Parametrizing GP Trees for Better Symbolic Regression Performance through Gradient Descent

Federico Julian Camerota Verdù[*]
Gloria Pietropolli[*]
federicojulian.camerotaverdu@phd.units.it
gloria.pietropolli@phd.units.it
Dipartimento di Matematica e Geoscienze,
Università degli Studi di Trieste
Trieste, Italy

Luca Manzoni
lmanzoni@units.it
Dipartimento di Matematica e Geoscienze,
Università degli Studi di Trieste
Trieste, Italy

Mauro Castelli
mcastelli@novaims.unl.pt
NOVA Information Management School (NOVA IMS),
Universidade NOVA de Lisboa
Lisboa, Portugal

ABSTRACT

Symbolic regression is a common problem in genetic programming (GP), but the syntactic search carried out by the standard GP algorithm often struggles to tune the learned expressions. On the other hand, gradient-based optimizers can efficiently tune parametric functions by exploring the search space locally. While there is a large amount of research on the combination of evolutionary algorithms and local search (LS) strategies, few of these studies deal with GP. To get the best from both worlds, we propose embedding learnable parameters in GP programs and combining the standard GP evolutionary approach with a gradient-based refinement of the individuals employing the Adam optimizer. We devise two different algorithms that differ in how these parameters are shared in the expression operators and report experimental results performed on a set of standard real-life application datasets. Our findings show that the proposed gradient-based LS approach can be effectively combined with GP to outperform the original algorithm.

CCS CONCEPTS

• **Computing methodologies** → *Machine learning approaches*; **Genetic programming**.

KEYWORDS

genetic programming, gradient descent, local search, adam, memetic search

ACM Reference Format:
Federico Julian Camerota Verdù, Gloria Pietropolli, Luca Manzoni, and Mauro Castelli. 2023. Parametrizing GP Trees for Better Symbolic Regression Performance through Gradient Descent. In *Genetic and Evolutionary Computation Conference Companion (GECCO '23 Companion), July 15–19, 2023, Lisbon, Portugal.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3583133.3590574

[*]Both authors contributed equally to this research.

1 INTRODUCTION

Genetic Programming (GP) [9] is a powerful population-based evolutionary algorithm (EA) for solving many real-world optimization problems by the automatic generation of computer programs. One of the main strengths of GP, which still makes it one of the most widely used algorithms nowadays, is that its solutions are naturally interpretable. However, the GP algorithm only employs a syntactic search that is intrinsically unable to efficiently adjust the (implicit) parameters of a given expression. Powerful local search (LS) algorithms, integrated into GP as additional search operators, have been developed over the past years to overcome such limitations. The resulting combination is usually referred to as *memetic algorithm* in evolutionary computation literature [4]. Despite the promising results obtained through the integration of LS in the GP framework [8, 13, 16], LS strategies remain still seldom used, if at all, in GP systems [15]. In this work, we chose a well-known gradient descent algorithm to perform LS over the individuals of a given GP population. A fundamental characteristic of gradient-based search is that the solutions can be improved gradually and steadily in a continuous fashion [2]. GP and gradient-based optimizers present complementary strengths: the former, thanks to the recombination operators, leads to the exploration of new areas in the solution space, preventing the algorithm from getting stuck in local optima, while the latter typically performs small shifts in the local area of the solution space. Their combination should guarantee a jump in promising areas (where good-quality solutions lie) and a subsequent refinement of these solutions obtained with the gradient-based algorithm. We propose an embedding of the gradient-based optimizer in the GP framework that is quite intuitive: firstly, we parameterize the GP representation trees and, then, optimize the parameters with LS. The optimization of the parameters with gradient-based techniques is possible since GP individuals encode mathematical formulas that can be differentiated, thus making it possible to compute their gradients – as long as the function set provided is composed of differentiable functions. Furthermore, due to its simplicity, the GP tree embedding introduced in this work can be easily integrated into other GP-based state-of-the-art techniques that have proven successful in symbolic regression [10], e.g., leveraging more powerful evolutionary operators. The proposed hybridization is tested on three benchmark problems and compared to a standard GP algorithm to study its effectiveness.

2 SYMBOLIC REGRESSION WITH GP

Symbolic regression consists of learning a model in the form of a mathematical formula that describes the relationship between an input-output pair. Let $\mathbb{T} = \{(\mathbf{x}_i, y_i)\}_{i=1\dots n}$ be the training set, the goal of symbolic regression is to search for the symbolic expression $K^O : \mathbb{R}^p \rightarrow \mathbb{R}$ that fits well $\mathbb{T}$. The GP approach [1] consists in defining a function set and a terminal set, so that the generic regression problem state as follows:

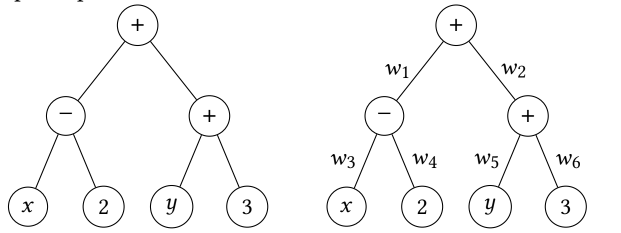
$$(K^O, \theta^O) \leftarrow \underset{K \in \mathbb{G}; \theta \in \mathbb{R}^m}{\operatorname{argmin}} f(K(\mathbf{x}_i, \theta), y_i) \quad \text{with } i = 1, \dots, p \qquad (1)$$

where $\mathbb{G}$ is the solution space, f is the fitness function, and θ is a parametrization of K, assuming m real-valued parameters. In the GP workflow, parameter optimization is not performed explicitly, as operators only focus on the syntax of the individuals [3]. This approach leads to some performance limitations: let's consider the GP individual $K(x) = x + \sin(x)$, and let's assume that the optimal solution is $K^*(x) = 3.3x + 1.003 \sin(0.0001x)$. Since the parameters in traditional GP are left unchanged during the whole evolution, the solution K^* might be easily lost. The above argument lies at the core of our investigation based on embedding an LS procedure to improve the performance achieved by plain GP.

The combination of Evolutionary Algorithms (EAs) and local optimizers attracted great interest in recent years [5]. Although the number of studies based on the combination of EAs and LS is quite large, the subset concerning the combination of GP and LS is still very limited. Specifically, focusing on the use of gradient descent in GP, the contributions found in the literature deal with particular components or task-specific solutions. In Topchy et al. [14], the effectiveness of gradient-based optimization of numeric leaf values in GP is analyzed. In Graff et al. [6], the authors used resilient back-propagation combined with GP in the wind speed forecasting domain. In Smart and Zhang [13], authors included weight parameters (called inclusion factors) for each function node aiming at modulating the importance of each node belonging to the tree. More recently, in Trujillo et al. [16], a straightforward parametrization for GP trees is proposed. In Kommenda et al. [8], a gradient-based optimization algorithm is used to adjust only the constant values in trees during their evolution. Finally, a way to combine a gradient-based method with Geometric Semantic Genetic Programming (GSGP) has been proposed in Pietropolli et al. [12].

3 COMBINING GP AND GRADIENT DESCENT

Plain GP is unable to efficiently perform small adjustments on the (implicit) parameters of a given expression. Also, gradient-based approaches come with some limitations such as the tendency to get stuck in local optima. To enhance the expressive capacity of a GP individual, we propose the addition of learnable parameters in its operators. These parameters act as multiplicative factors for the operands of the operations which define the GP tree. In doing so, the resulting GP individuals are interpretable as parametric functions which can be optimized via gradient descent. We can, therefore, use a gradient-based optimizer to find suitable values for the tree parametrization. To better frame our method, let us consider a canonical GP individual: it can be represented as a tree where all the edge connections between nodes take a constant value of 1. Yet, the possibility of modifying those values leads to a wider spectrum of possible solutions and, thus, to a possible improvement in the quality of the individual. Let us provide an example of the proposed parametrization.



On the left is represented a GP individual that encodes the following equation:

$$(x - 2) + (y + 3) \qquad (2)$$

On the right, is represented the same GP individual, with the addition of the parameters w_i over all the edge connections. Now, this tree is encoding the following equation:

$$w_1 \cdot (w_3 \cdot x - w_4 \cdot 2) + w_2 \cdot (w_5 \cdot y + w_6 \cdot 3) \qquad (3)$$

that would correspond to Equation 2 if all the weights w_i were set to 1. Because the individual on the right represents a parametric function, we can apply a gradient-based optimizer to tune the parameter values and, consequently, improve the solution's quality. We propose and investigate two alternatives to embed parameters on a GP individual. The first one is *Genetic Program Gradient Descent* (**GPGD**), which assigns a set of weights to each operator in the GP function set. These weights are the same for different instances of the same operator within a single program but may change among different individuals. In the previous example, both sum operators would have the same parameters, thus $w_1 = w_5$ and $w_2 = w_6$. The second one is *OPerators Gradient Descent* (**OPGD**), which assigns a different set of weights to each instance of the GP operators. In the previous example, the two sum operators would have different parameters, thus $w_1 \neq w_5$ and $w_2 \neq w_6$.

Another important aspect that needs to be considered is at which point of the evolutionary process the gradient-based optimizer should be applied. For a complete investigation of the proposed methods, we consider two alternatives. First of all, we propose an *alternated* (**A**) variant, where the gradient-based optimizer is applied after each generation, and before the following generation begins. This means that the local optimization of individuals from generation t impacts the search at generation $t+1$. In this version of the algorithm, the optimizer is applied to all the individuals. Then, in the *consecutive* (**C**) variant, all the GP genetic steps are performed, and only after the evolutionary process stops the gradient-based optimizer algorithm is applied for a predetermined number of epochs. Here, the optimizer is applied only to the best individual. These two different alternatives will be considered for both GPGD and OPGD.

Lastly, it is necessary to decide how much of the total computational budget (here defined as the total number of fitness evaluations) to allocate to the GP routine and how much to leave to the optimization phase. Hence, let us also introduce the tuple (k_1, k_2), which defines the ratio of fitness evaluations used for the evolutionary process (k_1) and the ones employed for updating the parameters through the Adam optimizer (k_2). In the experimental phase, different values for this ratio will be considered.

Table 1: Principal characteristics of datasets: number of variables, number of instances, domain, and the task.

Dataset	Variables	Instances	Area	Task
yac	7	308	Physics	Regression
conc	9	1030	Physics	Regression
air	6	1503	Physics	Regression

4 EXPERIMENTAL STUDY

This Section describes the datasets as well as all the experimental settings. The code, for the complete reproducibility of the proposed experiments, is available at https://github.com/federico-camerota/gpxgp.

Three complex and multidimensional real-world datasets, widely used in the literature as benchmarks for GP, are employed to assess the validity of the proposed technique. Table 1 reports a short description of their principal features. For a more detailed discussion, the reader is referred to McDermott et al. [11].

A total of 30 (random) partitions of the data into train and test sets, according to a proportion of 70% and 30% respectively, has been considered for each dataset. Moreover, experiments over each partition have been repeated with 5 different random seeds. Thus, for each dataset, a total of 150 runs are available for the experimental study. A population of 100 individuals is considered. The function set is composed of +, −, ×, and / (protected). To preserve interpretability, we prune trees to enforce a maximum tree depth – equal to 10 –during the whole process. Survival from one generation to the other is always guaranteed for the best individual in the population (elitism). To optimize the individual's parameters, we use the Adam optimizer [7] with a learning rate equal to 0.01 (while the remaining hyperparameters for the optimization algorithm are the default ones). The optimizer and the hyperparameter values were chosen after a preliminary tuning phase, in which different combinations were tested. The total number of generations considered for the GP algorithm equals 200. To render the comparison fair, the total number of fitness evaluations must be equal for every method considered. Recall that (k_1, k_2) indicates the ratio between GP and gradient-based optimizer steps. We always consider $k_1 = 1$, while k_2 ranges in $\{1, 5, 10\}$. Let n denote the total number of fitness evaluations, n_1 be the number of GP generations, and n_2 be the number of Adam steps. The value of n_1 and n_2 is univocally determined by n, k_1, and k_2, as follows:

$$n_1 = \frac{n}{k_1 + k_2} k_1 \quad n_2 = \frac{n}{k_1 + k_2} k_2 \tag{4}$$

This allows a fair comparison between different techniques, ensuring that the number of fitness evaluations is the same for each choice of k_2. Fitness is calculated as the root mean square error (RMSE) between the target and the predicted values.

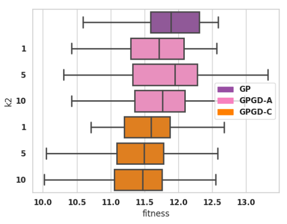
5 EXPERIMENTAL RESULTS

In Figure 1, we report a series of results obtained during our investigation. Each row shows results related to one of the different benchmark problems considered. The first (second) row compares, via boxplots, the test performance after 200 fitness evaluations of plain GP and the GPGD (OPGD) method. Furthermore, the results of the GPGD (OPGD) method are analyzed for both the A and C variants, for k_2 values ranging in the set $\{1, 5, 10\}$.

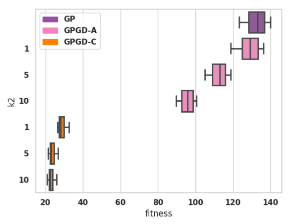
Table 2: P-values returned by the Wilcoxon ran-sum test and the correspondent improvement in percentage over the GP median error. Results are reported solely for $k_2 = 10$.

		GPGD-A	GPGD-C	OPGD-A	OPGD-C
yac	p-value	0.09	2.84 e-08	0.0	2.85 e-08
	%	-1	-6	-15	-6
conc	p-value	0.0	0.0	0.0	0.0
	%	-28	-83	-89	-85
air	p-value	0.0	0.0	0.0	0.0
	%	-21	-48	-49	-49

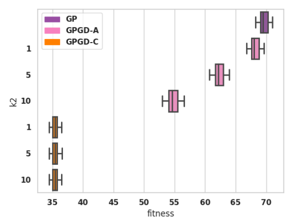
Table 2 reports the p-values obtained from the Wilcoxon rank-sum test for pairwise data comparison, for $\alpha = 0.05$, under the alternative hypothesis that the samples do not have equal medians. Furthermore, we include the percentage improvement w.r.t. GP, computed as the difference between the median error of our method and that of GP, over the GP median error. These values are reported exclusively for $k_2 = 10$, as it results in the best choice for this hyperparameter. Across all the datasets, the proposed methods achieve results equal to or better than simple GP, with at least one of the configurations outperforming it. In general, the best performance is achieved for both GPGD and OPGD when k_2 is set to 10. Thus, assigning a greater fitness evaluation budget to the gradient-based optimizer improves the quality of the solution. For all the considered benchmarks – except for *yac* – OPGD-C with $k_2 = 10$ leads to a test fitness that, also in the worst case (upper whisker in the boxplot), is better than the one achieved in the best case (lower whisker in the boxplot) by plain GP. All these considerations allow us to conclude that the LS introduced in this work can effectively explore the solutions landscape and steer the GP-based search toward more promising areas. Specifically, considering the *conc* and *air* datasets, all methods lead to a significant performance improvement. Boxplots of all the variants of GPGD and OPGD show very similar distributions in the test fitness. For the *yac* dataset, different methods lead to different fitness results, demonstrating a sensitivity to the choice of the variant (between A and C) and the value of k_2. The C variant seems to outperform the A one, and a higher value of k_2 seems to improve the algorithm performance.



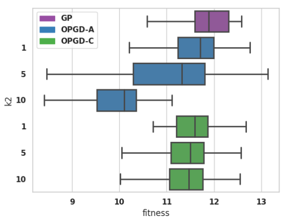
yac (testing RMSE with GPGD)



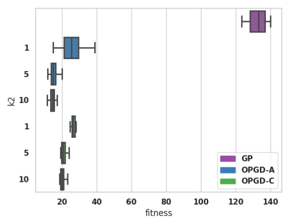
conc (testing RMSE with GPGD)



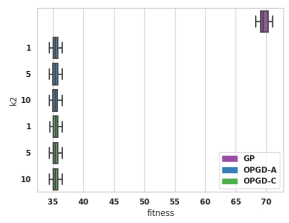
air (testing RMSE with GPGD)



yac (testing RMSE with OPGD)



conc (testing RMSE with OPGD)



air (testing RMSE with OPGD)

Figure 1: Boxplots of testing RMSE, obtained over 150 independent runs. In the first (second) row, classic GP is compared to GPGD methods (OPGD methods) for all the considered datasets.

6 CONCLUSIONS

In this paper, to enhance GP performance in symbolic regression tasks, we propose the addition of learnable parameters in its operators. Subsequently, through the use of a gradient-based algorithm, we perform the optimization of the introduced parameterization of the individuals. The intuition behind this work relies on the possibility of exploiting and combining the advantages of evolutionary-based algorithms and LS to achieve faster convergence of the evolutionary search process. To assess the performance of our methods, we consider complex real-life datasets. Experimental results show that our methods outperform plain GP with statistical significance both in the training and test set. Considering the promising results obtained in this first analysis, this work paves the way for multiple possible future developments. The most exciting research direction, which is also one of the main motivations for the approach we chose, is to combine the proposed methods with current state-of-the-art GP-based algorithms that do not use any LS strategy. In any case, the aim of this work is not limited to proving the effectiveness of embedding a gradient-based optimizer in the evolutionary process. The most important message that we want to convey is that there is an unexplored (yet extraordinary) potential in integrating powerful local search approaches into GP. This topic deserves more attention from the GP community.

REFERENCES

[1] Douglas Adriano Augusto and Helio JC Barbosa. 2000. Symbolic regression via genetic programming. In *Proceedings. Vol. 1. Sixth Brazilian Symposium on Neural Networks*. IEEE, 173–178.
[2] Yoshua Bengio. 2000. Gradient-based optimization of hyperparameters. *Neural computation* 12, 8 (2000), 1889–1900.
[3] Mauro Castelli, Leonardo Trujillo, Leonardo Vanneschi, Sara Silva, Emigdio Z-Flores, and Pierrick Legrand. 2015. Geometric semantic genetic programming with local search. In *Proceedings of the 2015 annual conference on genetic and evolutionary computation*. 999–1006.
[4] Xianshun Chen, Yew-Soon Ong, Meng-Hiot Lim, and Kay Chen Tan. 2011. A multi-facet survey on memetic computation. *IEEE Transactions on evolutionary computation* 15, 5 (2011), 591–607.
[5] Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. 2013. Exploration and exploitation in evolutionary algorithms: A survey. *ACM computing surveys (CSUR)* 45, 3 (2013), 1–33.
[6] Mario Graff, Rafael Pena, and Aurelio Medina. 2013. Wind speed forecasting using genetic programming. In *2013 IEEE Congress on Evolutionary Computation*. IEEE, 408–415.
[7] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[8] Michael Kommenda, Gabriel Kronberger, Stephan Winkler, Michael Affenzeller, and Stefan Wagner. 2013. Effects of constant optimization by nonlinear least squares minimization in symbolic regression. In *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*. 1121–1128.
[9] John R Koza. 1994. Genetic programming as a means for programming computers by natural selection. *Statistics and computing* 4, 2 (1994), 87–112.
[10] William La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabricio de Franca, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason Moore. 2021. Contemporary Symbolic Regression Methods and their Relative Performance. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, J. Vanschoren and S. Yeung (Eds.), Vol. 1.
[11] James McDermott, David R White, Sean Luke, Luca Manzoni, Mauro Castelli, Leonardo Vanneschi, Wojciech Jaskowski, Krzysztof Krawiec, Robin Harper, Kenneth De Jong, et al. 2012. Genetic programming needs better benchmarks. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*. 791–798.
[12] Gloria Pietropolli, Luca Manzoni, Alessia Paoletti, and Mauro Castelli. 2022. Combining geometric semantic gp with gradient-descent optimization. In *European Conference on Genetic Programming (Part of EvoStar)*. Springer, 19–33.
[13] Will Smart and Mengjie Zhang. 2004. Continuously evolving programs in genetic programming using gradient descent. In *Proceedings of the 7th Asia-Pacific Conference on Complex Systems*.
[14] Alexander Topchy, William F Punch, et al. 2001. Faster genetic programming based on local gradient search of numeric leaf values. In *Proceedings of the genetic and evolutionary computation conference (GECCO-2001)*, Vol. 155162. Morgan Kaufmann San Francisco, CA.
[15] Leonardo Trujillo, Perla S Juárez-Smith, Pierrick Legrand, Sara Silva, Mauro Castelli, Leonardo Vanneschi, Oliver Schütze, Luis Muñoz, et al. 2018. Local search is underused in genetic programming. In *Genetic Programming Theory and Practice XIV*. Springer, 119–137.
[16] Leonardo Trujillo, Oliver Schütze, Pierrick Legrand, et al. 2014. Evaluating the effects of local search in genetic programming. In *EVOLVE-A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation V*. Springer, 213–228.