Memetic Semantic Boosting for Symbolic Regression

Alessandro Leite, Marc Schoenauer

Memetic Semantic Boosting for Symbolic Regression

Alessandro Leite[1,2*] and Marc Schoenauer[1]

[1]TAU, Inria Saclay, LISN, Paris-Saclay University, France
firstname.lastname@inria.fr .
[2]INSA Rouen Normandie, France.

Abstract

This paper introduces a novel approach called semantic boosting regression (SBR), leveraging the principles of boosting algorithms in symbolic regression using a Memetic Semantic GP for Symbolic Regression (MSGP) algorithm as weak learners. Memetic computation facilitates the integration of domain knowledge into a population-based approach, and semantic-based algorithms enhance local improvements to achieve targeted outputs. The fusion of memetic and semantic approaches allows us to augment the exploration and exploitation capabilities inherent in Genetic Programming (GP) and identify concise symbolic expressions that maintain interpretability without compromising the expressive power of symbolic regression. Our approach echoes the boosting algorithm's characteristic, where weak learners (e.g., MSGP) are sequentially improved upon, focusing on correcting previous errors and continuously enhancing overall performance. This iterative strategy, intrinsic to boosting methods, is adeptly adapted to our SBR model. Experimental results demonstrate that our memetic-semantic approach has equal or better performance when compared to state-of-the-art evolutionary-based techniques when addressing real-world symbolic regression challenges. This advancement helps tackle the bloating issue in GP and significantly improves generalization capabilities. However, akin to classic boosting algorithms, one limitation of our approach is the increased computational cost due to the sequential training of boosting learners.

Keywords: Genetic Programming, Gradient Boosting, Memetic Semantic, Symbolic Regression

1 Introduction

This paper addresses the challenge of finding a function $f(X) : \mathbb{R}^n \mapsto \mathbb{R}$ that represents the underlying relationship between the input features (X) and an output (y) of a given dataset through Genetic Programming (GP) [1]. GP has gained the attention of the machine learning (ML) community due to its capacity to learn both the model structure and its parameters without making assumptions about the data [2, 3]. Moreover, the symbolic aspects of its solutions and their flexible representation enable it to learn complex data relationships. These properties have made it a candidate solution to replace neural networks, which are usually considered a black-box solution and, consequently, hard to understand and explain.

Traditional GP-based methods rely on the outcome of a program to decide how well it solves a task, ignoring intermediate results such as the semantics of its subtrees [4, 5]. The semantics of an individual and its subtrees represent its behavior concerning a set of input values. Thus, one can consider them to guide the search during its exploration process and to favor short expressions that are usually easier for users to understand and analyze. Furthermore, semantics can contribute to improving subtrees' reuse based not only on the performance of the whole tree but also on their effectiveness in approximating a desired output. Semantic backpropagation (SB) algorithm [5, 6] has shown to be an effective strategy for dealing with such endeavors. Semantic backpropagation tries to find a set of subtrees that better approximate the desired outputs for a given node in a supervised setting. In other words, for each subtree of a tree, SB computes the desired semantics, which they should have to minimize the fitness of the tree, assuming all the rest is unchanged. Many works have used this strategy to design different operators in GP. Random Desired Operator (RDO) [5, 7] is one example. RDO randomly chooses a subtree in a tree. Then, it uses SB to compute the desired semantics of its parent. Finally, it replaces this subtree with one from a predefined library of trees that is mostly closest to the desired semantics. As this semantics replacement occurs locally, one can see the RDO operator as a first direction to define a memetic operator [6].

At the same time, small changes in a GP solution can dramatically change its fitness and, thus, harm the search efficiency. Memetic algorithms (MAs) [2] provide an effective way to compensate for the capability of global exploration of general evolutionary methods with the increased exploitation that can be obtained through local search. They combine population-based evolutionary algorithms and individual-based local search strategies. In this context, *this paper proposes a novel evolutionary multi-objective algorithm that combines both memetic and semantic backpropagation algorithms for symbolic regression problems.* We tackle the challenge of finding shorter expressions with a lower error, representing a way to tackle the accuracy vs. bloat trade-off of symbolic regression approaches. Bloat in this context can be seen as the excessive growth of a symbolic expression without improvement in its fitness [8].

Our proposed approach, named memetic semantic algorithm, iteratively fine-tunes a randomly created solution (i.e., a tree) by using the semantics of other small trees available in a pre-computed library. The library comprises a set of randomly generated trees up to a predefined height. Likewise, each library tree includes its outputs (i.e., fitness cases) and desired semantics vectors. This strategy helps us find shorter and

more diverse expressions. Thus, we decide to keep the fine-tuned individual at each iteration based on its performance in the validation set and size (i.e., number of terms in the expressions).

Different from traditional semantic backpropagation operators (e.g., RDO [5]) that selects a target subtree in a tree at randomly, our approach (Section 4.1) selects the best possible semantics match between all possible subtrees in the tree and all the trees in the library. Likewise, it only tunes the real-valued constants after a suitable tree has been found for the problem, and it computes them through linear scaling (LS) (i.e., regression) [9], and not randomly. Finally, it computes them at each iteration, as implemented by the RDO operator [5]. Linear scaling aims to minimize the mean squared error (MSE) of a tree by performing a linear transformation on its outputs [9, 10]. Consequently, it frees GP from the time-consuming task of adjusting the ephemeral constants, allowing it to focus exclusively on the tree's shape that fits the data structure without bothering to find a scale that approximates the target output. Last but not least, linear scaling helps in dealing with GP bloat problem [9, 10].

Based on our Memetic Semantic GP for Symbolic Regression (MSGP) method, this paper proposes semantic boosting regression algorithm (Section 4.2). It combines a set of MSGP learners trying to improve the generalization performance.

Experimental results on various real-world benchmark datasets show that our proposed methods can have equal or better performance compared to state-of-the-art (SOTA) non GP-based (e.g., Decision Tree and Random Forest [11]), GP-based methods (e.g., GP-GOMEA [12], `gplearn` [13]), and SyRBO [14], which is a GP-based boosting method.

The paper is organized as follows. In the next section, we present the background of the paper. Section 3 reviews the related works on managing bloat in GP, focusing on obtaining short and possibly interpretable symbolic expressions. The memetic algorithm and two memetic methods are presented in Section 4. Section 5 describe the experimental setup, and Section 6 analyzes and compares the performance of the proposed methods with standard GP, SOTA GP-based methods, classical machine learning methods, and boosting algorithms. Finally, Section 7 concludes the paper and describes the envisioned future works.

2 Background

2.1 Semantic GP

In GP, semantics describes the behavior of a program on a specific dataset. In other words, it is the output vector for the fitness cases of a problem [15]. More formally, in a supervised setting, assume the data is a set $\mathcal{D}$ made of N fitness cases: $\mathcal{D} = \{(X_1, y_1), (X_2, y_2), \ldots, (X_N, y_N)\}$, where $X_i \in \mathbb{R}^n$ and $y_i \in \mathbb{R}^1$ are the inputs, and the corresponding desired outputs. The semantics (s) of a program p is the vector of outputs values computed by p from the set of all fitness cases $\mathcal{D}$, defined as [5]:

$$s(p) = [p(X_1), p(X_2), \ldots, p(X_N)] \tag{1}$$

[1]We are focusing in this work on the specific case of symbolic regression (SR), but X_i and y_i could belong to some other spaces, for instance, discrete spaces in the case of classification or Boolean functions.

Similar operations can be performed for every subtree of a given tree: the semantics can be computed from the tree terminals up to the subtree root sequentially, defining the semantics for every node (i.e., subtree) of a GP tree.

Semantic backpropagation algorithms [5, 6] try to find the subtrees whose semantics better approximate the desired outputs (d_i^N) of a node $\mathcal{N} \in p$. A prerequisite is that one can compute the desired outputs for every node in p, conditional on the target output and the semantics of the other nodes in the program. This operation can be done downward from the root node (where the desired outputs are the target values y_i of the problem definition given in the initial dataset). For all the other nodes, this is done by performing the inverse operation of the function implemented in the node, assuming that the semantics of all other nodes are fixed: from the target values are the root, semantic backpropagation recursively computes the desired output for a node $\mathcal{N}$ at depth D as [5, 16]:

$$d_i^{\mathcal{N}} = F_{A_{D-1}}^{-1}(d^{A_{D-1}}, S_d) \tag{2}$$

where A represents the ancestor of $\mathcal{N}$ at depth D_i, S the siblings of A, and F^{-1} comprehends the inverse of the function implemented by node A.

It is fundamental to highlight the difference between the semantics of subtrees and the semantics of contexts. On the one hand, in the **semantics of subtrees**, the semantics of a node $\mathcal{N}$ only depends on its output for each fitness case, which means that if nodes $\mathcal{N}_1$ and $\mathcal{N}_2$ have the same semantics and a program p contains the former, replacing it with the latter will not change the semantics of p. On the other hand, in the **semantics of contexts**, given a node $\mathcal{N} \notin p$, it is usually hard to know how it will impact the semantics of the entire program (i.e., tree) since such information is conditioned to the semantics of the node that it will replace, as well as the semantics of its ancestors and siblings [4]. In some contexts, it can remain the same (i.e., a fixed context independent of the replaced node) or change (i.e., variable context). Consequently, the semantics of a node are uniquely defined by the function it implements and the value of its arguments, and they are independent of the position in the tree. In contrast, the contexts depend on the function implemented by the immediate parent, the parent semantics, and the semantics of the siblings [4]. As a result, local improvements may degrade the global performance.

2.1.1 Library building and searching

For the desired outputs of a given node, we want to search for a tree that better approximates these outputs than the current subtree. This can be achieved by building a library in a static or dynamic setting. In the static setting, the semantics of all possible subtrees with a maximum height are pre-computed, and redundant semantics are pruned to keep only one tree for each unique semantics. In the dynamic setting, also known as population-based, new trees are added based on the observed subtrees of every generation [5, 17]. This strategy keeps only the subtrees with the smallest number of nodes if different ones exist in the library with the same output. Moreover, both strategies ignore the subtrees with constant outputs.

Once a library has been built, the search process looks for the individuals whose outputs o are the closest to the desired semantics d_i^j based on some distance metric (e.g., Euclidean or Minkowski distance). It means, finding a minimal distance d_i^j that minimizes $|d_i^j - o_i|^k, \forall k \in \{1, 2, \ldots, N\}$ [5]. Additionally, if the distance value remains the same, whatever the subtree, its value is defined as zero (i.e., $|* - o_i|^k = 0$). Finally, as subtrees with constant outputs are ignored, the search process checks if a constant semantics could reduce the distance between the tree outputs and the desired ones.

2.2 Memetic algorithm

Memetic algorithms combine population-based search strategies with local search heuristics inspired by the concept in genetics [18]. They have been used across different domains due to their capacity to establish a good balance between exploration and exploitation when looking for a solution for a complex optimization problem [19, 20]. In this case, a meme represents transferable knowledge built through local refinement procedures, which one can see as domain-specific expert knowledge on how a solution can be improved [19]. From an optimization viewpoint, prototypical memetic algorithm comprises three main phases named creation, local improvement, and evolution (Algorithm 1)[2]. A population of randomly created individuals is set up in the creation phase. Then, each individual is locally improved up to a predefined level in the improvement phase. Finally, the evolution phase is the usual phase of evolutionary algorithms. It starts by selecting individuals based on their fitness and combining/mutating them through variation operators (e.g., crossover and mutation), enabling them to share information in a cooperative manner. The last two phases repeat until they meet a stopping criterion [19].

Algorithm 1 Memetic algorithm

```
create a population of individuals
repeat
    improve some or all individuals with some local search algorithm
    select, then combine and/or mutate the individuals
until stopping criteria
```

3 Related Work

Fighting the bloat has been considered an issue in GP since the early days of the field [8, 21–23]. However, with the advent of powerful machine learning techniques like deep learning (DL), the niche for GP has shifted from accuracy to interpretability. Moreover, because the interpretability is closely related to the length of the models, many works in the Genetic Programming aim to reduce the sizes of the obtained models, equivalent to some extreme bloat fighting.

[2]Though other types of hybridization between Evolutionary Computation (EC) and local search have been proposed, like using the local search as pre- or post-processor, as a mutation operator, among others that are beyond the focus of this work.

Several approaches have been proposed to fight bloat in Genetic Programming. For example, Bleuler et al. used SPEA2 to identify candidate solutions based on fitness and size. Experimental results showed that the proposed strategy could reduce GP bloat and speed up convergence. In [22], the authors proposed a pseudo-hill-climbing strategy to control the tree size during the crossover operation. In this case, the proposed approach discards an offspring if it either degrades the fitness or increases a tree's size. Although this approach can slow down GP bloating, it penalizes the running time of GP algorithms. In [25], the authors integrated a local search strategy into Geometric Semantic Genetic Programming (GSGP) to speed up convergence and limit overfitting. Experimental results showed that although the proposed strategy only required a few generations to find suitable candidate solutions, they were lost over the next generations, leading to a performance drop. Other works have proposed semantic-based operators to handle both bloat and generalization issues. Some examples include [5, 15, 26, 27] among others.

Uy et al. proposed a semantic-based crossover operators, named Semantic Aware Crossover (SAC) and Semantic Similarity-based Crossover (SSC). Their main difference is in the definition of the semantic distances, which, when exchanging two subtrees, must be different but not widely different. Similarly, Moraglio et al. suggested a semantic crossover operator that creates offspring with a weighted average of their parents' semantics. Notwithstanding, it cannot properly handle GP bloat without further simplification procedures. Geometric Semantic Crossover (AGX) [28] tries to handle these endeavors by replacing the ancestor's subtrees with those semantically close to their parents' midpoint semantic. Random Desired Operator (RDO) [5] introduces back-propagation. In this case, after a crossover operation, the semantics of the new subtree are back-propagated as described in Section 2.1. Different studies have shown that RDO outperforms the other operators on both regression [5] and Boolean [6] problems.

Nguyen and Chu [29] rely on semantic similarity for controlling bloat in GP. In this case, a random subtree of an individual is selected during crossover. Then, the first strategy replaces it with a small tree of approximate semantics, whereas the second one grows the tree to approximate the desired semantics of the selected subtree. The Gene-pool Optimal Mixing Evolutionary Algorithm (GP-GOMEA) [12] combines GP with linkage learning to learn a model of interdependencies and thus estimates the patterns to propagate that lead to small solutions. Finally, Sipper and Moore [14] proposes a Symbolic-regression boosting (SyRBO). SyRBO combines boosting with symbolic regression to improve the performance through a few boosting stages.

In this work, we rely on the idea of the RDO [5] to improve the fitness of a candidate solution. Likewise, a memetic algorithm selects the solutions based on their fitness on a validation set and size. Moreover, we use linear scaling [28] to compute the scale of the constants once a candidate solution has been found. Finally, we propose to combine memetic semantics and a boosting algorithm to improve the accuracy without penalizing the size of the expressions. It is different from SyRBO [14], which relies on standard GP and does not consider the size of the expressions. Nevertheless, we observe increased computational cost during the training phase. To the best of our

knowledge, this is the first work to propose a semantic boosting algorithm for symbolic regression problems.

4 Methods

This section describes how semantic backpropagation (SB) and memetic algorithm (MA) are combined to evolve GP models for symbolic regression (SR) problems that are interpretable and have a lower learning error.

4.1 Memetic Semantic for Symbolic Regression

Although semantic backpropagations and memetic algorithms have been separately used on SR problems, combining them can improve the interpretability and the generalization efficiency of GP-based model. While SB helps one in finding programs (i.e., trees) with the approximated desired output, MAs contribute to improving them by considering the semantics of their components (i.e., subtrees).

In a standard SB-based approach, once a subtree is selected to replace a node in a tree, it adds some constants to enable the tree to output the desired output. Consequently, as evolution proceeds, the trees often undergo excessive growth, known as bloat, which penalizes the search process, drastically increases the evaluation cost, and hinders the generalization of the trees (i.e., the accuracy on unseen data). We handle these issues by using LS [9] to search for constants that correct the residual errors of the tree. As a result, at each iteration, semantic backpropagation and memetic algorithm can concentrate on the structure of the tree, leaving the scaling of the coefficients to linear scaling.

Given a dataset composed of N independent samples (X_i) with m independent input variables ($X_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,m}]$) and a corresponding target output (y_i), the task of symbolic regression is to find a tree ($\mathcal{T}(.)$) that minimizes the distance between its outputs and the target output (y) [30, 31]. Such tree $\mathcal{T}(.)$ is built from a set of predefined functions and a set of terminals (a.k.a. the input variables and ephemeral constants).

For instance, using the mean squared error (MSE) as the distance metric (a.k.a. fitness function) for $\mathcal{T}(.)$, and denoting $\hat{y}$ the outputs of tree $\mathcal{T}$, the task of symbolic regression is to find a tree $\mathcal{T}(.)$ that minimizes $MSE(\mathcal{T})$ defined as:

$$MSE(\mathcal{T}) \equiv MSE(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{3}$$

4.1.1 Algorithm

The **semantic** part of the proposed Memetic Semantic GP for Symbolic Regression (MSGP) algorithm (Algorithm 2), maintains a tree $\mathcal{T}$ with a set of subtrees $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$, and it iteratively tries to improve its accuracy (MSE) using a predefined library $\mathcal{L}$ composed of l small subtrees. The goal of Algorithm 2 is to improve this main tree interactively $\mathcal{T}$ by checking for each subtree $s_i \in \mathcal{S}$ if there exists a subtree $s^\star \in \mathcal{L}$ whose semantics are closest to the desired ones for s_i. At each iteration, after

attempting to locally improve a tree (Line 4, MSGP calculates the constants of the improved tree (Line 5), and replaces the current tree with the new one (Line 6) when the latter demonstrates superior performance in terms of accuracy (MSE) and size in the validation set.

The starting tree $\mathcal{T}$ is usually created randomly. However, it can also be, for instance, the output of another GP-based SR approach to make it simpler and consequently more straightforward to understand by the users. In other words, MSGP does not make any assumption about the size or nature of the initial tree when it uses the library ($\mathcal{L}$) to search the nodes that can better replace a given subtree of the main tree. Consequently, the size and heterogeneity of the library $\mathcal{L}$ plays a vital role.

The **memetic** part of MSGP, linear scaling, computes a scaled version of the MSE [9] with a computing cost that is linear with the dataset size N, (i.e., $\mathcal{O}(n)$):

$$\mathrm{MSE}^{a,b}(y,\hat{y}) = \frac{1}{N}\sum_{i=1}^{n}(y_i - (a + b\hat{y_i}))^2 \tag{4}$$

With a and b defined as:

$$a = \bar{y} - b\bar{\hat{y}} \tag{5}$$

$$b = \sum_{i=1}^{N}\frac{(y_i - \bar{y_i})(\hat{y_i} - \bar{\hat{y}})}{(\hat{y_i} - \bar{\hat{y}})^2} \tag{6}$$

These coefficients a and b are then added to the final tree. Moreover, the algorithm ignores trees with constant outputs.

Algorithm 2 Memetic semantic for symbolic regression

Require: Initial tree ($\mathcal{T}$), library ($\mathcal{L}$)
Require: # *epochs*, and fitness cases ($\mathbf{X}, y$)

1: $\mathcal{T}' \leftarrow clone(\mathcal{T})$
2: Evaluate($\mathcal{T}'$)
3: **while** $e \leq epochs$ **do**
4: $\quad\mathcal{T}^\star \leftarrow lti(\mathcal{T}', \mathcal{L}, \mathbf{X}, y)$ ▷ local tree improvement (Algorithm 3)
5: $\quad\mathcal{T}^\star \leftarrow$LS($\mathcal{T}^\star, y$) ▷ linear scaling [9] computes the coefficients of $\mathcal{T}^\star$ (Equations (5) and (6))
6: $\quad\mathcal{T}' \leftarrow$ best($\mathcal{T}^\star, \mathcal{T}'$)
7: **return** $\mathcal{T}'$

4.1.2 Local tree improvement

Local Tree Improvement (LTI) algorithm (Algorithm 3) identifies the subtrees with equal or better semantics than a randomly selected subtree of the main tree. It is thus repeatedly applied to the target tree to improve its fitness gradually. It performs

an exhaustive search in the library $\mathcal{L}$, which contains a set of small trees with pre-computed semantics using the ancestor semantics of the target subtree. Hence, LTI algorithm finds a subtree $s^\star$ in $\mathcal{L}$ that realizes

$$\arg\min_{s^\star \in \mathcal{L}} \min_{t \in \{t_i, \ldots t_n\}} d(t, s(s^\star)) \tag{7}$$

where t is the desired semantics and $s(z)$ represents the semantics of subtrees $z \in \mathcal{L}$.

Consequently, similar to the SB algorithm, LTI requires (a) a fitness measure function and (b) a function to compute from the current context at each subtree and for each fitness case, the desired semantics which subtrees should have to enable the target tree evaluates to the expected semantics.

During the search process, the algorithm keeps track of the semantics distance between the already analyzed subtrees to avoid replacing them several times. The error function computes the distance between the semantics of the subtrees, which, in this case, comprises the semantics of the candidate subtree and the ancestor semantics of the selected subtree. If no local improvement is identified, the algorithm randomly replaces a subtree in $\mathcal{T}$ by one also randomly selected from the library, which, in this case, can be seen as a mutation operation. One can observe that local enhancement may degrade global criteria (e.g., accuracy, size, and generalization). Thus, it is up to the superior level to keep or ignore the newly proposed modified tree. Further work will evolve each proposed tree during a predefined number of generations and then crossover them using the LTI algorithm.

This work considers a static library composed of trees up to a certain height and with heterogeneous semantics. Only the smallest tree is included in the library when two candidates have the same semantics. Moreover, we also individually include the problems' variables and the operators (i.e., functions) into the library.

4.2 Memetic Semantic Boosting

We propose a boosting version of MSGP, called semantic boosting regression (SBR). Boosting algorithms comprise a class of ensemble machine learning techniques designed to improve the accuracy of "weak learners" when combined with a "strong learner". Weak learners are simple models with limited predictive power capacity. It means that their output hypotheses are only slightly better than those of random guessing. In contrast, strong learners are models with significant accuracy and robustness, which means they can, with high probability, output hypotheses that are correct in all but limited samples of the observations [32]. The essence of boosting methods relies on their iterative strategy. In this case, each subsequent model focuses on correcting the errors made by its predecessors, thus continuously improving the overall performance. This process continues until a predefined number of models are built. Boosting algorithms are popular in various application domains due to their effectiveness and adaptability in handling complex datasets. One drawback is the increased computing time to train the boosting learners sequentially.

In our case, the proposed SBR algorithm (Algorithm 4) includes a set of MSGP models (Algorithm 2) as weak learners. Hence, training a model comprises fitting

Algorithm 3 Local tree improvement

Require: Tree ($\mathcal{T}$), library ($\mathcal{L}$)

1: COMPUTE-SEMANTICS($\mathcal{T}$) ▷ $\forall$ node $N \in \mathcal{T}$
2: $S \leftarrow$ SUBTREES($\mathcal{T}$)
3: $\tau \leftarrow$ SORT(S) ▷ by error ascending and height descending
4: $s \leftarrow$ RANK-SELECT(τ)
5: $best[s] \leftarrow \emptyset$
6: $k \leftarrow \emptyset$
7: $min\_error \leftarrow$ ERROR(S)
8: **for all** $s^\star \in \mathcal{L}$ **do**
9: **if** $(s, s^\star) \in k$ **then**
10: continue
11: $e \leftarrow$ ERROR($s, s^\star$)
12: **if** $e < min\_error$ **then**
13: best[s] $\leftarrow (s^\star, 0)$
14: $min\_error \leftarrow e$
15: **else if** $e == min\_error$ **then**
16: best[s] $\leftarrow \cup \{(s^\star, e)\}$
17: **if** $|best[s]| > 0$ **then**
18: $k \leftarrow k \cup \{(s, s^\star)\}$
19: $\mathcal{T}^\star \leftarrow$ CROSSOVER(ancestor(s), $s^\star, \mathcal{T}$)
20: SEMANTICBACKPROGRATION(ancestor(s), $s^\star, \mathcal{T}^\star$)
21: **return** $\mathcal{T}^\star$
22: $s \leftarrow$ RANDOM(τ)
23: $s^\star \leftarrow$ RANDOM($\mathcal{L}$)
24: $\mathcal{T}^\star \leftarrow$ CROSSOVER(ancestor(s), $s^\star, \mathcal{T}$)
25: SEMANTICBACKPROGRATION(ancestor(s), $s^\star, \mathcal{T}^\star$)
26: $k \leftarrow k \cup \{(s, s^\star)\}$
27: **return** $\mathcal{T}^\star$

an MSGP model (Line 3) and updating the target values for the next stage to be the residuals of the current model (Line 4), as usually done by traditional gradient boosting algorithms. The output model is a large tree made of the sum of each individual learner tree (Line 5) after undergoing a global linear scaling phase (Line 6). The prediction phase then simply applies the trained model to the fitness cases.

Algorithm 4 Semantic Boosting Regression Training

Require: fitness cases $(\mathbf{X}, y)$
Require: stages ▷ number of boosting stages
Require: kwargs ▷ arguments of MSGP

boosters ← []
for $i \leftarrow 1$ to stages **do**
    boosters[i] ← MSGP(kwargs).train(X,y) ▷ includes "local" linear scaling
    y ← y - boosters[i].predict(X) ▷ updates the target values to the remaining residuals (Algorithm 2)
$\mathcal{T}$ ← join_trees(boosters) ▷ concatenates (i.e., sum) the trees of all boosters
$\mathcal{T}$ ←LS($\mathcal{T}$, y) ▷ computes the coefficients of the tree $\mathcal{T}$ (Equations (5) and (6))
return $\mathcal{T}$

5 Experimental settings

We evaluate the proposed semantic boosting regression algorithm on different real-world regression dataset benchmarks. The datasets have heterogeneous features and samples, as depicted in Table 1. Moreover, they are commonly used in the GP literature [12, 33, 34] as overfitting the training set occurs either when complex models are learned or when models are built using discontinuous functions. Furthermore, the Dow Chemical and Tower datasets are recommended benchmarks [35], taken from the UCI machine learning repository (*archive.ics.uci.edu*) and the repository (*shortest.link/8n9V*) provided by Martins et al. [33].

Preliminary experiments on a few datasets lead to consider for SBR the parameter settings given in Table 6 to define the library, the initial tree, and the number of iterations. Note that following [12, 36], we use analytical quotient (AQ) instead of protected division to avoid discontinuous behaviors but keep the general properties of the division. Likewise, the literature has shown that it helps generalize at prediction time [12, 36, 37]. It is defined as:

$$AQ(x_1, x_2) = \frac{x_1}{\sqrt{1 + x_2^2}} \tag{8}$$

As baselines, we consider both evolutionary and non-evolutionary algorithms: Decision Tree (DT) and Random Forest (RF) [11]) for non-evolutionary approaches, and GP-GOMEA [12] and *gplearn* [13] as the GP-based approaches. We use the *scikit-learn* [38] implementations of decision tree and random forests as they are commonly used in the GP and machine learning literature [14, 39, 40]: While decision trees are generally considered to be interpretable models, *random forests* are considered as black-box models. Nevertheless, the latter often outperforms the former. Consequently, they are more often employed by practitioners across different domains. We rely on the *scikit-learn* [41] implementation of these methods. We use grid search with cross-validation equal to five for fine-tuning their main parameters to minimize the impact of

experimental parameters on their performance and interpretability. Table 2 describes the range of values for each tuned parameter.

Table 3 describes the parameters settings for GP-GOMEA. They are the same as in [12] to avoid impacting its performance on the benchmark datasets. Similarly, for SyRBO, we follow the parameters' values defined in [14] and depicted in Table 4. SyRBO uses `gplearn` as the weak learners. We highlight that the SyRBO algorithm only exposes the parameters: number of stages, population size, and number of generations. All the other parameters described in Table 4 are the default values defined by `gplearn`, and we list them here for completeness reasons.

In this work, we also compare the performance of `gplearn` in the selected benchmarks, using its default parameters, as illustrated in Table 5. Finally, we include the XGBoost [42] algorithm as another boosting machine learning method, using its Python implementation version with its default parameters.

In this work, each experiment comprised 30 independent runs. For all datasets, the fitness was evaluated such that lower values represent the best fitness, and the median of the results are reported. MSGP was implemented in `python` using the `DEAP` library [43], and we performed the experiments on a MacBook Pro with one Apple M1 processor (8 cores) and 16 GB of RAM memory. Table 6 describes the parameters of MSGP. An initial tree was created with the Ramped Half-and-Half (H&H) method [1] with a maximum depth of 4. We consider a library with 200 subtrees also initialized with the Ramped (H&H) approach with a height between 2 and 4. The terminal set includes all the covariates of the dataset, and we consider a maximum of 10,000 iterations (i.e., epochs) in which each candidate tree is tuned using the training set and evaluated on the validation set.

We use the Wilcoxon signed-rank test [44] to assess the statistical significance of the null hypothesis of no difference when comparing the performance of two methods on a given dataset. This statistical test is set up to compare competing methods based on the same prior conditions. In particular, we employ pairs of executions where a dataset is split into identical training, validation, and test sizes for the tested approaches. We consider a difference significant if a smaller p-value than $0.05/\beta$ is found, with β being the Bonferonni correction coefficient used to prevent false positives.

Table 1: Regression datasets benchmarks considered by this work

Name	Acronym	# Features	# Samples
Airfoil	AF	5	1503
Boston housing	BH	13	506
Concrete compressing strength	CCS	8	1030
Dow Chemical	DC	57	1066
Energy cooling	EC	8	768
Energy heating	EH	8	768
Tower	TW	25	4999
Wine red	WR	11	1599
Wine white	WW	11	4898
Yacht hydrodynamics	YH	6	308

Table 2: Range of values of the grid search for decision tree (DT) and random forest (RF) algorithms

	Parameter	Values
DT & RF	max_depth	$[3, 5, 7, 10]$
	min_samples split	$[2, 5, 7, 10]$
	min_samples leaf	$[1, 2, 3, 4, 5, 6]$
RF	n_estimators	$[10, 20, 50, 100]$

Table 3: GP-GOMEA parameters

Parameter	Value
Function set	$\{+, -, \times, \div(AQ)\}$
Terminal set	$\boldsymbol{x} \cup \{\text{ERC}\}$
ERC bounds	$[\min \boldsymbol{x}, \max \boldsymbol{x}]$
Initialization for GP-GOMEA	Half-and-Half
Tree height	4
Initial population size	64
Batch size	256
Linear scaling	True
Max number of generations	-1
Max number of evaluations	-1
Train-validation-test-split	50%-25%-25%
Interleaved multi-start scheme	4:1

Table 4: SyRBO parameters

Parameter	Value
Function set	$\{+, -, \times, \div, \sqrt{}, \log, \lvert x\rvert, \max, \min, \frac{1}{x}\}$
Terminal set	$\boldsymbol{x} \cup \{\text{ERC}\}$
Population size	200
Number of generations	200
Initial depth	[1, 6]
# Trials	30
Loss function	MSE
Tournament size	20
Initialization	Ramped H&H [1-2]
Train-test-split	70%-30%

Table 5: gplearn parameters

Parameter	Value
Function set	$\{+, -, \times, \div\}$
Terminal set	$\boldsymbol{x} \cup \{\text{ERC}\}$
Population size	1000
Number of generations	20
Initial depth	[1, 6]
# Trials	30
Loss function	MSE
Tournament size	20
Initialization	Ramped H&H [1-2]
Train-test-split	70%-30%

Table 6: MSGP parameters

Parameter	Value
Function set	$\{+, -, \times, \div(AQ)\}$
Terminal set	Features
Initial tree height	2
# epochs	$1e4$
Max time	500 seconds
# Trials	30
Loss function	MSE
Library size	200
Initialization	Ramped H&H [1-2]
Train-validation-test-split	50%-25%-25%

Table 7: SBR parameter settings

Parameter	Value
Function set	$\{+, -, \times, \div(AQ)\}$
Terminal set	Problem's variables
Initial tree height	2
# epochs	$6e3$
Max time	300 seconds
# Trials	30
Loss function	MSE
Library size	300
Initialization	Ramped H&H [1-2]
Train-validation-test-split	50%-25%-25%

6 Results

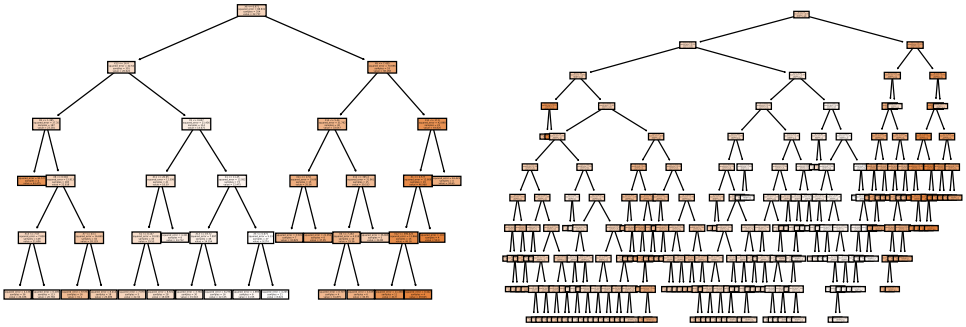
Table 8 shows the median error on the testing set for each of the considered methods. The experimental results confirm that XGBoost [42] and Random Forest (RF) [11] outperform evolutionary-based methods and also decision tree for almost all

the benchmark datasets. The exceptions include the results of the dow chemical (DC), wine red (WR), and yacht hydrodynamics datasets, in which our methods MSGP and SBR achieve similar performance (e.g., DC and WR). Although XGBoost and RF achieve high accuracy, they sacrifice the intrinsic interpretability of a single decision tree. For example, while it may be easy to follow the path taken by a decision tree to make a decision, it may be much more challenging to follow the paths of dozens or even hundreds of trees. For example, Section 6 illustrate the decision tree for the Boston housing dataset and one of the trees of RF. It is essential to highlight that both approaches were fine-tuned to balance the interpretability and accuracy trade-off, as described in Section 5. Moreover, a decision tree requires users to translate the decision paths into some if-then rules, which represents a workload when compared to a symbolic expression outputted by an evolutionary method such as GP-GOMEA, MSGP, and SBR, as we can see in Tables 9 and 10. Nonetheless, MSGP and SBR have a high computational training cost when compared to classical machine learning methods such as Random Forest and XGBoost due to their need to evaluate the different candidate trees during the refinement of the target tree. The experiments

Table 8: Performance on the test set for each benchmark dataset. **Boldface** highlights the best result in each row, while underlines indicate superior performance compared to the corresponding algorithm types. XGBoost demonstrates overall superior performance across datasets. Our boosting method (SBR) achieves comparable or better performance than evolutionary-based (EA) methods, without penalizing the size of the discovered symbolic expressions, as detailed in Table 10.

						Boosting methods				
						EA-based methods				
	Non EA		**EA**			**SBR**		**SyRBO**		
DS	**DT**	**RF**	**gplearn**	**GP-GOMEA**	**MSGP**	**5**	**10**	**5**	**10**	**XGB**
AF	8.7	4.3	607.14	33.3	20.0	19.3	17.81	93.49	59.66	**2.61**
BH	21.63	12.9	51.05	20.04	26.2	23.09	20.24	120.55	144.89	**11.2**
CCS	50.91	28.4	173.55	98.26	87.8	74.14	61.2	73.54	66.02	**23.1**
DC	0.03	**0.02**	0.14	0.08	0.05	0.04	0.03	0.12	0.12	**0.02**
EC	3.72	3.19	17.07	12.25	10.89	10.78	10.4	14.49	14.5	**0.77**
EH	0.33	0.29	14.98	10.92	8.97	8.12	7.33	9.56	8.15	**0.14**
TW	488	269	4451	1714	1577	1473	1265	2459	31596	**260**
WR	0.48	0.36	0.53	0.62	0.42	0.42	0.41	0.49	0.48	**0.39**
WW	0.55	0.43	0.69	0.69	0.55	0.55	0.55	0.71	0.78	**0.42**
YH	1.83	1.44	40.85	**0.58**	12.1	54.08	52.21	16.61	13.39	0.95

reveal that MSGP and SBR have a performance comparable to the one obtained by GP-GOMEA, and that they outperform `gplearn` and SyRBO. Moreover, Figure 2 shows that over the trials, the expressions' size and error stay concentrated around small values for the different datasets. Such a result confirms that our algorithms can improve the fitness of a tree while keeping it smaller. Such behavior allows users to pick a tree that includes some features—for example, a tree with features invoking causal relationships or any other feature of interest. In an ideal setting, one would



(a) Decision tree generated by a decision tree algorithm (b) A single tree from a random forest model

Fig. 1: Examples of decision trees produced by the decision tree and random forest algorithms on the Boston housing dataset

expect to see most points concentrated around the bottom left of the plot, representing small trees with a minimal training error.

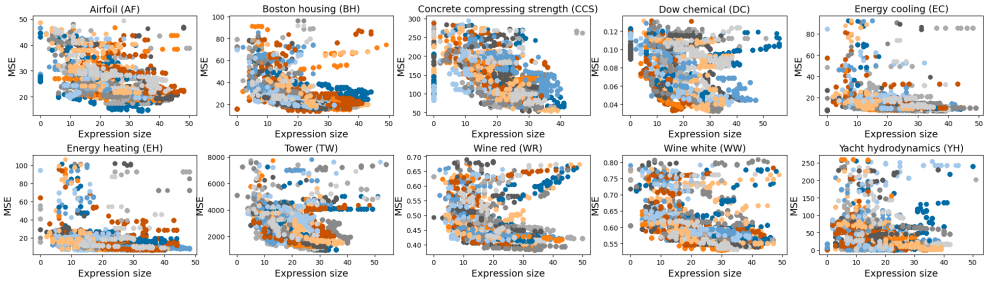


Fig. 2: Fitness versus MSGP expression sizes across all the benchmarks. Each dot represents a trial, with different colors distinguishing the trials.

Figure 3 illustrates the performance of the best solutions over the 30 runs on the training, validation, and test sets. We can note a small variance across the different runs. Moreover, Figure 4 shows that MSGP requires only a few iterations to fine-tune a target tree for an acceptable performance.

Figure 5 show that SBR algorithm outperforms MSGP and SyRBO except for the yacht hydrodynamics dataset for five and ten stages. As SyRBO solely focuses only on performance, we observe outputted expressions with hundreds and even thousands of terms, as shown in Figure 6a. It may be due to its reliance on `gplearn` as the weak learner. Contrarily, Figure 6b shows that SBR can output short symbolic expressions.

Table 9: Examples of the best symbolic expressions find by MSGP (Section 4.1) and GP-GOMEA [12]

DS	MSGP	GP-GOMEA
AF	$-55.66*X2*X3*(X4-X5)*(X1*X5+X2*X3-X2-X3+2*X5)\div(X4*X5*(X1*X5+X4+X5\div X4)*(X2*X3-X2-X4))+131.67$	$(108.14*X0*X3-25.89*X1-1055.88*X2+25.89*X3)\div(X0*X3)$
BH	$54.67-2.70*(X11+X5)*(X13+X8)\div X11$	$1.0*(10.59*X10*X9-2.11*X12*X9-92.10*X5)\div(X9*(X10+X12))$
CCS	$4.63+9.25*(X1+X5)*(X4*X8\div X6+X3\div(X2*X4*X8))*(X2+X4+X5+X6)\div(X4*(X4\div X1+X4\div(X1*X2*X8))*(X4-X5+X6+X8))$	$(-152.56*X0-152.56*X1-152.56*X2-190.26*(X2+3.94)*(X3-X7)-199.95)\div((X2+3.94)*(X3-X7))$
DW	$-6623.32*X35*X6\div(X37*X9*(-X14+X31)*(X19-X32))+6.75$	$-1710.01*X15-1710.01*X18-1710.01*X20+1710.01*X48+1710.01*(X20-X34)*(X23+0.13)+2.94$
EC	$6.18*X5*(X3\div X7+X6*X8)\div((X1-X7)*(-X2+X6))-1.17$	$(X2*(-37894.76*X0+13312.61*X2*X4-267.99)+13312.61*X7*(X6+0.02))\div X2$
EH	$11.06*X5\div(X1+X1\div X7-X7)-12.86$	$(X1*(-26642.41*X0+16280.46*X6-97.05)+3492.47*X4+102.44)\div X1$
TW	$-13.77*X1-13.77*X21*(X1+X4)\div(-X12+X15+X23)+13.77*X6+659.31$	$-12927.68*(-2*X0+X22+X5)*(X0-X10-X14+X8)-1210.43$

Table 10: Examples of the best expressions found by SBR for each benchmark dataset.

DS	5 stages	10 stages
AF	$-0.006411*X1*X3-0.006411*X1*X5+0.006411*X2-0.006411*X4+127.71$	$-0.006711*X1*X3-0.006711*X1*X5-0.006711*X2-0.013422*X3\div X4+0.006711*X4+127.257606-0.006711*X2\div X1-0.013422*X3\div X1$
BH	$5.17*X1*X4+10.86+5.17*X6\div X13+5.17*X6\div X11+5.17*X12\div X10$	$-0.14*X1-0.14*X11\div X4+0.14*X13-0.14*X3-0.14*X5\div X9-0.14*X8+26.14-0.14*X6\div X3-0.14*X6\div X11-0.14*X13\div X10$
CCS	$0.026*X1\div X4+0.026*X2+0.026*X4\div X8+0.026*X5*X8+0.026*X8+25.40+0.026*X8\div X5$	$0.021*X1-0.01*X2+0.01*X3\div X5+0.01*X4+0.02*X4\div X8+0.03*X5*X8+21.17$
DW	$-0.01*X16+0.01*X29+0.01*X30\div X4+0.01*X49+0.017203*X55-7.772227+0.01*X55\div X35$	$-9.3e-5*X16\div X7-9.3e-5*X18-9.3e-5*X18\div X19-9.3e-5*X29-9.3e-5*X32-9.3e-5*X35-9.3e-5*X36-9.3e-5*X36\div X53-9.3e-5*X49+3.16-9.3e-5*X9\div X37-9.3e-5*X5\div X30-9.3e-5*X47\div X16$
EC	$1.55*X1*X2+1.55*X1\div X3-762.86+1.55*X5\div X1$	$0.10*X1*X2+0.15*X1\div X3+0.05*X3*X7+0.05*X4*X5+0.05*X4*X7-80.28+0.05*X5\div X1$
EH	$-0.12*X1*X2-0.24*X1\div X5+83.96-0.12*X7\div X4$	$0.19*X1*X2+0.19*X1\div X3+0.09*X2\div X4+0.09*X3*X7+0.09*X5+0.09*X7-87.27+0.09*X7\div X5$
TW	$-5.86*X1-5.86*X1\div X23-5.86*X2\div X25-5.86*X23\div X5+5.86*X6+27.51-5.86*X6\div X4$	$-2.54*X1+2.54*X1\div X3-2.54*X13+2.54*X15+2.54*X24+2.54*X4\div X7+2.54*X5+2.54*X8-76.27+2.54*X8\div X22+2.54*X16\div X13+2.54*X2\div X1+2.54*X6\div X1$
WR	$0.36*X10*X5+0.36*X11\div X2+0.36*X2\div X4+2.28$	$-0.007*X10*X5-0.003*X11-0.003*X11\div X9-0.007*X3-0.003*X7*X8-0.003*X9+5.90-0.003*X9\div X2$
WW	$0.01*X1\div X4+0.01*X11**2+0.02*X2\div X6+4.39+0.01*X2\div X11$	$0.11*X10+0.11*X11+0.11*X2+0.11*X2\div X4+0.11*X4\div X7+0.11*X5*X9+0.11*X9+3.76+0.11*X9\div X6+0.11*X2\div X11+0.11*X6\div X11+0.11*X2\div X1$
YT	$85.80*X6**2+86.77$	$18.12*X2\div X3+72.49*X6^2+67.801051$

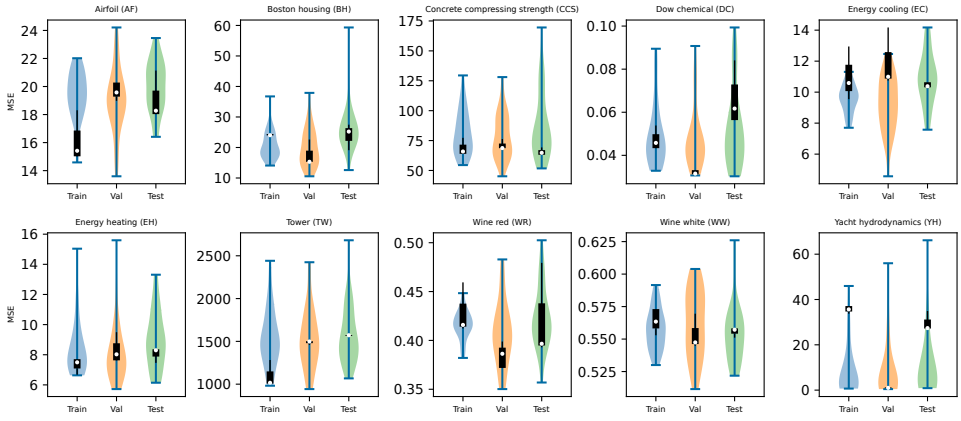


Fig. 3: Accuracy of the best solution across 30 runs on the training, validation, and test sets for MSGP. White dots indicate the median and the bars represent the first and third quartiles.

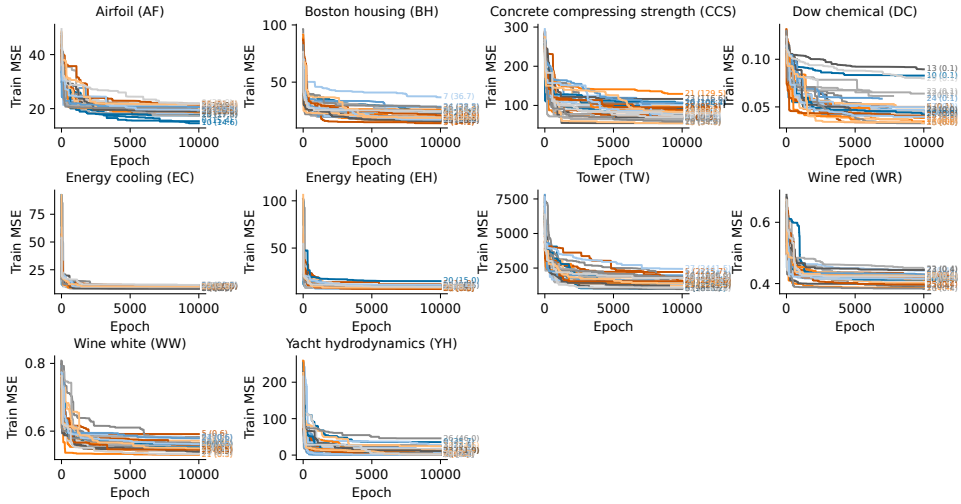


Fig. 4: Performance of the best MSGP trees during training across 30 trials for each dataset. Each line represents a trial, with different colors distinguishing the trials. The MSGP method requires a few iterations to identify a suitable tree. The y-axis represents the MSE during training, and the x-axis represents the epochs (i.e., iterations of the MSGP algorithm).

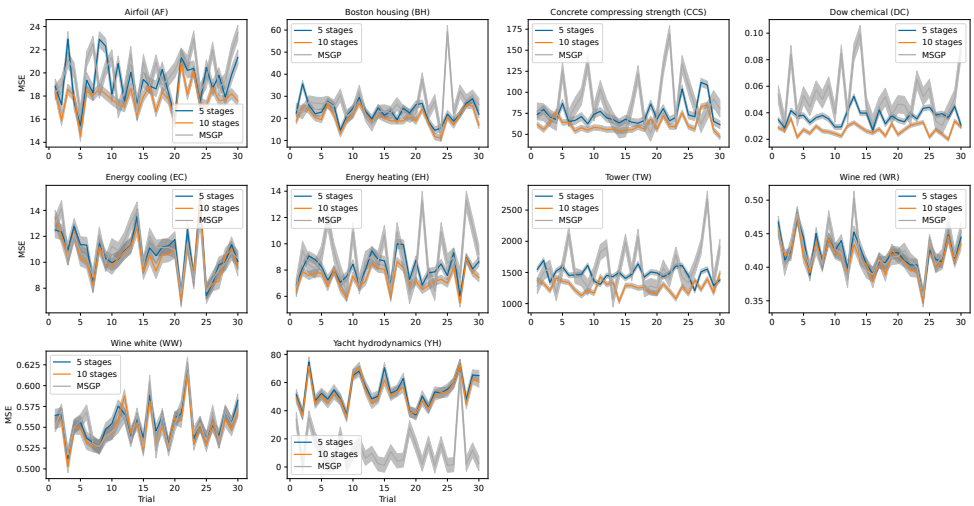
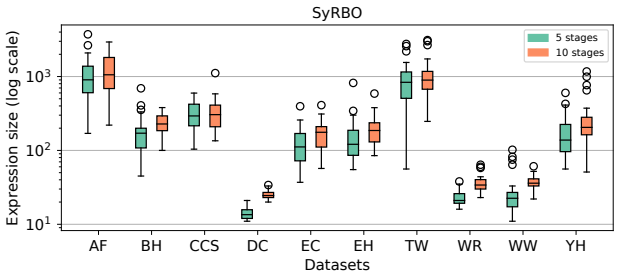
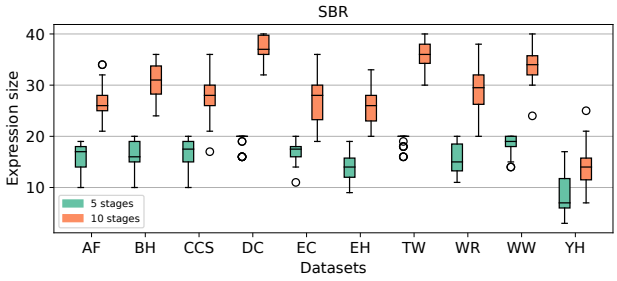


Fig. 5: Accuracy of semantic boosting regression on the test set across 30 independent runs, using one MSGP, five, and ten boosters for each benchmark dataset. The shaded area represents the 95% confidence interval.



(a) SyRBO expression sizes (y-axis in logarithmic scale)



(b) SBR expression sizes

Fig. 6: Sizes of expressions outputted by SBR and SyRBO for each benchmark dataset.

7 Conclusion

Symbolic regression (SR) searches for a set of mathematical expressions that better approximate a target variable of a given dataset. It is commonly implemented through Genetic Programming due to its characteristics in exploring the search space free of constraints' assumptions about the underlying data distribution. Nevertheless, GP-based approaches still face the challenge of overfitting the data expressed through complex symbolic expressions (a.k.a. bloat). Semantic-based strategies [5, 28] have been seen as a way to handle this issue. This paper proposes and evaluates a memetic semantic boosting algorithm for symbolic regression in this context. The memetic semantic algorithm combines a population-based search strategy with semantics-guided ones to output short symbolic expressions without penalizing the accuracy. In this case, rather than selecting a subtree in a target tree at random, as semantic-based operators [5, 7] usually do, our method repeatedly selects the closest semantics match between all possible nodes in the tree and all trees in a pre-computed library. Our semantic boosting-style algorithm integrates a set of Memetic Semantic GP for Symbolic Regression (MSGP) algorithms as weak learners. Consequently, the final model comprises a tree compound of the individual learner tree. Experimental results show that the proposed algorithm can achieve equal or better performance when compared to state-of-the-art evolutionary-based and traditional machine learning methods on various real-world datasets. Notwithstanding, our methods are restricted to tabular and continuous data. Moreover, the size of the predefined library and the length of its subtrees may also penalize the computational cost of our proposed methods. Furthermore, one might observe a high training time in a big data setting compared to the training time of classical boosting algorithms such as XGBoost [42].

Future works include investigating the performance when using a dynamic library, new strategies to select an individual in a library, and new strategies to guide the construction of the semantics library to enhance the exploration and exploitation features of memetic semantic-based algorithms. We are also interested in integrating the Operator Equalisation [8, 45, 46] into our memetic semantic-based approaches as a new strategy to speed up discovering short symbolic expressions. Likewise, we plan to evaluate the performance of our proposed approach on reinforcement learning (RL) problems, such as done in [47]. Finally, we want to apply our methods to different real-world applications, including temporal data.

Declarations

Ethical statements

The authors have no competing interests to declare relevant to this article's content.

Authors' contributions

A.L. wrote the manuscript, prepared the figures, and wrote the code accompanying the manuscript. All authors reviewed the manuscript.

Acknowledgements

The authors thank the European Commission for partially funding this work through the TRUST-AI project. We also sincerely thank the reviewers for their insightful and constructive feedback, which has significantly contributed to improving the quality of this paper.

Funding information

The European Commission partially funded this research within the HORIZON program (TRUST-AI Project, Contract No. 952060).

Data availability

The data utilized in this paper were sourced from two repositories: the UCI Machine Learning Repository (*archive.ics.uci.edu*) and the repository (*bit.ly/47wQHgO*) provided by Martins et al. [33]

References

[1] Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Evolution. MIT Press, Massachusetts (1992)

[2] Ong, Y.-S., Lim, M.-H., Neri, F., Ishibuchi, H.: Special issue on emerging trends in soft computing: memetic algorithms. Soft Computing **13**(8), 739–740 (2009)

[3] Udrescu, S.-M., Tegmark, M.: AI Feynman: A physics-inspired method for symbolic regression. Science Advances **6**(16), 2631 (2020)

[4] McPhee, N.F., Ohs, B., Hutchison, T.: Semantic building blocks in genetic programming. In: European Conference on Genetic Programming, pp. 134–145 (2008)

[5] Pawlak, T.P., Wieloch, B., Krawiec, K.: Semantic backpropagation for designing search operators in genetic programming. IEEE Transactions on Evolutionary Computation **19**(3), 326–340 (2014)

[6] Ffrancon, R., Schoenauer, M.: Memetic semantic genetic programming. In: Annual Conference on Genetic and Evolutionary Computation, pp. 1023–1030 (2015)

[7] Nguyen, Q.U., Pham, T.A., Nguyen, X.H., McDermott, J.: Subtree semantic geometric crossover for genetic programming. Genetic Programming and Evolvable Machines **17**, 25–53 (2016)

[8] Silva, S., Dignum, S., Vanneschi, L.: Operator equalisation for bloat free genetic programming and a survey of bloat control methods. Genetic Programming and Evolvable Machines **13**, 197–238 (2012)

[9] Keijzer, M.: Improving symbolic regression with interval arithmetic and linear scaling. In: European Conference on Genetic Programming, pp. 70–82 (2003)

[10] Keijzer, M.: Scaled symbolic regression. Genetic Programming and Evolvable Machines **5**(3), 259–269 (2004)

[11] Breiman, L.: Random Forests. Machine Learning **45**(1), 5–32 (2001)

[12] Virgolin, M., Alderliesten, T., Witteveen, C., Bosman, P.A.: Improving model-based genetic programming for symbolic regression of small expressions. Evolutionary computation **29**(2), 211–237 (2021)

[13] Stephens, T.: Genetic programming in python with a scikit-learn inspired API: gplearn. github.com/trevorstephens/gplearn (2016)

[14] Sipper, M., Moore, J.H.: Symbolic-regression boosting. Genetic Programming and Evolvable Machines **22**, 357–381 (2021)

[15] Moraglio, A., Krawiec, K., Johnson, C.G.: Geometric semantic genetic programming. In: International Conference on Parallel Problem Solving from Nature, pp. 21–31 (2012)

[16] Krawiec, K.: Semantic genetic programming. In: Behavioral Program Synthesis with Genetic Programming, pp. 55–66 (2016)

[17] Chen, Q., Zhang, M., Xue, B.: Geometric semantic genetic programming with perpendicular crossover and random segment mutation for symbolic regression. In: Asia-Pacific Conference on Simulated Evolution and Learning, pp. 422–434 (2017)

[18] Dawkins, R.: The Selfish Gene. Oxford University Press, Oxford, U.K. (1976)

[19] Moscato, P.: On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report 826, Caltech Concurrent Computation Program, California Institute of Technology (1989)

[20] Chen, X., Ong, Y.-S., Lim, M.-H., Tan, K.C.: A multi-facet survey on memetic computation. IEEE Transactions on Evolutionary Computation **15**(5), 591–607 (2011)

[21] Langdon, W.B., Poli, R.: Fitness causes bloat. In: Second On-line World Conference on Soft Computing in Engineering Design and Manufacturing, pp. 13–22 (1997)

[22] Langdon, W.B., Poli, R.: Genetic programming bloat with dynamic fitness. In: 1st European Workshop on Genetic Programming, pp. 97–112 (1998)

[23] Langdon, W.B., Poli, R.: Fitness causes bloat: Mutation. In: 1st European Workshop on Genetic Programming, pp. 37–42 (1998)

[24] Bleuler, S., Brack, M., Thiele, L., Zitzler, E.: Multiobjective genetic programming: Reducing bloat using SPEA2. In: Congress on Evolutionary Computation, vol. 1, pp. 536–543 (2001)

[25] Castelli, M., Trujillo, L., Vanneschi, L., Silva, S., Z-Flores, E., Legrand, P.: Geometric semantic genetic programming with local search. In: Annual Conference on Genetic and Evolutionary Computation, pp. 999–1006 (2015)

[26] Krawiec, K., Lichocki, P.: Approximating geometric crossover in semantic space. In: 11th Annual Conference on Genetic and Evolutionary Computation, pp. 987–994 (2009)

[27] Uy, N.Q., Hoai, N.X., O'Neill, M., McKay, R.I., Galván-López, E.: Semantically-based crossover in genetic programming: application to real-valued symbolic regression. Genetic Programming and Evolvable Machines **12**(2), 91–119 (2011)

[28] Krawiec, K., Pawlak, T.: Approximating geometric crossover by semantic backpropagation. In: 15th Annual Conference on Genetic and Evolutionary Computation, pp. 941–948 (2013)

[29] Nguyen, Q.U., Chu, T.H.: Semantic approximation for reducing code bloat in genetic programming. Swarm and Evolutionary Computation **58**, 100729 (2020)

[30] Schmidt, M., Lipson, H.: Distilling free-form natural laws from experimental data. Science **324**(5923), 81–85 (2009)

[31] Korns, M.F.: A baseline symbolic regression algorithm. In: Genetic Programming Theory and Practice X, pp. 117–137 (2013)

[32] Schapire, R.E.: The strength of weak learnability. Machine learning **5**, 197–227 (1990)

[33] Martins, J.F.B., Oliveira, L.O.V., Miranda, L.F., Casadei, F., Pappa, G.L.: Solving the exponential growth of symbolic regression trees in geometric semantic genetic programming. In: Genetic and Evolutionary Computation Conference, pp. 1151–1158 (2018)

[34] Liu, D., Virgolin, M., Alderliesten, T., Bosman, P.A.N.: Evolvability degeneration in multi-objective genetic programming for symbolic regression. In: Genetic and Evolutionary Computation Conference, pp. 973–981 (2022)

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

[36] Ni, J., Drieberg, R.H., Rockett, P.I.: The use of an analytic quotient operator in

genetic programming. IEEE Transactions on Evolutionary Computation **17**(1), 146–152 (2013)

[37] Chen, Q., Xue, B., Niu, B., Zhang, M.: Improving generalisation of genetic programming for high-dimensional symbolic regression with feature selection. In: IEEE Congress on Evolutionary Computation, pp. 3793–3800 (2016)

[38] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)

[39] Ferreira, J., Pedemonte, M., Torres, A.I.: A genetic programming approach for construction of surrogate models. In: Computer Aided Chemical Engineering vol. 47, pp. 451–456 (2019)

[40] Sathia, V., Ganesh, V., Nanditale, S.R.T.: Accelerating Genetic Programming using GPUs (2021)

[41] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., *et al.*: Scikit-learn: Machine learning in python. The Journal of Machine Learning Research **12**, 2825–2830 (2011)

[42] Chen, T., Guestrin, C.: XGBoost: A scalable tree boosting system. In: 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785–794 (2016)

[43] Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A.G., Parizeau, M., Gagné, C.: DEAP: Evolutionary algorithms made easy. Journal of Machine Learning Research **13**(1), 2171–2175 (2012)

[44] Demšar, J.: Statistical comparisons of classifiers over multiple data sets. The Journal of Machine Learning Research **7**, 1–30 (2006)

[45] Dignum, S., Poli, R.: Operator equalisation and bloat free GP. In: 11th European Conference on Genetic Programming, pp. 110–121 (2008)

[46] Silva, S., Dignum, S.: Extending operator equalisation: Fitness based self adaptive length distribution for bloat free GP. In: 12th European Conference on Genetic Programming, pp. 159–170 (2009)

[47] Videau, M., Leite, A., Teytaud, O., Schoenauer, M.: Multi-objective genetic programming for explainable reinforcement learning. In: European Conference on Genetic Programming, pp. 278–293 (2022)