Journal of Big Data

RESEARCH

Open Access

RILS-ROLS: robust symbolic regression via iterated local search and ordinary least squares

Aleksandar Kartelj[1*] and Marko Djukanović[2]

*Correspondence:
kartelj@matf.bg.ac.rs

[1] Department of Informatics, Faculty of Mathematics, University of Belgrade, Belgrade, Serbia
[2] Faculty of Sciences and Mathematics, University of Banja Luka, Banja Luka, Bosnia and Herzegovina

Abstract

In this paper, we solve the well-known symbolic regression problem that has been intensively studied and has a wide range of applications. To solve it, we propose an efficient metaheuristic-based approach, called RILS-ROLS. RILS-ROLS is based on the following two elements: (*i*) iterated local search, which is the method backbone, mainly solving combinatorial and some continuous aspects of the problem; (*ii*) ordinary least squares method, which focuses on the continuous aspect of the search space—it efficiently determines the best—fitting coefficients of linear combinations within solution equations. In addition, we introduce a novel fitness function that combines important model quality measures: R^2 score, *RMSE* score, size of the model (or model complexity), and carefully designed local search, which allows systematic search in proximity to candidate solution. Experiments are conducted on the two well-known ground-truth benchmark sets from literature: Feynman and Strogatz. RILS-ROLS was compared to 14 other competitors from the literature. Our method outperformed all 14 competitors with respect to the symbolic solution rate under varying levels of noise. We observed the robustness of the method with respect to noise, as the symbolic solution rate decreases relatively slowly with increasing noise. Statistical analysis of the obtained experimental results confirmed that RILS-ROLS is a new state-of-the-art method for solving the problem of symbolic regression on datasets whose target variable is modelled as a closed-form equation with allowed operators. In addition to evaluation on known ground-truth datasets, we introduced a new randomly generated set of problem instances. The goal of this set of instances was to test the sensitivity of our method with respect to incremental equation sizes under different levels of noise. We have also proposed a parallelized extension of RILS-ROLS that has proven adequate in solving several very large instances with 1 million records and up to 15 input variables.

Keywords: Symbolic regression, Iterated local search, Ordinary least squares, Ground-truth benchmark sets

Introduction

The problem of symbolic regression (SR) [1] has attracted a lot of research interest over the last decade. SR can be seen as a generalization of more specific variants of regression in which the functional form is fixed, e.g., the well-known linear

regression, polynomial regression [2], etc. All regression models have the same goal: given a set of n-dimensional input data and its corresponding continuous output target variable, the aim is to find a mathematical expression (function) of n (input) variables that best *fits* the target variable. In the case of linear regression, the model is presumed to be a linear combination of input variables. This is in general not well enough, since the target variable might be dependent on a nonlinear function among input variables. Unlike linear regression, SR allows for the search over a wider space of possible mathematical formulas to find the best-fitting ones, i.e., those able to predict the target variable from input variables. The basis of constructing an explicit formula is in elementary operations like addition and multiplication, as well as polynomial, trigonometric, exponential, and other operations.

Coefficients inside SR formulas can indicate the absolute or relative importance of certain input variables. The appearance of an exponential function can be associated with a specific physical phenomenon such as the intensity of radiation or acceleration over time (see [3]). Additionally, SR models often have high generalization power unlike some models with fixed functional forms like polynomial regression.

Practical applications of SR in chemical and biological sciences are listed in [4]. That paper in particular describes the discovery of a series of new oxide perovskite catalysts with improved activities. The application of SR to discovering physical laws from the data extracted from a video is studied in [5]. The revealing complex ecological dynamics by SR is presented in [6]. The paper [7] presents the application of the SR to model the effects of mutations on protein stability in the domain of fundamental and applied biology. A recent study in [8] is concerned with auto-discovering conserved quantities using trajectory data from unknown dynamical systems. The application of the SR to model analytic representations of exciton binding energy is shown in [9]. The use of SR in material science is described in [10–13]. SR application to wind speed forecasting is given in [14].

There are many different ways to tackle SR. Most of them are based on machine learning (ML) techniques, genetic programming (GP), or some other metaheuristics. Koza [15] introduced the SR problem as a specific application of GP. GP is also used to optimize nonlinear structures such as computer programs represented by syntax trees consisting of functions/operations over input variables and constants. Age-fitness Pareto Optimization approach is proposed by Schmidt and Lipson [16]. Application of the artificial bee colony programming to solve SR is proposed by Karaboga et al. [17]. Application of local-based heuristics for solving SR is reported by Kommenda in his PhD thesis [18]. A GP-based approach, the gene-pool optimal mixing evolutionary algorithm (GOMEA) is studied by Virgolin et al. [19]. Another evolutionary algorithm, the interaction-transformation EA (ITEA) has been proposed by de Franca et al. [20]. Simulated annealing to solve SR is proposed by Kantor [21]. Kommenda et al. [22] proposed a method called Operon algorithm, which uses nonlinear least squares for parameter identification of SR models further integrated into a local search mechanism in tree-based GP. The C++ implementation of Operon is discussed in [23]. A GP approach that uses the idea of semantic back-propagation (SBP-GP) is proposed in [24]. It is also worth to mention the GP-based Eureqa commercial solver [25, 26] that uses age-fitness Pareto optimization with co-evolved fitness estimation.

Table 1 SR methods overview

Algorithm	Paper/year	Short details
GP	[15] (1994)	Application of GP to SR
APF-FE	[25, 26] (2009, 2011)	Age-fitness Pareto optimization approach using co-evolved fitness estimation
APF	[16] (2010)	Age-fitness Pareto optimization approach
FFX	[33] (2011)	The fast function extraction algorithm – non-evolutionary technique based on a machine learning technique called path-wise regularized learning
ABCP	[17] (2012)	Artificial bee colony programming approach
EPLEX	[32] (2016)	A parent selection method called ϵ-lexicase selection
MRGP	[34] (2014)	It decouples and linearly combines a program's subexpressions via multiple regression on the target variable
Local optimization NLS	[18] (2018)	Constants optimization in GP by nonlinear least squares
FEAT	[35] (2018)	Features are represented as networks of multi-type expression trees comprised of activation functions; differentiable features are trained via gradient descent
SBP-GP	[24] (2019)	The idea of semantic back-propagation utilized in GP
BSR	[27] (2019)	ML-based approach; Bayesian symbolic regression
DSR	[28] (2019)	Deep symbolic regression based on a RNN approach further utilizing the policy gradient search
Operon	[22] (2020)	Utilizing nonlinear least squares for parameter identification of SR models with LS
OccamNet	[30] (2020)	A fast neural network approach; the model defines a probability distribution over a non-differentiable function space; it samples functions and updates the weights with back-propagation based on cross-entropy matching in an EA strategy
AI-Feynman	[3] (2020)	A physics-inspired divide-and-conquer method; it also uses neural network fitting
GOMEA	[19] (2021)	A model-based EA framework called gene-pool optimal mixing evolutionary algorithm
ITEA	[20] (2021)	EA based approach called the interaction-transformation EA
SA	[21] (2021)	Simulated annealing approach

The method based on Bayesian symbolic regression (BSR) is proposed in [27]—this method belongs to the family of Markov Chain Monte Carlo algorithms (MCMC). Deep Symbolic Regression (DSR), an RNN approach that utilizes the policy gradient search, is proposed in [28]. This mechanism of search is further investigated in [29]. A fast neural network approach, called OccamNet is proposed in [30].

Powerful hybrid techniques for solving SR are also well studied; among them, we emphasize the Eplex, a GP-based SR implementation that uses eps-lexicase selection, proposed by [31, 32] and AI-Feynman algorithm, based on a physics-inspired divide-and-conquer method combined with the neural network fitting, proposed by [3]. The latter is one of the most efficient methods for physically-inspired models. We would also like to mention the Fast Function Extraction (FFX) algorithm developed by McConaghy [33], which is a non-evolutionary method combined with a machine learning technique called path-wise-regularized learning, which quickly prunes a huge set of candidate basis functions down to compact models.

A short overview of the most important literature methods to solve SR is given in Table 1.

SR research lacks uniform, robust, and transparent benchmarking standards. Recently, La Cava et al. [36] proposed an open-source, reproducible benchmarking platform for SR called `SRBench`. `SRBench`[1] is an open source project that brings together a large number of different benchmark datasets, contemporary SR methods and ML methods for joint model evaluation, and an environment for analysis. `SRBench` considers two types of symbolic regression problems: 1) *ground-truth* problems, which are modelled as known closed-form equations, and 2) *black-box* problems, where the exact model is not known. Note that in our work, we consider only the former one, since RILS-ROLS was not designed to solve black-box problems. La Cava et al. [36] extended PMLB [37], a repository of standardized (mostly black-box) regression problems, with 130 ground-truth SR datasets. In their extensive experimental evaluation, 14 SR methods and 7 ML methods are compared on the set of 252 diverse regression problems. The remaining 122 SR datasets represent black-box problems. One of the most interesting conclusions from [36] is that algorithms specialize either in solving problems with an underlying closed-form equation (tested using ground-truth problem instances) or in black-box problems, but not both.

In this work, we present a new approach to solving closed-form SR problems (with ground-truth not necessarily known), which combines the popular iterated local search metaheuristic (ILS) [38, 39] with the ordinary least square method (OLS) [40]. ILS mostly handles combinatorial (discrete) aspects of search space, while OLS deals with the process of coefficient determination, so it handles some continuous parts of the search space. As will be shown later, the proposed method has shown its robustness w.r.t. introduced noise, which is why we called the method RILS-ROLS (with letters R for regression and robust). Additionally, to navigate the search toward the exact model (and not only the accurate one) the algorithm is equipped with a carefully constructed fitness function that combines important model characteristics.

The summary of main contributions is as follows:

1. The proposed method outperforms 14 comparison methods on two ground-truth benchmark sets from the literature—these methods and benchmarks are used in the `SRBench` platform.
2. The method exhibits high relative robustness to noise, as evidenced by a comparison to other algorithms w.r.t. obtained symbolic solution rate, in the presence of varying degrees of noise in the data.
3. A new set of unbiased instances is introduced—it consists of randomly generated formulae of various sizes and a number of input variables. This set was employed to analyze the effect of model size and the level of noise on the solving difficulty.
4. A new set of very large datasets with 1 million records is generated to test the performance of RILS-ROLS for a Big Data scenario. Our method is adapted to run multiple RILS-ROLS regressors in parallel, thus acting as a kind of ensemble method.

[1] https://cavalab.org/srbench/.

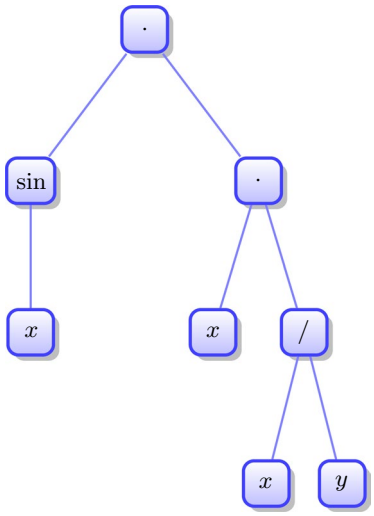


Fig. 1 Syntax tree representation for the expression $\sin x \cdot (x \cdot x/y) = \frac{x^2 \sin x}{y}$

Problem definition and search space

In this section we formally define the SR problem.

Definition 1 Given is a dataset $D = \{(\mathbf{x_i}, y_i)\}_{i=1}^{n}$, where $\mathbf{x_i} \in \mathbb{R}^d$ represents the i-th feature (input) vector while $y_i \in \mathbb{R}$ is its corresponding target (output) variable. Suppose that there exists an analytical model of the form $f(\mathbf{x}) = g^*(\mathbf{x}, \theta^*) + \epsilon$ that is a generator of all observations from D. The goal of SR is to learn the mapping $\tilde{f}(\mathbf{x}) = \tilde{g}(\mathbf{x}, \tilde{\theta}) : \mathbb{R}^d \mapsto \mathbb{R}$ estimated by searching through the space of (mathematical) expressions $\tilde{g}$ and parameters $\tilde{\theta}$ where ϵ is the observed white noise within the given input data.

Any mathematical expression can be represented by a syntax tree. As an example of such a syntax tree, see Fig. 1. In essence, all valid syntax trees form the solution search space of SR. That is, each sample model $\tilde{f}$ may be seen as a point in the search space, represented by a respective syntax tree. The solution accuracy can be computed on the basis of historical data D and the chosen error measure such as *MSE*, *RMSE*, R^2, their combination, etc. Interestingly, the nature of SR search space is twofold: discrete and continuous. It is primarily modeled as a problem of discrete (combinatorial) optimization, since the number of possible solution functional forms is countable. However, it may also include elements solved by means of continuous (global) optimization, e.g., constants (coefficients) fitting. It is common to use the set of the following elementary mathematical functions: $\sqrt{x}, x^2$, sin, cos, log, exp, arcsin, arccos, a^x, and a set of standard arithmetic operators: $+$, $-$, $\cdot$, and $/$.

The proposed RILS-ROLS method

Our method relies on the following operations: $+$, $-$, $\cdot$, $/$, $\sqrt{x}$, x^2, sin, cos, log and exp. Beside these, the following set of constants enters the search space explicitly: -1, 0, 1, 2, π, and 10. Before we provide details of our method for solving SR, we will explain

the two building blocks of RILS-ROLS: (1) iterated local search (ILS) metaheuristic and (2) ordinary least squares (OLS) method.

Iterated local search

ILS [38] is an efficient, single-point search metaheuristic that iteratively generates a sequence of solutions produced by the (embedded) heuristic, such as local search (LS) or randomized greedy heuristics. When the search gets *stuck* in the local optimum, *perturbation* is performed, which is usually a non-deterministic step. This simple idea was proposed by Baxter [41] in the early 1980s, and has since then been re-invented by many researchers under different names. These are some of the names that were used: iterated descent search [42], large-step Markov chains [43], chained local optimization [44], or in some cases, combinations of those [45]. The most popular version of ILS is shown in Algorithm 1. (Note that we use this version of ILS as the backbone of our RILS-ROLS algorithm.)

Algorithm 1 General ILS method.

```
Input: problem instance
Output: feasible solution
s ← Initialize()
s ← LocalSearch(s)
while stopping criteria are not met do
    s′ ← Perturbation(s)
    s′ ← LocalSearch(s′)
    s ← AcceptanceCriterion(s, s′)
end while
return s
```

An initial solution may be generated randomly or by using a greedy heuristic, which is afterwards improved by local search. At each iteration, ILS applies three steps. First, the current incumbent solution s is perturbed, i.e., partially randomized, yielding a new solution s'. Next, the solution s' is potentially improved by an LS procedure. Thirdly, the newly obtained solution s' possibly becomes a new incumbent—this is decided upon the acceptance criterion. Sometimes, ILS incorporates the mechanism of a tabu list, which prevents the search from getting back into already visited solutions.

Ordinary least squares method

The ordinary least square method (OLS) is a linear regression technique. It is based on applying the least-squares method to minimize the square residual (error) sum between actual and predicted values (given by the model). More precisely, given the dataset D of n points $(\mathbf{x_i}, y_i)$ where each $\mathbf{x_i}$ is d-dimensional, the task is to determine linear mapping $\hat{y} = \mathbf{kx} + b$, that is coefficients (line slope) $\mathbf{k} = (k_1, \ldots, k_d)$ and b (intercept), so that $\sum_i^n (\hat{y}_i - y_i)^2$ is minimized. This sum is also known as the sum of squared errors (SSE). There are many methods to minimize SSE. One of the analytical approaches is calculus-oriented so it takes into account the partial derivatives of SSE w.r.t. k_j, $j \in \{1, \ldots, d\}$ and b. This leads us to a system of $d + 1$ linear equations with $d + 1$ unknowns. The obtained linear system can be solved efficiently since the expensive matrix inversion can

be avoided by using, for example, the SVD technique which generates the pseudoinverse solution that solves the original system (see [46]).

RILS-ROLS method

We will now explain the proposed RILS-ROLS method in detail. The overall method scheme is given in Algorithm 2.

Algorithm 2 RILS-ROLS method.

Input: input training dataset D_{tr}
Control parameters: size penalty $penalty_{size}$, perturbations order $order_{random}$, initial sample share $sampleShare_{init}$
Output: best symbolic formula solution bs

procedure RILS-ROLS (D_{tr})
  $n_{tr} \leftarrow |D_{tr}|$
  $sample_{size} \leftarrow \max(\lfloor sampleShare_{init} \cdot n_{tr} \rfloor, \min(n_{tr}, 100))$
  $D'_{tr} \leftarrow$ `Sample`$(D_{tr}, sample_{size})$
  $s \leftarrow NodeConstant(0)$
  $s_{fit} \leftarrow$ `Fitness`(s, D'_{tr})
  $bs, bs_{fit} \leftarrow s, s_{fit}$
  $start_{tried}, perturbations_{tried} \leftarrow \emptyset, \emptyset$
  while *stopping criteria is not met* **do**
    $start_{tried} \leftarrow start_{tried} \cup \{s\}$
    $s_{perturbations} \leftarrow$ `All1Perturbations`(s)
    if $order_{random}$ **then**
      $s_{perturbations} \leftarrow$ `RandomShuffle`$(s_{perturbations})$
    else
      $s_{perturbations} \leftarrow$ `FitOLS`$(s_{perturbations}, D'_{tr})$
      $s_{perturbations} \leftarrow$ `OrderByR2`$(s_{perturbations})$
    end if
      $improved \leftarrow false$
      for $p \in s_{perturbations}$ **do**
        if $p \in perturbations_{tried}$ **then**
          continue
        end if
        $p \leftarrow$ `Simplify`(p)
        $p \leftarrow$ `LocalSearch`(p, D'_{tr})
        $p_{fit} \leftarrow$ `Fitness`(p, D'_{tr})
        if $p_{fit} < bs_{fit}$ **then**
          $bs, bs_{fit}, improved \leftarrow p, p_{fit}, true$ // new best solution
          break
        end if
        $perturbations_{tried} \leftarrow perturbations_{tried} \cup \{p\}$
      end for
      if improved **then**
        $s \leftarrow bs$
      else
        $start_{candidates} \leftarrow$ `All1Perturbations`(bs)
        if $start_{candidates} \setminus start_{tried} = \emptyset$ **then** // all 1-perturbations around bs tried
          $s' \leftarrow$ `RandomPick`$(start_{candidates})$
          $s \leftarrow$ `RandomPick`(`All1Perturbations`$(s'))$ // random 2-perturbation
        else
          $s \leftarrow$ `RandomPick`$(start_{candidates} \setminus start_{tried})$
        end if
          if not improved for too many iterations **then**
            $sample_{size} \leftarrow$ `IncreaseSampleSize`$(sample_{size}, n_{tr})$
            $D'_{tr} \leftarrow$ `Sample`$(D_{tr}, sample_{size})$
          end if
        end if
        if R^2 almost 1 and *RMSE* almost 0 **then**
          break // early exit
        end if
      end while
      $bs \leftarrow$ `Simplify`(bs)
      $bs \leftarrow$ `RoundModelCoefficients`(bs)
      return bs
    end procedure

The RILS-ROLS algorithm receives the training dataset D_{tr} as input. In addition, it has three key control parameters: $penalty_{size}$, $order_{random}$ and $sampleShare_{init}$. There are some other standard parameters like the maximum execution time, the maximum number of iterations, the pseudo-random number generator seed, etc. The first key parameter quantifies the importance of solution expression complexity in the overall solution quality measure (more on this in "Fitness function" section). The second refers to the way the candidate perturbations are arranged: if *False* (default), the perturbations are arranged w.r.t. R^2, otherwise, the arrangement is random. The third parameter specifies the proportion of the training sample to be used. If this value is set to 1, the entire training dataset is used. The default value of this parameter is 0.01 (1 percent). To avoid useless samples, there is a lower limit for the sample size—100. (Of course, in cases where the size of the training dataset is less than 100, the entire training dataset is used.) The size of the sample is later dynamically adjusted through the algorithm's iterations—when there are no solution improvements for a number of iterations, sample size is doubled (lines 43–44 of Algorithm 2).

As previously stated, solution is represented by means of a tree. We use a simple solution initialization where the tree root node is set to zero constant. We interchangeably use two solution variables: (*i*) *s* denotes the starting (or working) solution and (*ii*) *bs* stands for the best solution so far, also known as the incumbent solution. Solution quality is measured by evaluating the fitness function (more about it in the subsequent "Fitness function" section). Before entering the main loop, the best solution *bs* is set to the initial solution (line 7).

The main loop iterates as long as none of the termination criteria are met: (*i*) the maximal running time has been reached; (*ii*) the maximal number of fitness calculations has been made; (*iii*) the best solution is sufficiently good w.r.t. its R^2 and *RMSE* scores. More precisely, if R^2 is sufficiently close to 1 and, at the same time *RMSE* is sufficiently close to 0, the algorithm stops prematurely, which significantly reduces the running times for the majority of tested instances in the no-noise scenario.

One of the first steps in the main loop is to generate perturbations near the starting solution *s* (line 11). As the name of this procedure (`All1Perturbations`) suggests, the perturbation step is local, meaning that the closeness of the starting solution *s* and any of perturbations is 1 (we sometimes call it 1-perturbation). The precise way of generating perturbation is described separately in "Perturbations" section.

Candidate perturbations are improved by performing OLS coefficient fitting (procedure `FitOLS`). This means that the coefficients in any of the linear combinations of the current solution are being set by applying the ordinary least square method already described in "Ordinary least squares method" section. Note that our implementation uses the OLS method included as part of the popular Python package `statsmodels.api`. After this step, perturbations are usually better suited to the given sample data D'_{tr}. Further, these perturbations are sorted w.r.t. R^2 metric in the descending order (line 16).

Now the algorithm enters the internal loop—it iterates over the ordered perturbations and aims to find the one which improves the best solution *bs*. But before comparing candidate perturbation solution p with *bs*, p is first simplified (line 23), after which local search is performed (line 24). Solution simplification is done in a symbolic fashion by popular Python package `SymPy` [47]. The local search tries to find local optima expressions close to the given p, explained in detail in "Local search" section. Finally, the fitness function value of p is compared to the fitness function value of *bs*. If fitness of p is better, *bs* is updated correspondingly and the internal loop, which goes across ordered perturbations, is immediately terminated (it works in a *first-improvement* strategy). Otherwise, the next perturbation is probed. Note that the probed perturbations are stored in a set denoted by $perturbations_{tried}$. The goal is to avoid checking the same perturbation multiple times (lines 26–29), i.e., $perturbations_{tried}$ serves as a kind of tabu list, which is known from the Tabu search metaheuristic (see [48]).

If some of $s_{perturbations}$ around the starting solution s yielded an improvement, *bs* becomes the starting solution s in the next iteration of the main loop (line 33). Otherwise, it makes no sense to set the starting solution to *bs* as the search becomes *trapped* in a local optimum. Randomness is introduced in order to avoid this undesirable situation. First, a set of local perturbations around *bs* is generated ($start_{candidates}$) in the same manner as before (procedure `All1Perturbations`). If at least one of these was not previously used as a starting solution ($start_{tried}$), a single perturbation from the $start_{candidates} \setminus start_{tried}$ is randomly picked (line 40). There is a minor chance that $start_{candidates} \setminus start_{tried} = \emptyset$. In this case, a random perturbation at distance 2 from *bs* is used.

Before returning the symbolic model, RILS-ROLS performs the final symbolic simplification and rounding of model coefficients.

Fitness function

The objective of symbolic regression is to determine the expression that fits available data. It is also allowed to obtain an equivalent expression, since there are multiple ways to express a symbolic equation. Although logically sound and intuitive, this objective is not quantifiable during the solution search/training phase, because the goal expression is not known at that point but only target values for some of the input data. Therefore there are various numerical metrics in the literature that guide the symbolic regression search process. The most popular ones are the coefficient of determination, also known as R^2, and the mean squared error (MSE) or root mean squared error (RMSE). The important aspect of solution quality is solution expression complexity, which may correspond to the model size of its tree representation. This follows the Occam's razor principle [30] that a simpler solution is more likely to be the correct one. The search process of RILS-ROLS is guided by the non-linear combination of R^2, *RMSE*, and solution expression size (complexity) presented in Eq. (1).

$$fit(s) = (2 - R^2(s)) \cdot (1 + RMSE(s)) \cdot (1 + penalty_{size} \cdot size(s)) \quad (1)$$

$$R^2(s) = 1 - \frac{\sum_{i=1}^{n} \left(s(\mathbf{x_i}) - y_i\right)^2}{\sum_{i=1}^{n} \left(y_i - \bar{y}\right)^2} \tag{2}$$

$$RMSE(s) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(s(\mathbf{x_i}) - y_i)^2} \tag{3}$$

Where n stands for the number of records in the dataset under consideration, $s(\mathbf{x_i})$ for the value predicted by the candidate model s for an input vector $\mathbf{x_i}$, y_i for the (exact) target value, and $\bar{y}$ for the mean of all target values.

The size of expression is calculated by counting all nodes in the respective expression tree—this includes leaves (variables and constants) and internal nodes (operations).

The presented fitness function needs to be minimized. The motivation behind the structure of the fitness function is based on the following observations:

- higher R^2 is preferred, ideally when $R^2(s) = 1$, the effect of term $2 - R^2(s)$ is neutralized;
- lower $RMSE$ is preferred, ideally when $RMSE(s) = 0$, the whole term $(1 + RMSE(s))$ becomes 1;
- since $penalty_{size} > 0$, larger expressions tend to have higher fitness (which follows the Occam's razor principle); therefore, simpler solutions are favorable.

Expression caching

In order to speed up fitness evaluation, we employ *expression caching*. This means that values attached to expression trees or subtrees are stored in key-value structure, such that the key is tree (subtree) textual representation, while the value is the $|D'_{tr}|$-size vector of corresponding expression values on the sample training dataset D'_{tr}. Of course, once the D'_{tr} changes, which does not happen very frequently, the whole cache is cleared. Caching is performed in a partial way—when determining the value of a given expression T, it is not required to find the exact *hit* inside the cache. So, if a subexpression (subtree) of T is present in the cache, its value will be reused and further combined to calculate the whole fitness function value.

For example, let $T = y^2 \cdot (x + y)/z - \sin(x + y)$ be an expression, and $D'_{tr} = \{([1, 2, 5], 7), ([3, 4, 3], 5), ([4, 5, 3], 6), ([6, 7, 4], 3), ([3, 3, 6], 2)\}$ be a sample training dataset (here, input feature vector is labeled by $[x, y, z]$); let expression cache consist of the following key-value entries $cache = \{(x + y, [3, 7, 9, 13, 6]), (y^2, [4, 16, 25, 49, 9])\}$. Expression T does not need to be fully evaluated since some of its parts are inside the cache: y^2 and $x + y$. Note that single variables do not need to enter the cache, since they are already available as columns of D'_{tr}. Each newly evaluated (sub)expression (except for the constant or variable alone) enters the cache. In this example, the new entries will correspond to keys $(x + y)/z$, $y^2 \cdot (x + y)/z$, $\sin(x + y)$ and $y^2 \cdot (x + y)/z - \sin(x + y)$. The maximal number of cache entries is set to 5000. As soon as this number is reached,

the cache is cleared, although a more sophisticated LRU (Least Recently Used) caching strategy can be used.

Perturbations

Perturbations allow the algorithm to escape from local optima. As previously described, perturbations are performed on two occasions: (*i*) during the exhaustive examination of neighboring solutions around the starting solution, (*ii*) during selection of the next starting solution, a non-exhaustive case. In both cases, the same Algorithm 3 is used.

Algorithm 3 Generation of all 1-perturbations of a given solution.

Input: solution s
Output: local perturbations (1-perturbations) of solution s – $s_{perturbations}$
1: **procedure** ALL1PERTURBATIONS(s)
2: $s_{perturbations} \leftarrow \emptyset$
3: $s \leftarrow$ `NormalizeConstants`(s)
4: $s \leftarrow$ `Simplify`(s)
5: $s_{subtrees} \leftarrow$ `SubTrees`(s)
6: **for** $n \in s_{subtrees}$ **do**
7: $n_{perturbations} \leftarrow$ `All1PerturbationsAroundNode`(s, n)
8: $s_{perturbations} \leftarrow s_{perturbations} \cup n_{perturbations}$
9: **end for**
10: **return** $s_{perturbations}$
11: **end procedure**

Initially, the set of perturbations $s_{perturbations}$ is empty (line 2 of Algorithm 3).

This is followed by constant normalization during which coefficients that enter multiplication, division, addition or subtraction are set to 1, while those entering the power function are rounded to integer, with the exception of square root, which is kept intact. For example, for expression $3.3 \cdot (x + 45.1 \cdot y^{3.2}) \cdot 81 \cdot x/\sqrt{y}$ the normalized version is $1 \cdot (x + 1 \cdot y^3) \cdot 1 \cdot x/\sqrt{y}$. The reason for performing normalization is reducing the search space of possible perturbations. This reduction is reasonable, since normalization preserves the essential functional form. Note that coefficients get tuned later: the linear coefficient during the OLS phase, and the remaining ones during local search, see "Local search" section.

After performing the normalization process, the expression is simplified—getting the compact expression is more likely after normalization than before it. The previous expression will take the form $(x + y^3) \cdot x/\sqrt{y}$. In this particular case, the simplification will usually only remove unnecessary coefficients, but in general it can also perform some non-trivial symbolic simplification.

Perturbations are generated by making simple changes on the per-node level of s expression tree. Depending on the structure of the expression tree (note that the expression does not need to have unique tree representation), the set of subtrees of the previous expression $(x + y^3) \cdot x/\sqrt{y}$ might be $\{(x + y^3) \cdot x/\sqrt{y}, (x + y^3), x/\sqrt{y}, x, y^3, \sqrt{y}, y\}$. Further, the set of perturbations is generated around each subtree n (lines 6–9 in Algorithm 3).

Algorithm 4 shows how perturbations are generated around the given subtree.

Algorithm 4 Generation of 1-perturbations of a given solution around given node.

```
    Input: solution s, node n
    Output: 1-perturbations of solution s around node n
 1: procedure All1PerturbationsAroundNode(s, n)
 2:     n_perturbations ← ∅
 3:     if n = s then
 4:         n_perturbations ← n_perturbations ∪ NodeChanges(n)
 5:     end if
 6:     if n.arity ≥ 1 then
 7:         n_changes ← NodeChanges(n.left)
 8:         for nc ∈ n_changes do
 9:             new ← Replace(s, n.left, nc)
10:             n_perturbations ← n_perturbations ∪ {new}
11:         end for
12:     end if
13:     if n.arity = 2 then
14:         n_changes ← NodeChanges(n.right)
15:         for nc ∈ n_changes do
16:             new ← Replace(s, n.right, nc)
17:             n_perturbations ← n_perturbations ∪ {new}
18:         end for
19:     end if
20:     return n_perturbations
21: end procedure
```

It can be seen that there are three possibly overlapping cases when performing perturbations on the per-node level.

Case 1. The observed node n is the whole tree s (see line 3 in Algorithm 4). Based on the previous exemplary expression tree, this means that the multiplication node that connects $(x + y^3)$ and $x/\sqrt{y}$ is to be changed. For example, multiplication can be replaced by addition, which forms a 1-perturbation expression (tree) $(x + y^3) + x/\sqrt{y}$.

Case 2. Node n has the arity of at least 1 (see line 6 in Algorithm 4). This means that the left subtree exists, so the left subtree node is to be changed. For example, if $n = x + y^3$, the overall perturbation might be $(x/y^3) + x/\sqrt{y}$ (addition is replaced by division). Another example would be the case of unary operation, e.g., when $n = \sqrt{y}$. In that case, some possible perturbations could be $(x + y^3) \cdot x/\sqrt{\ln y}$ (application of logarithm to the left subtree y) or $(x + y^3) \cdot x/\sqrt{x}$ (changing variable y to x), etc.

Case 3. Node n is a binary operation, meaning that the right subtree must exist (Line 13 in Algorithm 4). The analogous idea is applied as in *Case 2*.

The algorithm allows for the following set of carefully chosen per-node changes (method named `NodeChanges` in Algorithm 4):

1. Any node to any of its subtrees (excluding itself). For example, if $(x + y^3)$ is changed to x, the perturbation is $(x + y^3) \cdot x/\sqrt{y} \rightarrow x \cdot x/\sqrt{y}$.
2. Constant to variable. For example, $(1 + y^3) \cdot x/\sqrt{y} \rightarrow (x + y^3) \cdot x/\sqrt{y}$.
3. Variable to the unary operation applied to that variable. For example, $(x + y^3) \cdot x/\sqrt{y} \rightarrow (x + y^3) \cdot \ln x/\sqrt{y}$.
4. Unary operation to another unary operation. For example, $(x + y^3) \cdot x/\sqrt{y} \rightarrow (x + y^3) \cdot x/\sin y$.
5. Binary operation to another binary operation. For example, $(x + y^3) \cdot x/\sqrt{y} \rightarrow (x + y^3) \cdot (x + \sqrt{y})$.

6. Variable or constant enter the binary operation with an arbitrary variable. For example, $(x + y^3) \cdot x/\sqrt{y} \rightarrow (x + x/y^3) \cdot x/\sqrt{y}$.

The method named `Replace`(s, n, nc) inside Algorithm 4 is simply used to replace the node n with node nc inside the given expression tree s.

Local search

Perturbations are further improved by means of the local search procedure (Algorithm 5).

Algorithm 5 Local search procedure.

Input: perturbation p, sample training dataset D'_{tr}
Output: local optimum bp in the vicinity of perturbation p

procedure LocalSearch(p, D'_{tr})
  $bp, bp_{fit} \leftarrow p$, `Fitness`$(p, D'_{tr})$
  $improved \leftarrow true$
  while $improved$ **do**
    $improved \leftarrow false$
    $bp_{candidates} \leftarrow \emptyset$
    $bp_{subtrees} \leftarrow$`SubTrees`(bp)
    for $n \in bp_{subtrees}$ **do**
      $n_{candidates} \leftarrow$ `All1PerturbationsAroundNodeExtended`(bp, n)
      for $new \in n_{candidates}$ **do**
        $new \leftarrow$ `FitOLS`(new, D'_{tr})
        $new_{fit} \leftarrow$ `Fitness`(new, D'_{tr})
        if $new_{fit} < bp_{fit}$ **then**
          $bp, bp_{fit} \leftarrow new, new_{fit}$
          $improved \leftarrow true$
        end if
      end for
    end for
  end while
  return bp
end procedure

For a given perturbation p, local search systematically explores the extended set of 1-perturbations around p. It relies on the *best-improvement* strategy, meaning that all 1-perturbations (for all subtrees) are considered. Before checking if the candidate solution (*new*) is better than the actual best *bp*, the OLS coefficient fitting (`FitOLS`) takes place.

The set of used 1-perturbations is expanded in comparison to those used previously. Namely, in addition to those six possible types of node changes, the following four are added:

7. Any node to any constant or variable. For example, $(x + y^3) \cdot x/\sqrt{y} \rightarrow y \cdot x/\sqrt{y}$ or $(x + y^3) \cdot x/\sqrt{y} \rightarrow (x + \pi) \cdot x/\sqrt{y}$.
8. Any node to the unary operation applied to it. For example, $(x + y^3) \cdot x/\sqrt{y} \rightarrow (x + y^3) \cdot x/\ln\sqrt{y}$.
9. Any node to the binary operation applied to that node and a variable or constant. For example, $(x + y^3) \cdot x/\sqrt{y} \rightarrow (x + y^3) \cdot x/\sqrt{y} - x$.
10. Constant to its multiple, whereby possible multipliers are {0.01, 0.1, 0.2, 0.5, 0.8, 0.9, 1.1, 1.2, 2, 5, 10, 20, 50, 100}. For example, $(1 + y^3) \cdot x/\sqrt{y} \rightarrow (1.2 + y^3) \cdot x/\sqrt{y}$.

Change under point 10 is very important as it performs general coefficient tuning, unlike OLS, which considers only coefficients in linear combinations.

Computational complexity of RILS-ROLS algorithm

The most expensive part of the algorithm is the invocation of the local search for a given subtree (line 24 of Algorithm 2). The total computational complexity incurred by calling this line of code, during a single iteration of the main loop, is equal to the multiple of $|s_{perturbations}|$ and the complexity of a single call to the local search.

Let us compute the number of $s_{perturbations}$ set, generated by Algorithm 3. The number $C(\cdot)$ of subtrees (line 5) of a given binary tree T, with $|V|$ vertices, depends on the tree structure. The upper bound on the number of subtrees is proved by Székely and Wang in [49]. The result states that the star tree $K_{1,|V|-1}$ has $2^{|V|-1} + |V| - 1 = O(2^{|V|})$ subtrees. On the other hand, $|V|$-vertex path $P_{|V|-1}$ has the smallest number of subtrees, only $\binom{|V|+1}{2}$. Also, a constant number of perturbations (procedure `All1PerturbationsAroundNode`) is generated for each subtree, which does not influence the complexity in terms of O notation.

For a given perturbation, the local search tries to find the best improvement for each of the subtrees. Therefore, in the worst case, the local search call itself is exponential.

In the worst case, the computational complexity of a complete iteration of the overall algorithm is therefore exponential to the size of the current solution, i.e., the number of tree vertices. However, real closed-form equations are usually relatively small, so the exponential computational complexity is not such a major practical obstacle for RILS-ROLS. Moreover, the number of subtrees will not be maximal on average, i.e., the expression will rarely take the form of star tree $K_{1,|V|-1}$.

RILS-ROLS for parallel execution

RILS-ROLS can be executed in the parallel environment, which can be especially useful for Big Data scenarios. This method, called RILS-ROLS Ensemble, is based on multiple RILS-ROLS regressors with different settings of the pseudo-random number generator. In addition, the $order_{random}$ parameter is set to *True* (the default case is *False*). The use of randomly ordered perturbations allows the regressors to have different search trajectories through the search space. Once all regressors have finished their executions, their symbolic models are evaluated on the whole training dataset, and the best one is selected as the final symbolic model. The number of regressors, i.e., the degree of parallelism, is given as the control parameter. Each regressor runs in a separate process, so the host operating system decides how parallelism is controlled. In our experiments, described in "Results on big data" section, each process was given its own logical core when the number of processes was less than the number of available logical cores.

Experimental evaluation

Our RILS-ROLS algorithm is implemented in Python 3.9.0, run and evaluated by `SRBench` platform. All experiments concerning our method are conducted in the single-core mode, on a PC with Intel i9-9900KF CPU @3.6GHz, 64GB RAM, under Windows 10 Pro OS.

Table 2 Some Feynman instances

Output	Formula
d	$\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$
θ_1	$\arcsin(n\cdot\sin\theta_2)$
E_n	$\frac{m\cdot c^2}{1-\frac{v^2}{c^2}}$
ω	$\frac{1+\frac{v}{c}}{\sqrt{1-\frac{v^2}{c^2}}}\cdot\omega_0$
A	$x_1\cdot y_1+x_2\cdot y_2+x_3\cdot y_3$
Pwr	$(1/2\cdot\epsilon\cdot c\cdot E_f^2)\cdot(8\cdot\pi\cdot r^{2/3})\cdot(\omega^4/(\omega^2-\omega_0^2)^2)$

The following 14 algorithms are compared to our approach: AI-Feynman, GOMEA, Afp-Fe, Itea, Afp, Dsr, Operon, the gplearn from python package gplearn [50], Sbp-Gp, Eplex, Bsr, Feat, Ffx, Mrgp.

The results of the competing methods are published in [36] and are also available within the `SRBench` platform. The maximum computation time allowed for each run of RILS-ROLS is set to 2 h, while the maximal number of fitness function evaluations is set to 1 million. `SRBench` sets termination criteria to 8 h runtime and 1 million iterations, but some methods tested within `SRBench` also use more restrictive settting, as we do.

RILS-ROLS was run ten times for each problem instance, using a different setting of a random number generator for each run (the seeds are the same as for other comparison algorithms). As with other comparison algorithms tested within `SRBench`, the training/test data were randomly split in a 75%:25% ratio.

Datasets and `SRBench`

There are two groups of instances in the `SRBench` ground-truth problem set:

- Feynman instances are inspired by physics and formulas/models that describe various natural laws. There are 116 instances, where each one consists of 10^5 samples (see [3] for more details). Some exact models (equations) of Feynman instances are listed in Table 2.
- Strogatz instances are introduced in [51]. Each instance represents a 2-state system of first-order, ordinary differential equations. The aim of each problem is to predict the rate of change of the subsequent state. These equations describe the natural processes modeled by non-linear dynamics exhibiting chaos. The equations for some of the datasets that belong Strogatz are shown in Table 3. There are 14 Strogatz instances in total.

The above-mentioned benchmark sets may be biased since the models which describe physical laws usually impose various symmetries, periodicity, or internal separability on some variables, etc. In order to test RILS-ROLS in unbiased setting, we generated a set of random SR problem instances called Random. These instances vary in size (total number of expression nodes) and number of variables. In total, there are 235 random instances, or 5 randomly generated instances for each of the

Table 3 Some STROGATZ ODE instances

Output	Formula
v'	$-0.05 \cdot v^2 \sin(\theta)$
θ'	$v - cos(\theta)/v$
x'	$10 \cdot (y - \frac{1}{3}(x^3 - x))$
y'	$-\frac{y}{10}$
x'	$20 - x - \frac{x \cdot y}{1+0.5x^2}$
y'	$10 - \frac{x \cdot y}{1+0.5x^2}$
θ'	$\cot(\phi)\cos(\theta)$
ϕ'	$(\cos^2(\phi) + 0.1 \cdot \sin^2(\phi))\sin(\theta)$

Table 4 Some RANDOM instances

Instance name	Formula
random_04_02_0010000_00	$\ln(x_0 + x_1)$
random_07_02_0010000_00	$(x_0 + 10) \cdot (x_1 + 1)$
random_08_02_0010000_00	$(x_0 + \ln x_0) * (x_1 + 10)$
random_10_03_0010000_02	$\ln(\sqrt{x_0} \cdot x_1 / \cos x_2)$
random_12_04_0010000_04	$\sqrt{x_3 \cdot \exp x_1 + \cos(x_2 \cdot \sin x_0)}$
random_15_03_0010000_04	$x_0 \cdot x_1/2 + (\cos(x_2) - \pi^2)^2$

following 47 (size, number of variables) combinations $\{(3,1),\{4,5\} \times \{1,2\},\{6\} \times \{1,2,3\},\{7,8,9,10,11,12\} \times \{1,2,3,4\},\{13,14,15\} \times \{1,2,3,4,5\}\}$. Each of the RANDOM instances has 10,000 randomly generated samples. Some random instances are listed in Table 4, while all instances can be found in the RILS-ROLS GitHub repository. [2]

Parameter tuning

The parameter $penalty_{size}$ is essential for making a trade-off between the solution accuracy and its complexity. It was empirically set to 0.001 for the considered benchmarks. This value is also used as the default in the corresponding Python package (described in the "Appendix"). For new datasets, one can either keep the default settings or use a wrapper tuning algorithm around the RILS-ROLS regressor, e.g., grid-search w.r.t. training or validation accuracy.

It should be noted that several other fitness functions were considered, including structural risk minimization (SRM) loss [52] and Bayesian information criterion (BIC) [53], R^2, *RMSE*, Huber loss [54], MAE, MAPE, etc. These functions did not require a $penalty_{size}$ parameter. Although attractive because of their simplicity, these measures produced results worse than the proposed penalty-based approach. In the end, we

[2] https://github.com/kartelj/rils-rols.

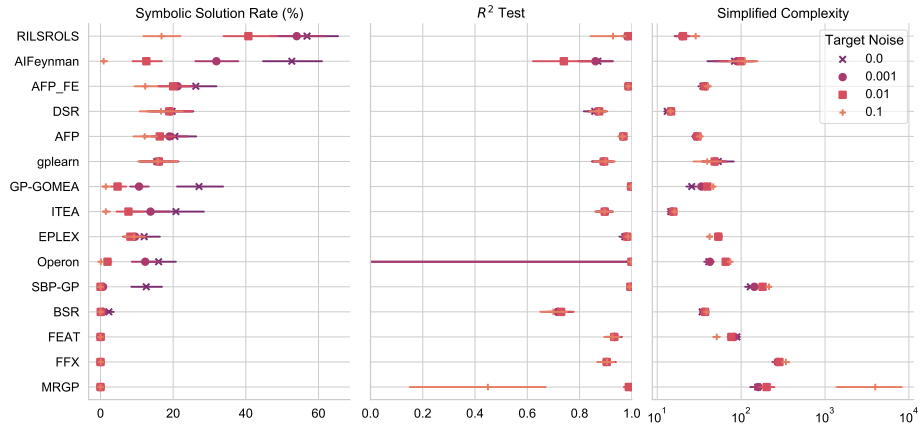


Fig. 2 Overall comparison w.r.t. (three) different metrics

included SRM and BIC measures into the RILS-ROLS Python package as alternatives to the proposed fitness function. Moreover, detailed results for BIC measure on Feynman and Strogatz instances are available in the GitHub repository.[3]

Comparison to other methods

In this section we evaluate our RILS-ROLS algorithm and compare it to 14 other competitors from the literature. The results for Feynman and Storgatz benchmark are given w.r.t. the following levels of noise: 0.0 (no-noisy data), 0.001 (low-level noise), 0.01 (medium-level noise) and 0.1 (high-level noise). More precisely, the white Gaussian noise is added to the target variable the same way as in [36].

The symbolic solution rate (or just solution rate) shows how often the method reaches the true formula (ground-truth). The second measure, solution accuracy (or R^2 test), considers the solution to be accurate if it produces an R^2 score greater than 0.999 on the test set. Note that no noise is added to the target variable of the test set. The overall results, concerning symbolic solution rate, R^2 test and simplified complexity (formula complexity after simplification) are shown in Fig. 2.

One can notice that the simplest (final) models are found by DSR and ITEA, regardless of the noise level. RILS-ROLS ranks third in terms of simplified complexity. Interestingly, AI-Feynman, one of the best approaches to closed-form equation regression problems, often yields larger models than RILS-ROLS. This could be the reason why RILS-ROLS performs better than AI-Feynman on the Strogatz benchmark, where the exact models are not large considering the size of their tree representations.

The numerical results are given in Tables 5 and 6. The first table contains the comparisons in terms of symbolic solution rate, while the second table contains the comparison in terms of solution accuracy ($R^2 > 0.999$). Each row of the tables contains the results of the respective algorithm (indicated by the name in the first column). In the three remaining blocks, the results of the respective algorithm are given for all

[3] https://github.com/kartelj/rils-rols/blob/master/paper_resources/srbench/tables

Table 5 Comparison w.r.t. symbolic solution rate (%)

Algorithm	All				Feynman				Strogatz			
	Noise				Noise				Noise			
	0	0.001	0.01	0.1	0	0.001	0.01	0.1	0	0.001	0.01	0.1
RILS-ROLS	**56.85**	**54**	**40.69**	**16.79**	53.88	**51.98**	**40.17**	**17.1**	**81.43**	**70.71**	**45**	**14.29**
AI-Feynman	52.65	31.89	12.61	0.86	**55.78**	33.08	13.03	0.7	27.14	22.14	9.29	2.14
GPGOMEA	27.12	10.62	4.69	1.46	26.83	11.03	5.09	1.64	29.46	7.14	1.43	0
AFP-FE	26.23	21.23	20	12.31	26.98	21.9	20.78	13.53	20	15.71	13.57	2.14
ITEA	20.77	13.77	7.69	1.46	22.41	14.57	7.84	1.47	7.14	7.14	6.43	1.43
AFP	20.48	19	16.31	12.15	21.12	19.66	16.9	13.1	15.18	13.57	11.43	4.29
DSR	19.71	19.15	18.92	16.62	19.72	19.14	18.97	16.81	19.64	19.29	18.57	15
Operon	16	12.31	1.92	0.08	16.55	13.19	2.07	0.09	11.43	5	0.71	0
gplearn	15.48	16.16	16.05	15.77	16.27	16.99	16.87	16.55	8.93	9.29	9.29	9.29
SBPGP	12.6	0.69	0	0	12.72	0.78	0	0	11.61	0	0	0
EPLEX	12.02	9.54	8.23	9.31	12.39	9.57	8.71	10.17	8.93	9.29	4.29	2.14
BSR	2.31	0.62	0.08	0	2.48	0.6	0.09	0	0.89	0.71	0	0
FEAT	0.1	0	0	0	0	0	0	0	0.89	0	0	0
FFX	0	0	0	0.08	0	0	0	0.09	0	0	0	0
MRGP	0	0	0	0	0	0	0	0	0	0	0	0

considered instances, the Feynman benchmark set, and the Strogatz benchmark set, respectively. Each block shows the results for all four noise levels separately in terms of (4) corresponding columns. The best results are marked in bold.

The following conclusions may be drawn from Table 5:

- As for the symbolic solution rate (in percentage) in the no-noise scenario, the best algorithm is RILS-ROLS, which achieves a symbolic solution rate of 56.85%. The second best approach is AI-Feynman, which achieves a symbolic solution rate of 52.65%. Far behind these two are the other approaches. For example, the best approach among the others is GPGOMEA, which achieves a solution rate of only 27.12%. As for the results for the benchmark set Feynman, the best approach is AI-Feynman, which achieves a symbolic solution rate of 55.78%. Slightly worse results are obtained by our RILS-ROLS , which succeeds in 53.88% of the cases. Among the remaining approaches, AFP-FE and GPGOMEA are the best, achieving a symbolic solution rate of 26.98% and 26.83%, respectively. As for the results for the Strogatz benchmark set, the most convincing approach is RILS-ROLS, which achieves a remarkable solution rate of 81.43%, compared to the second best approach, GPGOMEA, which yields a rate of only 29.46%. Note that AI-Feynman achieves an accuracy of 27.14% and ranks third in this benchmark set.
- Concerning symbolic solution rate at low noise (0.001) for all problem instances, RILS-ROLS performs quite well, achieving a solution rate of 54%, which is only slightly worse than that of the non-noisy input data. The second best approach is again AI-Feynman, whose solution rate is now 31.89%, which is a significant drop

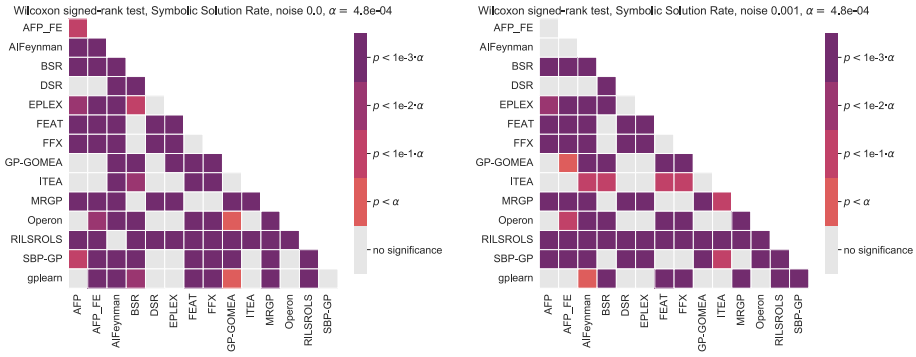
compared to the rate achieved for noisy input data. The best approach among the others is AFP-FE with 21.23% solution rate. The symbolic solution rate of the other approaches is less than 20%.

- As for the medium noise scenario (level 0.01), RILS-ROLS still performs well—it yields a solution rate of 40.69% on all problem instances. The second best approach is AFP-FE; it achieves a solution rate of only 20%. The best approach among the others is DSR, which achieves a symbolic solution rate of 18.92%. AI-Feynman's performance deteriorates significantly at this level of noise, so it comes in 6th place with a symbolic solution rate of only 12.61%.
- Concerning the scenario with the high-level noise (level of 0.1), RILS-ROLS still performs best considering all real problem cases, achieving a symbolic solution rate of 16.79%. The second best performing approach is DSR, which achieves a slightly worse symbolic solution rate of 16.62%, followed by Gplearn and AFP-FE, which achieve 15.77% and 12.31% symbolic solution rate, respectively. Note that DSR is shown to be robust to noise, with a consistent symbolic solution rate between 15% and 19% across all 8 combinations of benchmark datasets and noise levels. The same is true for Gplearn, but with lower symbolic solution rates.

The following conclusions concerning the obtained solution accuracy (i.e., $R^2 > 0.999$) may be drawn from Table 6:

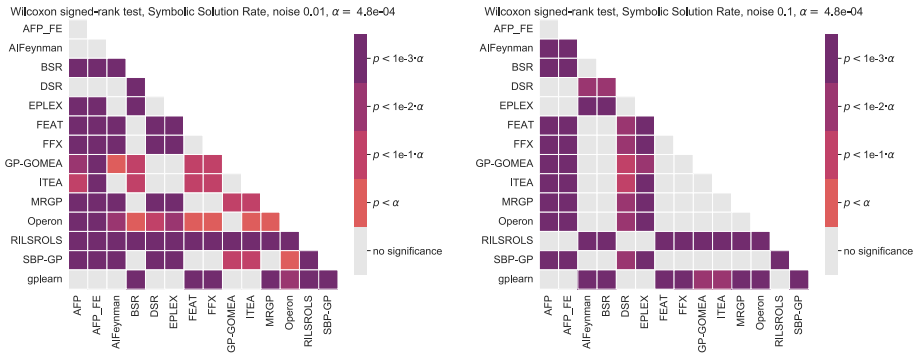
Table 6 Comparison w.r.t. ($R^2 > 0.999$) solution accuracy (%)

Algorithm	All				Feynman				Strogatz			
	Noise				Noise				Noise			
	0	0.001	0.01	0.1	0	0.001	0.01	0.1	0	0.001	0.01	0.1
MRGP	**92.69**	**91.54**	**88.46**	1.92	**93.1**	**92.24**	**91.81**	2.16	89.29	85.71	60.71	0
Operon	86.92	86.54	86.54	**73.46**	86.21	85.78	85.78	**82.33**	**92.86**	**92.86**	**92.86**	0
RILS-ROLS	80	81.15	80	21.92	78.45	79.74	78.45	19.4	**92.86**	**92.86**	**92.86**	**42.86**
SBPGP	74.23	74.23	75	53.85	73.71	75.43	75	60.34	78.57	64.29	75	0
AI-Feynman	73.83	73.64	67.86	10.16	78.51	77.39	71.43	10.53	35.71	42.86	39.29	7.14
GPGOMEA	71.54	70.38	73.46	67.69	71.55	70.26	73.71	71.98	71.43	71.43	71.43	32.14
AFP-FE	55.77	50.38	50.38	50	59.05	52.16	52.59	53.45	28.57	35.71	32.14	21.43
EPLEX	44.23	45.38	52.31	46.92	46.98	47.84	56.03	51.72	21.43	25	21.43	7.14
AFP	42.69	41.92	40.38	40.77	44.83	44.4	42.67	43.97	25	21.43	21.43	14.29
FEAT	40	43.08	40.77	13.46	39.66	42.24	40.52	13.36	42.86	50	42.86	14.29
gplearn	30	29.23	27.13	21.92	32.76	31.9	29.13	23.71	7.14	7.14	10.71	7.14
ITEA	26.92	26.92	26.92	25.77	27.59	27.59	27.59	26.72	21.43	21.43	21.43	17.86
DSR	23.85	24.62	25	25	25	25.86	26.29	26.29	14.29	14.29	14.29	14.29
BSR	11.92	10.77	11.92	6.92	10.78	10.34	12.07	7.76	21.43	14.29	10.71	0
FFX	0	0	2.69	17.69	0	0	2.59	19.83	0	0	3.57	0



(a) No-noise scenario (b) Low-level noise

Fig. 3 Statistical evaluation



(a) Medium-level noise (b) High-level noise

Fig. 4 Statistical evaluation

- In a no-noise scenario, our RILS-ROLS method achieves 80% solution accuracy (3rd among 15 competitors). Better methods are MRGP and Operon with 92.69% and 86.92%, respectively. On the other hand, the symbolic solution rates of these two approaches are rather low: 0% and 16%, respectively. These two algorithms are therefore more suitable for so-called black-box regression.
- In the presence of low-level noise, our method RILS-ROLS has a solution accuracy of 81.15% (third place again). As before, MRGP and Operon are the two best, with 91.54% and 86.54% solution accuracy, respectively.
- In the presence of medium intensity noise, the situation is almost the same. MRGP and Operon are the two best approaches, achieving 88.46% and 86.54% solution accuracy, respectively. The proposed RILS-ROLS achieves 80% solution accuracy, ranking third again. AI-Feynman, one of the leading approaches for solving closed-form equation regression, ranks 6th, with a solution accuracy of 67.86%.
- In the presence of the high-level noise, Operon is the best approach with 73.46% solution accuracy. GPGOMEA achieves 67.69% solution accuracy as the second best approach. The solution accuracy of MRGP, the leading approach for black-box regression at low and medium noise, deteriorates significantly for the high noise input data,

where its solution accuracy is only 1.92%. RILS-ROLS ranks 9th with a solution accuracy of 21.92%

- Note that RILS-ROLS and Operon perform equally well (and better than the other approaches) on the Strogatz benchmark set in terms of solution accuracy with up to moderate noise in the input data. With high noise on the Strogatz benchmark set, RILS-ROLS outperforms all approaches in terms of solution accuracy.
- We can conclude that RILS-ROLS achieves state-of-the-art performances regarding symbolic solution rate, see Fig. 2. It also shows high robustness to noise. In terms of solution accuracy, which is not so relevant for ground-truth problems, RILS-ROLS performs quite well even in the presence of low to medium noise.

Statistical evaluation

Figures 3, 4 give summary significance levels of pairwise tests of significance between the (15) competitors on ground-truth problems w.r.t. obtained symbolic solution rate. Statistical analysis is based on the Wilcoxon signed-rank test in all (105) cases. Note that a Bonferroni correction was applied. We emphasize that this methodology for valuating statistical significance of multiple estimators over many datasets is proposed by Demšar [55].

The following conclusions can be drawn from the statistical analyses applied:

- In the results for the no-noise scenario on all 130 ground truth problem instances (i.e., 1300 runs), RILS-ROLS and AI-Feynman achieve a significantly better solution rate than the other (13) approaches. The difference between these two approaches is not statistically significant.
- In the low noise scenario, RILS-ROLS claims its dominance as the obtained symbolic solution rate is statistically significantly better than that of all other approaches. Note that the second best approach according to the solution rate, AI-Feynman, is statistically on par with DSR, but performs statistically better than the other (12) approaches.
- In the results for the medium noise scenario, RILS-ROLS achieved a significantly better solution rate than all other approaches. Concerning other approaches, DSR, AFP-FE, AFP, gplearn, and AI-Feynman are the best, with no significant differences among them, while they are significantly better than the remaining (8) approaches.
- In the high noise scenario, RILS-ROLS, as mentioned earlier, provides the best solution rate. However, it is not statistically better than the following approaches: DSR and gplearn, AFP-FE, AFP, and EPLEX. The remaining (9) approaches are significantly outperformed by RILS-ROLS.

RILS-ROLS sensitivity to noise and solution sizes

In this section we study the sensitivity of RILS-ROLS w.r.t. size of instance problems under different (4) levels of noise and solution sizes. Since we do not compare RILS-ROLS to other competitors here, but just assess method sensitivity to noise and solution size, experimental setup is different. First, we run each instance only once—note that there are 5 randomly generated instances for each of 47 combinations (size,

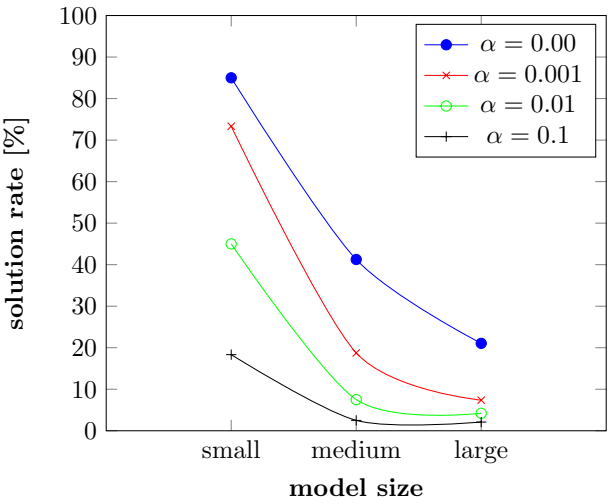


Fig. 5 Solution rates for varying levels of noise and solution sizes

variable count). Second, the exit criteria are different: (1) maximal running time is 300 s instead of 2 hours, and (2) maximal number of fitness evaluations is 100,000 instead of 1,000,000. As before, the test set takes 25% of total data. For the sake of simplicity, we split the instances from the RANDOM benchmark into three parts as follows:

- *Small-sized instances*: instances with solution size from 3 to 7.
- *Medium-sized instances*: those with solution size from 8 to 11.
- *Large-sized instances*: instances with solution size from 12 to 15.

The results are displayed in Fig. 5, grouped in accordance to the above-mentioned parts (x-axis). The obtained solution rate of RILS-ROLS is shown on the y-axis.

This leads to the following conclusions:

- As expected, solution rate is the highest for small-sized instances—it is sightly below 90% in a no-noise scenario. For noisy data, the solution rate naturally decreases as the level of noise increases. For example, on the small-sized problem instances with the low and medium level of noise, the solution rate achieved by RILS-ROLS is still reasonably high, at about 75% and 46%, respectively. In the case of the high noise, the solution rate for the small instances is only about 20%.
- For medium-sized instances when there is no noise, the solution rate is about 45%. It also decreases as the noise level increases—for the medium-noise level, solution rate of RILS-ROLS method is just about 8%. In case of the high-level noise, the rate drops to 2%.
- For large-sized instance problems, where no noise is added, the solution rate is about 25%, which means that the increase in size affects the overall algorithm performance, but not dramatically. In case of the high-level noise, the solution rate drops to 2%.
- We can conclude that increased formula size and noise level jointly contribute to deterioration of overall RILS-ROLS solution rate, which is expected and natural.

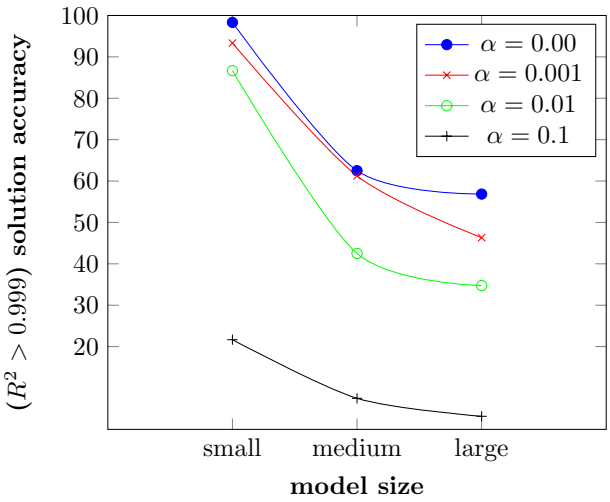


Fig. 6 Solution accuracy for varying levels of noise and formulae sizes

The diagram suggests that this deterioration behaves similarly across different levels of noise when instance size changes from small to medium almost linearly, and with almost exact slope. On the other hand, the relative deterioration of solution rate under no-noise level, when going from a medium to a large instance, is higher than under noisy data. A possible explanation for this is that the solution rate of medium instances has already significantly deteriorated in a noisy scenario. A further increase in size therefore does not have a significant impact.

Results for $R^2 > 0.999$ accuracy, therefore called solution accuracy, are presented in Fig. 6.

The following conclusions may be drawn from these results:

- For small-sized instances with no noise, RILS-ROLS almost delivers the perfect score. In the presence of low and medium noise, the solution accuracy is still holding high, at about 95% and 88%, respectively. In the case of high-level noise, the solution accuracy is about 24%, which is significantly (negatively) correlated with the increase in noise level.
- Concerning medium-sized instances in a no-noise scenario, the solution accuracy is around reasonable 65%. Having low noise does not affect the results considerably. However, under the higher noise, the impact is significant and very similar to the effect observable in small-sized instances—it reduces solution accuracy to about 45% in the presence of the medium-level noise and to just about 8% in the presence of the high noise.
- In the case of large-sized instances without noise, solution accuracy is slightly above 60%, which is very small relative deterioration in comparison to middle-sized instances. Not surprisingly, large instances under the medium and high noise are the most difficult to solve, where solution accuracies of about 40% and 4% are obtained, respectively.

Table 7 Results for big data instances

Instance	Sample size 0.0001				Sample size 0.001				Sample size 0.01			
	R^2	*RMSE*	Size	$t[s]$	R^2	*RMSE*	Size	$t[s]$	R^2	*RMSE*	Size	$t[s]$
`large1`	1	0.0019	31	2904.3	1	0.0019	31	7312.1	1	0.0019	31	57282.9
`large2`	0.9165	0.9918	88	5383	0.9759	0.5329	65	10664.7	0.9588	0.6968	62	79454.5
`large3`	0.8473	1.3414	75	5640.3	0.9588	0.6969	62	10513.7	0.9533	0.7415	72	78778.4
`large4`	0.6506	2.0288	62	6154.8	0.7628	1.6717	50	10115.2	0.7714	1.6412	54	66820.5
`large4'`	0.592	2.1924	50	5613.1	0.7638	1.668	50	10291.8	0.7721	1.6384	57	66682.8

Results on big data

The RILS-ROLS Ensemble method, introduced in "RILS-ROLS for parallel execution", was run on four datasets designed to be difficult to solve with a small data sample.

All generated datasets correspond to a function $f(x_1, \ldots, x_5) = \sum_{i=1}^{5} \sin(1/x_i)$, which was chosen because $\sin(1/x)$ becomes very unstable as x approaches zero; each dataset consists of 1 million records.

- Dataset `large1` has five input variables $x_1, \ldots, x_5$ and the target variable y is equal to the true value of $f(\cdot)$ for all input data.
- Dataset `large2`, on the other hand, has 10 additional *decoy* input variables $x_6, \ldots, x_{15}$, where the variables $x_6, \ldots, x_{10}$ are obtained by adding high-noise level to $x_1, \ldots, x_5$ while medium-level noise was applied to the variables $x_{11}, \ldots, x_{15}$.
- Dataset `large3` contains only *decoy* variables $x_6, \ldots, x_{15}$, with no real variables $x_1, \ldots, x_5$.
- Dataset `large4` contains only *decoy* variables with high-level noise $x_6, \ldots, x_{10}$. This dataset was also run in a mode with the medium-noise level applied to the target variable—this setting is denoted by `large4'`.

Experiments were performed with a parallelism level of 10 on an Intel i9-9900KF @3.6GHz processor with 8 physical and 16 logical cores (threads). Thus, the processor utilization (dedicated to this task) was approximately 10/16. Each instance was run three times, once for each value of the initial sample size $sampleShare_{init} \in \{0.0001, 0.001, 0.01\}$ to show that sampling was associated with a loss of solution quality. The exit criterion was 1 million iterations. As expected, the memory requirement (RAM) depended on $sampleShare_{init}$—for the value 0.01 it was about 20 GB (out of 64GB available).

Table 7 reports the findings, and its structure is as follows: the first column contains the name of the instance; the next three blocks contain results for different values of $sampleShare_{init}$, where each block reports on R^2, *RMSE* scores on the test set, model size, and total running time ($t[s]$) in seconds.

The following conclusions can be drawn from the results:

- The R^2 value reaches the perfect value in the case of the data set `large1` for all three sample sizes. The produced models are symbolically correct after rounding the coefficients. Thus, as expected, the `large1` is the simplest to solve. Regarding other

datasets, when the sample size is small (0.0001), the R^2 scores are much worse than R^2 scores for samples of size 0.001 or 0.01.

- The *RMSE* value is nearly perfect in the case of `large1`. For other data sets, similar to R^2, the values are better (smaller) when the sample size is larger, although there are some exceptions. For example, for the data set `large2`, the *RMSE* of 0.001 is better than when the sample size is 0.01 (0.5329 versus 0.6968). However, the model size for sample 0.001 is worse than the model size for sample 0.01. This is to be expected given that the fitness function balances accuracy and model complexity.
- As expected, the runtimes of RILS-ROLS increase with increasing sample size. This is also true for memory consumption.

Conclusions and future work

In this paper we dealt with solving the well-known symbolic regression (SR) problem. We proposed a metaheuristic approach called RILS-ROLS, built upon the iterated local search (ILS) scheme. The crucial element of this scheme is the ordinary least square method (OLS), used to efficiently determine the best-fitting linear coefficients within solution equations. Another key aspect is the utilization of a carefully designed fitness function which combines three important measures of the solution: *RMSE*, R^2, and solution size. Further, we designed an efficient local search that systematically explores the solution space around the candidate solution.

The RILS-ROLS method is compared to 14 other competing methods from the literature on two ground-truth benchmark sets from the literature: Feynman and Strogatz. Experimental evaluation confirmed the quality of our method—RILS-ROLS produced the best overall solution rate under varying levels of noise in data. Our algorithm was able to obtain 56.85%, 54%, 40.69%, and 16.79% solution rate under no-noise, low-level, medium-level, and high-level noise on all ground-truth problem instances, respectively. The second best approach, AI-Feynman, obtains fairly worse percentages: 52.65%, 31.89%, 12.61%, and 0.86%. In addition, statistical analysis of the obtained results confirmed that RILS-ROLS performed statistically significantly better than the other 14 approaches at low and medium noise.

In addition to well-known benchmarks from the literature, we introduced a new non-biased benchmark set of randomly generated instances. This benchmark was used to test RILS-ROLS sensitivity, by varying size of the models and noise levels. To test our algorithm for a Big Data scenario, we proposed RILS-ROLS for parallel execution and generated some very large datasets with 1 million records, designed to be difficult to solve with a small data sample.

In the future, one could think of constructing a hybrid of RILS-ROLS method with some other metaheuristics to further boost the quality of the obtained results. For example, replacing the ILS scheme of RILS-RILS with a more general Variable neighborhood search (VNS) scheme is a reasonable option. Also, RILS-ROLS could be used to solve practical problems from various domains. For example, it can be used in physics or chemistry to help set up reasonable hypotheses or obtain valuable insights into obtained experimental evaluations. Another improvement direction is incorporating top-level

constraints into the method. For example, if one knows that the modeled function needs to be monotonically decreasing or differentiable, or both, this knowledge can be used to alter the search—eliminate inadequate parts of the search space (hard constraint) or more subtly penalize this behavior through fitness function (soft constraint).

Appendix: RILS-ROLS Python package

RILS-ROLS algorithm is available for installation in the well-known Python package repository https://pypi.org, so it can be easily installed by typing commands:

```
pip install rils-rols
```

In this work, we used version 1.2 of RILS-ROLS. So if someone wants to reproduce the results shown, the safest thing to do is to specify the version during installation:

```
pip install rils-rols==1.2
```

RILS-ROLS PyPI project page is available at https://pypi.org/project/rils-rols. Here, one can find a minimal working example on how to use RILS-ROLS:

```
from rils_rols.rils_rols import RILSROLSRegressor
from rils_rols.rils_rols_ensemble import RILSROLSEnsembleRegressor
from math import sin, log

regressors = [
                RILSROLSRegressor(),
                RILSROLSEnsembleRegressor()
            ]

# toy dataset
X = [[3, 4], [1, 2], [-10, 20], [10, 10], [100, 100], [22, 23]]
y = [sin(x1)+2.3*log(x2) for x1, x2 in X]

# RILSROLSRegressor and RILSROLSEnsembleRegressor
# inherit BaseEstimator (sklearn), so we have well-known fit method
for regressor in regressors:
    regressor.fit(X, y)

    # this prints out the learned simplified model
    print("Final model is:\t"+str(regressor.model_simp))

    # this prints some additional information as well
    output_string = regressor.fit_report_string(X, y)
    print("Detailed output:\t"+output_string)

    # applies the model to a list of input vectors,
    #also well-known predict method
    X_test = [[4, 4], [3, 3]]
    y_test = regressor.predict(X_test)
    print(y_test)
```

Python sources, experimental results and all other RILS-ROLS resources can be found at the project GitHub page https://github.com/kartelj/rils-rols.

Acknowledgements
Not applicable.

Author contributions
Both AK and MD contributed to the conception of the work. AK worked on the implementation of method and the design of experiments. MD worked on analysis and discussion of the results, introduction and literature review. All authors read and approved the final manuscript.

Funding
A. Kartelj was supported by Grant 451-03-47/2023-01/200104 funded by Ministry of Science Technological Development and Innovations of the Republic of Serbia.

Availability of data and materials
All accompanying resources regarding this paper can be found in the GitHub repository https://github.com/kartelj/rils-rols.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare no competing interests.

Received: 14 December 2022 Accepted: 2 May 2023
Published online: 22 May 2023

References

1. Billard L, Diday E. Symbolic regression analysis. In: Jajuga K, Sokołowski A, Bock H-H, editors. Classification, clustering, and data analysis. Berlin, Heidelberg: Springer; 2002. p. 281–8.
2. Stimson JA, Carmines EG, Zeller RA. Interpreting polynomial regression. Sociol Methods Res. 1978;6(4):515–24.
3. Udrescu SM, Tegmark M. AI Feynman: a physics-inspired method for symbolic regression. Sci Adv. 2020;6(16):1–16.
4. Weng B, Song Z, Zhu R, Yan Q, Sun Q, Grice CG, Yan Y, Yin W-J. Simple descriptor derived from symbolic regression accelerating the discovery of new perovskite catalysts. Nat Commun. 2020;11(1):1–8.
5. Udrescu SM, Tegmark M. Symbolic pregression: discovering physical laws from distorted video. Phys Rev E. 2021;103(4): 043307.
6. Chen Y, Angulo MT, Liu Y-Y. Revealing complex ecological dynamics via symbolic regression. BioEssays. 2019;41(12):1900069.
7. Louis BB, Abriata LA. Reviewing challenges of predicting protein melting temperature change upon mutation through the full analysis of a highly detailed dataset with high-resolution structures. Mol Biotechnol. 2021;63(10):863–84.
8. Liu Z, Tegmark M. Machine learning conservation laws from trajectories. Phys Rev Lett. 2021;126(18): 180604.
9. Liang J, Zhu X. Phillips-inspired machine learning for band gap and exciton binding energy prediction. J Phys Chem Lett. 2019;10(18):5640–6.
10. Wang Y, Wagner N, Rondinelli JM. Symbolic regression in materials science. MRS Commun. 2019;9(3):793–805.
11. Wang C, Zhang Y, Wen C, Yang M, Lookman T, Su Y, Zhang T-Y. Symbolic regression in materials science via dimension-synchronous-computation. J Mater Sci Technol. 2022;122:77–83.
12. Burlacu B, Kommenda M, Kronberger G, Winkler S, Affenzeller M. Symbolic regression in materials science: Discovering interatomic potentials from data. 2022: arXiv preprint arXiv:2206.06422.
13. Kabliman E, Kolody AH, Kronsteiner J, Kommenda M, Kronberger G. Application of symbolic regression for constitutive modeling of plastic deformation. Appl Eng Sci. 2021;6: 100052.
14. Abdellaoui IA, Mehrkanoon S. Symbolic regression for scientific discovery: an application to wind speed forecasting. In: Proceedings of IEEE SSCI 2021—the 2021 IEEE Symposium Series on Computational Intelligence (SSCI), 2021:1–8. IEEE.
15. Koza JR. Genetic programming as a means for programming computers by natural selection. Stat Comput. 1994;4(2):87–112.
16. Schmidt MD, Lipson H. Age-fitness pareto optimization. In: Proceedings of GECCO 10—the 12th Genetic and Evolutionary Computation Conference, 2010;543–544.
17. Karaboga D, Ozturk C, Karaboga N, Gorkemli B. Artificial bee colony programming for symbolic regression. Inf Sci. 2012;209:1–15.
18. Kommenda M. Local optimization and complexity control for symbolic regression. PhD thesis, Johannes Kepler University 2018.
19. Virgolin M, Alderliesten T, Witteveen C, Bosman PA. Improving model-based genetic programming for symbolic regression of small expressions. Evol Comput. 2021;29(2):211–37.
20. de França FO, Aldeia GSI. Interaction-transformation evolutionary algorithm for symbolic regression. Evol Comput. 2021;29(3):367–90.
21. Kantor D, Von Zuben FJ, de Franca FO. Simulated annealing for symbolic regression. In: Proceedings of GECCO 21—the 23rd Genetic and Evolutionary Computation Conference, 2021;592–599.

22. Kommenda M, Burlacu B, Kronberger G, Affenzeller M. Parameter identification for symbolic regression using nonlinear least squares. Genet Program Evolvable Mach. 2020;21(3):471–501.
23. Burlacu B, Kronberger G, Kommenda M. Operon C++ an efficient genetic programming framework for symbolic regression. In: Proceedings of GECCO 20—the 22nd Genetic and Evolutionary Computation Conference Companion, 2020;1562–1570.
24. Virgolin M, Alderliesten T, Bosman PA. Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression. In: Proceedings of GECCO 19—the 21st Genetic and Evolutionary Computation Conference, 2019;1084–1092.
25. Schmidt M, Lipson H. Distilling free-form natural laws from experimental data. science 324(5923), 2009;81–85.
26. Schmidt M. Machine science: Automated modeling of deterministic and stochastic dynamical systems. PhD thesis, Cornell University 2011.
27. Jin Y, Fu W, Kang J, Guo J, Guo J. Bayesian symbolic regression. 2019; arXiv preprint arXiv:1910.08892.
28. Petersen BK, Larma ML, Mundhenk TN, Santiago CP, Kim SK, Kim JT. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. 2019; arXiv preprint arXiv:1912.04871.
29. Landajuela M, Petersen BK, Kim SK, Santiago CP, Glatt R, Mundhenk TN, Pettit JF, Faissol DM. Improving exploration in policy gradient search: Application to symbolic optimization. 2021; arXiv preprint arXiv:2107.09158.
30. Costa A, Dangovski R, Dugan O, Kim S, Goyal P, Soljačić M, Jacobson J. Fast neural models for symbolic regression at scale. 2020; arXiv preprint arXiv:2007.10784.
31. La Cava W, Helmuth T, Spector L, Moore JH. A probabilistic and multi-objective analysis of lexicase selection and ε -lexicase selection. Evol Comput. 2019;27(3):377–402.
32. La Cava W, Spector L, Danai K. Epsilon-lexicase selection for regression. In: Proceedings of GECO 16—the 18th Genetic and Evolutionary Computation Conference, 2016; pp. 741–748.
33. McConaghy T. FFX: Fast, scalable, deterministic symbolic regression technology. In: Proceedings of GEVO 11—the 13th Genetic and Evolutionary Computation Conference, 2011; pp. 235–260.
34. Arnaldo I, Krawiec K, O'Reilly U-M. Multiple regression genetic programming. In: Proceedings of GECCO 14—the 16th Genetic and Evolutionary Computation Conference, 2014; pp. 879–886.
35. La Cava W, Singh TR, Taggart J, Suri S, Moore JH. Learning concise representations for regression by evolving networks of trees. 2018; arXiv preprint arXiv:1807.00981.
36. La Cava W, Orzechowski P, Burlacu B, de França FO, Virgolin M, Jin Y, Kommenda M, Moore JH. Contemporary symbolic regression methods and their relative performance. 2021; arXiv preprint arXiv:2107.14351.
37. Olson RS, La Cava W, Orzechowski P, Urbanowicz RJ, Moore JH. Pmlb: a large benchmark suite for machine learning evaluation and comparison. BioData mining. 2017;10(1):1–13.
38. Lourenço HR, Martin OC, Stützle T. Iterated local search. In: Handbook of Metaheuristics, Springer, New York, 2003; pp. 320–353.
39. Lourenço HR, Martin OC, Stützle T. Iterated local search: framework and applications. New York: Springer; 2010. p. 129–68.
40. Leng L, Zhang T, Kleinman L, Zhu W. Ordinary least square regression, orthogonal regression, geometric mean regression and their applications in aerosol science. J Phys: Conf Ser. 2007;78(1): 012084. https://doi.org/10.1088/1742-6596/78/1/012084.
41. Baxter J. Local optima avoidance in depot location. J Oper Res Soc. 1981;32(9):815–9.
42. Baum E. Iterated descent: a better algorithm for local search in combinatorial optimization. Technical report 1998.
43. Martin O, Otto SW, Felten EW. Large-step markov chains for the traveling salesman problem. Complex Syst. 1991;5:299–326.
44. Martin OC, Otto SW. Combining simulated annealing with local search heuristics. Ann Oper Res. 1996;63(1):57–75.
45. Applegate D, Cook W, Rohe A. Chained lin-kernighan for large traveling salesman problems. INFORMS J Comput. 2003;15(1):82–92.
46. Golub GH, Van Loan CF. Matrix Computations. JHU press, 2013.
47. ...Meurer A, Smith CP, Paprocki M, Čertík O, Kirpichev SB, Rocklin M, Kumar A, Ivanov S, Moore JK, Singh S, Rathnayake T, Vig S, Granger BE, Muller RP, Bonazzi F, Gupta H, Vats S, Johansson F, Pedregosa F, Curry MJ, Terrel AR, Roučka V, Saboo A, Fernando I, Kulal S, Cimrman R, Scopatz A. Sympy symbolic computing in python. PeerJ Comput Sci. 2017;3:103. https://doi.org/10.7717/peerj-cs.103.
48. Glover F, Laguna M. Tabu Search. New York: Springer; 1998. p. 2093–229.
49. Székely LA, Wang H. On subtrees of trees. Adv Appl Math. 2005;34(1):138–55.
50. Stephens T. Genetic Programming in Python With a Scikit-Learn Inspired API: GPLEARN 2016.
51. La Cava W, Danai K, Spector L. Inference of compact nonlinear dynamic models by epigenetic local search. Eng Appl Artif Intell. 2016;55:292–306.
52. Shawe-Taylor J, Bartlett PL, Williamson RC, Anthony M. Structural risk minimization over data-dependent hierarchies. IEEE Trans Inf Theory. 1998;44(5):1926–40.
53. Bartlett DJ, Desmond H, Ferreira PG. Exhaustive symbolic regression. 2022; arXiv preprint arXiv:2211.11461.
54. Huber PJ. Robust estimation of a location parameter. Breakthroughs in statistics: Methodology and distribution, 1992;492–518.
55. Demšar J. Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res. 2006;7:1–30.

Publisher's Note