



# On the hybridization of geometric semantic GP with gradient-based optimizers

Gloria Pietropolli<sup>1</sup> · Luca Manzoni<sup>1</sup> · Alessia Paoletti<sup>1</sup> · Mauro Castelli<sup>2</sup>

Received: 2 November 2022 / Revised: 27 June 2023 / Accepted: 23 September 2023 /

Published online: 28 October 2023

© The Author(s) 2023

## Abstract

Geometric semantic genetic programming (GSGP) is a popular form of GP where the effect of crossover and mutation can be expressed as geometric operations on a semantic space. A recent study showed that GSGP can be hybridized with a standard gradient-based optimizer, Adam, commonly used in training artificial neural networks. We expand upon that work by considering more gradient-based optimizers, a deeper investigation of their parameters, how the hybridization is performed, and a more comprehensive set of benchmark problems. With the correct choice of hyperparameters, this hybridization improves the performances of GSGP and allows it to reach the same fitness values with fewer fitness evaluations.

**Keywords** Geometric semantic genetic programming · Stochastic gradient descent · Adam · Evolutionary algorithm

---

✉ Gloria Pietropolli  
gloria.pietropolli@phd.units.it

Luca Manzoni  
lmanzoni@units.it

Alessia Paoletti  
paoletti.alessia@gmail.com

Mauro Castelli  
mcastelli@novaaims.unl.pt

<sup>1</sup> Dipartimento di Matematica e Geoscienze, Università degli Studi di Trieste, Via Alfonso Valerio 12/1, Trieste 34127, TS, Italy

<sup>2</sup> NOVA Information Management School (NOVA IMS), Universidade NOVA de Lisboa, Campus de Campolide, 1070-312 Lisbon, Portugal

## 1 Introduction

Genetic programming (GP) [1] is one of the most prominent evolutionary computation techniques, with the ability to evolve programs, usually represented as trees, to solve specific problems given a collection of input and output pairs. Traditionally, operators in GP have focused on manipulating the syntax of GP individuals, like swapping subtrees for crossover or replacing subtrees for mutation. While simple to describe, these operations produce effects on the *semantics* [2] of the individuals that can be complex to predict, with small variations in the syntax that may significantly affect the semantics. To address this problem, semantic operators were introduced. In particular, geometric semantic operators, first introduced in [3], have been used for defining Geometric Semantic GP (GSGP), a new kind of GP where crossover and mutation operators directly act on the semantics of the candidate solutions. Successively, the work described in [4] introduced an implementation of GSGP that allows overcoming the limitation caused by the growth of the individuals at each generation, thus making it possible to address complex real-world problems.

While the introduction of GSGP establishes a clear effect of recombination and mutation operators on the semantics of the candidate solutions and improves the quality of the generated solutions, there is still a largely untapped opportunity for combining GSGP with local search methods. In particular, we can observe that, given two GP trees  $T_1$  and  $T_2$ , their recombination is obtained as  $\alpha T_1 + (1 - \alpha)T_2$ , and the mutation of one of them is given by  $T_1 + ms(R_1 - R_2)$ , where  $R_1$  and  $R_2$  are two random trees. As we can observe, three parameters,  $\alpha$ ,  $\beta = 1 - \alpha$ , and  $ms$  are either fixed or randomly selected during the evolution process. As long as each function used in the generation of the individuals is derivable, we can compute the *gradient* of the error with respect to the parameters used in crossover and mutation. Thus, we can employ a gradient-based optimizer to update the parameters of each crossover and mutation.

In this paper, we investigate the combination of GSGP with two gradient-based optimizers: stochastic gradient descent and Adam. In some sense, by combining GSGP with a gradient-based optimizer, we are leveraging the strengths of each of the two methods: GSGP (and GP in general) is good at providing structural changes in the shape of the individuals, while gradient-based methods are perfect for optimizing a series of parameters of the individuals that the evolutionary process has difficulty in optimizing.

GSGP, thanks to the recombination operators, leads to the exploration of multiple zones in the solution space. Geometric semantic crossover allows the creation of a child on the segment that connects two parents (in the semantic space) even if they are a long way apart. Geometric semantic mutation introduces random perturbations, leading to solution space exploration. On the other side, gradient-based methods perform local changes, so no big jumps in the solution space are done, but only small shifts in the local area of the solution space.

Concerning the gradient-based optimizer choice, we compare two of the most commonly used in the neural networks' training process: SGD and Adam. We

selected the former because, despite its simplicity, it remains one of the most performing methods. On the other hand, Adam consists of a more elaborated technique, which takes into account not only the directions of the gradient but also the first and the second moment, thus resulting in a more suitable choice—but also more computationally expensive—when dealing with challenging tasks.

This work is the natural consequence of a preliminary study [5] that presented promising results achieved by combining GSGP with gradient-based optimization. This study aims to provide a clear picture concerning the combination of GSGP and gradient-based optimizers. In the preliminary study, only the Adam optimizer was considered, while here, we want to study also the effect of a combination with Stochastic Gradient Descent to investigate if a powerful—yet expensive—method is necessary to render this combination more performing w.r.t. classic GSGP or if it is sufficient an easier—yet cheaper and faster—optimizer. We will also investigate the performance of both gradient-based methods by choosing different hyperparameters settings, specifically considering different learning rate values. Moreover, we will study different alternatives to alternate between GSGP steps and gradient-based steps, gradually increasing the number of gradient descent steps at the expense of the total amount of GSGP generations.

We experimentally show that the proposed method can provide better performance with respect to plain GSGP, thus suggesting that a combination of local search (via gradient-based optimizers) with GSGP is a new promising way to leverage knowledge from other areas of artificial intelligence.

Recalling that we were inspired by the neural network framework when introducing gradient-based optimization among GSGP, we will also observe some strong analogies with the deep learning world. Specifically: experimental results will show that the quality of the proposed hybridization will depend on the choice of some hyperparameters of the algorithm. In turn, the optimal value of the hyperparameters will not be universal and unique but will change depending on the benchmark problem under consideration.

This paper is structured as follows: Sect. 2 provides an overview of the applications of local search to evolutionary methods and GP in particular. Section 3 recalls the reliant notions of GSGP (Sect. 3.1) and the gradient-based algorithms studied in this work (Sect. 3.2). Section 3.3 introduces the proposed hybridized algorithms combining GSGP and gradient-based optimizers. The experimental settings and the dataset used in the experimental validation are described in Sect. 4, and the results of the experimental campaign are presented in Sect. 5. Section 6 summarizes the main contributions of the study and provides directions for further research.

## 2 Related works

The combination of evolutionary algorithms (EAs) and local search strategies received greater attention in recent years [6–8]. While EAs can explore large areas of the search space, the evolutionary search process improves the programs in a discontinuous way [9]. On the other hand, when considering local optimizers, solutions can be improved gradually and steadily in a continuous way. Thus, as stated

by Z-Flores et al. [10], a hybrid approach that combines EAs with a local optimizer can result in a well-performing search strategy. Such approaches are a simple type of memetic search [6], and the basic idea is to include within the optimization process an additional search operator that, given an individual, searches for the local optima around it. Thanks to the possibility of fully exploiting the local region around each individual, memetic algorithms obtained satisfactory results over different domains [6, 7], and they outperform evolutionary algorithms in multimodal optimisation [11]. Despite these results, the literature presents a poor number of contributions dealing with GP [12], thus indicating that the GP community may not have addressed the topic adequately. Some examples are the works of Eskridge [13] and Wang [14] that are domain-specific memetic techniques not addressing the task of symbolic regression considered in this work. Muñoz et al. [15] proposed a memetic algorithm that, given a regression (or classification) problem, creates a new feature space that, subsequently, is considered for addressing the underlying optimization problem. The algorithm maximizes the mutual information [16] in the new feature space and shows superior results with respect to other state-of-the-art techniques.

Focusing on the use of gradient descent in GP, existing contributions concentrate on particular tasks or components of the solutions. For instance, Topcyy et al. [17] analyzed the effectiveness of gradient search optimization of numeric leaf values in GP. In particular, they tuned conventional random constants utilizing gradient descent and considered several symbolic regression problems to demonstrate the approach's effectiveness. Zhang et al. [18] applied a similar strategy to address object classification problems and obtained better results compared to the ones achieved with standard GP. Graff et al. [19] employed resilient backpropagation with GP to address a complex real-world problem concerning wind speed forecasting, showing improved results. In [20], the authors used gradient-descent search to make partial changes in certain parts of genetic programs during evolution. To do that, they introduced weight parameters for each function node, what the authors call inclusion factors. These weights modulate the importance that each node has within the tree. The proposed method, which uses standard genetic operators and gradient descent applied to the inclusion factors, outperformed the basic GP approach that only uses standard genetic operators (i.e., without gradient descent and inclusion factors).

The aforementioned contributions are related to syntax-based GP. In the context of semantics-based GP [2], the integration of a local search strategy into GP was proposed by Castelli et al. [21] with the definition of a specific semantic mutation operator. Experimental results showed satisfactory performance on the training set, but with important overfitting [22, 23].

To the best of our knowledge, the only attempt to integrate a local optimizer—specifically Adam—within semantic GP has been presented by Pietropolli et al. [5]. In their work, the authors only considered the Adam optimizer and proposed two different manners for combining it with GSGP. The first approach alternates one step of GSGP and one step of Adam. Experimental results demonstrated that this

combination outperforms classic GSGP with a statistical significance difference over all the considered benchmark problems. The second option proposes to perform all the GSGP steps initially, followed by an equal number of Adam optimizers steps. However, this different algorithm did not produce superior results compared to GSGP.

Driven by the promising results obtained by the former method, we decided to investigate its potentiality further in this paper. Here, we will consider the combination of GSGP also with another well-known optimizer and propose different possibilities for the alternation of GSGP and gradient-based optimizers. We will also test the results over a new dataset concerning a classification task (while in the previous paper only the regression task has been analyzed), characterized by a significantly higher number of variables and instances. With this extended study, we aim to provide a clear and complete analysis concerning the integration of GSGP and gradient-based optimizers.

### 3 Gradient descent GSGP

This section discusses the two tools that will be combined later in this work. Firstly: Sect. 3.1 describes geometric semantic GP. Afterward, two gradient descent optimizers techniques—gradient descent and Adam—are introduced and discussed. Finally, Sect. 3.3 presented a framework where these two powerful techniques are combined.

#### 3.1 Geometric semantic GP

Traditional genetic programming investigates the space of programs exploiting search operators that analyze their syntactic representation. Programs usually are represented as syntax trees, guaranteeing syntactically well-formed individuals. To improve the performance of GP, recent years have witnessed the integration of semantic awareness in the evolutionary process [2]. The semantics of a solution can be identified by the vector of its output values calculated on the training data. Thus, we can represent a GP individual as a point in a real finite-dimensional vector space, the so-called semantic space. The semantics concerns the meaning of the program, which is, on the other hand, completely ignored in traditional GP. Geometric semantic genetic programming (GSGP) is an evolutionary technique originating from GP that directly searches the semantic space of the programs. GSGP has been introduced by Moraglio and coauthors [3], together with the definition of the correspondent geometric semantic operators (GSOs). These operators replace traditional (syntax-based) crossover and mutation, inducing geometric properties in the semantic space. GSOs induce on the training data a unimodal error surface for any supervised learning problem where input data has to match with known targets.

Let us recall the definition of crossover and mutation in the semantic framework:

**Definition 1 (Geometric semantic crossover)** Given two parents functions  $T_1, T_2 : \mathbb{R}^n \rightarrow \mathbb{R}$ , *geometric semantic crossover* (GSC) generates the real function

$$T_{XO} = (T_1 \cdot T_R) + ((1 - T_R) \cdot T_2)$$

where  $T_R$  is a random real function whose output range in the interval  $[0, 1]$ .

**Definition 2 (Geometric semantic mutation)** Given a parent function  $T_1 : \mathbb{R}^n \rightarrow \mathbb{R}$ , *geometric semantic mutation* (GSM) generates the real functions

$$T_M = T_1 + ms \cdot (T_{R1} - T_{R2})$$

where  $T_{R1}$  and  $T_{R2}$  are random real functions whose output range in the interval  $[0, 1]$  and  $ms$  is a parameter called mutation step.

This means that GSC generates one offspring whose semantics stands on the line joining the semantics of the two parents in the semantic space, while GSM generates an individual contained in the hyper-sphere of radius  $ms$  centered in the semantics of the parent in the semantic space. An intrinsic GSGP's problem is that this technique leads to larger offspring with respect to their parents. Due to this issue, the algorithm becomes slower generation after generation, making it unsuitable for real-world applications. In [4, 24], Vanneschi and coauthors introduced a GSGP implementation that solves this problem and consists in storing only the semantic vectors of newly created individuals, besides storing all the individuals belonging to the initial population and all the random trees generated during the generations. This improvement turns the cost of evolving  $g$  generations of  $n$  individuals from  $\mathbf{O}(ng)$  to  $\mathbf{O}(g)$ . The same idea was subsequently adopted to reconstruct the best individual found by GSGP, thus allowing for its usage in a production environment [25].

### 3.2 Gradient-based optimizers

Consider a  $D$ -dimensional search space  $S \subseteq \mathbb{R}^D$  and a function  $f : \mathbb{R}^D \rightarrow \mathbb{R}$ . A *minimization* (maximization) problem consists in finding the optimal  $\bar{x} \in \mathbb{R}^D$  such that  $f(\bar{x}) \leq f(x)$  ( $f(\bar{x}) \geq f(x)$ )  $\forall x \in \mathbb{R}^D \setminus \{\bar{x}\}$ . A wide range of techniques has been introduced to successfully find the optimal solution, among which the so-called *gradient-based* optimizers. These algorithms are characterized by the fact that the search direction is defined by the gradient of the loss function at the current point.

Gradient-based optimizers are optimization algorithms used in the training phase of many machine learning models, usually employed to update the model's parameters, like linear regression coefficients and neural network weights.

In the following, two of the most well-known gradient-based optimizers will be described: *stochastic gradient descent* and *Adam*. The former is one of the most intuitive and straightforward techniques, while the latter exploits not only the gradient of the function but also its first and second momentum.

### 3.2.1 Stochastic gradient descent

Stochastic gradient descent (SGD) is a variant of the vanilla gradient descent (GD). Classic GD finds the solution to the minimization problem directly following the direction of the gradient of  $f$ , as follows:

$$\theta_{i+1} = \theta_i - \gamma \nabla f(\theta_i) \quad (1)$$

where  $\theta$  are the parameters and  $\gamma$  is the learning rate—the hyperparameter that controls how much to change the parameter in response to the estimated error. The core idea of SGD is to decompose the function that needs to be optimized as follows:

$$f(\theta_i) = \frac{1}{N} \sum_{j=1}^{N_f} f_j(\theta_i) \quad (2)$$

where  $N_f$  represents the dimension of the space to which  $f$  belongs. Consequently, the parameters can be updated as:

$$\theta_{i+1} = \theta_i - \gamma \nabla f_i(\theta_i) \quad (3)$$

This decomposition renders SGD drastically cheaper to compute w.r.t plain GD, without affecting the performance. Moreover, SGD can be thought as GD with noise, as we are replacing the actual gradient with an approximation of it, and this noise can prevent the optimizing from converging to bad local minima. Nowadays, SGD—despite its simplicity—remains one of the most powerful techniques to update weights and biases in neural network theory.

### 3.2.2 Adam

*Adam* (Adaptive moment estimation) [26] is an algorithm for first-order gradient-based optimization of stochastic objective functions based on adaptive estimates of lower-order models. Adam optimizer is efficient, easy to implement, requires little memory usage for its execution, and is well suited for problems dealing with a vast amount of data and/or parameters. The steps performed by the Adam optimizer are summarized in Algorithm 1. The inputs required for this method are the parametric function  $f(\theta)$ , the initial parameter vector  $\theta_0$ , the number of steps  $N_{\text{epochs}}$ , the learning rate  $\alpha$ , the exponential decay rate of the first momentum  $\beta_1$ , the one for the second momentum  $\beta_2$ , and  $\epsilon$ , set by default at  $10^{-8}$ . At every iteration, the algorithm updates first and second moment estimates using the gradient computed with respect to the stochastic function  $f$ . These estimates are then corrected to contrast the presence of an intrinsic initialization bias through the divisions described in lines 7 and 8, where  $\beta_1^{i+1}$  stands for the element-wise exponentiation. For further details about the implementation of the Adam optimizer and the demonstration of its properties, the reader can refer to [26].

---

**Algorithm 1** Pseudocode of the *Adam* algorithm.

---

**Require:**  $f(\theta)$ ,  $\theta_0$ ,  $N_{epochs}$ ,  $\alpha$ ,  $\beta_1 \in [0, 1]$ ,  $\beta_2 \in [0, 1]$ ,  $\epsilon$

- 1:  $m_0 \leftarrow 0$
- 2:  $v_0 \leftarrow 0$
- 3: **for**  $i = 0 \cdots N_{epochs}$  **do**
- 4:    $d_{i+1} \leftarrow \nabla_\theta f_{i+1}(\theta_i)$
- 5:    $m_{i+1} \leftarrow \beta_1 \cdot m_i + (1 - \beta_1) \cdot d_{i+1}$
- 6:    $v_{i+1} \leftarrow \beta_2 \cdot v_i + (1 - \beta_2) \cdot d_{i+1}^2$
- 7:    $\bar{m}_{i+1} \leftarrow m_{i+1}/(1 - \beta_1^{i+1})$
- 8:    $\bar{v}_{i+1} \leftarrow v_{i+1}/(1 - \beta_2^{i+1})$
- 9:    $\theta_{i+1} \leftarrow \theta_i - \alpha \cdot \bar{m}_{i+1}/(\bar{v}_{i+1})$
- 10: **end for**

---

### 3.3 GSGP hybridized with gradient descent

---

**Algorithm 2** Pseudocode of the hybridization of GSGP and Gradient Descent.

---

**Require:**  $n, p_1, p_2$

- 1: Create an initial random population
- 2: **for**  $p < n$  **do**
- 3:   Perform  $p_1$  **GSGP** steps
- 4:   Select parents
- 5:   Geometric semantic crossover
- 6:   Geometric semantic mutation
- 7:    $p += p_1$
- 8:   Perform  $p_2$  **gradient-based optimizers** steps
- 9:    $p += p_2$
- 10: **end for**
- 11: Return the best individual

---

The idea introduced in this work is to investigate the results of the combination of the strength of an evolutionary algorithm, i.e., GSGP, and gradient-based optimizers, which are one of the principal tools exploited in the neural network theory. Geometric semantic GP, thanks to the intrinsic nature of geometric semantic operators, allows big jumps in the solution space. Thus, new areas of the solution space can be explored, with GSOs also preventing the algorithm from getting stuck in a local optimum. Gradient-descent optimizers, on the other hand, are optimization techniques based on the direction of the gradient of the loss function. Thus, they typically perform small shifts in the local area of the solution space. A combination of these techniques should guarantee a jump in promising areas (i.e., where good-quality solutions lie) of the solution space, thanks to the evolutionary search of GSGP and subsequent refinement of the solution obtained with a gradient-based algorithm. Let us describe in more detail how to implement this combination. Let us consider an input vector  $\mathbf{x}$  of  $N$  features and the respective expected scalar value output  $y$ . By applying GSGP, an initial random population of functions in  $n$  variables is created.

After performing the evolutionary steps involving GSM and GSC, a new population  $T = (T_1, T_2, \dots, T_M)$  of  $M$  individuals is obtained. The resulting vector  $T$  is composed of derivable functions, as they are obtained through additions, multiplications, and compositions of derivable functions. At this point, to understand for which parameters we should differentiate  $T$ , it is necessary to introduce an equivalent definition of the geometric semantic operators presented in Sect. 3.1. In particular let us redefine the *Geometric Semantic Crossover* as  $T_{XO} = (T_1 \cdot \alpha) + ((1 - \alpha) \cdot T_2)$ , where  $0 \leq \alpha \leq 1$ , and the *geometric semantic mutation* as  $T_M = T + m \cdot (R_1 - R_2)$ , where  $0 \leq m \leq 1$ . As the values of  $\alpha$  and  $m$  are randomly initialised, we can derive  $T$  with respect to  $\alpha$ ,  $\beta = (1 - \alpha)$ , and  $m$ . Therefore, the gradient-based optimizer can be applied using as objective function  $f(\theta)$  at the considered generation, while the vector of parameters becomes  $\theta = (\alpha, \beta, m)$ . Thus, GSGP and a gradient-based optimizer can be combined to find the best solution for the problem at hand. We propose and investigate different ways to perform this integration. The most trivial one consists of alternating one step of GSGP and one step of the gradient-based optimizers.

Henceforth in the text, a single step of GSGP indicates one complete generation of GSGP, while a single step of the gradient-descent optimizer stands for one epoch during which the gradient of the function to be optimized is computed, and the weights are adjusted accordingly. Thus, when there is more than one step of the gradient-based optimizer, it signifies that the weights are being updated multiple times.

Anyway, there is also the possibility to alternate one step of GSGP with multiple steps of gradient-based optimizers. Improving the number of gradient-based steps—at the expense of GSGP steps—is an interesting option to understand how far the small shifts performed by a gradient-based optimizer can improve the overall performance. Let's indicate with the tuple  $(p_1, p_2)$  the number of steps that we alternate GSGP ( $p_1$ ) and gradient-based optimization ( $p_2$ ). Therefore, we will fix  $p_1 = 1$  while considering different values for the hyperparameter  $p_2$ . Let  $n$  be the total number of fitness evaluations. The procedure introduced in this paper can be summarized by Algorithm 2.

## 4 Experimental settings

This section describes the datasets considered for validating our technique (Sect. 4.1) and provides all the experimental settings (Sect. 4.2) to make the experiments completely reproducible. The code, for the complete reproducibility of the proposed experiments, is available at <https://github.com/gpietrop/GradientBasedGP>.

### 4.1 Dataset

To assess the validity of the technique proposed in Sect. 3.3, real-world, complex datasets, ranging from different areas, have been considered and tested. All of them have been widely used as benchmarks for GP, and their properties have been discussed in [27].

Table 1 summarizes the characteristics of the different datasets, such as the number of instances and variables. All of the proposed datasets but one deal with regression tasks. In fact, a classification problem is also considered to assert the validity of our method.

The objective of the first group of datasets is the prediction of pharmacokinetic parameters of potential new drugs. *Human oral bioavailability (%F)* measures the percentage of initial drug dose that effectively reaches the system blood circulation after passing through the liver; *Median lethal dose (LD50)*, also informally called *toxicity*, measures the lethal dose of a toxin, radiation, or pathogen required to kill half the members of a tested population after a specified test duration; *Protein-plasma binding level (%PPB)* corresponds to the percentage of the initial drug dose that reaches the blood circulation and binds the proteins of plasma. Also, datasets originating from physical problems are considered: *Yacht hydrodynamics (yac)* measures the hydrodynamic performance of sailing yachts starting from their dimension and velocity; *Concrete slump (slump)* measures the value of the slump flow of the concrete, that is influenced by the ingredients of the concrete itself; *Concrete compressive strength (conc)* measures values about the compressive strength of concrete (the most important material in civil engineering); *Airfoil self-noise (air)* is a NASA dataset obtained from a series of aerodynamic and acoustic test of two and three-dimensional airfoil blade sections, conducted in an anechoic wind tunnel. Lastly, the *Parkinson's disease dataset (park)*—which is the classification task dataset—collects biomedical voice measurements from a sample of individuals having (or not) Parkinson's disease.

## 4.2 Experimental study

For all the datasets described in Sect. 4.1, samples have been split among train and test sets: 70% of randomly selected data has been used as a training set, while the remaining 30% has been used as a test set. For each experiment, 30 runs have been performed, with a random train/test split in each run.

To assess the performance of the hybridization of GSGP and gradient-based optimizers, the results obtained within these methods are compared to the ones achieved with classical GSGP. The comparison with the performance achieved by standard GP

**Table 1** Main characteristics of the considered datasets: the number of variables, the number of instances, the domain, and the type of task

Dataset	Variables	Instances	Area	Task
%F	242	359	Pharmacokinetic	Regression
LD50	627	234	Pharmacokinetic	Regression
%PPB	627	131	Pharmacokinetic	Regression
Yac	7	308	Physics	Regression
Slump	10	102	Physics	Regression
Conc	9	1030	Physics	Regression
Air	6	1503	Physics	Regression
Park	19	5875	Biomedical	Classification

is not reported because, after some preliminary tests, it has been observed that standard GP is non-competitive against GSGP. To make the comparison fair, the total number of fitness evaluations must be equal for every method considered.

Recall that with the tuple  $(p_1, p_2)$ , we will indicate how we alternate steps of GSGP and the gradient-based optimizer. In this paper, we will always consider  $p_1 = 1$ , while  $p_2$  will take values in the range  $\{0, 1, 2, 5, 10\}$ . Setting  $p_2 = 0$  means that we are not performing any gradient-based optimization of the parameters, i.e., that we are considering vanilla GSGP.

Let  $n$  denote the total number of fitness evaluations we want to perform. Let  $n_1$  be the total number of GSGP steps performed during the training, and  $n_2$  the total number of gradient-based optimizer steps. The value of  $n_1$  and  $n_2$  will be determined by  $n$ ,  $p_1$ , and  $p_2$ , which are the three main hyperparameters of our method, as follows:

$$n_1 = \frac{n}{p_1 + p_2} p_1 \quad n_2 = \frac{n}{p_1 + p_2} p_2 \quad (4)$$

This allows a fair comparison between different techniques, ensuring that the number of fitness evaluations is the same for each choice of  $p_2$ . For example,  $(p_1, p_2) = (1, 1)$  means that we are alternating one step of GSGP with one step of the gradient-based optimizer, and we will perform—in total— $\frac{n}{2}$  steps of GSGP and  $\frac{n}{2}$  optimization steps.

Regarding the choice of the hyperparameter of the gradient-based optimizers, both for Adam and SGD, we will investigate the performance of the hybridization method proposed varying the learning rate  $\gamma$  over the following range  $\gamma \in \{0.1, 0.01, 0.001\}$ . The choice of exploring relatively higher learning rates is due to the small number of subsequent gradient descent iterations which are performed in the proposed methods (that are 1, 2, 5 or 10). In fact, higher learning rates can facilitate faster exploration of the solution space, where optimal solutions may reside.

In this experimental phase, we set  $n = 200$ . The population's size for all the considered systems is set to 50, and the individuals in the initial population are generated with the ramped half-and-half technique. Further details concerning the implementation of the semantic system and the gradient-based optimization algorithm are reported in Table 2. To maintain fairness and observe the method's performance in a general context, hyperparameters known to work well in classic GSGP were selected, and no dedicated tuning phase was conducted to avoid introducing biases in the comparison between the methods.

The considered fitness function is the root mean squared error (RMSE):

$$\text{RMSE} = \sqrt{\sum_{i=1}^n \frac{(\hat{y}_i - y_i)^2}{n}} \quad (5)$$

where  $y_i$  is the true value while  $\hat{y}_i$  is the predicted one.

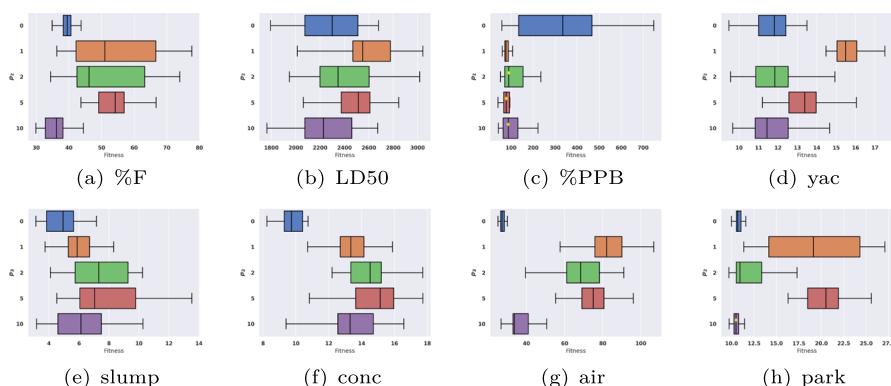
**Table 2** Experimental settings

Parameter	Value
Function set	$+, -, *, //$
Max. initial depth	6
Crossover rate	0.9
Mutation rate	0.3
Mutation step	0.1
Selection method	Tournament of size 4
Elitism	Best individuals survive
Learning rate ( $\gamma$ )	0.1, 0.01, 0.001
Exponential decay rate—first momentum ( $\beta_1$ )	0.9
Exponential decay rate—second momentum ( $\beta_2$ )	0.99
$\epsilon$	$10^{-8}$

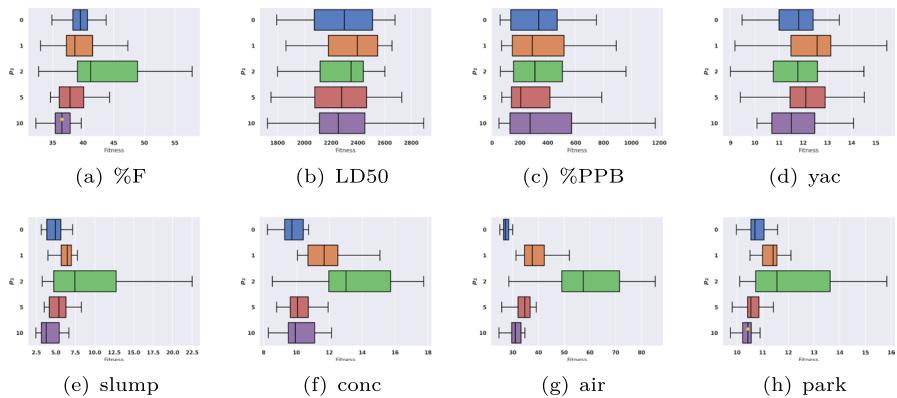
A horizontal line separates the parameters belonging to the GSGP algorithm and the ones of the gradient-based optimizers. The values for the learning rate are common for Adam and SGD, while other optimizer parameters are necessary only for the definition of the Adam algorithm

## 5 Experimental results

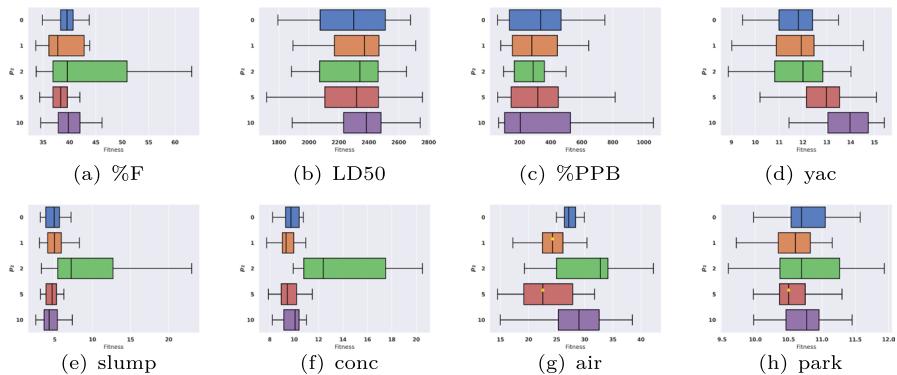
As stated in Sect. 3.3, this study aims to investigate the results of combining a gradient-based optimizer with GSGP. Figures 1, 2, and 3 show the box plots—obtained comparing the testing results over 30 independent runs—when GSGP is combined with SGD, with a learning rate value set to, respectively, 0.1, 0.01, and 0.001. Similarly, Figs. 4, 5, and 6 display the same study, but when the Adam optimizer is selected. Thus, each box plot allows comparing the fitness of standard GSGP and



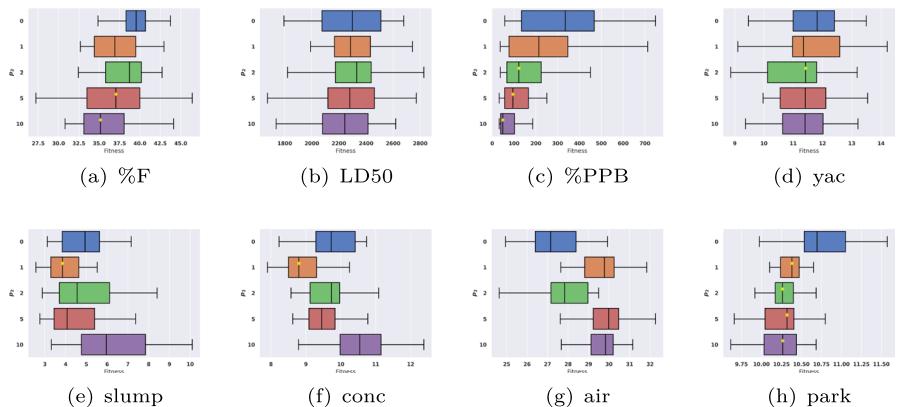
**Fig. 1** Boxplots of testing RMSE over 30 independent runs, with SGD and  $\gamma = 0.1$



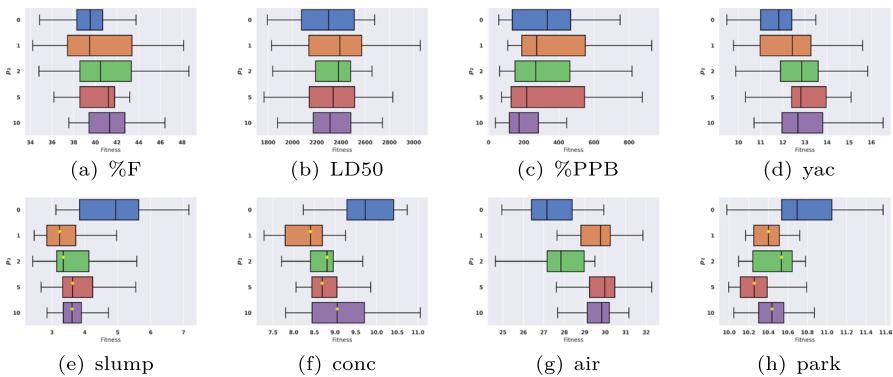
**Fig. 2** Boxplots of testing RMSE over 30 independent runs, with SGD and  $\gamma = 0.01$



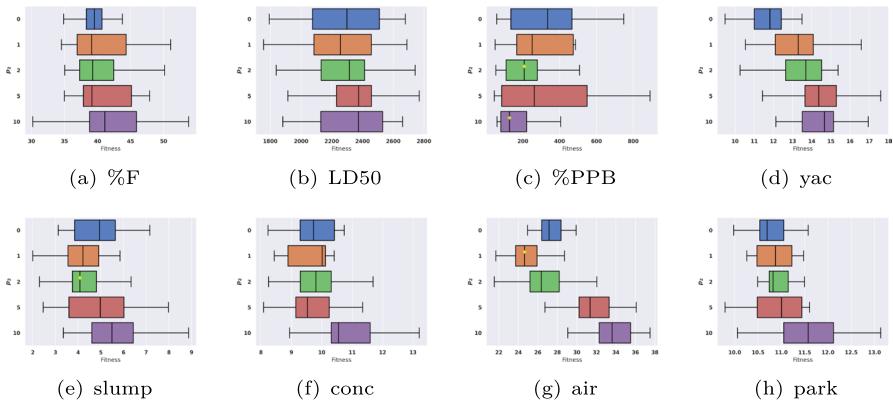
**Fig. 3** Boxplots of testing RMSE over 30 independent runs, with SGD and  $\gamma = 0.001$



**Fig. 4** Boxplots of testing RMSE over 30 independent runs, with Adam and  $\gamma = 0.1$



**Fig. 5** Boxplots of testing RMSE over 30 independent runs, with Adam and  $\gamma = 0.01$



**Fig. 6** Boxplots of testing RMSE over 30 independent runs, with Adam and  $\gamma = 0.001$

the one obtained by our algorithm over a different benchmark problem, considering different values assigned to  $p_2$ —the number of gradient-based optimizers steps performed after one step of GSGP.

To provide a statistical assessment of the obtained results, a yellow star was placed above the boxplots to denote the presence of statistically significant differences. The statistical analysis was performed through the Wilcoxon rank-sum test, with a significance level of  $\alpha = 0.05$ . The alternative hypothesis is that the distribution of the considered method is below the distribution of classic GSGP.

Moreover, we report the median fitness—of the 30 runs performed—over the total number of fitness evaluations for the test set, but only for the Adam optimizer. As we will explain later, we concentrate our discussion on the Adam optimizer considering the higher quality of the results obtained.

By observing the experimental results, it is possible to notice that the performance of the proposed algorithm is significantly influenced by the value of the learning rate. For example, the choice of SGD as a gradient-based optimizer

with  $\gamma = 0.1$  (Fig. 1) leads to results not satisfactory—with a few exceptions. This behavior is not surprising because a high learning rate value tends to result in instability also when used in the neural network framework [28]. On the contrary, considering SGD with  $\gamma = 0.001$  (Fig. 3)—which is the standard learning rate for this algorithm—results are comparable to (or outperforming) classic GSGP.

On the other hand, the integration of Adam optimizer in the GSGP framework shows a generally better performance—over all the benchmark problems considered—w.r.t. the use of classic SGD. These results confirm the necessity of choosing a more complex gradient-based optimizer (which leverages the first and second momentum for updating the parameters) to improve the algorithm's performance.

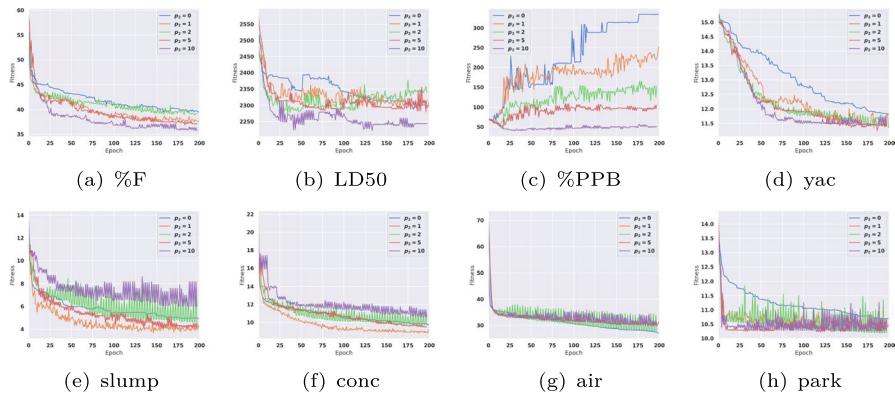
Adam is known to demonstrate advantages in convergence properties and robustness, particularly in complex and high-dimensional search spaces [29, 30], as the ones studied in this work. The adaptive learning rate mechanism and momentum in Adam provide it with the ability to dynamically adjust the step size during optimization, leading to faster convergence and improved performance. Moreover, Adam has been reported to demonstrate reduced oscillations during the optimization phase, contributing to its ability to achieve more stable convergence and avoid local minima [31].

The rest of the discussion will focus on the results obtained by the hybridization of GSGP within Adam.

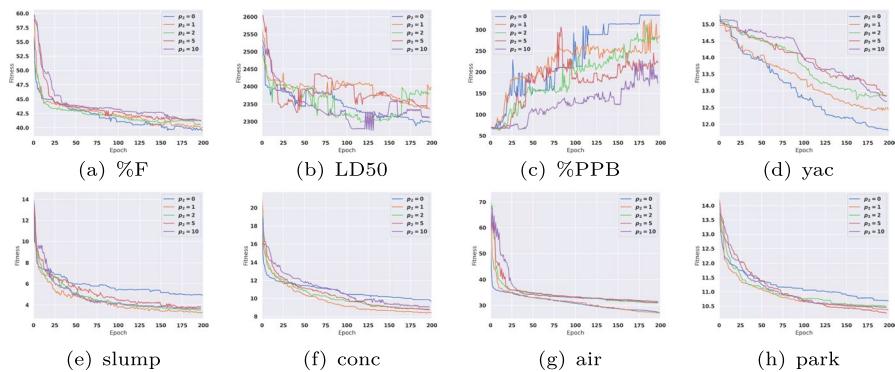
Concerning the best choice of the learning rate parameter  $\gamma$  for the Adam algorithm or the best value for the number of refinement steps  $p_2$ , experimental results do not provide a unique and universal answer. For example, let us focus on the results obtained with  $p_2 = 1$ . Considering Figs. 4, 5, and 6, we can observe that, for each problem, there is a learning rate value that makes the proposed method outperform standard GSGP. However, the best  $\gamma$  is not unique but depends on the specific benchmark problem under study. Specifically, for we improve the performance w.r.t. classic GSGP when **LD50** when  $\gamma = 0.1, 0.001$ , for **yac** when  $\gamma = 0.1$ , for **conc** when  $\gamma = 0.1, 0.01$ , for **air** when  $\gamma = 0.001$ , for **park** when  $\gamma = 0.01, 0.01$ . On the other hand, considering the **%F**, the **%PPB**, and the **slump** dataset, all the choices for  $\gamma$  lead to an improvement in terms of test fitness.

Again, these results are not surprising. In fact, the idea of applying a gradient-based optimizer is inspired by the neural network reality, where, typically, there is no better choice of parameters—such as learning rate—a priori. On the contrary, the most suitable value depends on the network's topology and the problem's complexity. Thus, it is plausible and reasonable that, in this neural network-inspired framework, different benchmark problems need different hyperparameters values to obtain a significant fitness improvement.

Concerning, on the other hand, the best choice for the number of gradient-based refinement  $p_2$  performed after a generation of GSGP, results indicate that  $p_2 = 1$  leads to better performance most of the times. Choosing  $p_2 = 1$  means that we are performing the same number of GSGP generations and Adam optimization. Thus, this combination reaches its maximum potential when takes full advantage of both methods in an equal way. This confirms the utility of hybridizing the two methods. In particular, the structural change of the individuals of GSGP and the ability of



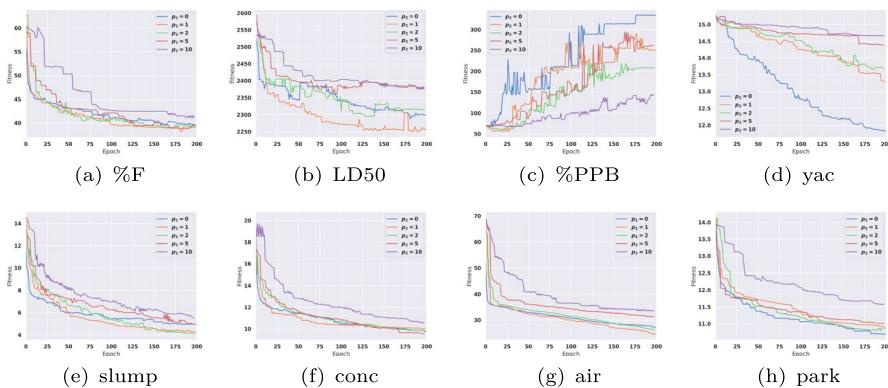
**Fig. 7** Median of the testing fitness over 30 independent runs, with Adam and  $\gamma = 0.1$



**Fig. 8** Median of the testing fitness over 30 independent runs, with Adam and  $\gamma = 0.01$

Adam to optimize slightly (but wisely) the parameters of the evolutionary process are both essential to achieving a good performance.

Furthermore, considering Figs. 7, 8, and 9, we can draw some conclusions on the stability and the rate of convergence. Firstly, Fig. 7 shows that despite the results obtained by our algorithm (which often outperform classic GSGP), there is a high level of instability, especially for higher values of  $p_2$ . This is caused (again) by the choice of  $\gamma = 0.1$  (a relatively high value for the learning rate), which tends to cause instability also in neural network applications. Another important conclusion we can draw is that once we find a good set of hyperparameter values for the proposed method, the convergence rate is way higher w.r.t. standard GSGP. In fact, most of the times, when our method outperforms GSGP, it reaches better fitness results after a few fitness evaluations, even if compared with the results obtained by GSGP at the last generation executed. Thus, we can conclude that using this hybridization—under the right choice of hyperparameters setting—allows for a reduction of the overall number of fitness evaluations.



**Fig. 9** Median of the testing fitness over 30 independent runs, with Adam and  $\gamma = 0.001$

Finally, one last important remark should be done about the increasing trend in fitness for the **%PPB** dataset, which can be attributed to the dataset's challenging nature, characterized by a propensity for overfitting, wherein the model achieves better performance on the training set but struggles to generalize effectively to the test set [21].

## 6 Conclusions

This paper investigates the results of the integration of gradient-based optimization methods, SGD and the Adam algorithm, within a genetic programming system, GSGP. The idea behind this work relies on the possibilities of exploiting and combining the advantages of these two methods to achieve faster convergence of the evolutionary search process.

Different ways of hybridizing these methods have been investigated, reported, and compared with vanilla GSGP on eight real-world, complex problems (concerning both the regression and the classification task) belonging to different applicative domains. Specifically, we considered two of the most famous optimizers in the neural network field (SGD and Adam) and studied their integration with GSGP assigning them different values to their learning rate—which is the most relevant parameter of both the SGD and Adam algorithm. Moreover, we compare the results obtained by our method by augmenting the number of gradient-based refinements after one generation of GSGP (performing a fair comparison by always taking into account the overall number of fitness evaluations as stopping criteria).

Experimental results show that, with the right choice of hyperparameters values, GSGP combined with the Adam algorithm outperforms classic GSGP. We also observe strong parallelism with the neural network framework, as the right choice of the optimizer and the learning rate drastically influence the results. Results confirm the necessity of choosing a more efficient optimizer—even though more expensive—to obtain better performance while avoiding instability

issues. As in deep learning, the optimal choice for these hyperparameters depends on the complexity of each benchmark problem under analysis.

We also observe that best results are achieved when we set  $p_2 = 1$ , thus when the contribution provided by GSGP and Adam refinement optimization is equal, confirming the utility of fully leveraging and combining the complementary strengths of these algorithms.

These results corroborate our hypothesis: the combination of GSGP with a gradient-based optimizer can improve the performance of GSGP. Moreover, this study highlights the importance of hybridizing the evolutionary process with other techniques of the artificial intelligence spectrum.

Even after this more detailed study of the integration of GSGP and gradient-based optimizer, multiple possible future developments focused on improving the benefits provided by this combination are still possible. It is interesting, as research directions, the idea of applying gradient-descent optimization not only for the parameters related to the GSC but to entire GP trees. Consider, for example, a normal functional  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ , like the sum. If the results of the subtrees on which  $f$  is applied are  $z_1, \dots, z_n$ , then the usual application is  $f(z_1, \dots, z_n)$ . But this can also be rewritten as  $f(\alpha_1 z_1, \dots, \alpha_n z_n)$  with all  $\alpha_i$  initially with value 1. The parameters  $\alpha_1, \dots, \alpha_n$  can then be optimized with methods like Adam or SGD, allowing us to perform parameter optimization on entire GP trees (non necessarily only for GSGP).

**Author contributions** All authors contributed to the study conception and design. Material preparation, data collection and analysis were performed by GP. The first draft of the manuscript was written by GP and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript

**Funding** Open access funding provided by Università degli Studi di Trieste within the CRUI-CARE Agreement. This work was supported by national funds through FCT (Fundação para a Ciência e a Tecnologia), under the Project—UIDB/04152/2020—Centro de Investigação em Gestão de Informação (MagIC)/NOVA IMS.

**Data availability** The datasets presented in this study can be found in online repositories. The names of the repository/repositories and accession number(s) can be found in the article.

## Declarations

**Conflict of interest** The authors have no relevant financial or non-financial interests to disclose.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

## References

1. J.R. Koza, J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, vol. 1 (MIT Press, Cambridge, 1992)
2. L. Vanneschi, M. Castelli, S. Silva, A survey of semantic methods in genetic programming. *Genet. Program Evol. Mach.* **15**(2), 195–214 (2014)
3. A. Moraglio, K. Krawiec, C.G. Johnson, Geometric semantic genetic programming, in *International Conference on Parallel Problem Solving from Nature* (Springer, 2012), pp. 21–31
4. L. Vanneschi, M. Castelli, L. Manzoni, S. Silva, A new implementation of geometric semantic GP and its application to problems in pharmacokinetics, in *European Conference on Genetic Programming* (Springer, 2013), pp. 205–216
5. G. Pietropolli, L. Manzoni, A. Paoletti, M. Castelli, Combining geometric semantic GP with gradient-descent optimization, in *European Conference on Genetic Programming (Part of EvoStar)* (Springer, 2022), pp. 19–33
6. X. Chen, Y.-S. Ong, M.-H. Lim, K.C. Tan, A multi-facet survey on memetic computation. *IEEE Trans. Evol. Comput.* **15**(5), 591–607 (2011)
7. F. Neri, C. Cotta, Memetic algorithms and memetic computing optimization: a literature review. *Swarm Evol. Comput.* **2**, 1–14 (2012)
8. M. Črepinský, S.-H. Liu, M. Mernik, Exploration and exploitation in evolutionary algorithms: a survey. *ACM Comput. Surv. (CSUR)* **45**(3), 1–33 (2013)
9. W. Smart, M. Zhang, Continuously evolving programs in genetic programming using gradient descent. Technical Report CS-TR-04-10, Computer Science, Victoria University of Wellington, New Zealand (2004)
10. L. Trujillo, O. Schütze, P. Legrand, Evaluating the effects of local search in genetic programming, in *EVOLVE-A Bridge Between Probability. Set Oriented Numerics, and Evolutionary Computation V* (Springer, Cham, 2014), pp. 213–228
11. P.T.H. Nguyen, D. Sudholt, Memetic algorithms outperform evolutionary algorithms in multimodal optimisation. *Artif. Intell.* **287**, 103345 (2020)
12. L. Trujillo, P.S. Juárez-Smith, P. Legrand, S. Silva, M. Castelli, L. Vanneschi, O. Schütze, L. Muñoz, Local search is underused in genetic programming, in *Genetic Programming Theory and Practice XIV* (Springer, Cham, 2018), pp. 119–137
13. B.E. Eskridge, D.F. Hougen, Imitating success: a memetic crossover operator for genetic programming, in *Proceedings of the 2004 Congress on Evolutionary Computation* (IEEE Cat. No. 04TH8753), vol. 1 (IEEE, 2004), pp. 809–815
14. P. Wang, K. Tang, E.P. Tsang, , X. Yao, A memetic genetic programming with decision tree-based local search for classification problems, in *2011 IEEE Congress of Evolutionary Computation (CEC)* (IEEE, 2011), pp. 917–924
15. L. Muñoz, L. Trujillo, S. Silva, M. Castelli, L. Vanneschi, Evolving multidimensional transformations for symbolic regression with M3GP. *Memet. Comput.* **11**(2), 111–126 (2019)
16. I. Kojadinovic, On the use of mutual information in data analysis: an overview, in *Proceedings of International Symposium Application of Stochastic Models and Data Analysis* (2005), pp. 738–47
17. A. Topchy, W.F. Punch, 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, 2001)
18. M. Zhang, W. Smart, Genetic programming with gradient descent search for multiclass object classification, in *European Conference on Genetic Programming* (Springer, 2004), pp. 399–408
19. M. Graff, R. Pena, A. Medina, Wind speed forecasting using genetic programming, in *2013 IEEE Congress on Evolutionary Computation* (IEEE, 2013), pp. 408–415
20. W. Smart, M. Zhang, Continuously evolving programs in genetic programming using gradient descent, in *Proceedings of the Second Asian-Pacific Workshop on Genetic Programming, Cairns, Australia* (2004), 16pp
21. M. Castelli, L. Trujillo, L. Vanneschi, S. Silva, E. Z-Flores, P. Legrand, Geometric semantic genetic programming with local search, in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation* (2015), pp. 999–1006
22. M. Castelli, L. Trujillo, L. Vanneschi, Energy consumption forecasting using semantic-based genetic programming with local search optimizer. *Comput. Intell. Neurosci.* (2015). <https://doi.org/10.1155/2015/971908>

23. P. Hajek, R. Henriques, M. Castelli, L. Vanneschi, Forecasting performance of regional innovation systems using semantic-based genetic programming with local search optimizer. *Comput. Oper. Res.* **106**, 179–190 (2019)
24. M. Castelli, S. Silva, L. Vanneschi, A C++ framework for geometric semantic genetic programming. *Genet. Program Evolvable Mach.* **16**(1), 73–81 (2015)
25. M. Castelli, L. Manzoni, GSGP-C++ 2.0: a geometric semantic genetic programming framework. *SoftwareX* **10**, 100313 (2019)
26. D.P. Kingma, J. Ba, Adam: A Method for Stochastic Optimization (2017)
27. J. McDermott, D.R. White, S. Luke, L. Manzoni, M. Castelli, L. Vanneschi, W. Jaskowski, K. Krawiec, R. Harper, K. De Jong, Genetic programming needs better benchmarks, in *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation* (2012), pp. 791–798
28. I. Kandel, M. Castelli, The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. *ICT Express* **6**(4), 312–315 (2020)
29. S.J. Reddi, S. Kale, S. Kumar, On the convergence of Adam and beyond. arXiv preprint [arXiv:1904.09237](https://arxiv.org/abs/1904.09237) (2019)
30. S. Ruder, An overview of gradient descent optimization algorithms. arXiv preprint [arXiv:1609.04747](https://arxiv.org/abs/1609.04747) (2016)
31. X. Chen, S. Liu, R. Sun, M. Hong, On the convergence of a class of Adam-type algorithms for non-convex optimization. arXiv preprint [arXiv:1808.02941](https://arxiv.org/abs/1808.02941) (2018)

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.