Evaluating methods for constant optimization of symbolic regression benchmark problems

Vinícius Veloso de Melo
Institute of Science and Technology
Federal University of São Paulo
São José dos Campos, São Paulo, Brazil
Email: vinicius.melo@unifesp.br

Benjamin Fowler
Department of Computer Science
Memorial University of Newfoundland
St. John's, NL, A1B 3X5, Canada
Email: b.fowler@mun.ca

Wolfgang Banzhaf
Department of Computer Science
Memorial University of Newfoundland
St. John's, NL, A1B 3X5, Canada
Email: banzhaf@mun.ca

***Abstract*—Constant optimization in symbolic regression is an important task addressed by several researchers. It has been demonstrated that continuous optimization techniques are adequate to find good values for the constants by minimizing the prediction error. In this paper, we evaluate several continuous optimization methods that can be used to perform constant optimization in symbolic regression. We have selected 14 well-known benchmark problems and tested the performance of diverse optimization methods in finding the expected constant values, assuming that the correct formula has been found. The results show that Levenberg-Marquardt presented the highest success rate among the evaluated methods, followed by Powell's and Nelder-Mead's Simplex. However, two benchmark problems were not solved, and for two other problems the Levenberg-Marquardt was largely outperformed by Nelder-Mead Simplex in terms of success rate. We conclude that even though a symbolic regression technique may find the correct formula, constant optimization may fail; thus, this may also happen during the search for a formula and may guide the method towards the wrong solution. Also, the efficiency of LM in finding high-quality solutions by using only a few function evaluations could serve as inspiration for the development of better symbolic regression methods.**

***Keywords*—Symbolic Regression, Genetic programming, Curve-fitting, Least-squares, Nonlinear regression**

I. Introduction

Symbolic regression has been one of the main subjects of research in evolutionary computation for many years, gaining more prominence recently. Techniques such as Kaizen Programming [1], Genetic Programming [2], Linear Genetic Programming [3], [4], [5], and Grammatical Evolution [6] have been studied to solve such task.

For symbolic regression problems, the goal is to find a symbolic description of a model, for instance, the exact equation used as benchmark problem. The general approach is to evolve both structure and the parameters (constant values) at the same time, in a coupled way. In traditional Evolutionary Computation (EC) techniques, the parameters of the model are naïvely adjusted by random modifications (mutation on the numerical values), ignoring any information that may be extracted from the model. As a result, these techniques usually require numerous objective function evaluations. One problem is that the correct structure with the wrong parameters may be of worse quality than another model presenting a very distinct structure. Once the method followed the wrong model, it may be stuck in a local optimum region.

After identifying this issue with the parameters, researchers decided to create hybrids by decoupling the symbolic regression problem and doing what was called Constant Optimization by local search, i.e., replacing the numerical values by variables, and running optimization algorithms to perform least-squares minimization on each trial model. That constant optimization is, in fact, the well-known regression technique studied in statistics [7].

It has already been shown that solving the constant optimization problem minimizes the prediction error of the models found by symbolic regression methods, resulting in better solutions. Attempts were made using hill-climbing, simulated annealing, local gradient search, genetic algorithms, differential evolution, and particle swarm optimization, among others [8], [9], [10], [11], [12]. More recently, Levenberg-Marquardt has been applied with success because it is usually faster than many other methods for least-squares minimization [13], [14]. By success, one means that the model found by the evolutionary method presents an acceptable error on the test set, which does not necessarily mean that the original expression was found. The acceptable error threshold is not a standard, but mean squared error below $1e$-08 and as $R^2 >= 0.99$ have been employed.

Given that there are many studies in the literature, we understand that there is no need to run experiments to confirm the results found by other researchers. On the other hand, we could not find reports in the literature that tested several constant optimization techniques on the symbolic regression benchmarks proposed by the community. Therefore, in this paper we compare the performance of some methods for constant optimization that may be used in symbolic regression.

We have selected well-known benchmark problems, and tested the performance of diverse optimization methods in finding constants that give low prediction error, assuming that the correct formula has been found. Therefore, the experiments were executed as a nonlinear regression task to optimize the parameters of the model. Thus, with the experiments in this paper the objective is to answer only the following question: *how good are the constant optimization methods supposing the correct expression has been found by the symbolic regression technique?* That question is important because if the constant optimization is not reliable when the model's structure is correct, then what can be expected when the model structure is wrong?

978-1-5090-0016-6/15 $31.00 © 2015 IEEE
DOI 10.1109/BRACIS.2015.55

The paper is structured as follows: In Section 2 we describe the constant optimization task in more detail, along with the optimization methods selected for this work. Section 3 has the experimental analysis. Finally, summary and conclusions are given in Section 4.

II. Constant Optimization

The usual way of solving a symbolic regression problem via evolutionary computation is to use ERCs (Ephemeral Random Constant), which are numbers inserted in the expressions that cannot be directly manipulated. Values are replaced either by other ERCs via mutation (new random number) or modified by functions; for instance, an ERC of value 10 is replaced by a randomly chosen function such as $log(10)$. Another possibility is ERC mutation by perturbing it with Gaussian noise [15] instead of generating an independent new value. Nevertheless, this perturbation is also random.

As introduced, local search mechanisms tend to improve symbolic regression methods by optimizing the values of the constants at some point of the process, i.e., the best individual of the population is optimized every ten generations. Assuming a tree-like structure to represent a trial solution (an individual in GP, for instance), one may have the original individual (with constants, see Figure 1a) transformed into an expression to be optimized (see Figure 1b), where the parameters are C_1 and C_2, and x is an input data.

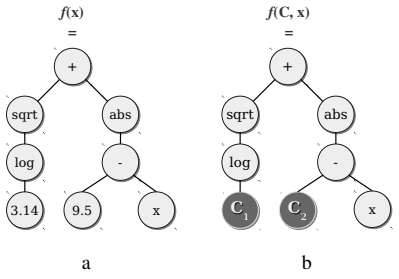

Figure 1. Transformation of an expression tree in a nonlinear function for constant optimization.

The transformed tree can then be treated as a multiple nonlinear optimization problem and solved by least-squares minimization:

$$min \, \|f(C,x) – y\|^2 = min \sum_{i=1}^{n} (f(C,x_i) - y_i)^2,$$

where i is the index for the n regression instances, f is the nonlinear function (GP solution), C is the set of parameters that are being optimized, for instance, C_1 and C_2, and y_i are the observed output data points. Consequently, the objective is to find the values for the constants that minimize the error of all predictions. One expects to reduce the risk of discarding good models due to wrong parameters by performing nonlinear regression on the trial solutions in a symbolic regression task.

Below we briefly describe the unconstrained minimization methods evaluated in this paper. The Nelder-Mead method, Powell's method, and Simulated Annealing do not rely on gradient evaluations. As such, they are useful to minimize functions whose derivatives cannot be calculated, although other methods that may exploit gradient evaluations will likely be faster. The other optimization methods used in this work require first-order gradients, which were numerically calculated through finite difference.

A. Simulated Annealing (SA)

Simulated annealing is an optimization method that involves evaluating function values on random points and keeping those that pass steadily increasing evaluation criteria. It is probable to move to points that perform worse in the earlier iterations, in order to search a wider space, but the probability of accepting worse performing points decreases with the number of iterations. This is analogous to the process of metallurgical annealing, where a metal is heated and slowly cooled to reduce defects that occur by cooling too quickly [16].

B. Broyden–Fletcher–Goldfarb–Shanno (BFGS)

The Broyden-Fletcher-Goldfarb-Shanno method is an optimization method that approximates second-order derivatives using gradient evaluations, which is used to generate a search direction. This generates a new point, closer to the optimum [17].

C. Conjugated Gradient (CG)

The conjugate gradient method is an optimization method that forgoes using local gradients, instead relying on conjugate directions. It works faster than other methods when the shape of the search space is a narrow valley [18].

D. Levenberg-Marquadt (LM)

The Levenberg-Marquardt method is an optimization method that relies on minimizing the sum of squares of nonlinear functions. The LM method searches in a direction determined by the partial derivatives of those functions. It adaptively varies how its search parameter updates between gradient descent and the Gauss-Newton method [19].

E. Nelder-Mead Simplex (NM)

The Nelder-Mead method (also known as the downhill simplex method) is an optimization method that compares function values at the vertices of a simplex, which gradually contracts the simplex as better values are found, continuing until some bound is obtained, or a particular number of iterations have been completed [20]. A simplex is a generalized tetrahedron, which in the Nelder-Mead method, has $n+1$ vertices, where n is the number of variables. The function value of each vertex are evaluated and compared, which indicate what new points to evaluate, and how to modify the simplex.

F. Powell (P)

Powell's method is an optimization method that uses n search vectors to identify which new points to evaluate. The search vectors then optimize the result for each variable. The best performing search vector is replaced with a new one indicating the difference between the new best point and the previous best point [21].

III. Experimental analysis

This section presents the experimental analysis performed to evaluate the methods and the discussion of results.

A. Benchmark functions

The symbolic regression problems tested here (see Table I) were taken from [22], where it is suggested that the input variables $x_i, i = 1, ..., 5$ should be renamed to x, y, z, v, w, respectively. Some benchmark problems intentionally omit variables from the function. For the training set, every variable is sampled from $U(-50, 50, n)$, where n is the number of uniform random samples drawn from -50 to 50, inclusive. There is no test set, only a training set with the same type of distribution for all problems because we are using the *ground truth* as the problem to be optimized. In order to do that, for each benchmark function we replace the numerical constants with symbols to be optimized. For instance, $f_1(\overrightarrow{C}, x, y, z, v, w) = 1.57 + 24.3v$ becomes $f_1(\overrightarrow{C}, x, y, z, v, w) = C_1 + C_2 v$. On the other hand, exponents were not changed because power functions are assumed to be multiplications: $f_{11}(\overrightarrow{C}, x, y, z, v, w) = 6.87 + 11cos(7.23x^3)$ becomes $f_{11}(\overrightarrow{C}, x, y, z, v, w) = C_1 + C_2 cos(C_3 x^3)$. Function *korns9* is missing in the benchmark set because it does not have numerical constants.

Table I. Korns benchmark functions and number of constants.

Function	D
$f_1 = 1.57 + 24.3v$	2
$f_2 = 0.23 + 14.2(v + y)/3w$	3
$f_3 = -5.41 + 4.9(v - x + y/w)/3w$	3
$f_4 = -2.3 + 0.13sin(z)$	2
$f_5 = 3 + 2.13ln(w)$	2
$f_6 = 1.3 + 0.13sqrt(x)$	2
$f_7 = 213.809408(1 - e^{-0.547237x})$	3
$f_8 = 6.87 + 11sqrt(7.23xvw)$	3
$f_{10} = 0.81 + 24.3\frac{2y+3z^2}{4v^3+5w^4}$	6
$f_{11} = 6.87 + 11cos(7.23x^3)$	3
$f_{12} = 2 - 2.1cos(9.8x)sin(1.3w)$	4
$f_{13} = 32 - 3\frac{tan(x)}{tan(y)}\frac{tan(z)}{tan(v)}$	2
$f_{14} = 22 - 4.2(cos(x) - tan(y))\frac{tan(z)}{sin(v)}$	2
$f_{15} = 12 - 6\frac{tan(x)}{e^y}(ln(z)\text{-}tan(v))$	2

B. Implementation and configuration of the methods

Our code was implemented in Python 2.7.6, using the optimization methods from SciPy 0.14.0 on a system using Arch Linux kernel 3.18.6-1, an Intel i7-920 CPU, and 6Gb DDR3 RAM. All methods were executed using the default configuration, except for the parameters in Table II. We are aware that the configuration is an important part of the process, but we are considering the recommended settings.

Tolerances were set as the value-to-reach, i.e., the desired accuracy in terms of the scoring function. The Mean Absolute Error is given by:

$$MAE = \frac{\sum_{i=1}^{n} |y_i - f(C, x_i)|}{n}, \quad (1)$$

where n is the number of samples$= D*100$, y_i is the expected result for the i^{th} sample, and $f(C, x_i)$ is the calculated result for the i^{th} sample using the parameters in C.

Table II. Configuration of the optimization methods.

Parameter	Value
scoring function	Mean Absolute Error
maxev	$D * 1000$
maxit	1000
VTR	$1e{-}08$
xtol	$1e{-}16$
ftol	$1e{-}16$
maxRuns	100

Algorithm 1 Algorithm to perform the experiment.

```
For each method
  For each problem
    Set D = amount of CONST in the problem
    Set MaxEvaluations and tolerances
    For run = 1 to maxRuns
      Generate the training Dataset
      Optimize using a random start point sampled from U(-1e5, 1e5, D)
      Save results
```

Because the methods may stop for another criterion, we keep track of all function evaluations to store MAE and the current number of function evaluations.

The algorithm employed to perform our experiments is shown in Algorithm 1. As explained before, there is no need to have a test dataset. One may notice that for each run a new dataset is created, and a new starting point is used. This is done to take stochastic effects into account.

C. Performance comparison

The comparison of the methods is done using the following measures and visual presentation.

1) Success rate: One considers a run successful when the result of the optimization is $MAE < 1e{-}08$. The success rate is simply

$$SR = \frac{number\ of\ successful\ runs}{maxRuns}. \quad (2)$$

where $maxRuns$ is the total number of runs. Average and Median were also calculated for each method considering all benchmark functions.

2) Success performance: described in [23], the success performance is the estimated mean number of function evaluations required to achieve a successful run. This measure allows a comparison among methods that perform a distinct number of function evaluations. The equation is:

$$SP = \frac{mean\ number\ of\ evaluations\ \times\ maxRuns}{number\ of\ successful\ runs}, \quad (3)$$

where the $mean\ number\ of\ evaluations$ is calculated only considering the successful runs.

3) Accumulated count: The final results of each method, for each benchmark function, are grouped into several bins of error precision level (MAE). The frequencies are accumulated for subsequent precision levels and then plotted (CUSUM, cumulative sum chart). In the case evaluated in this work, the chart presents only positive variation because frequencies

can only be greater than or equal to zero. The primary interpretation of this chart is related to the slope; the steeper, the greater the variation.

4) Error distribution : Error distributions are presented in log-scale violin plots, along with dot plots. Those plots, which are rotated kernel density plot on each side, are preferred over regular boxplots whenever the data present multi-modality or very irregular shapes.

D. Results and Discussion

The results and analysis of the experiment are presented next. In the success rates shown in Table III one can see that the best average was achieved by the Nelder-Mead Simplex method (NM), followed by the Levenberg-Marquardt method (LM) and the Powell's method (P). Simulated Annealing did not reach the desired precision in any run for any problem. Considering the median value, NM improved 24% over its average, P improved 31%, and LM showed 100%. On the other hand, Both BFGS and CG had median values lower than their average values, meaning that more than 50% of the benchmark problems were not solved.

Table III. SUCCESS RATES OVER 100 RUNS.

Fun	SA	BFGS	CG	LM	NM	P
f_1	0.00	0.04	0.00	**1.00**	**1.00**	0.90
f_2	0.00	0.28	0.00	**1.00**	0.60	**1.00**
f_3	0.00	0.36	0.00	**1.00**	0.66	**1.00**
f_4	0.00	0.96	0.39	**1.00**	**1.00**	**1.00**
f_5	0.00	0.16	0.00	**1.00**	**1.00**	0.89
f_6	0.00	0.10	0.00	**1.00**	**1.00**	0.84
f_7	0.00	0.00	0.00	0.09	**0.76**	0.01
f_8	0.00	0.00	0.00	**1.00**	0.89	0.13
f_{10}	0.00	0.00	0.00	**0.09**	0.00	0.04
f_{11}	0.00	0.00	0.00	0.00	0.00	0.00
f_{12}	0.00	0.00	0.00	0.00	0.00	0.00
f_{13}	0.00	0.01	0.00	**1.00**	**1.00**	0.97
f_{14}	0.00	0.03	0.00	**1.00**	**1.00**	0.99
f_{15}	0.00	0.00	0.00	0.01	**1.00**	0.05
Avg	0.00	0.14	0.03	0.66	**0.71**	0.56
Median	0.00	0.02	0.00	**1.00**	0.95	0.87

One may notice that some functions were solved to the desired precision for the large majority of runs (see $f_1 - f_6$, f_8, and $f_{13} - f_{15}$, around 70% of the problems), using either LM, NM, or P methods. On the other hand, two functions could not be solved at all (see f_{11} and f_{12}).

There is no clear winner between NM and LM when average and median success rates are evaluated over the 14 benchmark problems. However, individual comparisons show that LM was substantially superior to NM for problems f_2, f_3, and f_{10}, while NM was superior for problems f_7 and f_{15}. Therefore, one may suppose that LM and NM should be used together to increase the chance of finding adequate constant values.

Instead of looking at only the desired precision, one can count the final results at distinct precision levels shown in Figure 2. In that plot, the counts are accumulated from left to right to achieve 100 runs. For instance, the amount of runs where the precision was $MAE < 1e{-}05$ is shown in position $1e$-05. Naturally, the faster the method reaches the top of the plot the better.

As previously discussed, approximately 70% of the benchmark functions were easily solved by some of the tested optimization methods. In Figure 2, one may notice some relevant

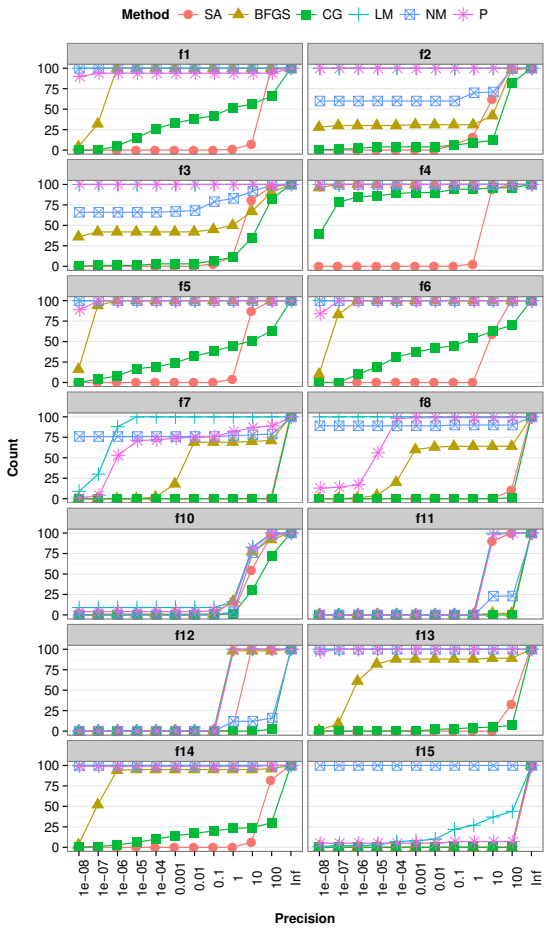

Figure 2. Accumulated counts versus precision.

characteristics when analyzing the slopes. A small variation in the slope means that the accumulated count changed just a little. This is a good indication for methods LM, P and NM in functions f_4-f_6 because most runs resulted in high-precision solutions. However, the flat line in f_{10} shows that LM found low error solutions in 9 runs, but the remaining solutions had much worse precision, increasing the count after $MAE > 1.0$.

Another visible characteristic is the linear or quasi-linear increasing slope, as happened with CG (see f_1, f_5, and f_6) and BFGS (see f_3). This kind of slope may be an indication that the method found the region containing the global optimum, but was not able to improve the solution to the desired precision. Modifications in the method's configuration could help the performance, but it is not the focus of this work.

An additional analysis of this chart is the vertical cut on the x axis, that gives the amount of solutions (in our case can also be seen as percentage since we performed 100 runs) that had that particular precision where the cut was performed. For instance, a cut on $1e{-}06$ on function f_7 gives that NM is worse than LM in terms of final solutions (76 vs. 88). A step to the left ($1e{-}07$) shows that NM was much more accurate than LM (76 vs. 30). However, moving to the right ($1e{-}05$) shows that

NM reaches that precision much fewer times than LM (same 76 vs. 100). So, this type of chart may be very useful to help the researcher choose the method and the desired precision.

Finally, Figure 3 shows the violin & dot plots of the solutions found, converting to VTR all values lower than VTR. One may observe that most distributions are asymmetric, skewed, and in some cases multi-modal with a few peaks in distinct regions (see f_2, f_3, and f_7). Those peaks (clusters) could be interpreted as local optima and are not visible in regular boxplots; that is the main reason for using a different visual representation.

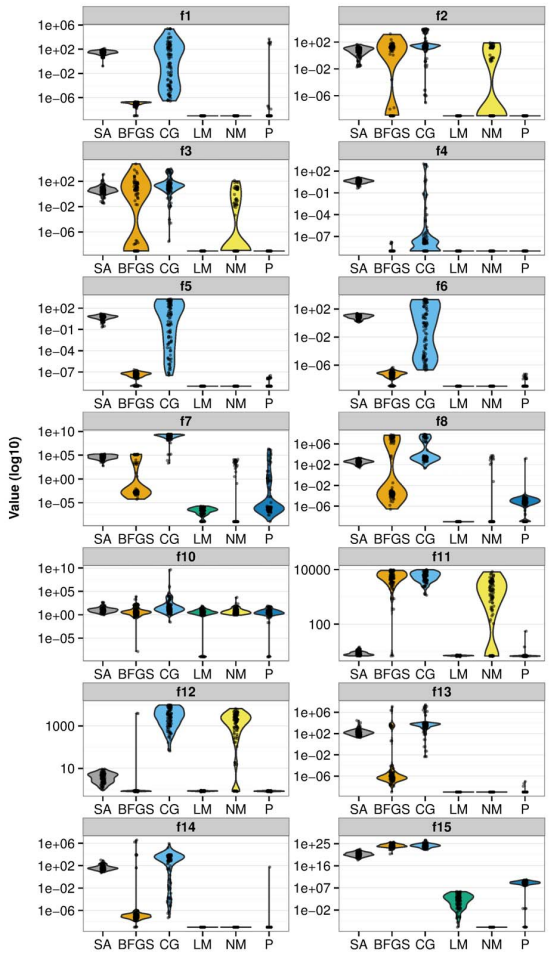

Figure 3. Violin & dot plots of the error distribution (log-scale).

It was very clear, based on Table III and the previous figures that the NM, LM, and P methods were the best approaches for the tested benchmark functions. However, those results do not consider the number of function evaluations. All methods were configured to execute a maximum number of function evaluations as presented in Table II. Nevertheless, the methods might stop earlier because of other termination criteria. That is why the success performance was calculated and is presented in Table IV.

The first characteristic to notice is that there are many Inf values, indicating that the success rate was zero; therefore, it was impossible to estimate the amount of function evaluations necessary to reach the desired precision. The second information is regarding the best results in bold face. As one can observe, LM was the method with the best performance for all problems. In Table III, NM method was the best for functions f_7 and f_{15}. Nevertheless, it required much more function evaluations than LM. Clearly, it is cheaper to perform several trials using LM than a single trial using NM.

Table IV. SUCCESS PERFORMANCE OVER 100 RUNS.

Fun	SA	BFGS	CG	LM	NM	P
f_1	Inf	7975.00	Inf	**9.00**	2001.50	1462.23
f_2	Inf	16,845.28	Inf	**35.10**	5003.22	622.25
f_3	Inf	12,700.23	Inf	**29.64**	4548.14	553.02
f_4	Inf	4168.33	1075.48	**9.00**	2001.64	1118.12
f_5	Inf	2195.70	Inf	**9.00**	2001.47	460.91
f_6	Inf	3540.00	Inf	**9.00**	2001.41	493.85
f_7	Inf	Inf	Inf	**280.25**	3950.03	73,000.00
f_8	Inf	Inf	Inf	**33.69**	3373.07	12,988.17
f_{10}	Inf	Inf	Inf	**1333.33**	Inf	64,450.00
f_{11}	Inf	Inf	Inf	Inf	Inf	Inf
f_{12}	Inf	Inf	Inf	Inf	Inf	Inf
f_{13}	Inf	36,800.00	Inf	**9.06**	2001.60	378.85
f_{14}	Inf	11,155.56	Inf	**9.00**	2001.44	494.52
f_{15}	Inf	Inf	Inf	**1500.00**	2001.46	5760.00

One important question that arises from the results is why LM is so efficient, reaching very high success rates while requiring only a few function evaluations? As can be seen in Table IV, the difference between the success performance of LM to the other methods may be a few orders of magnitude.

The use of gradient information is not the correct explanation because BFGS and CG also use it but showed much worse performance. In fact, those two methods were substantially worse than NM and P, which are gradient-free methods. Therefore, there must be another characteristic in LM that makes it so efficient.

A reasonable answer is that LM is truly solving a least-squares problem, while the other methods are solving a least-squares problem as an *unknown black-box* problem. LM uses the information in the residual of each example in the sample dataset, whereas the other methods use only the scoring function (MAE). The scoring function holds a condensed information of the error distribution that gives neither direction nor magnitude information for the parameter update. Thus, even tough all residuals are available, that valuable information is simply ignored.

IV. SUMMARY AND CONCLUSIONS

In this paper, an experimental evaluation of methods for constant optimization in symbolic regression benchmark problems is presented. Our study investigates the performance of several well-known optimization methods that can be used to perform nonlinear least squares. Some of those methods are gradient-free while others calculate gradients numerically. Only the constant optimization methods were tested because, as explained before, other researchers showed that their inclusion in symbolic regression methods largely improves model quality; therefore, it is not our objective to evolve models with the optimization techniques investigated here.

We performed tests on a set of fourteen symbolic regression problems commonly studied in the literature, which were

proposed by the symbolic regression community. The objective of this paper was then to check whether the optimization methods were reliable, finding constants that could result in small prediction error, without considering the computational complexity of the methods. The methods were executed using the default parameters of the optimization library. The main conclusions of this paper are:

1) There was no clear winner method in terms of success rate, but LM was the most efficient in terms of success performance. However, some problems were not solved correctly, and on two other problems LM was outperformed (in terms of success rate) by NM;
2) Because LM can fail, as should be expected since no method is perfect, more than one constant optimization trial should be done, using distinct initial guesses. However, even though NM was better than LM for problems f_7 and f_{15}, several LM trials would be cheaper;
3) The efficiency of LM serves as a lesson to us symbolic regression practitioners: a regression problem should not be solved as an unknown black-box problem. This leads to the conclusion that, at least for the examined benchmark problems, the use of any black-box optimizer such as metaheuristics should be avoided since they will likely be much less efficient. If a researcher wants to test a global optimizer, such as CMA-ES, PSO, DE, or GA on those problems, we suggest that LM be considered as the control method.
4) Although LM is considered a local search method, we believe that more efforts should be made to develop gradient-like information for GP and similar techniques. That kind of information will certainly have a substantial positive impact on the optimization process, reducing the randomness of the search and leading to large speedups.

In the future we intend to investigate more difficult symbolic regression problems that may require global optimization techniques to find adequate constant values. The next step is to add this procedure to improve Kaizen Programming performance.

ACKNOWLEDGMENTS

This paper was supported by the Brazilian Government CNPq (Universal) grant (486950/2013-1) and CAPES (Science without Borders) grant (12180-13-0) to V.V.M., CAPES grant (CNPq PIBIC 165293/2013-6) to V. C. C, and Canada's NSERC Discovery grant RGPIN 283304-2012 to W.B.

REFERENCES

[1] V. V. De Melo, "Kaizen programming," in *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*, GECCO '14, (New York, NY, USA), pp. 895–902, ACM, 2014.

[2] W. Banzhaf, "Artificial intelligence: Genetic programming," in *International Encyclopedia of the Social & Behavioral Sciences* (N. J. Smelser and P. B. Baltes, eds.), pp. 789–792, Oxford: Pergamon, 2001.

[3] M. Brameier and W. Banzhaf, "A comparison of linear genetic programming and neural networks in medical data mining," *IEEE Transactions on Evolutionary Computation*, vol. 5, pp. 17–26, Feb. 2001.

[4] L. F. D. P. Sotto and V. V. de Melo, "Investigation of linear genetic programming techniques for symbolic regression," in *Intelligent Systems (BRACIS), 2014 Brazilian Conference on*, pp. 146–151, IEEE, 2014.

[5] L. F. D. P. Sotto and V. V. de Melo, "Comparison of linear genetic programming variants for symbolic regression," in *Proceedings of the 2014 conference companion on Genetic and evolutionary computation companion*, pp. 135–136, ACM, 2014.

[6] M. O´Neil and C. Ryan, "Grammatical evolution," in *Grammatical Evolution*, vol. 4 of *Genetic Programming Series*, pp. 33–47, Springer US, 2003.

[7] D. M. Bates and D. G. Watts, *Nonlinear regression: iterative estimation and linear approximations*. Wiley Online Library, 1988.

[8] S. Mukherjee and M. J. Eppstein, "Differential evolution of constants in genetic programming improves efficacy and bloat," in *Companion Volume of the 14th International Genetic and Evolutionary Computation Conference (GECCO-2012)*, pp. 625–626, ACM, 2012.

[9] A. Topchy and W. F. Punch, "Faster genetic programming based on local gradient search of numeric leaf values," in *Proceedings of the 3rd International Genetic and Evolutionary Computation Conference (GECCO-2001)*, pp. 155–162, 2001.

[10] A. I. Esparcia-Alcazar and K. Sharman, "Learning schemes for genetic programming," *Genetic Programming*, pp. 57–65, 1997.

[11] H. Cao, L. Kang, Y. Chen, and J. Yu, "Evolutionary modeling of systems of ordinary differential equations with genetic programming," *Genetic Programming and Evolvable Machines*, vol. 1, no. 4, pp. 309–337, 2000.

[12] M. F. Korns, "Abstract expression grammar symbolic regression," in *Genetic Programming Theory and Practice VIII*, vol. 8 of *Genetic and Evolutionary Computation*, pp. 109–128, Springer New York, 2011.

[13] M. Kommenda, G. Kronberger, S. Winkler, M. Affenzeller, and S. Wagner, "Effects of constant optimization by nonlinear least squares minimization in symbolic regression," in *Companion Volume of the 15th International Genetic and Evolutionary Computation Conference (GECCO-2013)*, pp. 1121–1128, ACM, 2013.

[14] T. Worm and K. Chiu, "Prioritized grammar enumeration: Symbolic regression by dynamic programming," in *Proceeding of the 15th International Genetic and Evolutionary Computation Conference (GECCO-2013)*, pp. 1021–1028, ACM, 2013.

[15] M. Schoenauer, M. Sebag, F. Jouve, B. Lamy, H. Maitournam, *et al.*, "Evolutionary identification of macro-mechanical models," *Advances in Genetic Programming*, vol. 2, pp. 467–488, 1996.

[16] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, *et al.*, "Optimization by simmulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[17] R. Fletcher, *Practical methods of optimization*. John Wiley & Sons, 2013.

[18] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, vol. 49. National Bureau of Standards Washington, DC, 1952.

[19] K. Levenberg, "A method for the solution of certain problems in least squares," *Quarterly of Applied Mathematics*, vol. 2, pp. 164–168, 1944.

[20] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The Computer Journal*, vol. 7, no. 4, pp. 308–313, 1965.

[21] M. J. Powell, "An efficient method for finding the minimum of a function of several variables without calculating derivatives," *The Computer Journal*, vol. 7, no. 2, pp. 155–162, 1964.

[22] J. McDermott, D. R. White, S. Luke, L. Manzoni, M. Castelli, L. Vanneschi, W. Jaskowski, K. Krawiec, R. Harper, K. De Jong, *et al.*, "Genetic programming needs better benchmarks," in *Proceedings of the 14th International Genetic and Evolutionary Computation Conference (GECCO-2012)*, pp. 791–798, ACM, 2012.

[23] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari, "Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization.," Tech. Rep. KanGAL Report 2005005, Nanyang Technological University, Singapore, 2005.