



**HAL**  
open science

# Memetic Semantic Genetic Programming for Symbolic Regression

Alessandro Leite, Marc Schoenauer

► **To cite this version:**

Alessandro Leite, Marc Schoenauer. Memetic Semantic Genetic Programming for Symbolic Regression. 26th EuroGP - Part of EvoStar 2023, Species Society, Apr 2023, Brno, Czech Republic. pp.198-212, 10.1007/978-3-031-29573-7\_13 . hal-04563511

**HAL Id: hal-04563511**



**<https://hal.science/hal-04563511>**

Submitted on 29 Apr 2024

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Memetic Semantic Genetic Programming for Symbolic Regression

Alessandro Leite  and Marc Schoenauer 

TAU, Inria Saclay, LISN, Univ. Paris-Saclay, France  
firstname.lastname@inria.fr

**Abstract.** This paper describes a new memetic semantic algorithm for symbolic regression (SR). While memetic computation offers a way to encode domain knowledge into a population-based process, semantic-based algorithms allow one to improve them locally to achieve a desired output. Hence, combining memetic and semantic enables us to (a) enhance the exploration and exploitation features of genetic programming (GP) and (b) discover short symbolic expressions that are easy to understand and interpret without losing the expressivity characteristics of symbolic regression. Experimental results show that our proposed memetic semantic algorithm can outperform traditional evolutionary and non-evolutionary methods on several real-world symbolic regression problems, paving a new direction to handle both the bloating and generalization endeavors of genetic programming.

**Keywords:** Genetic Programming · Memetic Semantic · Symbolic Regression

## 1 Introduction

For a given dataset  $(X, y)$ , symbolic regression (SR) aims to find a function  $f(X) : \mathbb{R}^n \mapsto \mathbb{R}$  that represents the underlying relationship between the input features  $(X)$  and an output  $(y)$ . Over the last few years, genetic programming (GP) [13] 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 [25,33]. 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 black-box and, consequently, hard to understand and explain. Symbolic regression is usually implemented through genetic programming (GP).

Traditional GP-based methods rely on the outcome of a program to decide how well it solves the task, ignoring intermediate results such as the semantics of its subtrees [21,26]. However, one can consider them to guide the search during its exploration process and, thus, to generalize on unseen data, and to favor short expressions that are usually easier to understand and analyze by the users. 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 [26,8] 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 tree’s node in a supervised setting. In other words, it computes the desired outputs for each node on the path from the root regarding the target semantics and the semantics of the other subtrees in the tree.

At the same time, several mixtures of evolutionary and non-evolutionary methods have been proposed over the last few years. One example is memetics algorithms [25] that 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. In this context, *this paper proposes an evolutionary multi-objective algorithm that combines both memetics and semantic backpropagation algorithms for symbolic regression problems.*

Different from traditional semantic backpropagation operators (e.g., random desired operator (RDO) [26]), our *memetic semantic GP for symbolic regression (MSGP)* approach (Section 4) only tunes the real-valued constants after a suitable tree has been found for the problem. Likewise, it computes them through linear scaling (LS) (i.e., regression) [10] and not randomly, and at each iteration, as implemented by the RDO operator [26]. Linear scaling aims to minimize the mean squared error (MSE) of a tree by performing a linear transformation on its outputs [10,11]. Consequently, it frees GP from this time-consuming task, allowing it to focus exclusively on the shape of the tree that fits the structure of the data rather than on trying to find a scale that approximates the target output. Last but not least, linear scaling helps in dealing with GP bloat problem [10,11].

Additionally, instead of trying to build a library with all possible pre-computing subtrees up to a maximum height or a dynamic one, which increases the computing cost and interpretability due to the bloat problem, MSGP relies on a fixed library with a randomly generated population of subtrees up to a given height.

Experimental results (Section 6) on various real-world benchmark datasets show that MSGP either outperforms or is equal to traditional machine learning methods (e.g., decision tree (interpretable) and random forest (black-box method) [2]), and established GP-based methods (e.g., *gplearn*). Likewise, our approach leads to short expressions which improve the interpretability of the model without including any new parameter to be specified by the users. MSGP’s code is available at [gitlab.inria.fr/trust-ai/memetics/msgp](https://gitlab.inria.fr/trust-ai/memetics/msgp).

## 2 Semantic GP

In GP, semantics describes the behavior of a program on a specific dataset. In other words, it is the outputs’ vector for the fitness cases of a problem [22]. More formally, in a supervised setting, assume the data is a set  $\mathcal{D}$  made of  $N$

fitness cases:  $\mathcal{D} = \{(X_1, y_1), (X_2, y_2), \dots, (X_N, y_N)\}$ , 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 [26]:

$$s(p) = [p(X_1), p(X_2), \dots, p(X_N)] \quad (1)$$

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

Semantic backpropagation algorithms [26,8] 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  $o_i$  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 at the root, semantic backpropagation recursively computes the desired output for a node  $\mathcal{N}$  at depth  $D$  as [26,14]:

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

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 will be replaced, as well as the semantics of its ancestors and siblings [21]. 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 is 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, context semantics depend on the function implemented by the immediate parent, the parent semantics, and the semantics of the siblings [21]. As a result, local improvements may degrade the global performance.

---

<sup>1</sup> We are focusing in this work on the specific case of 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.

## 2.1 Library building and searching

For a given desired output at a given node, we want to search for a tree that better approximates these outputs than the current subtree. One 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 [26,4]. This strategy keeps only the subtrees with the smallest number of nodes if different ones exist in the population 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 output  $o$  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  $|d_i^j - o_i|^k, \forall k \in \{1, 2, \dots, N\}$  [26]. 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.

## 3 Memetic Algorithms

Memetic algorithms (MA) combine population-based search strategies with local search heuristics inspired by the concept in genetics [6]. They have been used across different domains due to their capacity to establish a good balance between exploration and exploitation when finding a solution for a complex optimization problem [23,5]. A meme, in this case, represents transferable knowledge built through local refinement procedures, which can be seen as a form of domain-specific expert knowledge on how a solution can be better improved [23]. For an optimization viewpoint, prototypical memetic algorithms comprise three main phases named creation, local improvement, and evolution (Algorithm 1)<sup>2</sup>. 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 that 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 [23].

---

<sup>2</sup> 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.

---

**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

```

---

## 4 Memetic Semantic for Symbolic Regression

This section describes how semantic backpropagation and memetic algorithms are combined to evolve GP models for symbolic regression problems that are interpretable and have a lower learning error.

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, MA contribute to improving them by considering the semantics of their parts (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 [10] to search for constants that correct the residual errors of the tree. As a result, at each iteration, the SB and memetic algorithms 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 ( $X_i = [x_{i,1}, x_{i,2}, \dots, x_{i,m}]$ ) and a corresponding target output ( $y_i$ ), the task of symbolic regression comprises in finding a tree ( $\mathcal{T}(\cdot)$ ) that minimizes the distance to an output ( $y$ ) [30,12]. Such tree  $\mathcal{T}(\cdot)$  usually includes a set of predefined functions and terminals (a.k.a constants and input variables).

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

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

### 4.1 Algorithm

Given a tree  $\mathcal{T}$  with a set of subtrees  $\mathcal{S} = \{s_1, s_2, s_n\}$  and a library  $\mathcal{L}$  composed of  $l$  individuals (i.e., small subtrees), the goal of the proposed memetic semantic GP for symbolic regression (MSGP) algorithm (Algorithm 2) is to interactively

improve  $\mathcal{T}$  by checking for each subtree  $s_i \in \mathcal{S}$  if there exists a subtree  $s^* \in \mathcal{L}$  whose semantics are closest to the desired ones for  $s_i$ .

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 easier 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 the one of a tree. Consequently, the size and heterogeneity of the library  $\mathcal{L}$  can play an important role.

Further, the memetic part of MSGP, linear scalings computes a scaled version of the MSE [10] with a computing cost that is linear with the dataset size  $N$ , (i.e.,  $\mathcal{O}(n)$ ):

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

With  $a$  and  $b$  defined as:

$$a = \bar{y} - b\bar{\hat{y}} \quad (5)$$

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

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

Finally, to avoid consuming computing resources to an already optimal tree, an early stopping strategy can be adopted in practice.

---

#### Algorithm 2 Memetic semantic for symbolic regression

---

**Require:** Initial tree ( $\mathcal{T}$ ), library ( $\mathcal{L}$ )

**Require:** # *epochs*, and fitness cases ( $\mathbf{X}, y$ )

- 1:  $\mathcal{T}' \leftarrow \text{clone}(\mathcal{T})$
  - 2: Evaluate( $\mathcal{T}'$ )
  - 3: **while**  $e \leq \text{epochs}$  **do**
  - 4:    $\mathcal{T}^* \leftarrow \text{lti}(\mathcal{T}', \mathcal{L}, \mathbf{X}, y)$  ▷ local tree improvement (Algorithm 3)
  - 5:    $\mathcal{T}^* \leftarrow \text{LS}(\mathcal{T}^*, y)$  ▷ linear scaling [10] computes the coefficients of  $\mathcal{T}^*$   
(Equations (5) and (6))
  - 6:    $\mathcal{T}' \leftarrow \text{best}(\mathcal{T}^*, \mathcal{T}')$
  - 7: **return**  $\mathcal{T}'$
- 

## 4.2 Local tree improvement

Local tree improvement (LTI) algorithm (Algorithm 3) identifies the subtrees with equal or better semantics than a randomly selected subtree of a given tree. It

performs an exhaustive search in a library  $\mathcal{L}$  with a set of pre-computed semantics using the ancestor’s semantics of a subtree as the target process. Hence, given a library  $\mathcal{L}$  and a tree  $\mathcal{T}$ , LTI finds a subtree  $s^*$  in  $\mathcal{L}$  that minimizes

$$\arg \min_{s^* \in \mathcal{L}} \min_{t \in \{t_i, \dots, t_n\}} d(t, s(s^*)) \quad (7)$$

where,  $t$  is the desired semantics and  $s$  represents the semantics of a subtree  $s^* \in \mathcal{L}$ . 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’s semantics of the selected subtree. If no local improvement was identified, the algorithm randomly replaces a subtree in  $\mathcal{T}$  by the 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, height, and generalization). Thus, it is up to the superior level to keep or ignore the new proposed tree. A further investigation may evolve each proposed tree during a predefined number of generations and then crossover them using the LTI algorithm.

We consider 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 candidate ones have the same semantics. Moreover, we also individually include the features of the problems, as well as the operators (i.e., functions), into the library.

## 5 Experimental Setup

We evaluated the proposed algorithm on different real-world regression dataset benchmarks. They have a heterogeneous number of features and sample sizes, as depicted in Table 1. Moreover, they are commonly used in the GP literature [35,18] as overfitting the training set occurs either when complex models are learned or when models are built using discontinuous functions. Furthermore, Dow Chemical and Tower datasets are recommended as benchmarks [36]. They come from the UCI machine learning repository ([archive.ics.uci.edu](http://archive.ics.uci.edu)) and from the repository ([shortest.link/8n9V](http://shortest.link/8n9V)) provided by Martins et al. [20].

Table 2 includes the parameters settings to define the library, the initial tree, and the one used in standard GP experiments. We use analytical quotient (AQ) instead of protected division to avoid discontinuous behaviors [24], but keeping the same general properties of division. Likewise, the literature has shown that using it helps generalize at prediction time [24,3,35]. It is defined as:

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

As baseline, we considered evolutionary (i.e., GP) and non-evolutionary (i.e., decision tree (DT) and random forest (RF) [2]) approaches. We relied on *gplearn* [32]



---

**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( $\tau$ )
5:  $best[s] \leftarrow \emptyset$ 
6: for all  $s^* \in \mathcal{L}$  do
7:   if  $(s, s^*) \in k$  then
8:     continue
9:    $e \leftarrow$  ERROR( $s, s^*$ )
10:  if  $e < min\_error$  then
11:     $best[s] \leftarrow (s^*, 0)$ 
12:     $min\_error \leftarrow e$ 
13:  else if  $e == min\_error$  then
14:     $best[s] \text{ best}[s] \leftarrow \cup \{(s^*, e)\}$ 
15:  if  $|best[s]| > 0$  then
16:     $k \leftarrow k \cup \{(s, s^*)\}$ 
17:     $\mathcal{T}^* \leftarrow$  CROSSOVER( $ancestor(s), s^*, \mathcal{T}$ )
18:    SEMANTICBACKPROGRATION( $ancestor(s), s^*, \mathcal{T}^*$ )
19:  return  $\mathcal{T}^*$ 
20:  $s \leftarrow$  RANDOM( $\tau$ )
21:  $s^* \leftarrow$  RANDOM( $\mathcal{L}$ )
22:  $\mathcal{T}^* \leftarrow$  CROSSOVER( $ancestor(s), s^*, \mathcal{T}$ )
23: SEMANTICBACKPROGRATION( $ancestor(s), s^*, \mathcal{T}^*$ )
24:  $k \leftarrow k \cup \{(s, s^*)\}$ 
25: return  $\mathcal{T}^*$ 

```

---

as the GP-based model, and on the *scikit-learn* [27] implementation of decision tree and random forests as they are commonly used in the GP and machine learning literature [7,31,29]. While decision trees are normally considered interpretable models, *random forests* are defined as black-box. Nevertheless, the latter often outperforms the former. Consequently, they are usually employed by practitioners across different domains. For these models, we used the default parameter values defined by *scikit-learn*. Each experiment comprised 30 independent runs, and the median of the results is reported. MSGP was implemented in `python` using the `DEAP` library [9]. Finally, we run the experiments on a MacBook Pro with one Apple M1 processor (8 cores) and 16 GB of RAM memory.

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: Parameter settings

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%
Data normalization	$L2$

## 6 Results

Table 3 shows the error on the testing set, and the symbolic expressions outputted by MSGP. The experiments reveal that MSGP outperformed the baseline methods except for the Boston housing (BH) dataset. Additionally, the expressions are short, showing that our proposed method could handle GP bloating on the selected problems. Moreover, it required less than 2000 epochs to find the best trees on most datasets, as depicted in Fig. 1. An early stopping strategy or other automatic options can be added to avoid running without further improvement. However, one can keep to users the decision of which tree to pick up for a given problem as an option to help them explore alternative solutions, as they all have distinct semantics.

Table 3: Performance on the test set for each benchmark

DS	DT	RF	GP	MSGP	Expression
AR	82.88	75.89	30587	<b>24.58</b>	$-90.09 * (X_1 + X_3 - X_4 + X_5) + 129.9$
BH	<b>10.96</b>	75.89	36.9	27.99	$1598 * X_6 * (-X_{13} + X_6) + 19.47$
CCS	583.32	349.41	441.13	<b>141.52</b>	$2052.89 * (X_1 + X_8) * (X_5 + X_6) + 15.52$
DC	0.59	0.49	0.1	<b>0.07</b>	$-166.12 * (X_{17} - X_{49} - \frac{X_{49}}{X_4}) - 7.16$
EC	101.27	115.46	179.51	<b>14.68</b>	$558.67 * ((X_1 * X_7) + X_2 + X_5) - 56.23$
EH	143.78	94.88	162.33	<b>15.37</b>	$576.3 * (X_2 + X_3 * X_7 + X_5) - 60.97$
TW	10586.48	10214.52	33253	<b>2783.19</b> ±369.93	$-27120.74 * (X_1 + X_{16} - X_{23} - X_6) + 346.49$
WR	2.48	1.53	0.7	<b>0.49</b>	$43.26 * X_{11} - 21.63 * X_2 - 21.63 * X_8 + 5.65$
WW	1.6	1.01	0.71	<b>0.62</b>	$127.95 - (125.5 * X_2)^2$
YH	791.61	699.25	100.77	<b>32.65</b>	$-28008.75 + 28014.61 * \frac{X_6}{X_2}$

Consequently, it is essential to check the contribution of the features chosen by the model from the prediction viewpoint. A common strategy comprises computing a score measure for all the input features. Examples include the Gini and mutual information measures. We used the random forest algorithm to compute the feature importance of the datasets. The goal was to understand if MSGP was relying on the relevant ones.

Figure 2 illustrates the feature importance computed by random forest for each dataset. We can observe that based on this metric, MSGP symbolic expressions (Table 3) include the important ones identified by random forest without needing external support. Indeed, we observed with some experiments that

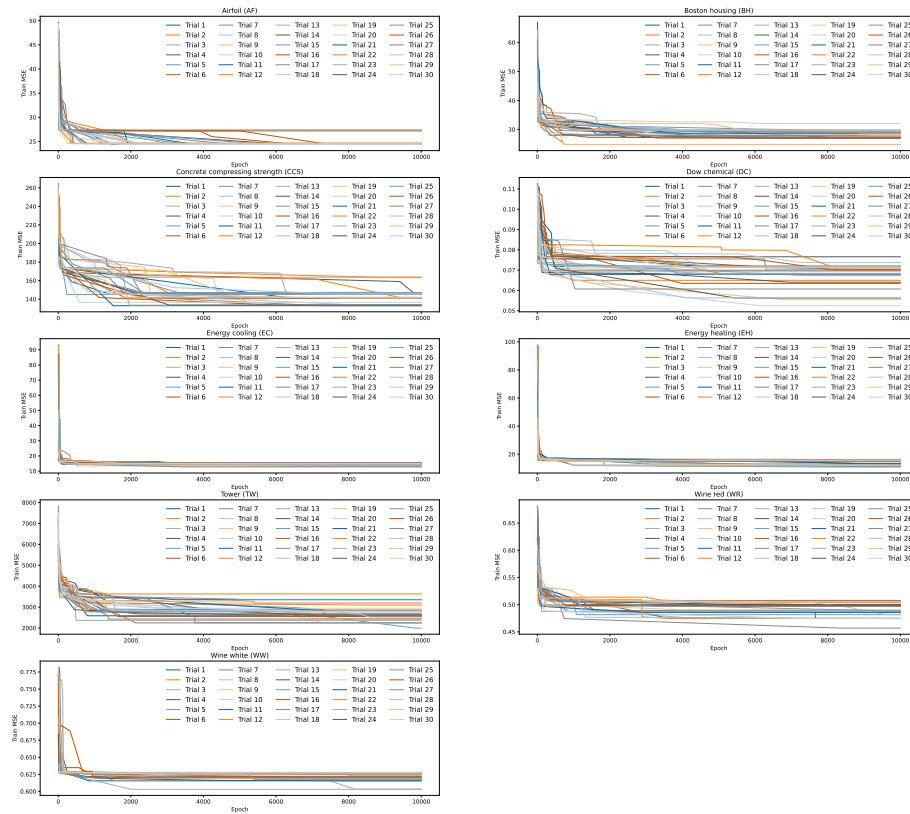


Fig. 1: Learning curves of the trees on the training set for each trial

building a library with only the most important features degrades the model’s performance and increases the output expressions’ size. This behavior suggests that the proposed method requires some freedom to explore the search space. However, further analyses are still necessary to understand the reasons. Furthermore, complementary studies can be done to assess the performance of MSGP as a feature selection approach, which plays a fundamental role in post-hoc explanation methods, including SHapley Additive exPlanations (SHAP) [19] and Local Interpretable Model-Agnostic Explanations (LIME) [28].

In addition, the results showed that for some problems (e.g., CCS and DC), the algorithm stays stuck in a local minimum. Such behavior suggests the need to be able to identify and handle such behavior by, for instance, introducing some transformations to enhance exploration. Finally, another improvement comprises quantifying the impact of replacing a subtree with another one by considering the semantics of the tree. It means identifying which operations are necessary to change it to the desired one in an optimization setting.

## 7 Related Work

Several works have tried to handle the bloat problem of genetic programming. For example, Bleuler et al. [1] 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 [17], the authors proposed a pseudo-hill-climbing strategy to control trees’ 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. Other works have proposed semantic-based operators to handle both bloating and generalization issues. Some examples include [15,34,22,26] among others. Uy et al. [34] proposed to 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. [22] suggested a semantic crossover operator that creates offsprings with a weighted average of their parents’ semantics. Notwithstanding, it cannot handle GP bloating properly without further simplification procedures. Geometric semantic crossover (AGX) [16] tries to handle these endeavors by replacing parents’ subtrees with other ones that are semantically closed to their parents’ midpoint semantic. Random desired operator (RDO) [26] introduces back-propagation. In this case, after a crossover operation, the semantics of the new subtree is back-propagated as described in Section 2. Different studies have shown that RDO outperforms the other operators on both regression [26] and boolean [8] problems.

This work is closest to the RDO operator [26]. Nonetheless, we use a similar idea to improve a candidate solution’s fitness locally instead of relying on crossover operation. Likewise, we use linear scaling [16] to compute the coefficients scale once a candidate solution was found. Finally, a memetic algorithm selects the

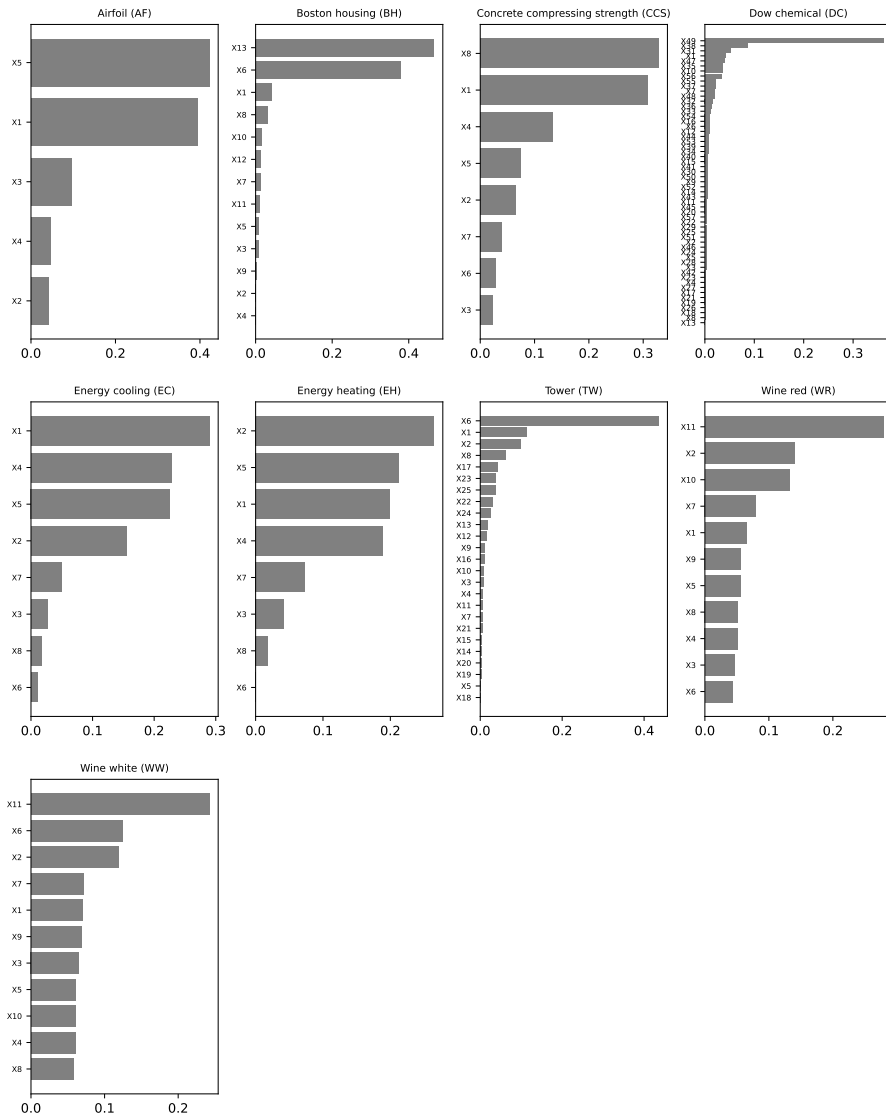


Fig. 2: Feature importance for each dataset obtained through random forest

solutions based on their fitness on a validation set and size. To the best of our knowledge, this is the first work to propose a memetic semantic algorithm for symbolic regression problems.

## 8 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 complicated symbolic expressions (a.k.a. bloat). Semantic-based strategies [16,26] have been seen as a way to handle this issue. In this context, this paper proposed and evaluated a memetic semantic algorithm for symbolic regression (MSGP). The proposed approach combines a population-based search strategy with semantics-guided ones to output short symbolic expressions without penalizing the accuracy.

Experimental results demonstrated that in addition to favoring short and interpretable expressions, the proposed algorithm could outperform traditional machine learning models (i.e., decision tree (DT) and random forest (RF) [2]) and evolutionary one on different real-world datasets. Additionally, they demonstrated that the proposed algorithm only required a few iterations to identify the most predictive features. Further works include employing it to search for counterfactual outputs, as the counterfactual response can be framed as desired semantics. Another one comprises investigating new strategies to guide the construction of the semantics library to enhance the exploration and exploitation features of memetic semantic-based algorithms.

## Acknowledgements

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

## References

1. 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)
2. Breiman, L.: Random Forests. *Machine Learning* **45**(1), 5–32 (2001)
3. 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)
4. 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)

5. 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)
6. Dawkins, R.: *The Selfish Gene*. Oxford University Press (1976)
7. 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. Elsevier (2019)
8. Ffrancon, R., Schoenauer, M.: Memetic semantic genetic programming. In: *Annual Conference on Genetic and Evolutionary Computation*. pp. 1023–1030 (2015)
9. 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 (jul 2012)
10. Keijzer, M.: Improving symbolic regression with interval arithmetic and linear scaling. In: *European Conference on Genetic Programming*. pp. 70–82 (2003)
11. Keijzer, M.: Scaled symbolic regression. *Genetic Programming and Evolvable Machines* **5**(3), 259–269 (2004)
12. Korn, M.F.: A baseline symbolic regression algorithm. In: *Genetic Programming Theory and Practice X*, pp. 117–137. Springer (2013)
13. Koza, J.R.: *Genetic Programming: On the Programming of Computers by means of Natural Evolution*. MIT Press, Massachusetts (1992)
14. Krawiec, K.: Semantic genetic programming. In: *Behavioral program synthesis with genetic programming*, pp. 55–66. Springer (2016)
15. Krawiec, K., Lichocki, P.: Approximating geometric crossover in semantic space. In: *11th Annual conference on Genetic and Evolutionary Computation*. pp. 987–994 (2009)
16. Krawiec, K., Pawlak, T.: Approximating geometric crossover by semantic backpropagation. In: *15th annual conference on Genetic and evolutionary computation*. pp. 941–948 (2013)
17. Langdon, W.B., Poli, R.: Genetic programming bloat with dynamic fitness. In: *First European Workshop on Genetic Programming*. pp. 97–112 (1998)
18. 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)
19. Lundberg, S.M., Lee, S.I.: A unified approach to interpreting model predictions. In: *NeurIPS*. pp. 4768–4777 (2017)
20. 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)
21. McPhee, N.F., Ohs, B., Hutchison, T.: Semantic building blocks in genetic programming. In: *European Conference on Genetic Programming*. pp. 134–145 (2008)
22. Moraglio, A., Krawiec, K., Johnson, C.G.: Geometric semantic genetic programming. In: *International Conference on Parallel Problem Solving from Nature*. pp. 21–31 (2012)
23. Moscato, P.: *On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms*. Tech. Rep. 826, Caltech Concurrent Computation Program, California Institute of Technology (1989)
24. Ni, J., Driehage, 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)



25. 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)
26. 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)
27. 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)
28. Ribeiro, M.T., Singh, S., Guestrin, C.: Why should I trust you? explaining the predictions of any classifier. In: *SIGKDD*. pp. 1135–1144 (2016)
29. Sathia, V., Ganesh, V., Nanditale, S.R.T.: Accelerating genetic programming using gpus (2021)
30. Schmidt, M., Lipson, H.: Distilling free-form natural laws from experimental data. *Science* **324**(5923), 81–85 (2009)
31. Sipper, M., Moore, J.H.: Symbolic-regression boosting. *Genetic Programming and Evolvable Machines* **22**, 357–381 (2021)
32. Stephens, T.: Genetic programming in python with a scikit-learn inspired API: gplearn. [github.com/trevorstevens/gplearn](https://github.com/trevorstevens/gplearn) (2016)
33. Udrescu, S.M., Tegmark, M.: Ai feynman: A physics-inspired method for symbolic regression. *Science Advances* **6**(16), eaay2631 (2020)
34. 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)
35. 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)
36. 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)