Expert Systems With Applications 198 (2022) 116712

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

DoME: A deterministic technique for equation development and Symbolic Regression

Daniel Rivero *, Enrique Fernandez-Blanco, Alejandro Pazos

Centro de investigación CITIC, Department of Computer Science and Information Technology, University of A Coruña, 15071, A Coruña, Spain

ARTICLE INFO

Keywords:
Symbolic regression
Machine learning
Artificial intelligence

ABSTRACT

Based on a solid mathematical background, this paper proposes a method for Symbolic Regression that enables the extraction of mathematical expressions from a dataset. Contrary to other approaches, such as Genetic Programming, the proposed method is deterministic and, consequently, does not require the creation of a population of initial solutions. Instead, a simple expression is grown until it fits the data. This method has been compared with four well-known Symbolic Regression techniques with a large number of datasets. As a result, on average, the proposed method returns better performance than the other techniques, with the advantage of returning mathematical expressions that can be easily used by different systems. Additionally, this method makes it possible to establish a threshold at the complexity of the expressions generated, i.e., the system can return mathematical expressions that are easily analyzed by the user, as opposed to other techniques that return very large expressions.

1. Introduction

¿What are the benefits of transparent ML models?

In Machine Learning, supervised learning makes it possible to discover models that represent the relationship between a series of inputs and outputs. Nowadays, there are many different techniques for developing these models, such as Artificial Neural Networks (Haykin, 2009). However, most of these methods do not give hints about the true relationship between inputs and outputs, even though they show good results in the modeling. In many environments, a black-box model is not enough, since the objective is to find out an equation that the expert can analyze and thus increase the knowledge about the system being modeled. In this sense, the search for transparent ML models, showing a clear relationship between inputs and outputs is a current hot topic in research, and nowadays there are conferences, such as FAccT (ACM Conference on Fairness, Accountability and Transparency), dedicated to this particular topic.

How do you call the technics that given a dataset, search for a expression that fits it?

When that relationship is a mathematical expression, it is called Symbolic Regression (SR). This term framed the few techniques which, given a dataset of observations, explore the search space to find an expression that fits that dataset (Orzechowski et al., 2018). Among those techniques, the most commonly used are based on Genetic Programming (GP) (Koza, 1992) (Poli et al., 2008) which is a general-purpose evolutionary technique. Although, other representations have arisen, such as Linear GP (Brameier & Banzhaf, 2010), Stack-based Genetic Programming (Perkis, 1994), Cartesian Genetic Programming (CGP) (Miller & Turner, 2015), or Positional Cartesian Genetic Programming (PCGP) (Wilson et al., 2018), the classical tree-shape representation is still the most used codification for mathematical expressions in Symbolic Regression.

what are some example of mathematical regression?

what is the most common representation for mathematical expressions?

GP starts from an initial population of tree-shaped individuals, which undergo an evolutionary process with the execution of the genetic operators (selection, crossover, mutation, and replacement). These operators are based on randomness making the whole process a highly time-consuming task. For instance, the original crossover operator proposes the random combination of mathematical expressions, although there are different approaches that try to combine useful parts (Kronberger et al., 2009)(Krawiec, 2012). The mutation operator also makes random changes in a mathematical expression. Therefore, even though the whole evolutionary process is a search driven by the fitness function, the computation of many mathematical expressions that are not going to be part of the final expression is clearly necessary. Attending to the convergence behavior of those solutions, it should be highlighted that, although the algorithms behind GP are well-known, there is no underlying theory for its convergence behavior, and the results obtained lack a mathematical basis. However, as a system capable of performing symbolic regression, it has been successfully applied to real-world problems, such as integrated circuit design (McConaghy & Gielen, 2009) or civil engineering (Pérez et al., 2008).

how does GP apply the genetic operators?

In recent years, a new type of GP called Geometric Semantic Genetic Programming (GSGP) has arisen (Moraglio et al., 2012). This approach

* Corresponding author.
E-mail addresses: daniel.rivero@udc.es (D. Rivero), enrique.fernandez@udc.es (E. Fernandez-Blanco), apazos@udc.es (A. Pazos).

https://doi.org/10.1016/j.eswa.2022.116712
Received 16 August 2020; Received in revised form 18 January 2022; Accepted 20 February 2022
Available online 4 March 2022

What's the main drawback of GSGP?

works in the so-called geometric space, which is made up of the outputs of the GSGP programs (semantics). In this space, the targets are another point in the semantic space. Its performance, measured in results and time, is much better than GP. However, a big drawback of this technique is the excessive number of nodes in the resulting tree, usually higher than 10^{15} (Pawlak, 2016). Although there are studies that try to reduce this number (Martins et al., 2018), the result is still too complex, and is often the sum of different mathematical expressions. On the other hand, it has been demonstrated that the resulting expression is a sum of the first generation expressions (Pawlak, 2016). This fact, which results in oversized and very complex expressions, is a major drawback of this technique, since those expressions are not understandable by a human being. However, it has been successfully applied in many real-world environments, such as financial (McDermott et al., 2014) or biomedical (Zhu et al., 2013).

is random number generation used in PGE?

Another interesting GP-based Symbolic Regression technique is called Prioritized Grammar Enumeration (PGE). Starting from an initial set of simple functions, this technique expands the best ones resulting in more complex expressions. Genetic operators are replaced with grammar production rules becoming a deterministic algorithm in which no random number generation is used by this algorithm (Worm & Chiu, 2013). However, to date it has not been possible to reproduce the results reported.

What's the advantage of FFX?

Symbolic Regression is a research field that has hardly been explored outside of GP literature, with very few papers reporting non-evolutionary approaches (McConaghy, 2011)(Pawlak, 2016). Those approaches, despite having a very short computation time and producing good training results, are based on a combination of several functions. The resulting expression is the weighted sum of a series of expressions, similar to what happens in GSGP (Pawlak, 2016). Although, FFX uses sparsity inducing L_1 penalty or a mixture of L_1 and L_2 penalty to limit the complexity, this method returns expressions that are not understandable by any human.

what does FFX/GP consist on?

This FFX approach has been improved in combination with GP, resulting in the FFX/GP algorithm. This technique is based on creating combinations of the input variables and their combination through linear regression by using ElasticNet (Zou & Hastie, 2005). This technique is applied several times, and as a result new data are obtained, which are used as input to GP (Icke & Bongard, 2013).

what does SymTree consist on?

In a different approach called SymTree, a tree structure is developed in which the root node contains a linear regression of its children. These nodes are iteratively expanded by creating new children with simple functions. To generate the child nodes, a greedy heuristic is used (Olivetti de França, 2018). As in previously described approaches, this technique is based on linear combinations of different expressions.

¿Can Deep Learning be used for Symb Regression?

There are also some interesting Deep Learning-based algorithms that leverage artificial neural networks for symbolic regression. In a system called Deep Symbolic Regression (DSR), a neural network is used to search symbolic expressions (Petersen, 2019). Using risk-aware reinforcement learning, a recurrent neural network is trained to output a distribution over mathematical expressions. From this distribution, some expressions are taken and evaluated, resulting in a fitness value that is used as the reward signal to train the neural network.

What is AI Feynman?

In a different study, the method called AI Feynman (Udrescu & Tegmark, 2020), symbolic regression is performed in two big steps: simplification of the problem, and finding a solution. The first step includes tasks to identify simplifying properties, such as dimensional analysis, translational symmetry or multiplicative separability. The objective of this part is to transform the problem so that it can be solved more easily in the next step. Then, a polynomial fit is performed, and brute force is used to find more complex expressions. A six-layer feed-forward neural network is used to perform tests for translational symmetry, generalizations, and separability.

How can a Variant Autoencoder be used?

In another paper, GrammarVAE, a Variant Autoencoder is used as a generative model to develop expressions according to a pre-defined grammar (Kusner et al., 2017). In this study, this is used for symbolic regression. Neural networks are also used to develop expressions by using symbolic operators as activation functions, which allows the development of complex expressions. This scheme was used in a system called Equation Learner (EQL) (Sahoo et al., 2018).

how can a context-free grammar be used?

One of the most recent advances in Symbolic Regression is described in (Kammerer et al., 2020). In this paper, a search space of semantically unique expressions is defined by context-free grammar. From this point, an exhaustive search is performed through all of the expressions and, the best-fitting one is selected. This search space is explored through a grammar and a hashing scheme and the result is a deterministic algorithm for symbolic regression.

¿How does CFR work?

Finally, one of the most important and recent approaches is called CFR (Continued Fraction Regression) (Moscato et al., 2021). This approach uses a memetic algorithm that uses continued fractions as a representation. The search does not occur in the whole search space of expressions. In other words, this method has a fixed representation as a CFR and the search occurs in the coefficient space of the CFR. Therefore, this method only uses algebraic operators, and is very similar to the method proposed in this paper and is therefore of particular interest for a comparison of both methods.

¿is it possible to find an algorithm that is better in all problems?

As shown in this literature review, GP and its variants have been very successful in SR. However, GP and, in general, population-based algorithms have practical limitations when used for SR. The most important issue is the dependency on their stochastic nature, which could return very different expressions in different executions on the same datasets, since randomness is inherent in the algorithm itself. In SR, the search space is usually very large and this complexity is enhanced by the fact that there are multiple expressions that are equivalent to each other. Even if population-based algorithms usually perform a search in many parts at the same time, there is no guarantee that the whole search space is explored. According to the No-Free-Lunch theorems, it is impossible for any algorithm to perform a universal optimization (Wolpert & Macready, 1997). This means that the performance of all optimization algorithms is equal when it is averaged across all possible problems. Therefore, there is no best optimization algorithm that can be used for any SR. For this reason, other algorithms will outperform GP and the rest of population-based and non-population-based algorithms in many problems. Thus, new algorithms are needed, even though they will also be surpassed by the existing ones in some problems. This last point can be seen in several studies comparing various ML regression techniques, including GP-based regression methods (La Cava et al., 2021; Moscato et al., 2021; Orzechowski et al., 2018).

The main aim of this paper is to propose a novel approach for Symbolic Regression. This method called DoME (Development of Mathematical Expressions), allows the extraction of the relationship between the inputs and outputs of a dataset in the form of a mathematical expression. The key elements of this novel approach are, firstly, the method is based on the iterative expansion of mathematical expressions with no randomness in this process. Consequently, this method is deterministic and has a solid mathematical background. Secondly, the method makes it possible to control complexity of the expression, which helps in the interpretation of the results. The representation and search bias used by DoME may work especially well for many real-world problems, particularly those with low dimensionality and smooth functions.

what are two benefits of DoME?

2. Model

2.1. Overview

Similarly to GP, the proposed approach develops the equation in a tree-shape where two types of nodes are distinguished: terminal, or leaves of the tree, and non-terminal, or functions. As terminal nodes, we use only the variables of the problem and constants. As non-terminal nodes, the four basic arithmetic operators are used: +,-,*,/. Note that

what types of nodes are used in DoME? What operations are allowed?

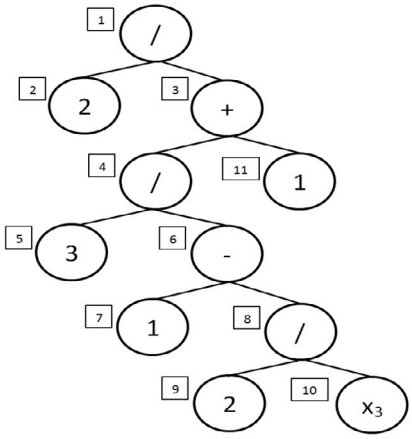


Fig. 1. Example of a tree.

we do not use "%" as protected division. In GP, it was needed to protect that operation because, as a result of many different combinations, there are chances to generate a divisions by zero. In this approach, the tree being developed will always be correct without illegal operations.

Fig. 1 shows an example of a tree. This tree has 11 nodes, 4 of which are non-terminal (labeled in squared tags as 1, 3, 4, and 8), and 5 are terminal (labeled in squared tags as 2, 4, 5, 7, 9, and 11). This tree represents the following function:

$$f(x_1, x_2, x_3, \ldots) = \frac{2}{\frac{3}{1-\frac{2}{x_3}} + 1} \tag{1}$$

Since the four basic arithmetic operators are used to construct the trees, all DoME expressions can be transformed algebraically into a rational polynomial. This is a restriction of this algorithm, which is not present in other SR systems such as GP-based methods. This restriction can lead to good results on many real-world problems, but poor performance on other problems where other behavior (such as periodic) is required. Therefore, a large benchmark set is needed to analyze its overall performance compared with other systems.

The proposed method starts from a minimum tree that, in successive iterations, is made to grow or shrink by replacing its nodes (either terminal or non-terminal) with other expressions. For example, a node can be replaced by a terminal node containing a constant (constant search), if this reduces the error of the tree. To do this, the best constant to replace that node is calculated. Another possibility is to replace that node with a terminal node containing a variable (variable search). Another possibility is to perform a mixture of these two searches, replacing the node with an expression, such as $k+x_2$ (constant-variable search).

A key element in this approach is the necessity to be able to calculate the error directly from the output of a node. This allows, firstly, to calculate the best constant for which that node could be substituted, and secondly, to calculate the error as a consequence of that substitution. The mathematical formulation of this method evolves around calculating the error from the output of a node and calculating the best constant for that node.

The algorithm can be summarized in the following steps:

What are the steps of the algorithm?

1. Construct an initial tree consisting of a constant.
2. For each node, calculate the coefficients that allow to evaluate the error from the outputs of that node.
3. Examine the nodes of the current tree following a strategy. According to this strategy, for each selected node p, perform one or more of these operations:
   - Compute the best constant with which to replace that node p, temporarily replace that node p with the constant, and compute the error of this new tree.
   - For each variable x_i, temporarily replace that node p with that variable and calculate the error of this new tree.
   - For each variable x_i and each of the four arithmetic operations, calculate the best constants with which to construct the operations $k + x_i$, $k - x_i$, $k * x_i$ and k/x_i, temporarily replace node p with each of the expressions and calculate the error in the new tree.
   - For the + and * operations, calculate the best constants with which to construct the expressions $k + p$ and $k * p$, substitute the node p temporarily for each of these two expressions, and calculate the error in the new tree.
4. If any of the previous temporal substitutions leads to a reduction in error greater than a fixed threshold, perform it as definitive.
5. If no stopping criterion is met, return to step 2.

As seen above, this algorithm is made up of different parts, such as the strategy of going through the nodes, the calculation of the coefficients for each node, or the substitution operations for each node. The rest of the subsections are intended to explain each of these parts separately.

2.2. Representation

Usually, when working in Machine Learning, the user has a dataset arranged as a matrix with dimensions $N \times P$ or $P \times N$, with P representing the number of variables or features, and N the number of data samples. In supervised learning, a target matrix is also needed with the dimension of $N \times T$ or $T \times N$ being T the number of outputs. In the case of SR, since a single equation is desired, $T = 1$, and the values of this matrix are real numbers. Mathematically, these matrices represent a set of pairs $\{(\mathbf{x_i}, t_i), i = 1, \ldots, N\}$, where $\mathbf{x_i} \in \mathbb{R}^L$ represents a data sample with P features and $t_i \in \mathbb{R}$ represents the target value for this data sample. Thus, an N-dimensional vector $\mathbf{t}$ represents the target values for each data sample. The objective of symbolic regression is to find a function $f : \mathbb{R}^L \to \mathbb{R}$ that minimizes a metric $\|\mathbf{o}, \mathbf{t}\|$, where the vector with $\mathbf{o}$ represents the output of the function for each data sample, i.e., $o_i = f(\mathbf{x_i}), i = 1, \ldots, N$.

what do the variables represent?

The most common way of working with this is creating a P-dimensional space, in which each dimension corresponds to each input variable. Thus, each data sample is a point in that P-dimensional space. However, in this paper, we work with an N-dimensional space, with one dimension for each data sample. Therefore, each variable corresponds to a single point in this space ($\mathbf{x_i} = (x_{i1}, \ldots, x_{iN})$), and the targets are also a point in this space ($\mathbf{t} = (t_1, \ldots, t_N)$). Moreover, the output of a model (not limited to being an equation) gives one value for each data sample, thus making a vector in this space ($\mathbf{o} = (o_1, \ldots, o_N)$). Therefore, each model is represented also as a point in this space. This vector $\mathbf{o}$ is called the semantics of this model, and $\mathbf{t}$ are the target semantics. Since both semantic vectors are points in the semantic space, the Euclidean distance between both points can be calculated corresponding to the square root of the SSE (Sum Squared Error) of the model:

$$\sqrt{SSE} = \sqrt{\sum_{i=1}^{N} (o_i - t_i)^2} \tag{2}$$

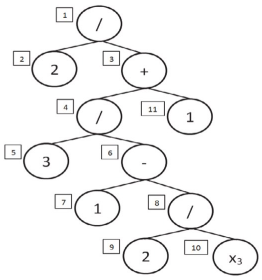
Thus, finding a better model is equivalent to finding a model closer to the target point. Therefore, an N-dimensional sphere is being created, and any model inside this sphere will be a better model (i.e., with a lower SSE). The radius of this sphere is the square root of the SSE (Root Sum Squared Error, $RSSE$). A different model with semantics $\mathbf{o'} = (o'_1, \ldots, o'_N)$ will be a better model if it is inside this sphere, i.e., if it complies with the following equation:

$$\sum_{i=1}^{N} (o'_i - t_i)^2 < SSE \tag{3}$$

Operation	Left Child				Right Child			
	a'_i	b'_i	c'_i	d'_i	a'_i	b'_i	c'_i	d'_i
+	a_i	$b_i - a_i \cdot y_i$	c_i	$d_i - c_i \cdot y_i$	a_i	$b_i - a_i \cdot x_i$	c_i	$d_i - c_i \cdot x_i$
−	a_i	$b_i + a_i \cdot y_i$	c_i	$d_i + c_i \cdot y_i$	a_i	$a_i \cdot x_i - b_i$	c_i	$c_i \cdot x_i - d_i$
*	$a_i \cdot y_i$	b_i	$c_i \cdot y_i$	d_i	$a_i \cdot x_i$	b_i	$c_i \cdot x_i$	d_i
/	a_i	$b_i \cdot y_i$	c_i	$d_i \cdot y_i$	b_i	$a_i \cdot x_i$	d_i	$c_i \cdot x_i$

Table 1
Description of the tree of the example.

Node	Semantic	Equation	S
1	(−1, 0.286, 0.8)	$\frac{1}{3}((o_i-5)^2+(o_i-4)^2+(o_i-1)^2)$	∅
2	(2, 2, 2)	$\frac{1}{3}((\frac{o_i+10}{2})^2+(\frac{o_i-28}{-7})^2+(\frac{o_i-2.5}{-2.5})^2)$	∅
3	(−2, 7, 2.5)	$\frac{1}{3}((\frac{-5\cdot o_i+2}{o_i})^2+(\frac{-4\cdot o_i+2}{o_i})^2+(\frac{-o_i+2}{o_i})^2)$	{(0, 0, 0)}
4	(−3, 6, 1.5)	$\frac{1}{3}((\frac{-5\cdot o_i-3}{o_i+1})^2+(\frac{-4\cdot o_i-2}{o_i+1})^2+(\frac{-o_i+1}{o_i+1})^2)$	{(−1, −1, −1)}
5	(3, 3, 3)	$\frac{1}{3}((\frac{-5\cdot o_i+3}{o_i-1})^2+(\frac{-4\cdot o_i-1}{o_i+0.5})^2+(\frac{-o_i+2}{o_i+2})^2)$	{(1, −0.5, −2)}
6	(−1, 0.5, 2)	$\frac{1}{3}((\frac{-3\cdot o_i-15}{o_i+3})^2+(\frac{-2\cdot o_i-12}{o_i+3})^2+(\frac{o_i-3}{o_i+3})^2)$	{(0, 0, 0), (−3, −3, −3)}
7	(1,1,1)	$\frac{1}{3}((\frac{-3\cdot o_i-9}{o_i+1})^2+(\frac{-2\cdot o_i-11}{o_i+2.5})^2+(\frac{o_i-2}{o_i+4})^2)$	{(2, 0.5, −1), (−1, −2.5, −4)}
8	(2, 0.5, −1)	$\frac{1}{3}((\frac{3\cdot o_i-18}{-o_i+4})^2+(\frac{2\cdot o_i-14}{-o_i+4})^2+(\frac{-o_i-2}{-o_i+4})^2)$	{(1, 1, 1), (4, 4, 4)}
9	(2, 2, 2)	$\frac{1}{3}((\frac{3\cdot o_i-18}{-o_i+4})^2+(\frac{2\cdot o_i-56}{-o_i+16})^2+(\frac{-o_i+4}{-o_i-8})^2)$	{(1, 4, −2), (4, 16, −8)}
10	(1, 4, −2)	$\frac{1}{3}((\frac{-18\cdot o_i+6}{4\cdot o_i-2})^2+(\frac{-14\cdot o_i+4}{4\cdot o_i-2})^2+(\frac{-2\cdot o_i-2}{4\cdot o_i-2})^2)$	{(0, 0, 0), (2, 2, 2), (0.5, 0.5, 0.5)}
11	(1, 1, 1)	$\frac{1}{3}((\frac{-5\cdot o_i+17}{o_i-3})^2+(\frac{-4\cdot o_i-22}{o_i+6})^2+(\frac{-o_i+0.5}{o_i+1.5})^2)$	{(3, −6, −1.5)}



observations = (1, 4, -2)
targets = (5, 4, 1)

what metric do I use to evaluate a tree?

Moreover, in order to reduce the impact of either having a large or low number of data samples, this number N is usually inserted into Eq. (3) resulting in the Mean Square Error. This is the metric used in this work to optimize Symbolic Regression, shown in Eq. (4):

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(o_i - t_i)^2 \tag{4}$$

Again, a new model with outputs $\mathbf{o}' = (o'_1, \ldots, o'_N)$ that complies with Eq. (3) is a model with a lower MSE, and closer to the target. If the model being processed can undergo different improvements, the best improvement will be the one that brings the model closer to the target point. This is the improvement that makes the new outputs maximize the reduction in MSE, given by Eq. (5)

$$reduction = MSE - \frac{1}{N}\sum_{i=1}^{N}(o'_i - t_i)^2 \tag{5}$$

how do I select the next step?

What is the stop condition?

This turns the problem of improving the model into an optimization problem: the more positive the result of Eq. (5) is, the better the performance of the new model will be. If different improvements are possible, then the one that maximizes Eq. (5) will be selected. If none of these possible improvements leads to having positive values in Eq. (5), then the improvement process has finished.

The tree has its semantics determined by the outputs to each data sample. As mentioned above, the root of this tree can be either a terminal or a non-terminal node. In the latter case, non-terminals, the root of the tree will be any of the four arithmetic operators, and each of its children form another tree, with their corresponding semantics and their corresponding points in the semantic space. Therefore, a tree with n nodes is represented in the semantic space as the semantic point of the root of the tree, but also as $n-1$ different points. If any of these nodes are modified, then the semantic points will be moved. Consequently, the overall evaluation values of the root of the tree are changed and the semantic point of the root of the tree will also be moved. Note that constants and variables (terminal nodes of the tree) also have a semantic value, representing one point in the semantic space.

Table 1 shows, in the second column, the semantics of each node of the tree shown in Fig. 1. All the nodes, including constants and variables, have their semantics, i.e., they are points in the search space. The variable x_3 takes the values of 1 for the first data sample, 4 for the second and −2 for the third. Therefore, its semantics are the vector (1,4,-2). Note that each terminal node representing a constant k has as semantics the vector $(k, k, \ldots, k)$, since it evaluates to k for each data point. The semantics for each non-terminal node are calculated by using the operation of this node and the semantics of each child. The semantics for each node on the tree is calculated from the terminal nodes to the root of the tree (bottom to top). This table contains two columns (Equation and S) whose contents are necessary to understand the formulation of this system, and will be described later in the paper.

2.3. Calculation of the equations for each node

The key idea of this study is that we could not have a tree that makes a positive reduction in Eq. (5) moving its semantics to get closer to the target point. However, it might be easier to modify one node in any branch to move the root towards the target. This modification is done by changing the chosen subtree into another. The question here is how to find the replacement subtree. To find it, it is necessary to calculate the MSE from the outputs of this node. In this sense, each node of the tree has an associated equation for calculating the MSE from its outputs. As happens with the root, this equation can measure the improvement of the overall result if any of the outputs o_i for this node are modified. This equation represents a particular region in the semantic space, and, for any node of the tree, is as follows:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{a_i \cdot o_i - b_i}{c_i \cdot o_i - d_i}\right)^2 \tag{6}$$

what is the general expression of the MSE of a node?

where o_i are the outputs of that particular node, and a, b, c and d are four vectors characterizing the equation for this node. These vectors will be different for each node of the tree, thus having a different equation on each node. However, given a tree, the result of the equation for each node with its own outputs o_i will be the same MSE value. This equation will be the basis for the calculation of the values of the constants that will appear in the tree.

As these four vectors (a,b,c,d) are different for each node of the tree, the first step of this algorithm is to calculate these vectors for each node. This process is recursively done from each non-terminal node to its child nodes, beginning with the root of the tree and going to each of its terminal nodes. For the root of the tree, $a_i = 1$, $b_i = t_i$, $c_i = 0$ and $d_i = -1$, leading to Eq. (4). From this node, for each non-terminal node, the four vectors of each of its two children are calculated. For each child, this calculation is done with the four vectors (a,b,c,d) of this non-terminal node and from the outputs o_i of the other child (whether

Table 2
Calculation of the a, b, c and d vectors for each node.

Operation	Left Child				Right Child			
	a'_i	b'_i	c'_i	d'_i	a'_i	b'_i	c'_i	d'_i
+	a_i	$b_i - a_i \cdot y_i$	c_i	$d_i - c_i \cdot y_i$	a_i	$b_i - a_i \cdot x_i$	c_i	$d_i - c_i \cdot x_i$
−	a_i	$b_i + a_i \cdot y_i$	c_i	$d_i + c_i \cdot y_i$	a_i	$a_i \cdot x_i - b_i$	c_i	$c_i \cdot x_i - d_i$
*	$a_i \cdot y_i$	b_i	$c_i \cdot y_i$	d_i	$a_i \cdot x_i$	b_i	$c_i \cdot x_i$	d_i
/	a_i	$b_i \cdot y_i$	c_i	$d_i \cdot y_i$	b_i	$a_i \cdot x_i$	d_i	$c_i \cdot x_i$

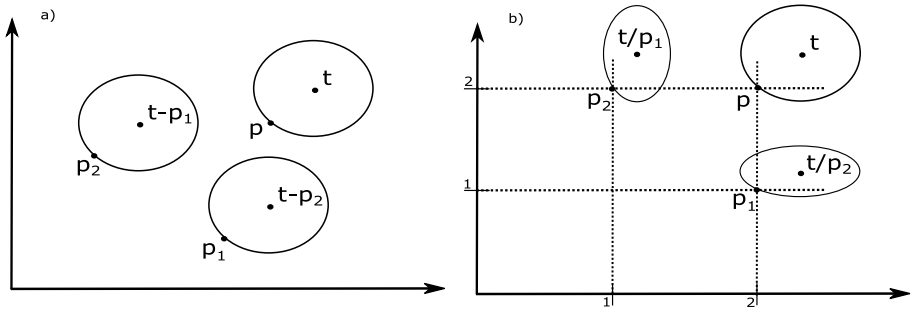


Fig. 2. Examples of calculating new shapes for the addition operation (a), and multiplication operation (b).

it is terminal or non-terminal). Table 2 shows the calculations to be performed to generate the vectors for each child of each non-terminal node containing an operation. In this table, a_i, b_i, c_i and d_i represent the coefficients of the non-terminal node, while a'_i, b'_i, c'_i and d'_i represent the coefficients of the child node being calculated. x_i and y_i are the outputs of the first and second child, respectively.

The rest of this subsection contains the mathematical basis used to generate the expressions shown in Table 2, as well as the geometric interpretation. Readers who are more interested in understanding the algorithm may skip this part and go directly to Section 2.4.

For each non-terminal node, the vectors for each child are calculated as follows:

- Addition operation. In this case, the output of the node is written as $o_i = x_i + y_i$, with x_i and y_i representing the outputs (semantics) of its two children. The equation for the first child becomes the following:

$$\frac{1}{N}\sum_{i=1}^{N}\left(\frac{a_i\cdot(x_i+y_i)-b_i}{c_i\cdot(x_i+y_i)-d_i}\right)^2=\frac{1}{N}\sum_{i=1}^{N}\left(\frac{a_i\cdot x_i-(b_i-a_i\cdot y_i)}{c_i\cdot x_i-(d_i-c_i\cdot y_i)}\right)^2 \quad (7)$$

which is like Eq. (6) with $a'_i = a_i$, $b'_i = b_i - a_i \cdot y_i$, $c'_i = c_i$ and $d'_i = d_i - c_i \cdot y_i$. For the second child, the equation is very similar:

$$\frac{1}{N}\sum_{i=1}^{N}\left(\frac{a_i\cdot(x_i+y_i)-b_i}{c_i\cdot(x_i+y_i)-d_i}\right)^2=\frac{1}{N}\sum_{i=1}^{N}\left(\frac{a_i\cdot y_i-(b_i-a_i\cdot x_i)}{c_i\cdot y_i-(d_i-c_i\cdot x_i)}\right)^2 \quad (8)$$

which is like Eq. (6) with $a'_i = a_i$, $b'_i = b_i - a_i \cdot x_i$, $c'_i = c_i$ and $d'_i = d_i - c_i \cdot x_i$.
If the operator sum is the root of the tree, then $a_i = 1$, $c_i = 0$, $d_i = -1$, and $b_i = t_i$. In this case, the resulting equations for the two children are the following

$$\frac{1}{N}\sum_{i=1}^{N}(x_i-(t_i-y_i))^2 \quad (9)$$

$$\frac{1}{N}\sum_{i=1}^{N}(y_i-(t_i-x_i))^2 \quad (10)$$

and can be interpreted for each child as "move the target value subtracting from it the value of the output of the other child" and apply the original equation. Thus, as was done in the root node, two spheres in the space are created, with centers in $t-y$ (for the first child) and $t-x$ (for the second child), i.e., the target values for the root of the tree have been moved to a different position for each of the children. If a new subtree is found which, applying the Eqs. (9) or (10) with its own outputs, return a lower value, then the MSE of the tree will be reduced. From another point of view, if a new subtree is found inside one of these spheres, the corresponding first or second child of the root can be replaced by this new subtree. Therefore, the semantics of the tree will move towards the target, having an improvement in the overall result. In general, from a shape given by Eq. (6) the sum operation creates two new identical shapes located in other points.
Fig. 2 (a) shows an example of a tree situated in p. This tree is $(p_1 + p_2)$. The calculation of the equations for each child leads to having a similar shape but moved according to the values of p_1 and p_2. The resulting shapes (in this case, spheres) still have the semantics in the border.

- Subtraction operation. This case is very similar to the previous one: the output of the node is written as $o_i = x_i - y_i$, with x_i and y_i as the outputs (semantics) of its two children. The equation for the first child becomes the following:

$$\frac{1}{N}\sum_{i=1}^{N}\left(\frac{a_i\cdot(x_i-y_i)-b_i}{c_i\cdot(x_i-y_i)-d_i}\right)^2=\frac{1}{N}\sum_{i=1}^{N}\left(\frac{a_i\cdot x_i-(b_i+a_i\cdot y_i)}{c_i\cdot x_i-(d_i+c_i\cdot y_i)}\right)^2 \quad (11)$$

which has the shape of Eq. (6) with $a'_i = a_i$, $b'_i = b_i + a_i \cdot y_i$, $c'_i = c_i$ and $d'_i = d_i + c_i \cdot y_i$. For the second child, the equation is:

$$\frac{1}{N}\sum_{i=1}^{N}\left(\frac{a_i\cdot(x_i-y_i)-b_i}{c_i\cdot(x_i-y_i)-d_i}\right)^2=\frac{1}{N}\sum_{i=1}^{N}\left(\frac{-a_i\cdot y_i-(b_i-a_i\cdot x_i)}{-c_i\cdot y_i-(d_i-c_i\cdot x_i)}\right)^2=\frac{1}{N}\sum_{i=1}^{N}\left(\frac{a_i\cdot y_i-(a_i\cdot x_i-b_i)}{c_i\cdot y_i-(c_i\cdot x_i-d_i)}\right)^2 \quad (12)$$

which has the shape of Eq. (6) with $a'_i = a_i$, $b'_i = a_i \cdot x_i - b_i$, $c'_i = c_i$ and $d'_i = c_i \cdot x_i - d_i$.
Again, if this operator is used as root of the tree, the sphere of the root will be moved to $t + y$ for the first child and $x - t$ for the second child, but the idea is the same: if a subtree is found inside any of these spheres, the corresponding child can be replaced with this subtree and the overall result will be improved. In general, the addition and subtraction operations creates two new identical shapes in other points.

- Multiplication operation. In this case, the output of the node is written as $o_i = x_i \cdot y_i$, with x_i and y_i as the outputs (semantics) of its two children. The equation for the first child becomes the following:

$$\frac{1}{N}\sum_{i=1}^{N}\left(\frac{a_i\cdot(x_i\cdot y_i)-b_i}{c_i\cdot(x_i\cdot y_i)-d_i}\right)^2=\frac{1}{N}\sum_{i=1}^{N}\left(\frac{(a_i\cdot y_i)\cdot x_i-b_i}{(c_i\cdot y_i)\cdot x_i-d_i}\right)^2 \tag{13}$$

which has the shape of Eq. (6) with $a'_i = a_i \cdot y_i$, $b'_i = b_i$, $c'_i = c_i \cdot y_i$ and $d'_i = d_i$. For the second child, the equation is very similar:

$$\frac{1}{N}\sum_{i=1}^{N}\left(\frac{a_i\cdot(x_i\cdot y_i)-b_i}{c_i\cdot(x_i\cdot y_i)-d_i}\right)^2=\frac{1}{N}\sum_{i=1}^{N}\left(\frac{(a_i\cdot x_i)\cdot y_i-b_i}{(c_i\cdot x_i)\cdot y_i-d_i}\right)^2 \tag{14}$$

If the multiplication operator is the root of the tree, then $a_i = 1$, $c_i = 0$, $d_i = -1$, and $b_i = t_i$. In this case, considering that these equations allow the calculation of the MSE, the resulting equations for the two children will consequently be the following:

$$\frac{1}{N}\sum_{i=1}^{N}(y_i\cdot x_i-t_i)^2=MSE \tag{15}$$

$$\frac{1}{N}\sum_{i=1}^{N}(x_i\cdot y_i-t_i)^2=MSE \tag{16}$$

These equations can be rewritten as

$$\sum_{i=1}^{N}\frac{\left(x_i-\frac{t_i}{y_i}\right)^2}{\left(\frac{SSE}{y_i}\right)^2}=1 \tag{17}$$

$$\sum_{i=1}^{N}\frac{\left(y_i-\frac{t_i}{x_i}\right)^2}{\left(\frac{SSE}{x_i}\right)^2}=1 \tag{18}$$

resulting in the equations of ellipsis. These equations can be interpreted for each child as "move the target value, and shrink or extend the radius of the sphere in each dimension, according to the other child outputs" and apply the original equation. Thus, the sphere of the root of the tree becomes ellipses in each of the two children. However, the reasoning is the same: if a new subtree is found inside these new shapes (ellipsis), then the result of the application of Eq. (15) or (16) will be equal to a lower MSE, and its semantics will move towards the target. In general, the multiplication operation creates new shapes in other points, which are distortions of this one.

Fig. 2 (a) shows an example of a tree situated in p. This tree is $(p_1 \cdot p_2)$, with $p_1 = (1, 2)$ and $p_2 = (2, 1)$. The calculation of the equations for each child leads to having the sphere translated and shrunk according to the values of p_1 and p_2. The resulting shapes are ellipses, and, as in the rest of the cases, have the semantics in the border.

The use of the operations of addition, subtraction, and multiplication makes it possible to build complex trees and having a semantic space with different N-dimensional ellipsoids. If a model is found inside one of these ellipsoids, the corresponding node can be changed with that model. When this is done, the semantics of the root of the tree gets closer to the target, the MSE lowers, the radius of the sphere of the root of the tree shrinks, and the rest of the spheres/ellipses also move from their places and shrink.

- Division operator. In this case, the output of the node is written as $o_i = x_i/y_i$, with x_i and y_i as the outputs (semantics) of its two children. The equation for the first child becomes the following:

$$\frac{1}{N}\sum_{i=1}^{N}\left(\frac{a_i\cdot(x_i/y_i)-b_i}{c_i\cdot(x_i/y_i)-d_i}\right)^2=\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\frac{a_i\cdot x_i-b_i\cdot y_i}{y_i}}{\frac{c_i\cdot x_i-d_i\cdot y_i}{y_i}}\right)^2=$$
$$=\frac{1}{N}\sum_{i=1}^{N}\left(\frac{a_i\cdot x_i-b_i\cdot y_i}{c_i\cdot x_i-d_i\cdot y_i}\right)^2 \tag{19}$$

which has the shape of Eq. (6) with $a'_i = a_i$, $b'_i = b_i \cdot y_i$, $c'_i = c_i$ and $d'_i = d_i \cdot y_i$. For the second child, the equation is:

$$\frac{1}{N}\sum_{i=1}^{N}\left(\frac{a_i\cdot(x_i/y_i)-b_i}{c_i\cdot(x_i/y_i)-d_i}\right)^2=\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\frac{a_i\cdot x_i-b_i\cdot y_i}{y_i}}{\frac{c_i\cdot x_i-d_i\cdot y_i}{y_i}}\right)^2=$$
$$=\frac{1}{N}\sum_{i=1}^{N}\left(\frac{b_i\cdot y_i-(a_i\cdot x_i)}{d_i\cdot y_i-(c_i\cdot x_i)}\right)^2 \tag{20}$$

which has the shape of Eq. (6) with $a'_i = b_i$, $b'_i = a_i \cdot x_i$, $c'_i = d_i$ and $d'_i = c_i \cdot x_i$.

If this operator is the root of the tree, then $a_i = 1$, $c_i = 0$, $d_i = -1$, and $b_i = t_i$. In this case, the resulting equations for the two children are the following:

$$\frac{1}{N}\sum_{i=1}^{N}\left(\frac{x_i-t_i\cdot y_i}{y_i}\right)^2=\frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{y_i}\cdot x_i-t_i\right)^2 \tag{21}$$

$$\frac{1}{N}\sum_{i=1}^{N}\left(\frac{-t_i\cdot y_i+x_i}{y_i}\right)^2=\frac{1}{N}\sum_{i=1}^{N}\left(\frac{x_i}{y_i}-t_i\right)^2 \tag{22}$$

In the first case, the effect is similar to the multiplication operation: the target value for the first child is moved, and the radius is extended/shrunk according to the values of the second child. In the second case, the sphere is turned into a completely different shape. Therefore, the division operation can transform ellipsoids into different shapes.

The third column of Table 1 shows the equations calculated for each of the nodes of the tree shown in Fig. 1. The vectors of these equations are calculated in the opposite direction as the semantics, i.e., from the root of the tree to the terminal nodes. The target vector used was (5,4,1), which allows the construction of the equation of node 1. Using the semantics of both children (x for the first child, node 2, and y for the second child, node 3), the vectors a, b, c, and d of nodes 2 and 3 can be calculated with the expressions shown in Table 2. It should be highlighted that, to calculate the vectors of node 2, the semantics of node 3 are used, but not its own semantics, and vice versa. Recursively, this process is repeated for each non-terminal node of the tree. Note also that the vectors of each node are calculated from its parent node. For this reason, the terminal nodes do not calculate any coefficients since they do not have children.

An alternative formulation can be used by this system. Instead of using Eq. (6) to calculate the MSE in each node, the following equation could be used:

$$MSE=\frac{1}{N}\sum_{i=1}^{N}\left(\frac{a'_i\cdot o_i-b'_i}{c'_i\cdot o_i-d'_i}-t_i\right)^2 \tag{23}$$

what's another way of calculating MSE?

Again, o_i are the outputs of that node with i identifying each of the N input patterns, while a', b', c' and d' are the four vectors characterizing the equation for each node. In this alternative formulation in the root of the tree $a'_i = 1$, $b'_i = 0$, $c'_i = 0$ and $d'_i = -1$, leading to Eq. (4). These four vectors and the S set can be calculated following the same steps of the previous formulation. Both ways of performing the calculations are equivalent, since Eq. (23) can be rewritten as

$$\frac{1}{N}\sum_{i=1}^{N}\left(\frac{a'_i\cdot o_i-b'_i-t_i\cdot(c'_i\cdot o_i-d'_i)}{c'_i\cdot o_i-d'_i}\right)^2$$
$$=\frac{1}{N}\sum_{i=1}^{N}\left(\frac{(a'_i-c'_i\cdot t_i)\cdot o_i-(b'_i-d'_i\cdot t_i)}{c'_i\cdot o_i-d'_i}\right)^2 \tag{24}$$

and thus Eq. (23) takes the shape of Eq. (6) with $a_i = a'_i - c'_i \cdot t_i$, $b_i = b'_i - d'_i \cdot t_i$, $c_i = c'_i$ and $d_i = d'_i$.

The difference between this form of representation and the one used in this paper is that the equation used here makes it possible to calculate the MSE from the output of each node, while this new

what's is the difference between the two equations we can use to calculate the SME from the output of each one?

formulation allows the calculation of the global output of the tree from the output of each node, and from there to calculate the MSE. However, to calculate the derivatives explained below in Section 2.5.1 in a simpler way, the previous formulation is used.

2.4. Avoiding out-of-domain operations

The division operator is different from the rest for a very important reason: in the other three, the domain is all $\mathbb{R}$. However, in the division operator, the domain is restricted, and must therefore be used with care. For example, Fig. 1 shows an example of a tree with the division operator. The expression representing this tree is shown on Eq. (1), and it can be simplified to the following expression:

$$\frac{x_3 - 2}{2 \cdot x_3 - 1} \tag{25}$$

However, both expressions are different. This last expression is defined in any value of x_3 that belongs to $\mathbb{R} - \{0.5\}$. In Eq. (1) the values of x_3 that are not in the domain are 0, 2 and 0.5. These values are forbidden because they would produce a division by 0 somewhere in the tree.

These "forbidden" values are produced by the way the equation for the second child of the division operator is constructed, shown in Eq. (20). In that equation, this step

$$\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\frac{a_i \cdot x_i - b_i \cdot y_i}{y_i}}{\frac{c_i \cdot x_i - d_i \cdot y_i}{y_i}}\right)^2 = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{a_i \cdot x_i - b_i \cdot y_i}{c_i \cdot x_i - d_i \cdot y_i}\right)^2 \tag{26}$$

is correct for values of $y_i \neq 0$. Therefore, for that node, any semantics in the second child with any $y_i = 0$ is outside the domain of this node. For instance, a node with semantics of $(1, -2, 0)$ cannot be used as a second child. Moreover, as the second child can be a tree, its semantics depends on its nodes and operations, since the result of these operations can lead to having semantics with any $y_i = 0$. For example, in the tree shown in Fig. 1, node 4 represents a division and if the variable x_3 takes the value of 2, then the denominator of this division operator will take the value of 0.

Therefore, when calculating the equations, it is necessary to take into account the domains of the operators already explored from the top of the tree. Since the +, - and * operators are defined in all $\mathbb{R}$, only the division operator has to be taken into account. The way to do this is to extend the definition of the equations of each node. Until now, these operations were defined with four vectors: a, b, c, and d. In addition to these vectors, it is necessary to add a set S with those semantics that would lead to any out of domain operation, in this case divisions by zero, in some upper node of the tree. For the root node of the tree, $a_i = 1$, $b_i = t_i$, $c_i = 0$ and $d_i = -1$, and also $S = \emptyset$.

As for the vectors a, b, c and d, the values of the S set must be calculated for each node. The process is similar to the one described: each non-terminal node makes the calculation of a_i, b_i, c_i, d_i and S for each child. Therefore, at the same time a'_i, b'_i, c'_i and d'_i are calculated for each child node, a new S' set is calculated for this node.

The rest of this subsection contains the mathematical basis used to calculate the S set for each node of the tree. The reader most interested in understanding the algorithm may skip this part and go directly to Section 2.5.

Given a node with an operation and a S set, for each child the calculation method follows these steps:

1. $S' = \emptyset$
2. For each semantics $s \in S$, calculate s' depending on the node operation. The way to calculate for one child is to perform the inverse of the node operation with respect to the other child. If x and y are the two semantics of the first and second children respectively, the process is, for each operation and each child:

   - Addition operation:
     - First child: $s = s' + y \Rightarrow s' = s - y$.
     - Second child: $s = x + s' \Rightarrow s' = s - x$.
   - Subtraction operation:
     - First child: $s = s' - y \Rightarrow s' = s + y$.
     - Second child: $s = x - s' \Rightarrow s' = x - s$.
   - Multiplication operation:
     - First child: $s = s' \cdot y \Rightarrow s' = s/y$.
     - Second child: $s = x \cdot s' \Rightarrow s' = s/x$.
   - Division operation:
     - First child: $s = s'/y \Rightarrow s' = s \cdot y$.
     - Second child: $s = x/s' \Rightarrow s' = x/s$.

   Once s' is calculated, add it to S'. Since s, s', x and y are vectors, the operations described are element-wise operations.
3. In the case of the second child of the division operation, add the semantics $s'_i = 0$ to S', since these semantics is not in the domain of this operation.

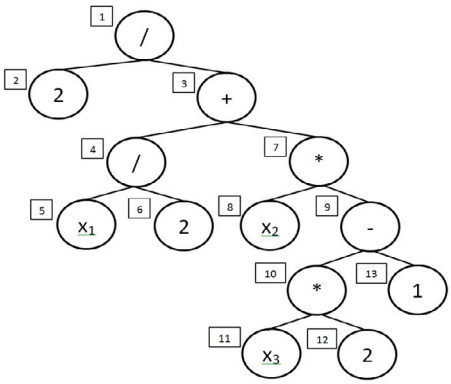


Fig. 3. Example of a tree with invalid operations.

For each node, the set S contains the semantics that make any operation of any previous node with an out of domain value, i.e., each s semantics in S has "forbidden" values. It is important to highlight that, if $s \in S$ has semantics of "forbidden" values s_i, invalid semantics x will be those semantics in which any value $x_i = s_i$, i.e., not all of the values have to be equal.

The fourth column of Table 1 shows the S sets for each node of the tree shown in Fig. 1. Nodes 1 and 2, before any division operation was performed, have $S = \emptyset$. Nodes 3, 4, 5 and 11, when only one division has been performed, have one element in S. Nodes 6, 7, 8 and 9, when 2 divisions have been performed, have 2 elements in S. Finally, when the calculation of the equations of node 10 is reached, 3 different divisions have been done, and therefore the number of elements in S is 3. Note that each node used as the divisor (nodes 3, 6 and 10) has the element (0,0,0) in S, meaning that no value of their semantics can be 0.

Another example is shown in Fig. 3, which has its semantics and S sets described in Table 3. Note that this tree is invalid since divisions by zero are performed. Thus, this tree will never be generated by this system. However, it is used here as an example to better explain the following descriptions.

Special care has to be taken when calculating each semantic value s' from s for the multiplication and division operators. In this case, divisions by zero could take place in the following situations:

- In the case of the multiplication operation, for the first child when any element of y_i (semantics of the second child) is 0 or, for the second child, when any element of x_i (semantics of the first

Table 3
Description of the tree of the example with invalid operations.

Node	Semantic	S
1	$(1.33, Inf, Inf)$	$\emptyset$
2	$(2, 2, 2)$	$\emptyset$
3	$(1.5, 0, 0)$	$\{(0, 0, 0)\}$
4	$(1.5, 1, 0)$	$\{(0, 1, 0)\}$
5	$(3, 2, 0)$	$\{(0, 2, 0)\}$
6	$(2, 2, 2)$	$\{(0, 0, 0), (Inf, 2, NaN)\}$
7	$(0, -1, 0)$	$\{(-1.5, -1, 0)\}$
8	$(0, 1, 0)$	$\{(-1.5, 1, 0)\}$
9	$(1, -1, 5)$	$\{(Inf, -1, NaN)\}$
10	$(2, 0, 6)$	$\{(Inf, 0, NaN)\}$
11	$(1, 0, 3)$	$\{(Inf, 0, NaN)\}$
12	$(2, 2, 2)$	$\{(Inf, Inf, NaN)\}$
13	$(1, 1, 1)$	$\{(Inf, 1, NaN)\}$

child) is 0, i.e., a multiplication of $0 \cdot y_i$ or $x_i \cdot 0$ is being made. In both cases, the result of the multiplication is equal to 0, and the calculations of the new out of domain values are $s' = s/y$ (for the first child) and $s' = s/x$ (for the second). Two situations are possible:

  - $s_i \neq 0$: the "forbidden value" is not 0. In this case, s_i/x_i or s_i/y_i equals infinite (Inf). This means that any value is valid for x_i or y_i because $x_i \cdot 0 = 0 \neq s_i$ or $0 \cdot y_i = 0 \neq s_i$ and therefore they are never out of the domain. An example of this can be seen in Fig. 3 and Table 3, in the S set of node 9 (first value of the only element).
  - $s_i = 0$: the "forbidden value" is 0. In this case, s_i/x_i or s_i/y_i equals "not a number" (NaN). This means that there are not valid values for x_i and y_i, since for any x_i or y_i values, $x_i \cdot 0 = 0 = s_i$ or $0 \cdot y_i = 0 = s_i$, and therefore they are always out of the domain. An example of this can be seen in Fig. 3 and Table 3, in the S set of node 9 (third value of the only element).

- In case of the division operation, for the first child, the reasoning is the same as in the multiplication operation.
- For the second child of the division operation, the operation for calculating the new "forbidden" semantic values is $s' = x/s$. Division by zero will take place when any $s_i = 0$: the "forbidden value" of the result of the division operation is 0. Two situations may occur:

  - $x_i \neq 0$. In this case, the result of this operation is $s'_i = x_i/s_i = Inf$, meaning that any y_i value is valid, since $x_i/y_i \neq s_i = 0$ if $x_i \neq 0$.
    This can be argued since a value of $y_i = 0$ is not valid for this division. However, the calculation of each s' corresponds to values out of the domain of previous division operators, not for this one. For the current division operation, as explained above, the semantics $s' = 0$ is added to S, and therefore the value of $y_i = 0$ is not valid.
    An example of this can be seen in Fig. 3 and Table 3, in the S set of node 6 (first value of the second element).
  - $x_i = 0$. In this case, the result of the division operation is $x_i/y_i = 0 = s_i$. The calculation of s'_i is the following: $s'_i = x_i/s_i = 0/0 = NaN$, meaning that there no valid values for y_i, i.e., for any value y_i, $x_i/y_i = 0 = s_i$.
    Again, these calculations are valid for $y_i \neq 0$. The case $y_i = 0$ is excluded with the addition of the semantics $s'_i = 0$ to S;
    An example of this can be seen in Fig. 3 and Table 3, in the S set of node 6 (third value of the second element).

Therefore, for each $s \in S$, a value labeled as Inf means that any value is valid, while a value labeled as NaN means that any value is invalid. The operations between these values are done as usual when $x \in \mathbb{R}$:

- Any operation between x and NaN has NaN as a result. This means that, if any value is out of domain in a specific node, then any value will also be out of domain in each child, no matter what operation is done in this node. An example of this can be seen in Fig. 3 and Table 3, in the S sets of nodes 10 and 13 (third value of the only element). Also, node 12 shows another example (third value of the only element).
- Addition and subtraction operations between x and Inf return Inf. This means that, if any value is valid (the domain is $\mathbb{R}$) for the result of the operation, then any value is also valid for each child. An example of this can be seen in Fig. 3 and Table 3, in the S sets of nodes 10 and 13 (first value of the only element).
- $s'_i = Inf/x_i = Inf$ ($x_i = 0$ or $x_i \neq 0$). This can happen when s' is being calculated as one of the children in a multiplication operation. This means that, if any value is valid as result of a multiplication operation (Inf), then each child can have any value. An example of this can be seen in Fig. 3 and Table 3, in the S set of node 12 (first and second values of the only element).
- $s'_i = x_i/Inf = 0$ ($x_i = 0$ or $x_i \neq 0$). This can happen when s' is being calculated as the second child in a division operation (i.e., the S set of the denominator) with the semantics of the first child x_i. This means that, even any value (denoted as Inf) is valid as a result of a division operation in which the numerator is a valid number, the denominator must be different from 0 (0 is not in the domain).
- $s'_i = Inf \cdot y_i = Inf$ when $y_i \neq 0$. This can happen when s' is being calculated as the first child in a division operation (i.e., the S set of the numerator) with the semantics of the denominator $y_i \neq 0$. This means that if the result of a division operation can be any value (Inf) and the denominator is different from 0, then the first child can take any value in $\mathbb{R}$.
- $s'_i = Inf \cdot y_i = NaN$ when $y_i = 0$. This can happen when s' is being calculated as the first child in a division operation (i.e., the S set of the numerator) with the semantics of the denominator $y_i = 0$. This means that even if the result of a division operation can be any value, if the second child evaluates as 0, then there is no valid value for the first child.
- Operations between NaN and/or Inf values will not happen because these operations take place between the semantics of the S set, which may contain NaN and Inf values, and semantics of the nodes of the tree, which may not. The definition of the S set is done to prevent the semantics of the nodes from having NaN and Inf values.

Any semantics x to be applied to Eq. (6) of any node in order to compute the MSE from that node must be first checked with its S set. For each semantics s in S, if there is any value in which $s_i = x_i$ or $s_i = NaN$, then those semantics cannot be used because it would lead to having out-of-domain values in preceding nodes. When any $s_i = NaN$, since there are no valid values, the whole S set is labeled as NaN for the sake of simplicity.

It is important to highlight that the use of this S set ensures that no divisions by zero are being performed within the tree with the training set. Therefore, the result of the evaluation of the training set will never be Inf or NaN. However, these values can appear in the evaluation of new data (the test set). For example, the expression $1/(x_1 - 10)$ does not return Inf if the variable x_1 in the training set does not take any value of 10. However, when evaluating this tree with a new data sample in which $x_1 = 10$, the result will be Inf. Since the future values to be taken by variables cannot be known, in this system, the selection of a consistent training set that represents the future data that will be used by the resulting expressions is very important.

2.5. Local search on the nodes of the tree

The evaluation process of the tree goes from the bottom of the tree to the top. Once the tree has been evaluated, the four vectors (a,b,c,d) of Eq. (6) and the S set are calculated for each node. This process goes from the top, with values of $a_i = 1$, $c_i = 0$, $d_i = -1$, $b_i = t_i$ and $S = \emptyset$, to the bottom of the tree, following the described equations.

Once these values have been found for each node, the search for subtrees that can substitute a node begins. For any subtree that could substitute a node, first its semantics are checked with the S set of the node. If all of the values are valid, then the semantics are evaluated on Eq. (6) of that node. This subtree can replace the node if the result of the following equation is positive:

$$MSE - \frac{1}{N}\sum_{i=1}^{N}\left(\frac{a_i \cdot o_i - b_i}{c_i \cdot o_i - d_i}\right)^2 \tag{27}$$

If there are several subtrees that could replace a node, the selected subtree will be the one with the highest positive value in this equation. The search for subtrees has 4 different methods: search for constants (constant search), search for variables (variable search), search for constants combined with variables (constant-variable search) and search for constants combined with expressions (constant-expression search). These searches do not take place in those nodes in which $S = NaN$, since there are no valid values. However, since this method always builds correct trees, no nodes with $S = NaN$ are expected. The following subsections discuss each of these searches.

2.5.1. Search for constants

One of the biggest problems in GP is the generation of constants. In the first approaches, the generation of constants was left to the evolutionary process. An ephemeral random constant was included in the terminal set, so each time it was selected in the building of a tree, a random constant in a predefined interval was generated. The building of a useful value was left to the evolutionary process, by a successive combination of these random constants. However, there are some approaches using gradient descent (Chen et al., 2015)(Kronberger et al., 2018) to optimize these constants.

Within a tree, constants take place as a terminal node. Therefore, they can be seen as trees with a single node, representing its constant value, and therefore they are also considered as a model, with its semantics. The particularity of the semantics of a constant k is that all the elements of the semantic vector take the same value: $o_i = k$. In other words, the semantics of all of the constants are situated on the line, $span(1, 1, \ldots, 1)$.

Given a node of the tree, with its corresponding equation, the objective is to find the constant that maximizes Eq. (27), i.e., the constant k that minimizes the following equation:

$$\frac{1}{N}\sum_{i=1}^{N}\left(\frac{a_i \cdot o_i - b_i}{c_i \cdot o_i - d_i}\right)^2 = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{a_i \cdot k - b_i}{c_i \cdot k - d_i}\right)^2 \tag{28}$$

The way to find this constant is to calculate the derivative of this expression, set it equal to zero, and calculate that value k. As the original equation is a sum of squared expressions, any value of k that makes the derivative equal to 0 will be a minimum, since Eq. (28) does not have a maximum. The derivative of this expression is the following equation:

$$\frac{2}{N}\sum_{i=1}^{N}(b_i \cdot c_i - a_i \cdot d_i)\frac{a_i \cdot k - b_i}{(c_i \cdot k - d_i)^3} \tag{29}$$

In general, calculating the values in which this expression becomes 0 is a time-consuming task, because it involves building a 3N-order polynomial and finding its roots, with N possibly being very high. Also, there are many possible values for k. However, this calculation can be simplified in some common situations:

- $c_i = 0$ and $a_i = 0$ (i=1, …,N). In this case, the function is constant and has no minimum.
- $c_i = 0$ (i=1, …,N) and any $d_i = 0$. In this case, the function cannot be calculated and therefore there is no minimum.
- $c_i = 0$ and $d_i \neq 0$ (i=1, …,N). In this case, the only minimum value of k is given by:

$$k = \frac{\sum_{i=1}^{N}\frac{a_i \cdot b_i}{d_i^2}}{\sum_{i=1}^{N}\frac{a_i^2}{d_i^2}} \tag{30}$$

  This equation can be simplified in the following situations:

  – a_i are constants ($a_i = k_a$) and d_i are constants ($d_i = k_d$). The case with $k_d = -1$ happens when no division has been performed yet. The minimum k is given by:

$$k = \frac{1}{k_a \cdot N}\sum_{i=1}^{N} b_i \tag{31}$$

  – a_i are constants ($a_i = k_a$) and d_i are not constants. The minimum k is given by:

$$k = \frac{1}{k_a}\frac{\sum_{i=1}^{N}\frac{b_i}{d_i^2}}{\sum_{i=1}^{N}\frac{1}{d_i^2}} \tag{32}$$

  – a_i are not constants and d_i are constants ($d_i = k_d$). The minimum k is given by:

$$k = \frac{\sum_{i=1}^{N} a_i \cdot b_i}{\sum_{i=1}^{N} a_i^2} \tag{33}$$

- Any $c_i = 0$ and $d_i = 0$ (i=1, …,N). In this case, the function cannot be calculated and therefore has no minimum.
- $c_i \neq 0$ and $d_i = 0$ (i=1, …,N). In this case, the only minimum value of k is given by:

$$k = \frac{\sum_{i=1}^{N}\frac{b_i^2}{c_i^2}}{\sum_{i=1}^{N}\frac{a_i \cdot b_i}{c_i^2}} \tag{34}$$

  As happens with Eq. (30), this equation can be simplified for the cases in which b_i and/or c_i are constants.
- c_i and d_i are constants $c_i = k_c$ and $d_i = k_d$ (i=1, …,N), and $k_c \neq 0$ and $k_d \neq 0$. In this case, the only minimum value of k is given by:

$$k = \frac{\sum_{i=1}^{N} b_i \cdot (b_i \cdot k_c - a_i \cdot k_d)}{\sum_{i=1}^{N} a_i \cdot (b_i \cdot k_c - a_i \cdot k_d)} \tag{35}$$

In any of these cases, once the minimum k has been found, it has to be checked against the S set. For each semantics $s \in S$, if any value $s_i = k$, then this search is unsuccessful.

If none of these situations occur, as was stated, the finding of the minimum values can be time-consuming. However, in this paper, we propose the alternative solution of evaluating a set of points that can have a low value in Eq. (6). These values are the zeros on each term inside the sum. These values are given by b_i/a_i. In general, the minimum value of Eq. (6) will not be any of these values. However, one of them will take a close value. To avoid rounding errors in these calculations, the zeros with values close to any poles (d_i/c_i) are excluded.

Therefore, in the general case the process is as follows:

1. Take the N $z_i = b_i/a_i$ zero values.
2. Compare each z_i with all the values of each $s \in S$. If there is any coincidence, delete z_i, since it is not valid.
3. Exclude the values that are too close to any root of the denominator of Eq. (6).

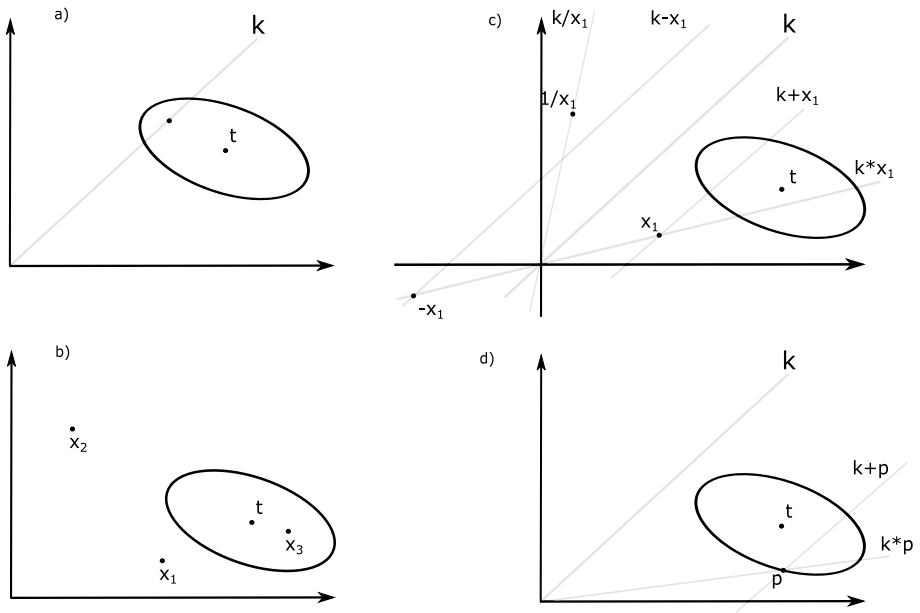


Fig. 4. Examples of (a) constant search, (b) variable search, (c) constant-variable search, (d) constant-expression search.

4. Evaluate the remaining z_i with Eq. (6) and select the one with the lowest value (lowest MSE) as k.

Once a value of k has been found for a node, if the result of Eq. (27) is positive, then this search was successful and a single-node terminal tree representing this constant can substitute this node in the tree, leading to an improvement in the MSE.

This process allows the creation and refinement of constants. However, this is not limited to changing the value of one constant for another (i.e., one terminal node representing a constant for another terminal node representing a constant). It is important to highlight that this process of finding a constant can be performed for each node of the tree to be improved, whether it is a constant, variable or a non-terminal node. Users may decide on which nodes they want this search to be performed. Therefore, the node selected to be replaced by a constant can be a non-terminal one. In this case, if this search is successful, the tree is being reduced, resulting in a lower number of nodes.

In general, it has been found that as a result of this search, most of the times a constant is changed into another. Therefore, this search technique is very useful for refining the constant values of the nodes of the tree. Sometimes a variable or a non-terminal node (thus reducing the tree) is changed by a constant. However, since the occurrence of these situations is rare, much computational time can be saved by performing this search only on constant nodes, and exploring the rest of the nodes only when needed. Users can decide whether they want to perform this search only on constant nodes or perform a full search on all of the nodes of the tree.

Fig. 4 (a) shows an example of a constant search. The shape given by Eq. (6) has an intersection with the gray line defined as $span(1, 1, \ldots, 1)$. Any constant node is on this line. Therefore, any constant node on this line and inside the ellipsis would give a lower MSE than the given tree. This method returns the constant that returns the lowest MSE for this shape.

2.5.2. Search for variables

This process is easier to calculate than the previous one. As in constants, each variable is represented by a terminal node, with its semantics. Therefore, variables are also points in the search space.

Given a node of the tree, with its Eq. (6) and its S set, the first thing to do is to check all the variables to see whether their semantics are valid with the S set. For those variables with valid semantics, Eq. (27) is evaluated with their semantics. Again, a positive value means a reduction in MSE. These are variables inside the shape of that node in the semantic space. The selected variable will be the one with the highest reduction in MSE.

As in the previous search process, any node of a tree can be selected to be changed with a variable, regardless of whether this node is constant, variable or non-terminal. Therefore, the number of nodes in the tree cannot grow with this method, and the tree can be reduced. As in the previous search, users may decide on which nodes they want to perform this search. Obviously, if this search is being performed on variable nodes, those variables equal to the nodes being explored are excluded.

Also, as what happened with the constant search, it has been found that this search is successful most of the times on constant nodes, and thus changing constant to variables. Therefore, computational time can be saved if this search is performed only on constant nodes, and exploring the remaining nodes only when needed.

Fig. 4 (b) shows an example of a variable search. The variable x_3 was found to be inside the shape given by Eq. (6). Therefore, the given tree can be replaced by this variable, thereby improving MSE. Variables x_1 and x_2 do not lead to improving MSE.

2.5.3. Search for variables combined with constants

With the two search methods already described, it is possible to reduce the number of nodes of the tree. However, it is often necessary to find larger and more complex trees to get closer to the solution. This subsection allows the finding of simple subtrees with 3 nodes to replace another node of the tree. If the node to be replaced is a terminal node, then the tree will grow.

The idea behind this search method is a combination of the two previous methods. After the calculation of each equation on the nodes of the tree, it is possible not to find a constant or a variable that improves the MSE (a constant or a variable inside any of the shapes in the space). However, although a variable x is not inside any of the shapes, the models $k + x$, $k - x$, $k \cdot x$ or k/x may be inside one of the shapes, for any value of k. Each of these four expressions represent, a line in the semantic space. This search method will look for the intersection of one of these lines with each of the shapes. Looking for intersections is equivalent to looking for the best value of k. Therefore, this third method proposes the search of a variable combined with a constant, with one of the four arithmetic operations. For each variable x, the possibilities are:

- Addition operation. Although the variable x represents a single point in the search space, the expression $(k + x)$ represents a line in the search space. This line is parallel to the line with the constants. Therefore, if the variable x is not inside any of the shapes, any section of the line $(k + x)$ may be inside the shapes. The expression $(x + k)$ is represented by a tree with three nodes: a non-terminal node representing the sum, and two nodes, with the constant k and the variable x.
  The objective here is, given a node with its equation calculated, to find a constant value that maximizes Eq. (27) for the semantics of the tree $(k + x)$. To do this, a new equation is calculated from the a_i, b_i, c_i, d_i and S values of the equation of this node. As the constant is going to be situated as the first child of the addition operation, this new equation is calculated as in Eq. (9), with $a'_i = a_i$, $b'_i = b_i - a_i \cdot x_i$, $c'_i = c_i$ and $d'_i = d_i - c_i \cdot x_i$, where x_i are the values of the variable x for each data sample. Also, S' is calculated from S with the method previously described for the addition operator. With this new equation, the constant optimization process described in Section 2.5.1 is performed. As a result, an improvement value is returned, a positive one indicates that this node can be replaced with $(k + x)$.
- Subtraction operation. As in the previous case, the expression $(k - x)$ represents a line in the semantic space, parallel to the constant line, and parallel to the $(k + x)$ line. It should be highlighted that in the previous case the position of the constant and variable is indifferent: $(k + x)$ and $(x + k)$ leads to the same expression. However, in this case, the constant must be the first argument of the subtraction operation, and the variable the second. Otherwise, we would be again in the previous case. To keep coherence, in all these four cases the constant is going to be the first argument.
  The process is similar in all of the four cases: given a node with an equation represented by a_i, b_i, c_i, d_i and S, a new equation is calculated, this time from Eq. (19): with $a'_i = a_i$, $b'_i = b_i + a_i \cdot x_i$, $c'_i = c_i$, $d'_i = d_i + c_i \cdot x_i$, and S' calculated from S with the method previously described for the subtraction operation Once this equation has been calculated, the constant optimization process is performed.
- Multiplication operation. In this case, the expression $(k \cdot x)$ represents a line in the search space in which the vector x is included, i.e., all of the vectors in $(k \cdot x)$ are collinear to x. The objective here is the same: find the value of k that minimizes Eq. (6).
  Given a node of the tree with an equation represented by a_i, b_i, c_i, d_i and S, a new equation is calculated, this time from Eq. (13): $a'_i = a_i \cdot x_i$, $b'_i = b_i$, $c'_i = c_i \cdot x_i$, $d'_i = d_i$, and S' calculated from S with the method previously described for the multiplication operation. With this equation, the previous constant search process is undergone, having as result a value of reduction in MSE.
- Division operation. In this case, the expression (k/x) represents a line in the search space in which the vector given by the values $1/x_i$ is included, i.e., all of the vectors in (k/x) are collinear to the vector given by $1/x_i$. With the same objective as in the previous cases, and given a node, from the equation of this node a new one is calculated, with Eq. (19): $a'_i = a_i$, $b'_i = b_i \cdot x_i$, $c'_i = c_i$, $d'_i = d_i \cdot x_i$, and S' calculated from S with the method previously described for the division operation. The same constant optimization is performed.
  As in the subtraction operation, this operation does not allow changing the order of the children. If (x/k) was chosen instead of (k/x), then an operation similar to the previous one (constant-variable search with multiplication operation) will be being performed, resulting in the value of $1/k$.

In this search, for each of the four operations, a set S' has to be calculated from the S set of the node. If $S' = NaN$, then the constant search for that operation, node and variable does not take place, since there are no possible values for the constant. A particular case happens in the division operator when the variable has any 0 in its semantics. In this case, the result of the calculation of S' is NaN, excluding variables with 0 values in semantics from being the second child of a division operation.

This process can be done for each node of the tree, whether it is a constant, variable or non-terminal. As a result, the combination selected is the one that returns a higher value in Eq. (27), in case it is higher than 0. This combination states which node can be changed, and which 3-node subtree can be inserted in its place. The node to be changed can be terminal or non-terminal, so this search process can lead to having the tree increase or decrease the number of nodes. However, it has been observed that most of the time this search is successful only on terminal nodes (constants and variables), and sometimes on non-terminal nodes. As in previous searches, performing this search only on terminal nodes may save computational time.

Fig. 4 (c) shows an example of a constant-variable search. The shape given by Eq. (6) does not have an intersection with the gray line defined as $span\{(1, 1, \ldots, 1)\}$, and there are not variables inside this shape. Therefore, constant and variable searches would not be successful. However, from the variable x_1 four different lines can be defined: $k+x_1$, $k-x_1$, $k \cdot x_1$ and k/x_1. In this case, the gray lines $k+x_1$ and $k \cdot x_1$ intersect the shape, so in both cases, a constant can be calculated to have an expression $k + x_1$ or $k \cdot x_1$ that improves MSE. The given tree would be replaced with this expression.

It is important to highlight that this search must be used carefully so that it does not "hide" a constant search. For instance, the branch "$(3.2 + x_3)$" can be selected for this search. As a result, this node could be changed by "$(3.5 + x_3)$" in this constant-variable search. However, in practice the constant node was changed from a value of 3.2 to a value of 3.5. For this reason, for a specific variable and each of the four operations (+,-,*,/), this search is not performed on nodes representing the same operation, with a constant as the first child and the same variable as the second child.

Moreover, even with the described precaution, it is possible to use this operation to "hide" a constant search. Another example could be in the tree $(3.4 + x_2)$, in which x_2 could be replaced by $(0.1 + x_2)$. This has as result the tree $(3.4 + (0.1 + x_2))$, which has its semantics equivalent to $(3.5 + x_2)$, and could be generated by a constant search. Another example could be $((1.2 \cdot x_1) + x_1)$. In this tree, the second x_1 could be changed into $(0.5 \cdot x_1)$, and the resulting tree would be $((1.2 \cdot x_1) + (0.5 \cdot x_1))$, which has its semantics equivalent to $(1.7 \cdot x_1)$ or $((0.7 \cdot x_1) + x_1)$. This last tree could be found from the original one, by performing a constant search on the constant leaf.

In general, it is hard to determine when a constant-variable search can "hide" a constant search. This situation makes the trees unnecessarily large, with possibly a high number of nodes (see Section 2.6). If a constraint to the size of the tree is set, then the algorithm could be prematurely stopped because of having too many nodes that could be simplified.

One way to avoid this problem is, before this search takes place, to optimize all of constants. This process is described in Section 2.6, and is based on constant search.

Another possibility is to perform a constant search at the same time this search is performed. With this approach, a modification of a constant and a constant-variable search that modifies the same constant will have the same MSE reduction. The first of them (constant search) would be chosen to modify the tree.

2.5.4. Search for expressions combined with constants

The previous search allows the tree to grow by replacing terminal nodes with small branches. However, once a part of the tree has been built, it will unlikely be modified, and its modification involves the deletion of a branch and substitution by a constant, variable or constant-variable branch. This can be undesirable since the deleted part of the tree has proved to be useful. Therefore, a method that allows the modification of a branch without deleting it is desirable.

This search allows the tree to grow by modifying a non-terminal node of the tree. This modification is done as in the previous cases: finding a better branch and replacing the whole subtree with this branch. However, in this case, the branch contains the previous subtree, so it is not deleted but modified. For instance, a node $2.5/(3+x_2)$ can be replaced by the branch $(3.4+2.5/(3+x_2))$, in which the second child is the previous subtree.

The idea behind this search method is similar to the constant-variable search. In this case, the node selected represents a point p in the semantic space in the border of the MSE shape. There may be a constant k that makes $(k+p)$ inside the MSE shape, thus improving the MSE. As in the previous search, $(k+p)$ is the line that intersects p, parallel to the constant line (1,1, ...,1). Similarly, there may be a constant k that makes $(k \cdot p)$ inside the MSE shape. $(k \cdot p)$ is the line that goes through p and (0,0, ...,0). In both cases, finding a value for k means finding a point in the corresponding line inside the MSE shape. Other approaches such as $(k-p)$ or (k/p) are not explored, because they do not begin in the border of the MSE shape and thus are not likely to return an improvement in MSE. In the first case, $(k-p)$, the line goes though $-p$ instead of p, and in the second case, (k/p), the line goes through $1/p$ instead of p.

Contrary to the previous searches, this one is performed only on non-terminal nodes, because when it is performed on a terminal node:

- If this terminal node is a constant, then a constant search is being performed.
- If this terminal node is a variable, then a constant-variable search is being performed.

The behavior of this search is very similar to the constant-variable search. In the case of performing a $(k+p)$ search on a node p with its equation, a constant value that maximizes Eq. (27) for the semantics of the tree $(k+p)$ has to be found. To do this, a new equation is calculated from the a_i, b_i, c_i, d_i and S values of the equation of this node. As the constant is going to be situated as the first child of the addition operation, this new equation is calculated as in Eq. (9), with $a'_i = a_i$, $b'_i = b_i - a_i \cdot p_i$, $c'_i = c_i$, $d'_i = d_i - c_i \cdot p_i$, where p_i are the values of the node p for each data sample, and S' is calculated from S with the method described for the addition operation. With this new equation, the constant optimization process described in Section 2.5.1 is performed. As a result, an improvement value is returned, a positive one indicates that this node can be replaced with $(k+p)$.

If a $(k \cdot p)$ search is performed on a node with an equation represented by a_i, b_i, c_i, d_i and S, a new equation is calculated, this time from Eq. (13): $a'_i = a_i \cdot p_i$, $b'_i = b_i$, $c'_i = c_i \cdot p_i$, $d'_i = d_i$, and S' is calculated from S with the method described for the multiplication operation. With this equation, a constant search process is undertaken, resulting in a value of reduction in MSE.

With this search, the size of the tree is always increased in two nodes.

Fig. 4 (d) shows an example of a constant-expression search. In this case, the node is represented by the semantics p. From this point, two lines can be defined: $k+p$ and $k \cdot p$, having a part of both inside the shape. Thus, a constant can be calculated for each line, and the tree can be replaced by either $k+p$ or $k \cdot p$. Only addition and multiplication operations are considered because on $k-p$ and k/p the nodes $-p$ and $1/p$ are not tangent to the shape and it is less likely to find a line intersecting the shape.

As happens with constant-variable searches, constant-expression searches can hide a constant search. For example, if the tree is $(3+2 \cdot x_1)$ and the operation $(k+p)$ is used, then the result would be $(k+(3+2 \cdot x_1))$, which has its semantics equivalent to $(k+3)+(2 \cdot x_1)$, and could be generated by a constant search on the first constant leaf.

One way to partially avoid this situation is not to perform constant-expression searches with the operation $(k+p)$ on non-terminal nodes with the operation "+" or "-", and one of the children is a constant. In the same way, constant-expression searches with the operation $(k \cdot p)$ on non-terminal nodes with the operation "*" or "/", and one of the children is a constant should be avoided too.

Even with these precautions, a constant-expression search can hide a constant search. For example, the tree $((2-x_4)+(4 \cdot x_4))$ with the operation $(k+p)$, would result $k+((2-x_4)+(4 \cdot x_4))$, which has its semantics equivalent to $((k+2)-x_4)+(4 \cdot x_4)$. This last tree could be found by a constant search.

Another example could be the tree $(3+(2 \cdot x_1))$, in which the node where the constant-expression search takes place is $p=(2 \cdot x_1)$. In this case, a $(k+p)$ constant-expression search could take place and p could be replaced by $(k+(2 \cdot x_1))$. In this example, the final tree would be $(3+(k+(2 \cdot x)))$, which has its semantics equivalent to $((3+k)+(2 \cdot x))$. This tree could be found by means of a constant search.

Therefore, the same situation as in the previous search repeats itself. It is hard to know in advance when a constant-variable search or a constant-expression search will hide a constant search. As stated previously, this leads to the problem of having excessively large trees and stopping the algorithm too early. The same two solutions can be applied: performing optimization of all of the constants before these searches or performing a constant search at the same time.

2.5.5. Deletion of parts of the tree

The deletion of a branch of the tree implies replacing its father with its "sibling" branch. For instance, in the tree $(2 \cdot x_1+3 \cdot x_2)$, the deletion of the branch $3 \cdot x_2$ leads to having $(2 \cdot x_1+?)$ and then replacing the root with the first child, resulting in $2 \cdot x_1$. In this example, this is equivalent to having performed a constant search in the $3 \cdot x_2$ branch and achieving as result an improvement in MSE with the constant 0, leading to the tree $(2 \cdot x_1+0)$. If the root were the operators of -, * or /, the corresponding constants would be 0, 1 and 1.

Therefore, in all the cases deleting a branch of the tree is similar to performing a constant search in that branch, and having as a result one of those constants, which is a very unlikely scenario. Therefore, deletion of parts of the tree is not considered in this method, because it can be performed by constant, variable, and constant-variable searches.

2.6. Optimization of constants

As the tree is being built, each time a constant is generated, its value is the best for that tree. However, as the improvement process continues, the tree will be modified, therefore this previously calculated constant value will not be the best for the new tree and, consequently, this tree will not return the best possible result with that structure. This happens with all the constants of the tree. Therefore, each time the tree is modified, an optimization of the constants could be performed to find their new best values. Also, it has been found that, after a tree has been modified, during several iterations in which only constant search is performed, the MSE is improved.

For this reason, a constant optimization process is proposed. This process is simply based on examining the tree, taking all the constant nodes and subsequently performing the following steps for each one:

1. Calculate the equation in this node.
2. Perform a constant search, having as result the value of MSE reduction.
3. If this reduction of MSE is higher than a value (see Section 2.8 for details of this parameter), then modify the constant of the node with the result of the search.

Once these steps are executed on a constant node, the process goes into the following one. When the last constant node has been processed, this process begins again with the first. This process finishes when none of the constant nodes can be optimized. Note that before that point is reached, each node may have been optimized several times.

A different constant optimization process could be considered, in which in each iteration a constant search is performed on all the

constant nodes, and the one with the highest reduction (in case it is higher than the parameter value) is modified. However, this would need the calculation of all the equations of the constant nodes in each iteration. Since the calculation of an equation needs the calculation of the equations of the previous nodes and the constant nodes are always leaves of the tree, this implies the calculation of a high number of equations of the tree. This can be a time-consuming task. Therefore, the approach described is proposed instead, which is much more efficient, since only a small number of equations need to be calculated in each iteration. However, if the user wishes to perform a constant optimization process in which in each iteration the constant to be changed is the one with the highest MSE reduction, this can be done by repeatedly executing constant searches on constant nodes.

As stated in Sections 2.5.3 and 2.5.4, the optimization of constants can be very useful for preventing some successful searches that in practice only modify the value of a constant, causing an artificial growth of the tree. One way to prevent these situations is, before the selected searches take place, to optimize all the constants of the tree. If this is done, obviously a constant search will be unsuccessful and will only consume computational time.

2.7. Constraints to the tree

As one of the aims of this paper is to find simple expressions that are easy to analyze by humans, it is interesting to limit the complexity of these expressions. With this objective, two constraints are used: the height of the tree and the number of nodes. The user may make use of either of these two, or none.

Both restrictions have an effect on in the constant-variable and constant-expression searches described in Sections 2.5.3 and 2.5.4. In the other two searches, there is no need to apply complexity constraints because in both searches the result would be a tree with the same or less complexity. Only on constant-variable and constant-expression searches, the complexity of the tree is increased.

The application of these two constraints is very straightforward. If the maximum height constraint is being used, instead of performing these two searches in all the nodes of the tree, they will be done only in the nodes whose depth is lower than the maximum height. If the maximum number of nodes (n) constraint is being used, the tree has r nodes and each node of the tree represents a subtree with s nodes, then the constant-variable search is performed only on those nodes that meet the constraint $n-r+s \geq 3$. The constant-expression search is performed if $n-r \geq 2$

These two constraints are closely related. Since only binary operations, arithmetic functions, are used, the height constraint sets a limit to the number of nodes. The number of nodes of a tree of height h can be up to 2^{h-1}. Moreover, in this tree these 2^{h-1} must be balanced, so a height limit is a constraint to the number of nodes, but also to the structure of these nodes in the tree. Also, it was already demonstrated that a binary tree with n nodes has an average height of $2\sqrt{\pi n}$ (Flajolet & Odlyzko, 1982). Therefore, to allow the building of a tree with n nodes without any structure constraint, the maximum height should be higher than $2\sqrt{\pi n}$, which would lead to having more than $2^{2\sqrt{\pi n}-1}$ nodes, which is a very large number. For this reason, the height limit is not used, and the only constraint is the number of nodes.

Setting a limit on the complexity of the tree has two interesting features. First, it makes it possible to obtain simple, easily understood expressions. Second, it is possible to obtain models with better generalization abilities and to control overfitting. The described system can grow until a good enough result in the training set is found. However, excessive growth, resulting in a very large tree and a very complex model, may lead to overfitting the training set. Therefore, the user may set a limit to this growth in the maximum number of nodes that the tree can have. Thus, this parameter will be important, and experiments will be performed with it.

2.8. Algorithm

The method proposed in this paper allows the creation of trees with a low MSE value. However, contrary to GP, in which many different trees are created, in DoME, a simple tree is continuously improved.

The algorithm begins with a simple initial tree made of a single node representing a constant. This constant is the point in the constant line closer to the target point. To calculate it, it is necessary to set the constraint that the vectors k and $k-t$ must be perpendicular, and therefore the dot product $\langle k, k-t\rangle = 0$. Developing this expression leads to

$$0 = \sum_{i=1}^{N} k \cdot (k - t_i) = \sum_{i=1}^{N} k^2 - \sum_{i=1}^{N} k \cdot t_i = N \cdot k^2 - k \cdot \sum_{i=1}^{N} t_i \tag{36}$$

And therefore $k = \frac{1}{N}\sum_{i=1}^{N} t_i$ is the average value of the targets. As this constant node is already the best value, it will not be optimized due to the constant search process. This value could also be obtained with a constant search from Eq. (30) ($a_i = 1$, $b_i = t_i$, $c_i = 0$, $d_i = -1$, $S = \emptyset$).

It has been found that in several datasets the algorithm sometimes stalls in the first iteration. This happens because this tree is a local minimum with a low number of nodes. As it has few nodes, the algorithm has few places to search and might not find any improvement in these places.

To solve this, one possibility is, when this happens, to change the initial tree to a more complex one. This new tree does not consist only of the constant k calculated above, but initially has combinations of the variables up to a specific order, but with the semantics of the k-tree calculated above. For example, for a problem with 2 inputs and order 2, the initial tree will be the following:

$$k + 0 \cdot x_1 + 0 \cdot x_2 + 0 \cdot x_1 \cdot x_1 + 0 \cdot x_1 \cdot x_2 + 0 \cdot x_2 \cdot x_2 \tag{37}$$

In general, with L inputs, the initial tree of order n will be as follows:

$$k + \sum_{i=1}^{L} 0 \cdot x_i + \sum_{i=1}^{L}\sum_{j=i}^{L} 0 \cdot x_i \cdot x_j + \sum_{i=1}^{L}\sum_{j=i}^{L}\sum_{k=j}^{L} 0 \cdot x_i \cdot x_j \cdot x_k + \cdots \tag{38}$$

This tree can be calculated by the following recursive relation, where $T_n(x_1, x_2, \ldots, x_L)$ is the tree of order n for L inputs:

$$\begin{aligned} T_0(x_1, x_2, \ldots, x_L) &= 0 \\ T_n(x_1, x_2, \ldots, x_L) &= \sum_{i=1}^{L} x_i \cdot T_{n-1}(x_i, x_{i+1}, \ldots, x_L) \\ &\quad + T_{n-1}(x_1, x_2, \ldots, x_L) \end{aligned} \tag{39}$$

And, therefore, for an order n and L possible variables, the initial tree will be $k + T_n(x_1, x_2, \ldots, x_L)$.

In this way, when the initial tree does not allow to leave this local minimum, successively more complex trees are generated (with a higher order) but with the same semantics, until one of them makes it possible for the search process to leave this local minimum, or until the maximum number of nodes is reached.

After this initial tree has been created, the iterative process begins. On each iteration, the constant search, variable search, constant-variable, and constant-expression search can be performed on any nodes, whether they are constant, variable or non-terminal nodes, following a specific strategy. For instance, some possible strategies could be:

1. "Exhaustive": perform the four searches at the same time and substitute the corresponding node with the result of the strategy with a higher reduction in MSE, in case it is positive. This is the strategy that makes a full search, however, it is also the most time-consuming strategy since much computational time is wasted in performing operations that will not

lead to improving the tree. It is important to note that in this strategy constant search is performed in constants, together with constant-variable search and constant-expression search, to avoid the above-mentioned problem of "hiding" changes in constants and tree growing.

2. "Exhaustive with constant optimization": perform the four searches at the same time; however, perform constant search only in variables and non-terminal nodes. After any modification is made to the tree, perform the constant optimization process. This optimization ensures that the next constant-variable and constant-expression searches do not "hide" a change in constants.
   These two strategies can be seen as similar in the sense that in the second one a constant optimization step is performed on each iteration, while in the first one a constant search on the constant nodes of the tree is performed on each iteration. Therefore, in the first strategy successive iterations may perform a constant refinement while in the second strategy the constant refinement is performed on each iteration. However, this constant refinement performed in the first strategy will be stopped when any other search returns a higher MSE reduction, while in the second strategy the constant refinement is stopped when no MSE reduction is found in the constant optimization process.
3. "Selective with constant optimization": To save computational time, searches that are found to be unsuccessful most of the time can be avoided. Also, the searches can be performed sequentially, and when one of the searches is successful, the rest are not performed. This strategy proposes to run the following steps on each iteration:

   (a) Perform variable search only on constant nodes.
   (b) If the previous search was not successful, perform a constant-expression search (this search is always only performed on non-terminal nodes).
   (c) If either of the two previous searches were unsuccessful, perform a constant-variable search on terminal nodes.
   (d) If none of the three previous searches was successful, perform together:

   - Constant search on variable and non-terminal nodes.
   - Variable search on variable and non-terminal nodes.
   - Constant-variable search on non-terminal nodes.

   (e) If any of the previous searches were successful, perform a constant optimization.

   The objective of this search is to perform the searches that are unlikely to be successful only when the rest are unsuccessful. This usually happens when the tree has reached the maximum limit. As can be seen, the searches that are performed when this happens are those that allow the tree to become smaller.
4. "Selective". In a similar way to the first and second strategies, this fourth strategy is like the third, but without the constant optimization process at the end. Instead, the first step of this strategy is a constant search on constant nodes (refinement of constant values). If a constant is modified, then no further searches are performed and this iteration is finished. If no constant is modified, then a strategy similar to strategy 3 is executed, except for the final constant optimization, which is not performed.

These four strategies were used in the experiments because they describe two different situations: an exhaustive strategy in which in each iteration all the searches are performed in all the nodes, and a lighter strategy in which in each iteration the whole tree is examined only when no previous search was successful. In both strategies, their variants with and without constant optimization are explored too.

However, users may decide to use other strategies they may consider useful. Other examples are:

- Perform the searches consecutively. Modify the tree accordingly for each search.
- Perform a constant and/or variable search and if the tree is not modified, perform a constant-variable search. If no modifications are made, perform a constant-expression search.
- Perform a constant search. If no modification is made, perform a variable search. If no modification is made in this second search, perform a constant-expression search. If no modification is made in this third search, perform a constant-variable search.

Once a strategy has been set, this process is iteratively performed until a stopping criterion is met. This criterion can be defined by the user. This paper proposes the use of the following criteria:

- The number of iterations exceeds a fixed value.
- The MSE in the current tree reaches the goal value set by the user.
- The tree could not be improved in the last iteration. A hyperparameter defining the minimum improvement to consider that a search has been successful is needed.

The result is a system whose configuration depends on a low number of hyperparameters:

- Maximum number of iterations to be executed. Default value: infinite.
- Goal in MSE. Default value: 0.
- Minimum improvement in the MSE for any search. A search is found to be successful if the reduction in MSE is positive and higher than the previous MSE value multiplied by this parameter. For example, if this parameter has a value of 10^{-2}, then a search will be successful only if the reduction in MSE is greater than 1% of the MSE. This value is also applied to the constant optimization process described in Section 2.6. Default value: 10^{-6}.
- Maximum number of nodes of the tree. This parameter sets a limit to the complexity of the equations to be developed by the system. The default value is infinite, which means that arbitrarily complex expressions may be developed to meet the MSE goal.
- Strategy to be used on each iteration.

A key feature of this algorithm is its deterministic nature, i.e., each time it is run with the same dataset it will return the same result. This means that it only has to be tun once for each parameter configuration.

This algorithm was implemented in Julia (Bezanson et al., 2017) v1.4.2. The source code of this technique can be downloaded from https://github.com/danielriveroc/DoMEv1 along with detailed instructions on how to repeat the experiments described in this paper, and how to run this system so that it can be used by anyone in their experiments.

3. Example of equation development

This section describes the application of DoME for developing an easy and well-known expression: Newton's law of universal gravitation (Newton, 1687). This equation measures the force of attraction between two masses. This force is proportional to the product of both masses and inversely proportional to the square of their distance. This expression is given by Eq. (40), in which M_1 and M_2 are the masses, d is the distance, and $G = 6.67392 \cdot 10^{-11}$ is the gravitational constant.

$$F = G \cdot \frac{M_1 \cdot M_2}{d^2} \tag{40}$$

This equation can be applied to arbitrarily large or small masses and/or distances, i.e., it can be applied to calculate the force between either small particles or big planets. In this example, this second case will be explored, in order to show that the system can return good results with arbitrarily different values since masses and distances belong to different ranges.

To apply this equation, 1000 random data samples were built. Each one is composed of:

Table 4
Summary of the execution for developing Newton's law of universal gravitation.

Iteration	Node selected for substitution	Resulting expression	MSE
0	–	$9.185106275464827 \cdot 10^{20}$	$2.7645 \cdot 10^{43}$
1	$9.185106275464827 \cdot 10^{20}$	$(1.1526861538137104 \cdot 10^{30}/d)$	$2.1717 \cdot 10^{43}$
2	$1.1526861538137104 \cdot 10^{30}$	$((434063.57187533064 \cdot M_2)/d)$	$1.9183 \cdot 10^{43}$
3	434063.57187533064	$(((3.93202929320367 \cdot 10^{-19} \cdot M_1) \cdot M_2)/d)$	$5.1733 \cdot 10^{42}$
4	$3.93202929320367 \cdot 10^{19}$	$((((6.673920000000007 \cdot 10^{-11}/d) \cdot M_1) \cdot M_2)/d)$	$3.1828 \cdot 10^{13}$

- M_1: Mass of the first planet: a random number between 10^{23} and 10^{25}
- M_2: Mass of the second planet: a random number between 10^{23} and 10^{25}
- d: Distance between both planets: a random number between 10^8 and 10^{12}
- target: the result of the application of Eq. (40)

With this dataset, the system was configured with the following hyperparameters:

- Maximum number of nodes: Infinite
- Strategy: The first strategy described in Section 2.8
- Goal in MSE. This parameter is very important in this example, since we are working with a wide range of values, from very small (G) to very large (M_1 and M_2). The targets were also in a large scale, from $10^{12}N$ to $10^{22}N$. Due to the limitation of the use of floating-point values, even when 64 bits are used for their storage, when working with such a wide range, inaccuracies may occur due to operating with numbers on very different scales. For instance, the operation $10^{20} + 10^{-10} - 10^{20}$ returns an incorrect value of 0. This may result that, even though the correct expression has been obtained, the MSE calculated is not 0 due to inaccuracies, and the system keeps trying to improve this expression with inaccurate calculations. For this reason, a target MSE of 0 was not used in this example. Instead, the target used was the average value of the targets.

Table 4 shows a summary of one execution of this system with the described dataset. As was described, before the execution is performed, an initial tree with a single constant node is built. In this execution, this constant had a value of $9.185106275464827 \cdot 10^{20}$. From this initial tree, four iterations were performed until an expression with an MSE below the goal was found. All the operations performed were constant-variable searches, and the nodes selected for substitutions were the constants. This example also shows the ability of this system to develop the constants needed for the expressions to be returned. For this reason, the constants are shown in this Table with a large number of decimals.

Finally, as a result of this execution, the expression $((((6.673920000000007 \cdot 10^{-11}/d) \cdot M_1) \cdot M_2)/d)$ is returned by the system, which is similar to Eq. (40).

4. Experiments and discussion

The previous section showed that the system described in Section 2.2 can be successfully used as a tool for developing mathematical expressions. These mathematical expressions can also be used as an ML model. For this purpose, a number of databases from the PMLB (Penn Machine Learning Benchmark) repository (Olson et al., 2017) have been taken and this method has been run with them, as they have been extensively used to test different SR techniques (La Cava et al., 2021; Moscato et al., 2021; Orzechowski et al., 2018). This makes it possible to provide a comparison of the results obtained with this system and those obtained with other already published SR models. In particular, the results of this system have been compared with those published in Moscato et al. (2021), since that paper shows the numerical values obtained by other models. However, it is interesting to evaluate the AI Feynman dataset since it is based on trigonometric operations, and this would allow to evaluate the previously mentioned problems that this system may have when other behavior (such as periodic) is required. This task is left as a future work.

From the 94 datasets used in Moscato et al. (2021), 64 were artificially generated from the Friedman function (Friedman, 1991; Friedman et al., 1983), with different values of colinearity degree, different number of samples, and different number of features (if greater than 5, the remainder are random). These 64 datasets have been set aside and have not been used as benchmarks in this study. The reason for this is to avoid the results in this particular function having an excessive impact on the overall performance of this system. If 64 out of 94 datasets refer to the same function, then this function has a weight of 68.1% in the overall results, when ideally the weight should be 1/94. For this reason, in this paper, the remaining 32 datasets will be used.

The experiments carried out in this part have the objective of demonstrating the applicability of this technique. In this sense, as shown in Section 2.7, it is important to limit the complexity of the trees. For this reason, the experiments will focus on studying the impact of the parameter described in Section 2.7 that limits the size of the tree: maximum number of nodes. In this sense, different numbers of nodes varying from 5 to 200 growing in intervals of 5 were used. Also, the hyperparameter of minimum improvement in MSE plays an important role. For this reason, it was used in the experiments. The values chosen for this parameter are 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6} and 10^{-7}. Finally, the experiments performed in this section were run with the four strategies explained in Section 2.8.

A grid search (Raschka, 2015) was performed with all the values of these three parameters, i.e., all of the possible combinations of the values of the three parameters were tried. For each combination of these parameter values studied, a 10-fold cross-validation was performed. Since this system is deterministic, only one training in each fold was done. Cross-validation is a widely used technique for comparing different methods and hyperparameter configurations. In a cross-validation approach, the results used for the comparison are the average of the test results in each fold. Thus, for each hyperparameter configuration, the 10 test results are averaged. However, this section will show the median of the results obtained in the 10 independent runs, rather than the mean, because the paper used for the comparison uses the median as the basis for comparison. In order to correctly compare the results, the same training and test sets were used in all of the experiments performed.

An important measure of the performance of this system is the computational time. All the experiments were run on an Intel(R) Gold(R) CPU 6240R, with a frequency of 2.40 GHz. A measure of time was performed on each of the experiments performed.

Table 5 shows the results obtained for each combination of problem and strategy. Due to the high number of experiments carried out, only the configurations that returned the best results are shown in these Tables. For each dataset, this Table shows the best median MSE result (in test, second column) and, in the following 3 columns, the hyperparameter configuration (strategy, minimum MSE reduction and maximum number of nodes, as described at the end of Section 2.8) in which the training process returned that MSE value. The hyperparameter related to the maximum number of nodes limits the complexity of the system. This is directly related to the generalization ability of the system. Therefore, the configurations that generalize best are not those that have used the highest number of nodes. With respect to

Table 5
Results obtained. For each problem, this table shows the best median MSE obtained in test, the hyperparameter configuration (strategy used to go through the nodes, minimum improvement in MSE for a search to be considered successful and maximum number of nodes in the tree) that returned this value, the median training time of this configuration (measured in seconds), and a stability measure of this configuration.

Problem	Best test MSE	Strategy	Min MSE reduction	Maximum num nodes	Average time (s)	Stability (100%)
ESL	0.254	2	10^{-5}	100	8.3	4.35%
SWD	0.368	1	10^{-5}	60	4.15	1.49%
LEV	0.35	3	10^{-4}	45	0.0356	0.89%
ERA	2.21	2	10^{-7}	100	1.49	2.03%
USCrime	193	3	10^{-3}	20	0.0124	2.48%
FacultySalaries	0.728	1	10^{-7}	155	165	52.44%
vineyard	4.1	4	10^{-3}	15	0.00136	16.33%
auto price	3.43e6	1	10^{-7}	65	0.978	19.50%
autoPrice	3.41e6	2	10^{-4}	160	7.94	17.61%
cloud	0.0676	4	10^{-7}	55	0.0903	27.87%
elusage	89.8	1	10^{-3}	15	0.00706	2.81%
machine cpu	1.27e3	4	10^{-7}	70	8.41	5.84%
analcatdata vehicle	5.32e3	4	10^{-5}	145	2.13	51.20%
vinnie	2.36	3	10^{-4}	5	0.000948	1.41%
pm10	0.545	2	10^{-4}	30	0.677	1.57%
analcatdata neavote	0.814	3	10^{-2}	10	0.00081	12.57%
analcatdata election2000	1.88e7	3	10^{-5}	100	1.91	99.83%
pollution	1.26e3	4	10^{-4}	30	0.0186	5.40%
no2	0.242	1	10^{-4}	50	3.4	3.23%
analcatdata apnea2	5.58e5	4	10^{-4}	115	0.749	9.53%
analcatdata apnea1	6.97e5	1	10^{-6}	125	5.99	8.98%
cpu	15.2	3	10^{-7}	140	140	79.49%
sleuth ex1714	7.66e5	3	10^{-2}	40	0.0691	13.25%
rabe 266	2.65	1	10^{-7}	185	262	34.92%
sleuth case2002	52.7	3	10^{-6}	30	0.109	2.81%
rmftsa ladata	2.81	1	10^{-3}	70	0.211	8.50%
visualizing environmental	6.82	3	10^{-5}	30	0.0472	6.63%
sleuth ex1605	85.2	2	10^{-7}	30	0.104	8.01%
visualizing galaxy	384	1	10^{-7}	75	19.3	8.13%
chatfield 4	212	3	10^{-6}	30	0.319	5.03%
sleuth case1202	1.56e3	3	10^{-3}	70	0.0875	11.89%
chscase geyser1	33.3	1	10^{-3}	45	0.0219	2.11%

the minimum MSE reduction, this hyperparameter seems to return better results as it takes lower values. This means that even very low improvements in the training sets may lead to improvements in the test sets. Thus, this hyperparameter does not influence the possible overfitting of the expressions to the training sets, and this overfitting is controlled mainly through the maximum number of nodes.

The strategy used is shown to be an important hyperparameter, as it is the driver of the search process through the semantic space. In order to compare the MSE results returned by the different strategies, the following process has been carried out. First, for each dataset, the four best MSE results (one for each strategy) were divided by the best of the four. Thus, all 4 values will be greater than or equal to 1, with 1 being the best value. With these values, a boxplot has been made for each of the strategies, shown in Fig. 5. In this figure, values closer to 1 indicate better results, in this case regarding MSE. As can be seen in this figure, strategy 1 offers the best results of the 4 strategies. However, it is also the most time-consuming strategy.

Table 5 also shows the median computational time (in seconds) used to generate the 10 expressions corresponding to that hyperparameter configuration. In order to assess the time spent by each strategy, similar processing was carried out, but instead of using the MSE values, the time measures were used. In this way, new boxplots were generated and are shown in Fig. 6. Again, values closer to 1 indicate shorter times. As this figure shows, the Exhaustive strategy has a much higher execution time than the others. Also, the Selective strategies are sensitively faster than the Exhaustive strategies. On the other hand, the constant optimization process seems to improve the computational time. This can be seen in that the strategies using constant optimization (2 and 3) have a much shorter execution time than their corresponding ones without optimization (1 and 2 respectively). This reinforces the usefulness of using a constant optimization process. Joining these two Figs. 5 and 6, it can be seen that strategy 1 performs a search that returns better results on average, at the cost of having a higher execution time. This is to be expected since this strategy performs a more exhaustive search in the semantic space.

The last column of this table shows a stability test. The aim is to observe how the results vary with small changes in the data used for training. When using a 10-fold cross-validation, 11.11% of the instances of a training set of one fold are different from those of any other fold; however, 100% of the instances used for testing are different in all folds. For this reason, the stability calculation is performed with the MSE values in the training sets, and not in the test sets. To do this, the median absolute deviation of the 10 MSE values in training was calculated for each hyperparameter configuration shown in the table. This gives a measure of the dispersion of training errors when using slightly different training datasets. This value was then divided by the median of the 10 MSE values in training in order to calculate a variability ratio, which is expressed in the last column as a percentage.

The results of this system have been compared with those published in a previous work in which several SR systems were used for comparison (Moscato et al., 2021). In that work, CFR (Continued Fraction Regression) is compared with 15 SR algorithms. five of them are GP-based methods and 10 are ML algorithms. However, only for the following three methods, other than CFR, numerical results are provided:

- eplex-1 m: ∈-Lexicase selection with stopping criteria of one million evaluations (La Cava et al., 2016).
- xg-b: Extreme Gradient Boosting (Chen & Guestrin, 2016).
- grad-b; Gradient Boosting Regression (Friedman, 2000).

The first is a GP-based method, while the other 2 are ML algorithms. In that paper, the numerical results of CFR and these 3 methods have been published because they were the "top four methods that showed no significant differences of performances" in a Critical Difference diagram with a significance threshold of p=0.05 (Moscato et al., 2021). Therefore, these 4 algorithms have been used in the comparison here

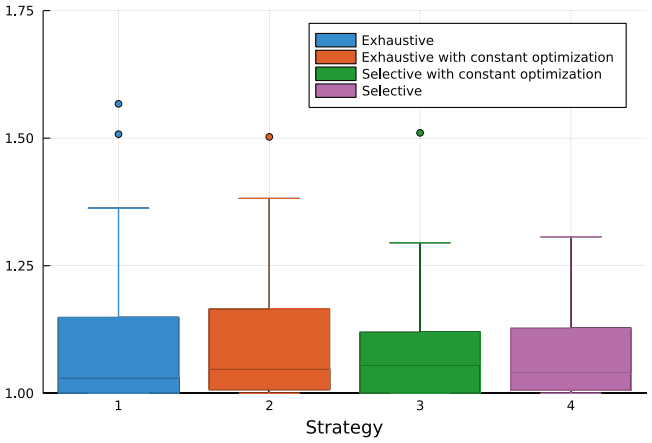


Fig. 5. Comparison of the relative MSE results with the different strategies.

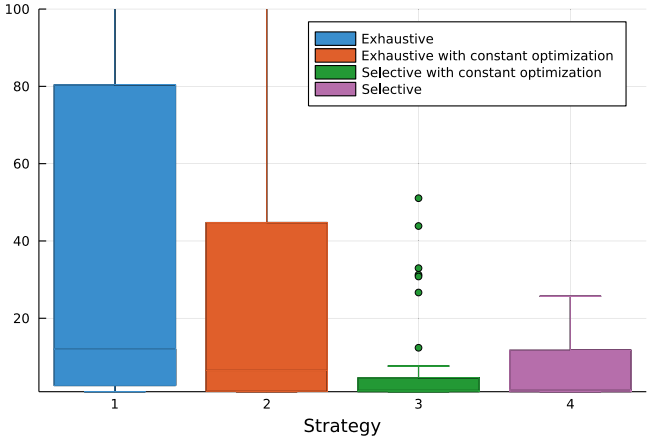


Fig. 6. Comparison of the relative execution time.

because they are the ones for which numerical data are provided in the reference work. Also, since that paper uses the median instead of the mean for comparison, this paper will also use the median statistic.

Table 6 shows a comparison between the best median MSE results obtained with the three different methods. The numerical results of the 4 algorithms used for comparison correspond to the median value for 10 independent runs of these four algorithms. In this table, for each dataset, the results of the method that returned the lowest median MSE are shown in bold. As expected according to NFL theorems, there is no technique that returns the best results in all problems. However, DoME is the algorithm that returns the best results in most cases.

In Fig. 7 a Critical Difference diagram is shown in which the comparison between the results obtained by the 5 methods can be seen (Calvo & Santafé, 2016; Demšar, 2006). In this diagram, after placing each method on the x-axis according to its MSE, the methods between which there is no significant difference with p≤0.05 are shown joined by a horizontal line. As this figure shows, although DoME offers better results, there is no significant difference between the results returned by DoME and CFR. However, there is difference with respect to the other techniques.

From the data in this Table, a similar analysis to that carried out with the strategies can be performed. In this case, for each dataset, the MSE results are divided by the best value obtained by any of the five methods. From this data, a boxplot is created for each of the methods, which can be seen in Fig. 8. As before, the lower the values and the closer to 1, the lower the MSE. As can be seen in this figure, DoME

Table 6
Comparison between different methods (MSE)

Problem	eplex-1 m	grad-b	xg-b	CFR	DoME
ESL	0.274	0.319	0.272	0.268	**0.254**
SWD	0.39	0.405	0.408	0.432	**0.368**
LEV	0.425	0.424	0.422	0.353	**0.35**
ERA	2.51	2.58	2.57	2.45	**2.21**
USCrime	393	257	378	220	**193**
FacultySalaries	4.04	8.07	4.11	1.28	**0.728**
vineyard	6.01	8.22	7.83	4.22	**4.1**
auto price	5.89e6	3.89e6	4.03e6	6.01e6	**3.43e6**
autoPrice	4.17e6	5.29e6	**2.87e6**	4.83e6	3.41e6
cloud	0.11	0.208	0.144	0.095	**0.0676**
elusage	135	199	137	**65.8**	89.8
machine cpu	3.8e3	2.23e3	2.69e3	2.1e3	**1.27e3**
analcatdata vehicle	4.14e4	2.41e4	4.2e4	1.55e4	**5.32e3**
vinnie	2.29	2.86	2.66	**1.93**	2.36
pm10	0.64	0.431	**0.399**	0.621	0.545
analcatdata neavote	1.18	0.818	0.917	**0.401**	0.814
analcatdata election2000	4.33e7	3.4e8	7.72e8	**5.09e5**	1.88e7
pollution	1.87e3	2.19e3	1.67e3	1.42e3	**1.26e3**
no2	0.272	0.227	**0.21**	0.295	0.242
analcatdata apnea2	1.12e6	9.42e5	7.86e5	6.09e5	**5.58e5**
analcatdata apnea1	8.16e5	9.98e5	**5.28e5**	6.96e5	6.97e5
cpu	175	2.36e3	883	164	**15.2**
sleuth ex1714	1.42e6	1.57e6	2.31e6	**6.83e5**	7.66e5
rabe 266	7.11	7.32	3.04	**2.64**	2.65
sleuth case2002	75.8	56.2	72.4	**41.7**	52.7
rmftsa ladata	3.01	3.51	3.21	**2.76**	2.81
visualizing environmental	9.62	9.8	9.54	**4.7**	6.82
sleuth ex1605	102	98.4	92	**84**	85.2
visualizing galaxy	313	268	**224**	434	384
chatfield 4	282	384	288	**189**	212
sleuth case1202	3.29e3	3.42e3	3.2e3	**1.39e3**	1.56e3
chscase geyser1	38.9	39.9	42.9	**31.1**	33.3

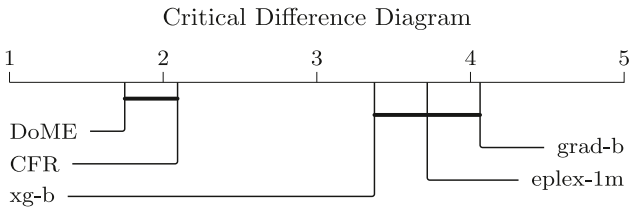


Fig. 7. Critical Difference diagram with the comparison of the 5 methods.

returns, on average, better results than the other techniques. However, it is important to note that the splits of the datasets are not the same as those used in the reference work because we have not had access to these partitions.

Also, an analysis of the expressions obtained can be carried out. Table 7 shows, for each dataset, the median number of nodes and the median height of the expressions obtained in each fold with the configuration that gave the best results, shown in Table 6. As can be seen, the number of nodes is usually very close to the maximum. This means that the system makes use of as much complexity as possible in order to fit the training data as much as possible. Interesting information is obtained when examining the height of the resulting trees. With respect to the height of the trees, as was expected, a higher number of nodes leads to higher heights. However, these heights are not on the scale of $log_2(n)$, with n representing the number of nodes. This shows that the system, with a limit of the number of nodes, finds better results with very unbalanced trees rather than with balanced trees. This fact supports the idea of not using a hyperparameter to limit the height of the tree. It is also interesting to note that in all datasets a maximum of 200 nodes has been allowed, but in no case has the best tree needed to reach that number. As can be seen in this table, this method not only returns expressions with a low MSE, but they are

Table 7
Summary of the expressions obtained.

Problem	Nodes	Height	Num. inputs	Problem	Nodes	Height	Num. inputs
ESL	99	17	4	analcatdata election2000	99	16	14
SWD	59	14	10	pollution	29	12	15
LEV	45	13.5	4	no2	49	12.5	7
ERA	78	18.5	4	analcatdata apnea2	115	23	3
USCrime	19	7	13	analcatdata apnea1	125	15.5	3
FacultySalaries	155	20	4	cpu	139	24	7
vineyard	9	4	2	sleuth ex1714	39	8.5	7
auto price	65	16.5	15	rabe 266	185	46.5	2
autoPrice	159	25.5	15	sleuth case2002	29	11	6
cloud	32	9	5	rmftsa ladata	15	7	10
elusage	15	5	2	visualizing environmental	24	9	3
machine cpu	69	15.5	6	sleuth ex1605	29	11	5
analcatdata vehicle	145	24	4	visualizing galaxy	75	13.5	4
vinnie	5	3	2	chatfield 4	29	10	12
pm10	15	6	7	sleuth case1202	69	13.5	6
analcatdata neavote	9	5	2	chscase geyser1	23	8	2

Table 8
Examples of the final expressions obtained.

Problem	Expression
LEV	$(-0.122) + ((((0.265 \cdot X_1) + (0.416 \cdot X_2)) + (0.152 \cdot X_3)) + (0.155 \cdot X_4))$
USCrime	$\left(\left(\frac{(-0.822)}{X_{10}} + \frac{\left(X_9 + \frac{648854.722}{X_2}\right)}{\frac{X_4}{X_6}}\right) \cdot X_1\right) + \frac{72690.856}{X_{13}}$
vineyard	$0.968 \cdot ((5.663 + X_1) + X_2)$
cloud	$((-0.147) \cdot X_5) + ((0.479 \cdot X_3) + ((-0.047) + (0.733 \cdot X_4)))$
elusage	$(((-0.002) \cdot X_1) \cdot X_1) + \frac{((2362.397 - X_1) - X_1)}{((-2.965) + X_1)}$
vinnie	$(-1.584) + (0.582 \cdot X_2)$
pm10	$0.351 \cdot \left(\frac{13.246}{((0.442 \cdot X_1) + (1.155 \cdot X_3))} + (1.021 \cdot X_1)\right)$
analcatdata neavote	$3.732 \cdot (2.602 - ((3.022 - X_1) \cdot X_1))$
visualizing environmental	$4.346 \cdot \left(1.335 + \frac{(112.678 - (1.017 \cdot X_1))}{X_3}\right)$
sleuth ex1605	$0.761 \cdot \left(\left(\frac{\left(\frac{31.767}{X_5} \cdot X_4\right)}{X_3}\right) \cdot \left(\frac{\left(\left(\left(\frac{(327.570 \cdot X_5)}{X_2} - X_2\right) - X_2\right) - X_2\right)}{((-95.548) + X_4)} + X_2\right)\right) + X_5$
chscase geyser1	$(((31.425 - (0.923 \cdot (0.984 \cdot X_2))) - X_2) - X_2) \cdot \left(\frac{(2.508 - ((((0.002 \cdot X_3) \cdot X_2) \cdot X_1) + X_2))}{X_2} + X_2\right)$

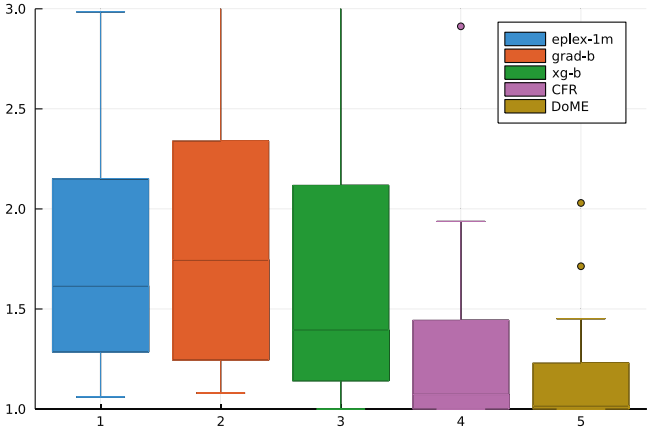


Fig. 8. Comparison of the relative MSE results with different methods.

also very compact, both in the number of nodes and in height. This table also shows the number of variables for each dataset. As can be seen, there is no correlation between the number of variables and the number of nodes needed to obtain the best test result.

Finally, for each problem, we took the configuration with the combination of hyperparameters that returned the best test results for each dataset, according to Table 5. With these configurations, this system was re-trained again, but this time using all the data as the training set. Table 8 shows some of the final expressions generated. To generate this table, only the datasets whose expressions fit on the page were taken. In addition, the number of decimal values in the expression constants has been limited to three. As can be seen, in many cases these expressions can be simplified to simpler ones, i.e. with fewer nodes. In this way, for a given limit on the maximum number of nodes, the expressions developed could have a lower training MSE using a simplification process. However, this task has been left as a future work.

5. Conclusions

This paper presents a novel technique for Symbolic Regression, called DoME. In this field, the most commonly used technique is GP, which is based on evolutionary processes and no mathematical explanation can be given for the equation development process. Thus, this field has an important lack of mathematical-based methods. The technique presented in this paper makes it possible to obtain mathematical expressions that can model an input–output relationship.

Also, the expressions obtained by this method can have a limit in complexity. This makes it possible to obtain expressions that can be easily analyzed by humans, in contrast with other techniques that

return very large expressions. As was shown, the resulting expressions are compact and, therefore, easier to analyze. The analysis of these expressions is usually one of the objectives of Symbolic Regression.

Results in Section 4 show that this technique can return good results in real-world problems. These results have also been compared with several SR methods, concluding that this method returns, on average, better results.

An additional advantage of this system used for Machine Learning purposes is that the returned model is a standard equation and thus it can be used in any programming language with no need to import any ML library. Moreover, this expression can be used in any calculation systems apart from programming environments. For instance, it can be easily used in a spreadsheet as opposed to other systems such as Neural Networks.

6. Future works

This paper opens a wide new research field in Symbolic Regression. As was described throughout the paper, much research is still to be done. Some of the possible developments could be:

- In the constant search, find an easy expression or method to compute the minimum of Eq. (6) in the case when c_i and d_i are vectors. One possibility could be to use gradient descent (Snyman & Wilke, 2018) to minimize this function.
- A new constant optimization algorithm could be proposed. In the one described in this paper, each constant is alternatively optimized in different iterations, which needs the computation of the equations of the corresponding nodes in each iteration. An alternative could be to change the value of several (or all) constants in each iteration, thus making this process faster.
- As the limit of the complexity of the tree has been proven to be an important factor, new ways of limiting this complexity can be found. For instance, setting a limit to the number of addition, subtraction, multiplication or division operations that can be used.
- Limiting the complexity of the models is a common way to avoid overfitting. However, other methods such as the use of a validation set can also be used. This possibility could be explored, with the advantage of not needing to set the limit of the number of neurons or height of the tree.
- In order to obtain expressions that are more easily understandable by humans, information about the structure of the desired expressions could be given. For instance, many times the desired expression is a division of two expressions, with no other division performed in these two parts. This structure, as well as any other, could be given to the system. As an additional feature, the search process could be made faster.
- An interesting possibility could be to extend the variable search not only with variables but also with any subtrees. The parts of the tree that are going to be replaced with another could be stored in a structure like a "node pool" and be used in the search later. The idea is that if they were useful once, they might become useful again later when the tree is modified.
- As mentioned above, GP is a very unstable system since it is non-deterministic. On the other hand, although DoME is deterministic, it could be interesting to study deeper its stability by analyzing the resulting expressions when there is a small change in the dataset. In this sense, the iterative improvement might make this algorithm more stable than others.
- As mentioned, the restriction of using arithmetic operators may make this system perform well on some problems, and poorly on others. For this reason, it would be interesting to evaluate it on other datasets, such as the AI Feynman benchmark, to study the limitations of this system.
- Finally, since this system returns algebraic expressions, these could be simplified. This simplification can be performed by a search operator that could be inserted into the strategy. In this sense, the use of Gröbner basis (Lin et al., 2008) could be useful to develop this search operator.

Additionally, this study could also be a basis for new ways of combining models that are not necessarily mathematical expressions. Any model (for instance, an ANN) can be represented in the semantic space as one point, that could fall inside one of the shapes. Therefore, the tree being developed could combine mathematical expressions with other types of models. This could be easily done with a search similar to the variable search, that could be called "model search", and a "constant-model search". In this sense, this technique would allow the building of ensembles of models.

CRediT authorship contribution statement

Daniel Rivero: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Enrique Fernandez-Blanco:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The experiments described in this paper were performed on computers in the Supercomputing Center of Galicia (CESGA). Daniel Rivero and Enrique Fernández-Blanco would also like to thank the support provided by the NVIDIA Research Grants Program, USA. Daniel Rivero created the whole model, programmed it in Julia, performed the experiments and wrote the paper. Enrique Fernandez-Blanco supervised the writing of the paper.

This study is partially supported by Instituto de Salud Carlos III, grant number PI17/01826 (Collaborative Project in Genomic Data Integration (CICLOGEN) funded by the Instituto de Salud Carlos III from the Spanish National Plan for Scientific and Technical Research and Innovation 2013–2016 and the European Regional Development Funds (FEDER)—"A way to build Europe". It was also partially supported by different grants and projects from the Xunta de Galicia [ED431D 2017/23; ED431D 2017/16; ED431G/01; ED431C 2018/49; IN845D-2020/03]. The authors thank the CyTED, Spain and each National Organism for Science and Technology for funding the IBEROBDIA project (P918PTE0409). In this regard, Spain specifically thanks the Ministry of Economy and Competitiveness for the financial support for this project through the State Program of I+D+I Oriented to the Challenges of Society 2017–2020 (International Joint Programming 2018), project (PCI2018-093284). Funding for open access charge: Universidade da Coruña/CISUG.

References

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Review*, *59*(1), 65–98. http://dx.doi.org/10.1137/141000671.

Brameier, M. F., & Banzhaf, W. (2010). *Linear genetic programming* (1st ed.). Springer Publishing Company, Incorporated.

Calvo, B., & Santafé, G. (2016). Scmamp: Statistical comparison of multiple algorithms in multiple problems. *The R Journal*, *8*, 1–8. http://dx.doi.org/10.32614/RJ-2016-017.

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *KDD '16, Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 785–794). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/2939672.2939785.

Chen, Q., Xue, B., & Zhang, M. (2015). Generalisation and domain adaptation in GP with gradient descent for symbolic regression. In *2015 IEEE congress on evolutionary computation* (pp. 1137–1144). http://dx.doi.org/10.1109/CEC.2015.7257017.

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

Flajolet, P., & Odlyzko, A. (1982). The average height of binary trees and other simple trees. *Journal of Computer and System Sciences*, *25*(2), 171–213. http://dx.doi.org/10.1016/0022-0000(82)90004-6, URL: http://www.sciencedirect.com/science/article/pii/0022000082900046.

Friedman, J. H. (1991). Multivariate adaptive regression splines. *The Annals of Statistics*, *19*(1), 1–67. http://dx.doi.org/10.1214/aos/1176347963.

Friedman, J. H. (2000). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, *29*, http://dx.doi.org/10.1214/aos/1013203451.

Friedman, J. H., Grosse, E., & Stuetzle, W. (1983). Multidimensional additive spline approximation. *Siam Journal on Scientific and Statistical Computing*, *4*, 291–301.

Haykin, S. S. (2009). *Neural networks and learning machines* (3rd ed.). Upper Saddle River, NJ: Pearson Education.

Icke, I., & Bongard, J. (2013). Improving genetic programming based symbolic regression using deterministic machine learning. In *2013 IEEE congress on evolutionary computation*. http://dx.doi.org/10.1109/CEC.2013.6557774.

Kammerer, L., Kronberger, G., Burlacu, B., Winkler, S., Kommenda, M., & Affenzeller, M. (2020). Symbolic regression by exhaustive search: Reducing the search space using syntactical constraints and efficient semantic structure deduplication. (pp. 79–99). http://dx.doi.org/10.1007/978-3-030-39958-0_5.

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA, USA: MIT Press.

Krawiec, K. (2012). Medial crossovers for genetic programming. In A. Moraglio, S. Silva, K. Krawiec, P. Machado, & C. Cotta (Eds.), *Genetic programming* (pp. 61–72). Berlin, Heidelberg: Springer Berlin Heidelberg.

Kronberger, G., Kommenda, M., Promberger, A., & Nickel, F. (2018). Predicting friction system performance with symbolic regression and genetic programming with factor variables. (pp. 1278–1285). http://dx.doi.org/10.1145/3205455.3205522.

Kronberger, G., Winkler, S., Affenzeller, M., & Wagner, S. (2009). On crossover success rate in genetic programming with offspring selection. In L. Vanneschi, S. Gustafson, A. Moraglio, I. De Falco, & M. Ebner (Eds.), *Genetic programming* (pp. 232–243). Berlin, Heidelberg: Springer Berlin Heidelberg.

Kusner, M. J., Paige, B., & Hernández-Lobato, J. M. (2017). Grammar variational autoencoder. In *ICML'17, Proceedings of the 34th international conference on machine learning - vol. 70* (pp. 1945–1954). JMLR.org.

La Cava, W., Orzechowski, P., Burlacu, B., de Franca, F. O., Virgolin, M., J.I.N., Y., Kommenda, M., & Moore, J. H. (2021). Contemporary symbolic regression methods and their relative performance. In *Thirty-fifth conference on neural information processing systems datasets and benchmarks track (Round 1)*. URL: https://openreview.net/forum?id=xVQMrDLyGst.

La Cava, W., Spector, L., & Danai, K. (2016). Epsilon-lexicase selection for regression. (pp. 741–748). http://dx.doi.org/10.1145/2908812.2908898.

Lin, Z., Xu, L., & Bose, N. K. (2008). A tutorial on gröbner bases with applications in signals and systems. *IEEE Transactions on Circuits and Systems. I. Regular Papers*, *55*(1), 445–461. http://dx.doi.org/10.1109/TCSI.2007.914007.

Martins, J. F. B. S., Oliveira, L. O. V. B., Miranda, L. F., Casadei, F., & Pappa, G. L. (2018). Solving the exponential growth of symbolic regression trees in geometric semantic genetic programming. CoRR abs/1804.06808, URL: http://arxiv.org/abs/1804.06808, arXiv:1804.06808.

McConaghy, T. (2011). FFX: Fast, scalable, deterministic symbolic regression technology. (pp. 235–260). http://dx.doi.org/10.1007/978-1-4614-1770-5_13.

McConaghy, T., & Gielen, G. G. E. (2009). Template-free symbolic performance modeling of analog circuits via canonical-form functions and genetic programming. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *28*(8), 1162–1175. http://dx.doi.org/10.1109/TCAD.2009.2021034.

McDermott, J., Agapitos, A., Brabazon, A., & O'Neill, M. (2014). Geometric semantic genetic programming for financial data. (pp. 215–226). http://dx.doi.org/10.1007/978-3-662-45523-4_18.

Miller, J., & Turner, A. (2015). Cartesian genetic programming. In *GECCO Companion '15, Proceedings of the companion publication of the 2015 annual conference on genetic and evolutionary computation* (pp. 179–198). New York, NY, USA: ACM, http://dx.doi.org/10.1145/2739482.2756571, URL: http://doi.acm.org/10.1145/2739482.2756571.

Moraglio, A., Krawiec, K., & Johnson, C. G. (2012). Geometric semantic genetic programming. In C. A. C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, & M. Pavone (Eds.), *Parallel problem solving from nature - PPSN XII* (pp. 21–31). Berlin, Heidelberg: Springer Berlin Heidelberg.

Moscato, P., Sun, H., & Haque, M. N. (2021). Analytic continued fractions for regression: A memetic algorithm approach. *Expert Systems with Applications*, *179*, Article 115018. http://dx.doi.org/10.1016/j.eswa.2021.115018, URL: https://www.sciencedirect.com/science/article/pii/S0957417421004590.

Newton, I. (1687). *Philosophiae naturalis principia mathematica*. J. Societatis Regiae ac Typis J. Streater, URL: https://books.google.es/books?id=-dVKAQAAIAAJ.

Olivetti de França, F. (2018). A greedy search tree heuristic for symbolic regression. *Information Sciences*, *442–443*, 18–32. http://dx.doi.org/10.1016/j.ins.2018.02.040, URL: http://www.sciencedirect.com/science/article/pii/S0020025516308635.

Olson, R. S., La Cava, W., Orzechowski, P., Urbanowicz, R. J., & Moore, J. H. (2017). PMLB: a large benchmark suite for machine learning evaluation and comparison. *BioData Mining*, *10*(1), 36. http://dx.doi.org/10.1186/s13040-017-0154-4.

Orzechowski, P., La Cava, W., & Moore, J. (2018). Where are we now? A large benchmark study of recent symbolic regression methods. http://dx.doi.org/10.1145/3205455.3205539.

Pawlak, T. (2016). *Geometric semantic genetic programming is overkill, vol. 9594*. http://dx.doi.org/10.1007/978-3-319-30668-1_16.

Pérez, J., Miguélez Rico, M., Rabuñal, J., & Abella, F. (2008). Applying genetic programming to civil engineering in the improvement of models, codes and norms. (pp. 452–460). http://dx.doi.org/10.1007/978-3-540-88309-8_46.

Perkis, T. (1994). Stack-based genetic programming. In *Proceedings of the first IEEE conference on evolutionary computation. IEEE world congress on computational intelligence, vol. 1* (pp. 148–153). http://dx.doi.org/10.1109/ICEC.1994.350025.

Petersen, B. K. (2019). Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. arXiv:1912.04871.

Poli, R., Langdon, W. B., & McPhee, N. F. (2008). *A field guide to genetic programming*. Lulu Enterprises, UK Ltd.

Raschka, S. (2015). *Python machine learning*. Birmingham, UK: Packt Publishing.

Sahoo, S. S., Lampert, C. H., & Martius, G. (2018). Learning equations for extrapolation and control. In *Proc. 35th international conference on machine learning, vol. 80* (pp. 4442–4450). PMLR, URL: http://proceedings.mlr.press/v80/sahoo18a.html.

Snyman, J., & Wilke, D. (2018). *Springer optimization and its applications*, *Practical mathematical optimization: basic optimization theory and gradient-based algorithms*. Springer International Publishing, URL: https://books.google.es/books?id=n1dLswEACAAJ.

Udrescu, S.-M., & Tegmark, M. (2020). AI Feynman: A physics-inspired method for symbolic regression. *Science Advances*, *6*(16), http://dx.doi.org/10.1126/sciadv.aay2631, URL: https://advances.sciencemag.org/content/6/16/eaay2631.

Wilson, D. G., Miller, J. F., Cussat-Blanc, S., & Luga, H. (2018). Positional cartesian genetic programming. CoRR abs/1810.04119, arXiv:1810.04119, URL: http://arxiv.org/abs/1810.04119.

Wolpert, D., & Macready, W. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*(1), 67–82. http://dx.doi.org/10.1109/4235.585893.

Worm, T., & Chiu, K. (2013). Prioritized grammar enumeration: Symbolic regression by dynamic programming. In *GECCO 2013 - Proceedings of the 2013 genetic and evolutionary computation conference* (pp. 1021–1028). http://dx.doi.org/10.1145/2463372.2463486.

Zhu, Z., Nandi, A. K., & Aslam, M. W. (2013). Adapted geometric semantic genetic programming for diabetes and breast cancer classification. In *2013 IEEE international workshop on machine learning for signal processing* (pp. 1–5). http://dx.doi.org/10.1109/MLSP.2013.6661969.

Zou, H., & Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society. Series B. Statistical Methodology*, *67*(2), 301–320, URL: http://www.jstor.org/stable/3647580.