the number of iterations, is detailed on these benchmark problems. Overall constant optimization improves the achieved quality of symbolic regression models as well as the success rate on hard problems considerably. However, it affects the runtime of the GP algorithm and therefore should be mainly used, if the problem cannot be easily solved with standard methods.

Future work regarding constant optimization involves further analysis on how the algorithm dynamics are changed. The assumption is that it reduces the disruptiveness of the crossover operation, but this has still to be verified. Another issue is the occurrence of overfitting on a few problems and how this could be counteracted.

In addition, techniques to reduce the overhead of constant optimization should be investigated to achieve better results in shorter time, for example by parallelization. Another option would be to reduce the number of training samples by sampling strategies, like random sampling [5] or coevolution of fitness predictors [16, 6]. A third way of accelerating constant optimization would be to intelligently distribute the effort across the individuals by stopping the constant optimization when little or no further improvement can be achieved.

Summarizing, empirical results show that constant optimization improves the quality of symbolic regression models significantly and the whole approach is implemented and ready-to-use in HeuristicLab.

6. ACKNOWLEDGEMENTS

The work described in this publication was mostly done in the Josef-Ressel-Center *Heureka!* for heuristic optimization, which is supported by the Austrian Research Promotion Agency (FFG).

7. REFERENCES

[1] M. Affenzeller, S. Winkler, S. Wagner, and A. Beham. *Genetic Algorithms and Genetic Programming - Modern Concepts and Practical Applications*, volume 6 of *Numerical Insights*. CRC Press, Chapman & Hall, 2009.

[2] C. L. Alonso, J. L. Montana, and C. E. Borges. Evolution strategies for constants optimization in genetic programming. In *21st International Conference on Tools with Artificial Intelligence, ICTAI '09*, pages 703–707, Nov. 2009.

[3] S. Bochkanov and V. Bystritsky. Alglib. `http://www.alglib.net/`.

[4] J. H. Friedman. Multivariate adaptive regression splines. *The annals of statistics*, pages 1–67, 1991.

[5] I. Goncalves, S. Silva, J. B. Melo, and J. M. B. Carreiras. Random sampling technique for overfitting control in genetic programming. In A. Moraglio, S. Silva, K. Krawiec, P. Machado, and C. Cotta, editors, *Proceedings of the 15th European Conference on Genetic Programming, EuroGP 2012*, volume 7244 of *LNCS*, pages 218–229, Malaga, Spain, 11-13 Apr. 2012. Springer Verlag.

[6] R. Harper. Spatial co-evolution in age layered planes (SCALP). In *IEEE Congress on Evolutionary Computation (CEC 2010)*, Barcelona, Spain, 18-23 July 2010. IEEE Press.

[7] L. M. Howard and D. J. D'Angelo. The ga-p: A genetic algorithm and genetic programming hybrid. *IEEE Expert*, 10(3):11–15, 1995.

[8] M. I. Keijzer. Improving symbolic regression with interval arithmetic and linear scaling. In *EuroGP'03: Proceedings of the 6th European conference on Genetic programming*, pages 70–82, Essex, UK, 2003. Springer-Verlag.

[9] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[10] S. Luke. Two fast tree-creation algorithms for genetic programming. *IEEE Transactions on Evolutionary Computation*, 4(3):274–283, Sept. 2000.

[11] D. W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial & Applied Mathematics*, 11(2):431–441, 1963.

[12] S. Mukherjee and M. J. Eppstein. Differential evolution of constants in genetic programming improves efficacy and bloat. In K. Rodriguez and C. Blum, editors, *GECCO 2012 Late breaking abstracts workshop*, pages 625–626, Philadelphia, Pennsylvania, USA, 7-11 July 2012. ACM.

[13] R. Poli. A simple but theoretically-motivated method to control bloat in genetic programming. In *Proceedings of the 6th European conference on Genetic programming*, EuroGP'03, pages 204–217, Berlin, Heidelberg, 2003. Springer-Verlag.

[14] L. B. Rall. *Automatic Differentiation: Techniques and Applications*, volume 120 of *Lecture Notes in Computer Science*. Springer, Berlin, 1981.

[15] C. Ryan and M. Keijzer. An analysis of diversity of constants of genetic programming. In C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, and E. Costa, editors, *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 404–413, Essex, 14-16 Apr. 2003. Springer-Verlag.

[16] M. D. Schmidt and H. Lipson. Co-evolving fitness predictors for accelerating and reducing evaluations. In R. L. Riolo, T. Soule, and B. Worzel, editors, *Genetic Programming Theory and Practice IV*, volume 5 of *Genetic and Evolutionary Computation*, pages 113–130. Springer, Ann Arbor, 11-13 May 2006.

[17] M. Schoenauer, M. Sebag, F. Jouve, B. Lamy, and H. Maitournam. Evolutionary identification of macro-mechanical models. In P. J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 23, pages 467–488. MIT Press, Cambridge, MA, USA, 1996.

[18] K. C. Sharman, A. I. Esparcia Alcazar, and Y. Li. Evolving signal processing algorithms by genetic programming. In A. M. S. Zalzala, editor, *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA*, volume 414, pages 473–480, Sheffield, UK, 12-14 Sept. 1995. IEE.

[19] A. Shtof. Autodiff. `http://autodiff.codeplex.com/`.

[20] S. Stijven, W. Minnebo, and K. Vladislavleva. Separating the wheat from the chaff: on feature selection and feature importance in regression random forests and symbolic regression. In S. Gustafson and E. Vladislavleva, editors, *3rd symbolic regression and modeling workshop for GECCO 2011*, pages 623–630, Dublin, Ireland, 12-16 July 2011. ACM.

[21] A. Topchy and W. F. Punch. Faster genetic programming based on local gradient search of numeric leaf values. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 155–162, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.

[22] V. V. Toropov and L. F. Alvarez. Application of genetic programming to the choice of a structure of multipoint approximations. In *1st ISSMO/NASA Internet Conf. on Approximations and Fast Reanalysis in Engineering Optimization*, June 14-27 1998. Published on a CD ROM.

[23] S. Wagner. *Heuristic Optimization Software Systems - Modeling of Heuristic Optimization Algorithms in the HeuristicLab Software Environment*. PhD thesis, Institute for Formal Models and Verification, Johannes Kepler University, Linz, Austria, 2009.

[24] D. R. White, J. McDermott, M. Castelli, L. Manzoni, B. W. Goldman, G. Kronberger, W. Jaskowski, U.-M. O'Reilly, and S. Luke. Better GP benchmarks: community survey results and proposals. *Genetic Programming and Evolvable Machines*, 14(1):3–29, Mar. 2013.

[25] S. M. Winkler. *Evolutionary System Identification - Modern Concepts and Practical Applications*. PhD thesis, Institute for Formal Models and Verification, Johannes Kepler University, Linz, Austria, Apr. 2008.

[26] Q. Zhang, C. Zhou, W. Xiao, and P. C. Nelson. Improving gene expression programming performance by using differential evolution. In *Sixth International Conference on Machine Learning and Applications, ICMLA 2007*, pages 31–37, Cincinnati, Ohio, USA, 13-15 Dec. 2007. IEEE.