RESEARCH

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
Full list of author information is available at the end of the article

Abstract

In this paper, we solve the well-known symbolic regression problem. This problem has been intensively studied in the literature and has a wide range of applications. The symbolic regression models could provide insights towards establishing physical theory behind experimentally observed data. Practical applications include revealing complex ecological dynamics in ecology, discovering mutations effects on protein stability in biology, modeling analytic representations of the exciton binding energy in energy science, finding models for band gaps of NaCl-type compounds in material science, just to name a few.

We propose a new meta-heuristic-based approach, called RILS-ROLS to solve this problem. RILS-ROLS is based on the following elements: (i) iterated local search is the method backbone, mainly solving combinatorial and some continuous aspects of the problem; (ii) ordinary least square method is focused on the continuous aspect of the search space – it efficiently determines the best–fitting coefficients of linear combinations within solution equations; (iii) a novel fitness function combines three important model quality measures: R^2 score, RMSE score, and the size of the model (or model complexity).

The experiments are conducted on the two well-known ground-truth benchmark sets called Feynman and Strogatz. RILS-ROLS was compared to 14 other competitors from the literature with presence of different levels of the Gaussian white noise in the input data. Our method outperformed all 14 competitors with respect to the true symbolic model accuracy percentage under all level of noise. In addition to evaluation on known ground-truth datasets, a new randomly generated set of problem instances is introduced. The goal of Random dataset was to test the scalability of our method with respect to incremental equation sizes and different levels of noise. RILS-ROLS can be considered as a new state-of-the-art method for solving the problem of symbolic regression when ground-truth equations are known.

Keywords: Symbolic regression; Iterated local search; Ordinary least square

1 Introduction

The problem of symbolic regression (SR) [1] has attracted many researchers 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 always a linear combination of input variables. This is in general not good enough, since target variable might be dependent on some nonlinear function among input variables.

Unlike linear regression, SR allows the search over much larger space of possible mathematical formulas to find the best-fitting ones, i.e. those able to predict the target variable from the input variables. The basis of constructing explicit formula is in elementary operations like addition and multiplication, as well as polynomial, trigonometric, exponential, and others.

SR models tend to be interpretable, unlike artificial neural networks. Coefficients inside SR formulas might indicate the absolute or relative importance of certain input variables. Moreover, relations among variables might be described in an elegant way. For example, the appearance of an exponential function can be associated with some physical phenomenon such as the intensity of radiation or acceleration over time, see [3]. Additionally, the SR models often have high generalization power unlike some models with fixed functional forms such as polynomial regression.

Practical applications of SR in chemical and biological sciences are listed in [4]. In particular, this paper describes the discovery of a series of new oxide perovskite catalysts with improved activities. The application of SR to discovering physical laws from a distorted video is studied in [5]. Revealing complex ecological dynamics by SR is presented in [6]. Paper [7] shows how SR is used to model the effects of the mutations on protein stability, in the domain of fundamental and applied biology. One recent study [8] is concerned of auto-discovering conserved quantities using trajectory data from unknown dynamical systems. Paper [9] shows the application of SR to model analytic representations of the exciton binding energy. The use of SR in material science is described in [10, 11, 12, 13]. SR application to wind speed forecasting is given in [14].

There are many different ways to tackle the SR. Most of them are based on the machine learning techniques, genetic programming (GP) or some other meta-heuristics. Among the first GP methods to tackle SR is the one of Raidl [15] based on a hybrid variant of genetic programming. A differential evolution algorithm was proposed by Cerny et al. [16]. Age-fitness Pareto Optimization approach is proposed by Smidt and Lipson [17]. Application of the artificial bee colony programming to solve SR is proposed by Karaboga et al. [18]. Application of local-based heuristics for solving SR is reported by Commenda in his PhD thesis [19]. A GP-based approach, the gene-pool optimal mixing evolutionary algorithm (GOMEA) is studied by Virgolin et al. [20]. Another evolutionary algorithm, the interaction-transformation EA (ITEA) has been proposed by de Franca et al. [21]. Simulated annealing to solve SR is proposed by Kantor [22]. A variable neighbourhood programming approach to solve SR is proposed by Elleurich et al. [23]; this technique is initially proposed in [24]. Kommenda et al. [25] 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 [26]. The method that utilizes the Taylor polynomials to approximate the symbolic equation that fits the dataset, called Taylor genetic programming, is proposed in [27]. A GP approach that uses the idea of semantic back-propagation (Sbp-Gp) is proposed in [28]. Empirical analysis of variance between many GP-based methods for SR is discussed in [29]. It is also worth to mention the GP-based Eurequa commercial solver [30, 31] that uses age-fitness pareto optimization with co-evolved fitness estimation. This solver is nowadays accepted as the gold standard of symbolic regression.

Method based on Bayesian symbolic regression (BSR) is proposed in [32] – this method belongs to a family of Markov Chain Monte Carlo algorithms (MCMC). Deep Symbolic Regression (DSR), an RNN approach, is proposed in [33] – it utilizes the policy gradient search. This mechanism of search is further investigated in [34]. A fast neural network approach, called OccamNet, is proposed in [35]. A deep reinforcement learning approach enhanced with genetic programming is proposed in [36].

Powerful hybrid techniques to solve SR are also well studied; among them, we emphasize the Eplex solver from [37, 38] and AI Feynman algorithm from [3], a physics-inspired divide-and-conquer method combined with the neural network fitting. The latter is one of the most efficient methods for physically-inspired models. We also mention the Fast Function Extraction (FFX) algorithm developed by McConaghy [39], 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.

Algorithm	Paper/year	Short details
GP	[40] (1994)	Application of GP to SR
Hybrid-GP	[15] (1998)	GP-based method; solutions are locally optimized with OLS to find optimum coefficients for top-level terms
DFE	[16] (2008)	Differential evolution algorithm
APF-FE	[30, 31] (2009, 2011)	Age-fitness Pareto optimization approach using co-evolved fitness estimation
APF	[17] (2010)	Age-fitness Pareto optimization approach
FFX	[39] (2011)	The fast function extraction algorithm – non-evolutionary technique based on a machine learning technique called path-wise regularized learning
ABCP	[18] (2012)	Artificial bee colony programming approach
EPLEX	[38] (2016)	A parent selection method called ϵ-lexicase selection
MRGP	[41] (2014)	It decouples and linearly combines a program's subexpressions via multiple regression on the target variable
Local optimization NLS	[19] (2018)	Constants optimization in GP by nonlinear least squares
FEAT	[42] (2018)	Features are represented as networks of multi-type expression trees comprised of activation functions; differentiable features are trained via gradient descent
SBP-GP	[28] (2019)	The idea of semantic back-propagation utilized in GP
BSR	[32] (2019)	ML-based approach; Bayesian symbolic regression
DSR	[33] (2019)	Deep symbolic regression based on a RNN approach further utilizing the policy gradient search
Operon	[25] (2020)	Utilizing nonlinear least squares for parameter identification of SR models with LS
VNP	[23] (2020)	VNS based GP approach
OccamNet	[35] (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	[20] (2021)	A model-based EA framework called gene-pool optimal mixing evolutionary algorithm
ITEA	[21] (2021)	EA based approach called the interaction-transformation EA
SA	[22] (2021)	Simulated annealing approach
DRLA	[36] (2021)	A deep reinforcement learning approach enhanced with genetic programming
Taylor-GP	[27] (2022)	Taylor polynomials approximations

Table 1: SR methods overview.

The SR researches lack uniform, robust, and transparent benchmarking standards. Recently, La Cava et al. [43] proposed an open-source, reproducible benchmarking platform for SR, called SRBench. The authors extended PMLB [44], a repository of standardized regression problems, with 130 SR datasets for which exact (*ground-truth*) models are known; see more in the aforementioned paper. In their extensive experimental evaluation, 14 symbolic regression methods and 7 machine learning methods are compared on the set of 252 diverse regression problems. The remaining 122 SR datasets belong to a class of so-called *black-box* problems for which the *ground-truth* is unknown. One of the most interesting conclusions from La Cava et al. research is that algorithms specialize in either solving *ground-truth* problems, or *black-box* problems, but not both.

In this work we present a novel approach to solve *ground-truth*-based SR, which combines the popular iterated local search meta-heuristic (ILS) [45, 46] with the ordinary least square method (OLS) [47]. ILS mostly handles combinatorial (discrete)

aspects of search space, while OLS helps in the process of a 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, therefore we called the method RILS-ROLS (with letters R corresponding to regression and robust). Additionally, in order to navigate the search toward exact model (and not only accurate one) the algorithm is equipped with a carefully constructed fitness function that utilizes three important model characteristics: RMSE and R^2 scores which participate into models accuracy, and a weighted solution size that penalizies too complex models.

The summary of main contributions is as follows:

1 The proposed method outperforms 14 comparison methods on two *ground-truth* benchmark sets from literature – these methods and benchmarks are used in SRBench platform.
2 The method shows its high robustness, which is proved by its comparison to the other algorithms in the presence of different levels of Gaussian white noise in the input data.
3 The method is very efficient, taking in average less than 4 minutes to reach the exact solution, on the problem instances where it was possible.
4 A new set of unbiased instances is introduced – it consists of randomly generated formulae of various sizes and number of input variables. This set was employed to analyze the effect of the model size and the level of noise on the solving difficulty.

2 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 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 a mapping* $\tilde{f}(\mathbf{x}) = \tilde{g}(\mathbf{x}, \tilde{\theta}) \colon \mathbb{R}^d \mapsto \mathbb{R}$ *estimated by searching through the space of (mathematical) expressions* $\tilde{g}$ *and parameters* $\tilde{\theta}$ *where* ϵ *is observed white noise within the given input data.*

Koza [40] introduced the problem of SR as a specific application of genetic programming. GP can be used to optimize nonlinear structures such as computer programs. In particular, the programs are represented by syntax trees consisting of functions/operations over input features and constants. As an example of a function represented by a syntax tree, see Figure 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 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, some combination of them, etc. Interestingly, the SR search space nature 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 $/$.

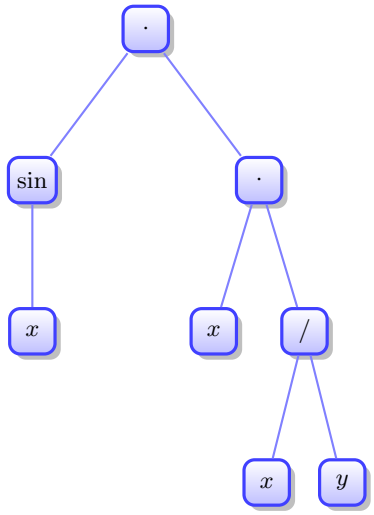


Figure 1: Syntax tree representation for the expression $\sin x \cdot (x \cdot x/y) = \frac{x^2 \sin x}{y}$

3 The proposed RILS-ROLS method

Our method relies on the following operations: $+$, $-$, $\cdot$, $/$, $\sqrt{x}$, x^2, sin, cos, log, exp, a^x, where a^x is only consequently used – it is never explicitly introduced in the search space. Beside this, the following set of constants enters the search space explicitly: -1, 0, 1, 2, π and 10. Before we give details on our method for solving SR, we will explain two building blocks of RILS-ROLS: 1) iterated local search (ILS) and 2) ordinary least squares (OLS).

3.1 Iterated local search

ILS [45] is an efficient meta-heuristics 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 local optimum, a *perturbation* is performed – this step is usually non-deterministic. This simple idea was proposed by Baxter [48] in early 1980s, and, since then, has been re-invented by many researches under different names. Some of the following names were used: iterated descent search [49], large-step Markov chains [50], chained local optimization [51], or, in some cases, combinations of previous [52]. 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.)

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 – it is partially randomized, yielding new solution s'. Next, the solution s' is potentially improved by a LS

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
```

procedure. Thirdly, the newly obtained solution s' possibly becomes new incumbent – this is decided upon the acceptance criterion. Sometimes, ILS incorporates a mechanism of tabu list, which prevents the search getting back into already visited solutions.

3.2 Ordinary least square method

Ordinary least square method (OLS) is a linear regression technique. It is based on applying the least-square method to minimize the square residual (error) sum between actual and predicted values (given by the model). More precisely, given previously introduced 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 $\tilde{y} = \mathbf{kx} + b$, that is coefficients (line slope) $\mathbf{k} = (k_1, \ldots, k_d)$ and b (intercept), so that $\sum_i^n (\tilde{y}_i - y_i)^2$ is minimized. This sum is also know as the sum of squared errors (SSE). There are many methods to minimize SSE. One of the analytical approaches is the calculus-oriented – it takes into account partial derivatives of SSE w.r.t. $k_j,\ j \in \{1, ..., d\}$:

$$\frac{\partial}{\partial k_j} \sum_i^n (\tilde{y}_i - y_i)^2 = \frac{\partial}{\partial k_j} \sum_{i=1}^n (\mathbf{kx_i} + b - y_i)^2 = \sum_{i=1}^n 2x_{ij}(\mathbf{kx_i} + b - y_i).$$

Similarly, by taking into account partial derivation w.r.t. b, we have:

$$\frac{\partial}{\partial b} \sum_{i=1}^n (\tilde{y}_i - y_i)^2 = \frac{\partial}{\partial b} \sum_{i=1}^n (\mathbf{kx_i} + b - y_i)^2 = \sum_{i=1}^n -2(y_i - \tilde{y}_i).$$

Finally, by setting each partial derivative to zero, we get the system of $d+1$ linear equations with $d+1$ unknowns ($\mathbf{k} = (k_1, ..., k_d)$ and b):

$$\begin{aligned}
&\sum_{i=1}^n -2x_{i1}(y_i - \tilde{y}_i) = 0 \\
&\vdots \\
&\sum_{i=1}^n -2x_{id}(y_i - \tilde{y}_i) = 0 \\
&\sum_{i=1}^n -2(y_i - \tilde{y}_i) = 0.
\end{aligned}$$

To solve this system one can use Cholesky factorization or LU decomposition for matrix inversion, and Winograd or Strassen algorithm for matrix multiplication,

depending on a solver of linear equations chosen as well as the sparsity of the matrix associated to the linear system [53]. For example, by using Cholesky decomposition, solving the system requires a matrix inversions and matrix multiplications. In that case the computation cost of $O(d^3)$ is required to find an inverse of the matrix associated to the linear system, which is the most consuming part of the algorithm. Thus, OLS using close-form matrix techniques performs in $O(d^3)$ time complexity. It is worth mentioning that when the number of input features becomes large, these closed-form matrix techniques are usually replaced by iterative ones, such as gradient descent (GD) [54]. When using GD the complexity becomes $O(n^2 d)$.

3.3 RILS-ROLS method

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

RILS-ROLS algorithms receives training dataset D_{tr} as input. In addition, it has two control parameters: $penalty_{size}$ and $tolerance_{error}$. The first one quantifies the importance of solution expression complexity in the overall solution quality measure (more on this in Section 3.3.1). The second parameter is related to the expected noise level in data – higher noise means that tolerance to errors should be higher. D_{tr} can have very large number of records, so the first step of RILS-ROLS is to choose a random sample $D'_{tr} \subseteq D_{tr}$ of size $sample_{size}$. Initial the sample size is chosen as $\max(0.01 \cdot n_{tr}, 100)$. The size of sample is later dynamically adjusted through the algorithm's iterations – when there are no solution improvements for some number of iterations, the sample size is doubled (lines 38-39 of Algorithm 2).

As previously stated, solution is usually represented by means of a tree. We use a simple solution initialization – tree root node is set to zero constant. We interchangeably use two solution variables: (*i*) s denotes 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 Section 3.3.1). Before entering the main loop, the best solution bs is set to the initial solution (line 7).

Main loop iterates as long as none of termination criteria is met: (*i*) maximal running time has been reached; (*ii*) maximal number of fitness calculations has been made; (*iii*) 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 – this significantly reduces the running times for the majority of tested instances. The sufficiency is controlled with parameter $tolerance_{error}$ – smaller value means that R^2 and $RMSE$ should be closer to 1 and 0, respectively.

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, perturbation step is local – meaning the closeness of starting solution s and any of perturbations is 1 (we call it 1-perturbation sometimes). The precise way of generating perturbation is described separately, in Section 3.3.3.

Candidate perturbations are being improved by performing OLS coefficient fitting (procedure `FitOLS`). This means that coefficients in any of linear combinations of current solution are being set by applying ordinary least square method, already

Algorithm 2 RILS-ROLS method.

Input: input training dataset D_{tr}
Control parameters: size penalty $penalty_{size}$, error tolerance $tolerance_{error}$
Output: best symbolic formula solution bs

procedure RILS-ROLS (D_{tr})
  $n_{tr} \leftarrow |D_{tr}|$
  $sample_{size} \leftarrow$ `InitialSampleSize`(n_{tr})
  $D'_{tr} \leftarrow$ `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)
    $s_{perturbations} \leftarrow$ `FitOLS`($s_{perturbations}$, D'_{tr})
    $s_{perturbations} \leftarrow$ `OrderByR2`($s_{perturbations}$)
    $improved \leftarrow false$
    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`($start_{candidates}$)
        $start_{candidates} \leftarrow$ `All1Perturbations`(s') // 2-permutations of *bs*
      end if
      $s \leftarrow$ `RandomPick`($start_{candidates} \setminus start_{tried}$)
      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 w.r.t. $tolerance_{error}$ **then**
      break // early exit
    end if
  end while
  $bs \leftarrow$ `Simplify`(bs)
  $bs \leftarrow$ `RoundModelCoefficients`(bs)
  return bs
end procedure

described in Section 3.2. After this step, perturbations are usually better suited to given sample data D'_{tr}. Further, these perturbations are sorted w.r.t. R^2 metric in the descending order (line 13).

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 19), after which the local search is performed (line 20). Solution simplification is done in a symbolical fashion by popular `SymPy` Python package [55]. Local search tries to find local optima expressions close to the given p – explained in detail in Section 3.3.4. 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, that 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 22-25), i.e. $perturbations_{tried}$ serves as a kind of tabu list, which is known from the Tablu search meta-heuristic, see [56].

If some of $s_{perturbations}$ around starting solution s yielded an improvement, bs becomes the starting solution s in the next iteration of the main loop (line 29). Otherwise, it makes no sense to set starting solution to bs, as nothing changed – the search is being *trapped* in a local optimum. Randomness is introduced in order to avoid this undesired 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 36). There is a minor chance that $start_{candidates} \setminus start_{tried} = \emptyset$. When that happens, the set of starting solution candidates is equal to perturbations of some randomly selected perturbation of bs (lines 33-34) – which effectively means that the perturbations with distance 2 from bs are used. There is a very small chance that these 2-perturbations will have an empty set difference with $start_{tried}$ set of starting solutions. Therefore, 3-perturbations are not considered in our algorithm.

Before returning the final symbolical model, RILS-ROLS performs the final symbolical simplification and rounding of model coefficients. The latter is sensitive to control parameter $tolerance_{error}$ – a smaller value means the information loss during rounding is smaller, i.e. a higher number of significant digits is preserved. Note that confidence in the symbolical model is reduced when expected noise in data is higher – the rule of thumb is: for higher expected noise $tolerance_{error}$ should be higher.

3.3.1 Fitness function

Symbolic regression objective is to determine the expression fitting available data. It is also allowed to obtain some equivalent expression, since there are multiple ways to express some symbolical 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 – only target values for some of the input data are known. Thus, various numerical metrics are being used in literature to

guide the symbolic regression search process. The most popular are the coefficient of determination, also known as R^2, and the mean squared error (MSE) or root mean squared error (RMSE). Also, the important aspect of the solution quality is the solution expression complexity, which may correspond to the model size of its tree representation. This follows the Occam's razor principle [35] that simpler solution is more likely to be a 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 Equation 1.

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

Since the presented fitness function needs to be minimized, the following conclusions may be drawn:

- 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.

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

3.3.2 Expression caching

In order to speed-up the fitness evaluation, we employ *expression caching*. This means that values attached to expression trees or subtrees are persisted 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 happens not that 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 some 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 consisting 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 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. Once this number is reached, the cache is cleared.

3.3.3 Perturbations

Perturbations allow the algorithm to escape from local optima. As previously described, perturbations are performed in 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 ← ∅
3:     s ← NormalizeConstants(s)
4:     s ← Simplify(s)
5:     s_subtrees ← SubTrees(s)
6:     for n ∈ s_subtrees do
7:         n_perturbations ← All1PerturbationsAroundNode(s, n)
8:         s_perturbations ← s_perturbations ∪ 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 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 during local search.)

After performing the normalization process, the expression is simplified – getting compact expression is more likely after normalization than before. The previous expression will take 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 expression does not have 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 around each subtree n is generated (lines 6-9 in Algorithm 3).

Algorithm 4 shows how perturbations around given subtree are generated.

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

Case 1. Observed node n is the whole tree s (see line 3 in Algorithm 4). Following on the previous exemplary expression tree, this means that 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) $(x+y^3)+x/\sqrt{y}$.

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
```

Case 2. Node n has 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 of possible perturbations could be $(x+y^3)\cdot x/\sqrt{\ln y}$ (application of logarithm to left subtree y) or $(x+y^3)\cdot x/\sqrt{x}$ (changing variable y to x), etc.

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

The algorithm allows 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 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 arbitrary variable. For example, $(x+y^3)\cdot x/\sqrt{y} \rightarrow (x+x/y^3)\cdot x/\sqrt{y}$.

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.

3.3.4 Local search

Perturbations are further improved by means of 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

```
1: procedure LocalSearch(p, D'_tr)
2:     bp, bp_fit ← p, Fitness(p, D'_tr)
3:     improved ← true
4:     while improved do
5:         improved ← false
6:         bp_candidates ← ∅
7:         bp_subtrees ←SubTrees(bp)
8:         for n ∈ bp_subtrees do
9:             n_candidates ← All1PerturbationsAroundNodeExtended(bp, n)
10:            for new ∈ n_candidates do
11:                new ← FitOLS(new, D'_tr)
12:                new_fit ← Fitness(new, D'_tr)
13:                if new_fit < bp_fit then
14:                    bp, bp_fit ← new, new_fit
15:                    improved ← true
16:                end if
17:            end for
18:        end for
19:    end while
20:    return bp
21: end procedure
```

For a given perturbation p, local search systematically explores 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 increased 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 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 binary operation applied to that node and some 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, where 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 number 10 is very important – it performs the general coefficient tuning, unlike OLS that considers only coefficients in linear combinations.

4 Experimental evaluation

Our RILS–ROLS algorithm is implemented in Python 3.9.0. 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. The RAM consumption was very small (up to few hundred megabytes) – beside memory space needed to load input dataset, the only considerable amount is reserved for expression cache.

The following 13 algorithms are compared to our approach: AI-Feynman, GOMEA, Afp-Fe, Itea, Afp, Dsr, Operon, the Gplearn from python package gplearn [57], Sbp-Gp, Eplex, Bsr, Feat, Ffx, Mrgp.

For the (13) competitors, the results are exported from SRBench `https://cavalab.org/srbench/results/`, reported in [43]. The maximum computation time allowed for each run of RILS-ROLS is set to 1 hour, while the maximal number of fitness function evaluations is set to 1 million. All other algorithms from SRBench used the time limit that was equal or bellow 8 hours runtime and the same fitness evaluation limit of 1 million (see [43]). We did not use the full time limit of 8 hours because we empirically concluded that there were almost no improvements after 1 hour RILS-ROLS runtime.

All non-deterministic algorithms (including RILS-ROLS) were run ten times per each problem instance, where each run used different setting of random number generator seed.

4.1 Datasets and SRBench

SRBench (`https://cavalab.org/srbench/`) is an open-source benchmarking project which merges a large set of diverse benchmark datasets, contemporary SR methods as well as ML methods around a shared model evaluation and an environment for analysis, see [43]. SRBench considers two types of symbolic regression problems: 1) *ground-truth* problems, for which the exact model is known, and 2) *black-box* problems, for which the exact model is not known. In our work, we consider only the former one, since RILS-ROLS was not designed to solve *black-box* problems. There are two groups of instances in SRBench *ground-truth* problem set:

- Feynman instances are inspired by physics and formulas/models that describes 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.

$x = \sqrt{x_1^2 + x_2^2 - 2x_1x_2\cos(\theta_1 - \theta_2)}$
$\theta_1 = \arcsin(n \sin\theta_2)$
$E = \frac{mc^2}{1-\frac{v^2}{c^2}}$
$\omega = \frac{1+\frac{v}{c}}{\sqrt{1-\frac{v^2}{c^2}}}\omega_0$

Table 2: Some Feynman instances.

- Strogatz instances are introduced in [58]. 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 natural processes modeled by non-linear dynamics exhibiting chaos. The equations for some of the datasets that belong Strogatz are shown in Table 3. In overall, there are 14 Strogatz instances.

The above-mentioned benchmark sets may be biased since the models which describe physical laws usually impose various symmetries, periodicity w.r.t. some variables, internal separability on some variables, etc. In order to test RILS-ROLS robustness and scalability in unbiased setting, we generated a set of random SR

Bacterial representation	$x' = 20 - x - \frac{x \cdot y}{1+0.5x^2}$
	$y' = 10 - \frac{x \cdot y}{1+0.5x^2}$
Shear Flow	$\theta' = \cot(\phi)\cos(\theta)$
	$\phi' = (\cos^2(\phi) + 0.1 \cdot \sin^2(\phi))\sin(\theta)$

Table 3: Some STROGATZ ODE instances.

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	$x_0 \cdot x_1/2 + (\cos(x_2) - \pi^2)^2$

Table 4: Some RANDOM instances.

problem instances called RANDOM. These instances have a varying 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 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 RANDOM instances has 10 thousands randomly generated samples.

Some RANDOM instances are shown in Table 4, while all instances can be found at RILS-ROLS GitHub repository: `https://github.com/kartelj/rils-rols`.

4.2 Parameter tuning

The RILS-ROLS algorithm employs two control parameters, $penalty_{size}$ and $tolerance_{error}$. The first parameter is empirically tuned for considered benchmarks to 0.001 – this is also used as a default value in the corresponding Python package (package is described in 6). The second parameter is negatively correlated with the expected level of noise in input data, i.e., for higher noise the $tolerance_{error}$ takes lower values and vice-versa.

4.3 Comparison with other methods on the ground–truth benchmark sets

In this section, we evaluate our RILS-ROLS algorithm and compare it to the 13 other competitors from the literature. The numerical results are reported in terms of the three Tables 5–7. The results for the two ground-truth benchmark sets, FEYNMAN and STORGATZ, are given w.r.t. the following level of noise: 0.0 (no-noisy data), 0.001 (low-level noise), and 0.01 (high-level noise), respectively. In details, the white Gaussian noise is added to the target (y-axis) values as a fraction of the signal RMS value, applied also in [43], i.e. for target noise level α we have

$$y_{noise} = y + \epsilon, \epsilon \sim \mathcal{N}\left(0, \alpha\sqrt{\frac{1}{n}\sum_{i=1}^{n} y_i^2}\right),$$

where n relates to the number of items in the input data.

Each table consists of three blocks. The performances of all (14) algorithms are displayed in respective rows. The first block gives the label of the algorithm whose performance is reported. The second block reports by two columns the relative

percentages of all (10) algorithm runs on all problem instances for which the exact model has been found by respective algorithm on Feynman and Strogatz benchmark set, respectively. The third column of this block provides the overall relative exact percentage of success of the algorithm over all runs on all problem instances from the both benchmark sets. The third block displays the results of the respective algorithm in terms of final R^2 score which indicate on accuracy of found solution. The solution is considered accurate if it produced an R^2 score larger that 0.999. Note that the termination of our algorithm may not always be triggered by reaching an appropriate R^2 score, but also due to the reached time limit. These kind of results are of a particular interest as they may point out on the presence of strong over-fitting of algorithms in contrast to the primary goal, that is finding the right model. This block consists of three columns that provide the relative percentage of algorithm runs on the instances where he algorithm is able to reach the desired accuracy (fulfilling the condition $R^2 > 0.999$ upon termination) on the both ground–truth benchmark sets separately, and the overall percentage ov all (10) runs on all considered problem instances.

In short, the exact results, thus the number of algorithm runs over all instances (in percentages) for which the output of respective algorithm is able to find the exact model, are displayed in the second block of each table. The numbers of runs (in percentages) for which an algorithm delivers a precise enough model (i.e. the outcome satisfying $R^2 > 0.999$) are displayed in the third block of each of the tables.

The following conclusions may be drawn from the numerical results:

- Concerning the exact percentages in case when no noise is utilized in the input data, the best performing algorithm is our RILS-ROLS, able to find the solution that match to the right model on 57.84% runs over all problems instances of the benchmark set Feynman; it was even more impressive in solving problem instances from benchmark set Strogatz, being successful on 83.57% runs. The second best performing approach is AI-Feynman, successful in 55.78% runs over all problem instances from Feynman set, and just 27.14% runs over the problem instances from benchmark set Strogatz, respectively. All other approaches are performing much worse, and none of them is able to delver the exact solution for more than 30% runs over the instances of any of the two considered ground-truth benchmark sets.
- Concerning the exact results in presence of the noise level of 0.001, our RILS-ROLS is still performing very well, having found an exact model for 42.08% runs considering all problem instances from the both ground-truth benchmark sets. The second best approach is AI-Feynman which is successful on 31.89% runs over all problem instances; as already noticed, this algorithm performs better when applied on the instances from Feynman benchmark set than it is the case of Strogatz benchmark set. Approach AFP-FE shows to be robust w.r.t. noise so its performances only slightly deteriorate in comparison to the no-noise scenario. This puts it on the third place with overall accuracy of 21.23%. Other approaches reach accuracy bellow 20%.
- Concerning the exact results when the presence of noise is large with the noise level of 0.01, our RILS-ROLS is still performing robustly, having found an exact model for 34.77% runes over all problem instances considering the

both ground-truth benchmark sets. The second best is AFP-FE reaching 20% correct solutions. AI-Feynman performances are degraded significantly under this level of noise, so it is ranked only 6th overall, reaching the accuracy of 12.61%.

- When comparing performances of the algorithms in terms of algorithm runs for which the obtained final model fulfills an accuracy of $R^2 > 0.999$, in case when no noise is integrated, our RILS-ROLS is successful with the rate of success of 83.38% (third place). However, in contrast with the exact model percentages comparisons from above, there are a few methods reaching better percentages of success than our algorithm: MRGP, and Operon, with 92.69%, and 86.92%, respectively. On the other hand, the exact model percentages of these two approaches are rather low (0% and 16%, respectively). These two algorithms are therefore more appropriate in scenarios when the underlying (ground-truth) model is unknown, so-called black-box regression.
- In the presence of the noise level of 0.001, our RILS-ROLS method has 78.62% of a success rate (ranked on the third place). As before, MRGP and Operon are showing good performances.
- When noise level is 0.01, the situation is almost the same, MRGP and Operon are the best and second best with 88.46% and 86.54% accuracy, respectively. GP-GOMEA now reaches third place with 73.46%, while the proposed RILS-ROLS is fifth with 62.92% R^2 accuracy.
- Following the above conclusions, our RILS-ROLS algorithm is more robust than the other competitors, which is confirmed by the produced results w.r.t. different levels of noise in the input data. Moreover, it can be considered as new state-of-the-art algorithm when one solves the problems of SR with known ground-truth models.

Table 5: Comparison without noise

	Exact model percentage			$R^2 > 0.999$ percentage		
Method	Feynman	Strogatz	Total	Feynman	Strogatz	Total
RILS-ROLS	**57.84**	**83.57**	**60.62**	82.59	90	83.38
AI-Feynman	55.78	27.14	52.65	78.51	35.71	73.83
GP-GOMEA	26.83	29.46	27.12	71.55	71.43	71.54
AFP-FE	26.98	20	26.23	59.05	28.57	55.77
ITEA	22.41	7.14	20.77	27.59	21.43	26.92
AFP	21.12	15.18	20.48	44.83	25	42.69
DSR	19.72	19.64	19.71	25	14.29	23.85
Operon	16.55	11.43	16	86.21	**92.86**	86.92
gplearn	16.27	8.93	15.48	32.76	7.14	30
SBP-GP	12.72	11.61	12.6	73.71	78.57	74.23
EPLEX	12.39	8.93	12.02	46.98	21.43	44.23
BSR	2.48	0.89	2.31	10.78	21.43	11.92
FEAT	0	0.89	0.1	39.66	42.86	40
FFX	0	0	0	0	0	0
MRGP	0	0	0	**93.1**	89.29	**92.69**

In Figure 2, average runtimes of the algorithms on those instance problems from the both ground-truth benchmark sets for which respective exact model has been found, are compared. The algorithms which obtained an exact percentage of at least 5% (considering all 1300 runs) are considered as relevant and therefore included here, while others are just omitted. The runtimes of each algorithm are averaged w.r.t different levels of noise, and presented by a bar plot. Algorithms are labeled on

Table 6: Comparison with noise level 0.001

	Exact model percentage			$R^2 > 0.999$ percentage		
Method	Feynman	Strogatz	Total	Feynman	Strogatz	Total
RILS-ROLS	**42.93**	**35**	**42.08**	80.26	65	78.62
AI-Feynman	33.08	22.14	31.89	77.39	42.86	73.64
AFP-FE	21.9	15.71	21.23	52.16	35.71	50.38
DSR	19.14	19.29	19.15	25.86	14.29	24.62
AFP	19.66	13.57	19	44.4	21.43	41.92
gplearn	16.99	9.29	16.16	31.9	7.14	29.23
ITEA	14.57	7.14	13.77	27.59	21.43	26.92
Operon	13.19	5	12.31	85.78	**92.86**	86.54
GP-GOMEA	11.03	7.14	10.62	70.26	71.43	70.38
EPLEX	9.57	9.29	9.54	47.84	25	45.38
SBP-GP	0.78	0	0.69	75.43	64.29	74.23
BSR	0.6	0.71	0.62	10.34	14.29	10.77
FFX	0	0	0	0	0	0
FEAT	0	0	0	42.24	50	43.08
MRGP	0	0	0	**92.24**	85.71	**91.54**

Table 7: Comparison with noise level 0.01

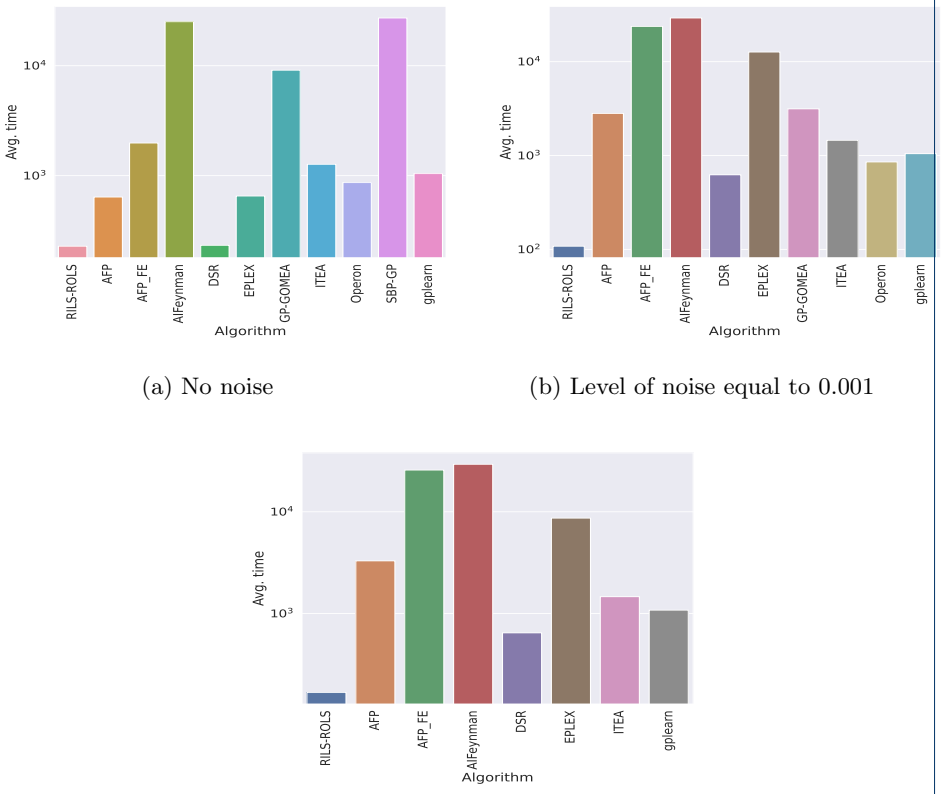
	Exact model percentage			$R^2 > 0.999$ percentage		
Method	Feynman	Strogatz	Total	Feynman	Strogatz	Total
RILS-ROLS	**36.29**	**22.14**	**34.77**	64.91	46.43	62.92
AFP-FE	20.78	13.57	20	52.59	32.14	50.38
DSR	18.97	18.57	18.92	26.29	14.29	25
AFP	16.9	11.43	16.31	42.67	21.43	40.38
gplearn	16.87	9.29	16.05	29.13	10.71	27.13
AI-Feynman	13.03	9.29	12.61	71.43	39.29	67.86
EPLEX	8.71	4.29	8.23	56.03	21.43	52.31
ITEA	7.84	6.43	7.69	27.59	21.43	26.92
GP-GOMEA	5.09	1.43	4.69	73.71	71.43	73.46
Operon	2.07	0.71	1.92	85.78	**92.86**	86.54
BSR	0.09	0	0.08	12.07	10.71	11.92
SBP-GP	0	0	0	75	75	75
FEAT	0	0	0	40.52	42.86	40.77
FFX	0	0	0	2.59	3.57	2.69
MRGP	0	0	0	**91.81**	60.71	**88.46**

x-axis, while respective average runtimes (oer all runs) are shown on y-axis. Note that y-axis is logarithmically scaled. The following conclusions are drawn from here.

- When considering the instance problems with no-noisy input data, the lowest average runtime is reported for RILS-ROLS algorithm ($228s$). A slightly worse runtime is obtained by Dsr algorithm ($232s$). However, note that the relative exact percentage produced by RILS–ROLS is far better from Dsr (60.62% vs. 19.71%). All remaining approaches deliver an order of magnitude larger average runtimes than that of the RILS–ROLS method in case of runs when the exact model has been reached.
- When considering the instance problems with the noise level of 0.001, the lowest average runtime to find the exact model is again delivered by RILS-ROLS algorithm ($108s$). The second best average runtime is produced by Dsr, but this time significantly larger ($622s$) from the former algorithm.
- When considering the instance problems with the noise level of 0.01, a similar conclusion may be drawn as that from above.

4.4 Statistical evaluation

In order to check the statistical significance of the results w.r.t. obtained exact models and their differences between the competitor approaches, we employed the



(a) No noise

(b) Level of noise equal to 0.001

(c) Level of noise equal to 0.01.

Figure 2: Average runtimes on the instance problems where the exact solution has been reached (considered only those algorithms whose the exact percentage is at least 5% of all runs).

statistical methodology comparing the results of RILS–ROLS to 13 other approaches over all algorithm runs on the problem instances from the both ground-truth benchmark sets, Feynman and Strogatz. Note that an indicator random variable is involved for each run of each algorithm. More precisely, the score that comes into the statistical observation is an indicator (0 or 1) indicating weather or not an algorithm was able to return the exact model at specific run. More precisely, we involved 1300 results into our statistical evaluation per each algorithm. Note that results of some of the 13 competitors are not known from SRBench `https://cavalab.org/srbench/results/` due to unknown reasons (most likely algorithm is terminated running out of memory or due to chosen algorithm configuration settings). For these runs, we simply indicate a 0 as their score.

Initially, Friedman's test was separately executed for all competitor approaches. In those cases in which the null hypothesis H_0 was rejected (H_0 states that there are no statistical differences between the obtained results of all competitors) pairwise comparisons are further performed by using the Nemenyi post-hoc test [59]. The outcome is represented by means of critical difference (CD) plots. In each CD plot, the (14) competitor approaches are placed on the horizontal axis according to their

average ranking. Thereafter, the CD score is computed for a significance level of 0.05. If the difference is small enough, meaning that no statistical difference is detected, a horizontal bar linking statistically equal approaches is drawn. Three (one for each level of noise) CD plots are shown in Figure 3 per each level of noise. The following conclusions may be drawn from there.

- Concerning the instance problems where no noise is included, RILS-ROLS method returns the best average ranking in terms of the number of obtained exact models over all runs. The second best average ranking is achieved by AI-Feynman algorithm. There is statistically significant difference between these two approaches w.r.t. number of found exact solutions. The third best approach is Apf-Fe, which is far behind RILS-ROLS w.r.t. delivered average ranking.
- Concerning the instance problems with the included level of noise of 0.001, RILS-ROLS method again obtains the best average ranking in terms of the found exact solutions. The second best ranking is obtained by AI-Feynman algorithm. There is again a statistically significant difference between the results obtained by these two approaches. The third best approach is Afp-Fe which performs statistically worse than AI-Feynman.
- Concerning the instance problems with the included noise level of 0.01, again, RILS-ROLS method produces the best average ranking according to positively evaluated runs, i.e. those runs with found exact models; the second best approach w.r.t. average ranking is Afp-Fe, followed by Dsr, and Afp. The latter three approaches perform statistically equal, but significantly worse than RILS-ROLS method. The fourth best approach according to delivered average ranking is AI-Feynman which is on pair with gplearn. Note that the average ranking of AI-Feynman method declines by increasing the level of noise in the input data. Thus, despite being efficient when comes to the no-noisy data, AI-Feynman shows to be very sensitive w.r.t. increase of the level of noise, which is not that dramatically conspicuous with our RILS-ROLS method.

4.5 Scalability of RILS-ROLS algorithm

In this section we study the scalability of our method in terms of the size of instance problems and the presence of different noise levels in the input data. First, for the shake of simplicity of presenting, we split the instances from Random benchmark sets into three parts as follows.

- *Small–sized instances*: any instance from Random benchmark whose corresponding exact model tree representation has from 3 to 7 nodes belongs to this sub-set.
- *Medium–sized instances*: any instance from Random benchmark whose corresponding exact model tree representation has from 8 to 11 nodes belongs to this sub-set.
- *Large–sized instances*: any instances from Random benchmark whose corresponding exact model tree representation has from 12 to 15 nodes belongs to this sub-set.

The results are displayed in Figure 4, grouped with accordance to the above (three) formed sub-groups (x-axis); for each of them relative exact percentage rate of RILS–ROLS algorithm is shown (y-axis). The following conclusions may be drawn from there.

- As expected, the exact percentage success rate is the highest for the small–sized instance problems; it is sightly bellow 90% (over all XXx runs) in case of no-noisy data. For the noisy data, the exact success rate is getting decreased as the level of noise increases. For example, on the small-sized problem instances with the noise level of 0.0001, and 0.01, the exact percentage achieved by RILS–ROLS algorithm is still reasonably high, at about 55% and 34 %, respectively.
- For the medium-sized instance problems where noise is not presented, the exact percentage rate of RILS-ROLS is slightly bellow 50%. It also decreases by increasing the level of noise in the target data; for the highest level of noise (of 0.01), the success rate of RILS-ROLS method is just at about 6%.
- For the large-sized instance problems where no noise is utilized, the exact percentage rate of RILS–ROLS algorithm is at about 25%, which means that the increase in the size of exact models affects the algorithm's performance, but not dramatically.
- From the above, we can conclude that the size of exact models (solutions) and the increase of level of noise both contribute to overall performance of our RILS–ROLS method. However, the exact percentage rate of the algorithm seems to decrease linearly in the size of (exact) model, regardless of the noise level values.
- Concerning the percentage of runs over all random instances for which RILS-ROLS terminates by producing an accurate model, that is a model with $R^2 > 0.999$, for the small–sized problem instances where no noise is included, RILS-ROLS delivers the perfect score almost. In the presence of a low noise level (equals to 0.001), this percentage is still holding high, at about 95%. In case when the high level of noise (equals to 0.01) is presented in the input data, these percentages stagnate, but still are at a respectable 66%. As pointed out already, for a half of these cases the respectable exact models were found by the algorithm.
- Concerning the medium-sized instance problems when the input data are free of the noise, the final outcome reached the high precision in 65% of all algorithm runs. The noisy the data are, these percentages, as expected, drop off; however, it is not so dramatically. For the level of noise of 0.01, the high–accuracy of the final solution is reported on ≈30% of all runs.
- Concerning the large-sized instance problems with no noise, the final solution reached the high precision in 60% of all runs. Not so surprisingly, the hardest case to obtain the desired high–accuracy (that is, ensuring $R^2 > 0.999$ for the outcome) is in the presence of a high level of noise (of 0.01), where the algorithm was successful on about 20% of all runs concerning the large–sized instance problems.

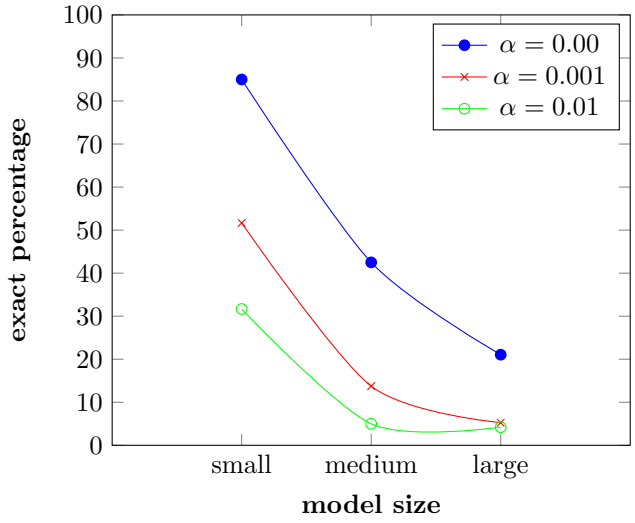


Figure 4: Exact solution percentages for varying levels of noise and formulae sizes

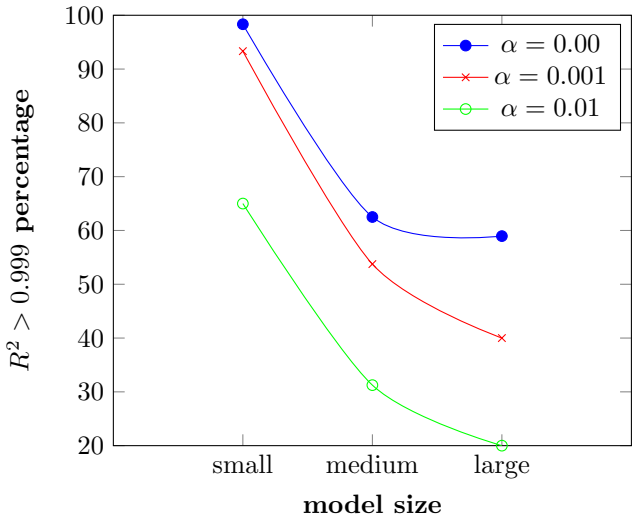


Figure 5: Percentages of solutions having $R^2 > 0.999$ for varying levels of noise and formulae sizes

In the rest of the section, we investigate scalability of our RILS–ROLS algorithm in terms of the number of variables and sizes of corresponding exact models of the problem instances varying levels of noise. There are three bar plots generated per each level of noise, shown by Figures 6–7. The instances are grouped according to the different variable count/size of respective exact models (x-axis) that correspond to them. For each group, the exact percentages obtained by RILS–ROLS algorithm group are shown on y-axis.

The following conclusions can be drawn from Figure 6.

- For the no-noisy data, the exact percentage rate of RILS–ROLS algorithm slightly decrease as the expected number of variables increases in the exact models.
- The exact percentages drop off with the presence of a larger noise level. As expected, the higher the noise level, the smaller the exact percentages are.
- Interestingly, in the presence of noise in the data, the highest exact percentages are not obtained for the problem instances which have exact models consisted

of a single variable, but two or three. We argue it by the fact that the early phase of the algorithm will more likely trying to include additional variables in current solution at the cost of increasing the model accuracy to ensure improving the fitness value over iterations. For the no–noisy data, OLS method will more likely find a small-sized high-accuracy model (in our experiments that happened on $\approx 50\%$ runs) that fits to the (no-noisy) input data than the case of noisy data. When the data is noisy, the decision of adding more variables at the beginning is likely to be forced than just dealing with short-size solutions.

The following conclusions may be drawn from Figure 7.

- The exact percentages are decreased with increase in the size of exact models, in case of any level of noise utilized on the target values.
- The highest exact percentages, which are larger than 70%, are noticed in case of the small-sized problem instances (the size of models ranges from 3 to 7). However, for the same sizes, for noisy data, these percentages are significantly lower; for example, in case of the instances whose exact models are of size 7 when utilizing the level of noise of 0.001 and 0.01, the exact percentages obtained are 20% and 10%, respectively.
- For the problem instances with the largest size of (exact) models in case of the no noisy data, exact percentages are never lower than 20%. The opposite holds for noisy data.

When one concerns of the runtime analysis of RILS–ROLS algorithm on the RANDOM benchmark set w.r.t number of variables, the results are displayed in Figure 8. The instances are grouped w.r.t. number of variables involved into corresponding exact models (x-axis). The following conclusions may be drawn from there.

- As one could expect, average runtime is getting increased with the increase of the number of variables in exact models.
- Note that the average runtime in case of the instances whose exact models have the largest number of variables (5) is just about $200s$.
- The two above-mentioned points indicate efficacy of our method especially in case of the instances with small-to-medium sized exact models in the presence of small–to–middle noise levels included in the input data. In these cases, the algorithm is able to deliver a reasonable exact percentage rate, which are always larger than 50%. Note that models which have more compact representations (thus, small in their size) are desirable in many areas of natural sciences as they are easy interpret-able than models represented by a more complex mathematical formula.

5 Conclusions and future work

In this paper, we dealt with solving the well–known Symbolic regression (SR) problem. SR problem is a generalization of more interdisciplinary known problems such as linear and logistic regression. We proposed a meta-heuristic approach, called RILS-ROLS, which is built upon the Iterated local search (ILS) scheme. The first crucial concept within this scheme is utilizing the ordinary least square method to efficiently determine best-fitting internal coefficients to the candidate solution.

Another key aspect is the utilization of a carefully constructed fitness function in the search, which appeared as a combination of three important scores: RMSE, R^2 scores and the size of the model (solution). Last but not least, a carefully constructed local search is applied systematically exploring solutions obtained as 1–perturbations of the considered solution, represented as trees. 1–perturbations of a solution are based on applying the six per-node change operations on the solution's tree representation.

RILS-ROLS method is compared to the 14 other competitor algorithms from the literature on the two ground–truth benchmark sets from the literature, namely Feynman and Strogatz. Additionally, we introduced a non-biased benchmark set of randomly generated instances, labelled by Random. The experimental evaluation confirmed the efficiency of our method which produced the best average ranking results in terms of exact percentage when compared to all other competitors on the two ground-truth benchmarks in the presence of all three different levels of noise in the input data. In all cases, new best exact percentages are obtained by our RILS-ROLS method. More in details, our algorithm was able to obtain the right model on 60.62%, 42.08%, and 34.77% runs over all considered problem instances in the presence of different noise levels: 0.0, 0.001, and 0.01, respectively. For the shake of comparisons, the second best approach AI-Feynman, when the noise level of 0.0 and 0.001 is included, obtains the exact percentages of 52.65%, and 31.89%, respectively. In case of the highest noise level, the second best approach was Afp-Fe, obtaining an exact percentage rate of 20%. Additionally, the scalability and robustness of RILS-ROLS method are checked and confirmed on the unbiased Random benchmark set in terms of a few different criteria such as the size of the model, the number of variables included in exact models, and obtained average runtimes.

In future work, one could think of constructing a hybrid of RILS-ROLS method with some other meta-heuristics to further boost the quality of the obtained results. For example, replacing ILS scheme of RILS-RILS with a more general Variable neighbourhood search (VNS) scheme is a reasonable option to give it a try. VNS utilizes a more stronger and systematic system of solutions diversification and thus escaping from local optima could be much easier and the convergence much faster. Additionally, as proof of the efficiency and usefulness of our algorithm, this tool could be applied to solving problems from various problem domains out of the area of computer science, for example from physics or chemistry, to help for setting up reasonable hypotheses or to obtain valuable insights on obtained experimental evaluations.

Appendix

6 Overview of RILS-ROLS python package

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

```
pip install rils-rols
```

RILS-ROLS project page is 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 math import sin, log

regr = RILSROLSRegressor()

# 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 inherits BaseEstimator (sklearn)
regr.fit(X, y)

# this prints out the learned model
print("Final model is "+str(regr.model))

# applies the model to a list of input vectors
X_test = [[4, 4], [3, 3]]
y_test = regr.predict(X_test)
print(y_test)
```

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

Acknowledgements
Text for this section...

Funding
Text for this section...

Abbreviations
Text for this section...

Availability of data and materials
Text for this section...

Ethics approval and consent to participate
Text for this section...

Competing interests
The authors declare that they have no competing interests.

Consent for publication
Text for this section...

Authors' contributions
Text for this section ...

Authors' information
Text for this section...

Author details
[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.

References

1. Billard, L., Diday, E.: Symbolic regression analysis. In: Classification, Clustering, and Data Analysis, pp. 281–288. Springer, ??? (2002)
2. Stimson, J.A., Carmines, E.G., Zeller, R.A.: Interpreting polynomial regression. Sociological Methods & Research **6**(4), 515–524 (1978)
3. Udrescu, S.-M., Tegmark, M.: Ai feynman: A physics-inspired method for symbolic regression. Science Advances **6**(16), 2631 (2020)
4. Weng, B., Song, Z., Zhu, R., Yan, Q., Sun, Q., Grice, C.G., Yan, Y., Yin, W.-J.: Simple descriptor derived from symbolic regression accelerating the discovery of new perovskite catalysts. Nature communications **11**(1), 1–8 (2020)
5. Udrescu, S.-M., Tegmark, M.: Symbolic pregression: Discovering physical laws from distorted video. Physical Review E **103**(4), 043307 (2021)
6. Chen, Y., Angulo, M.T., Liu, Y.-Y.: Revealing complex ecological dynamics via symbolic regression. BioEssays **41**(12), 1900069 (2019)
7. Louis, B.B., Abriata, L.A.: Reviewing challenges of predicting protein melting temperature change upon mutation through the full analysis of a highly detailed dataset with high-resolution structures. Molecular Biotechnology **63**(10), 863–884 (2021)

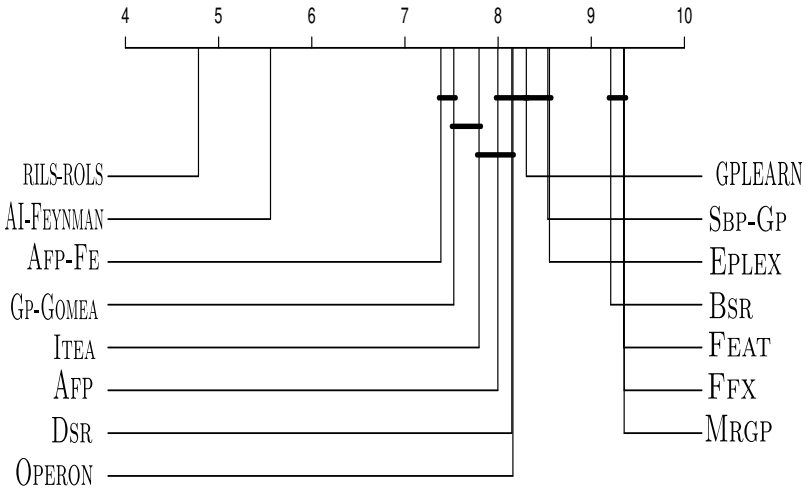
8. Liu, Z., Tegmark, M.: Machine learning conservation laws from trajectories. Physical Review Letters **126**(18), 180604 (2021)
9. Liang, J., Zhu, X.: Phillips-inspired machine learning for band gap and exciton binding energy prediction. The journal of physical chemistry letters **10**(18), 5640–5646 (2019)
10. Wang, Y., Wagner, N., Rondinelli, J.M.: Symbolic regression in materials science. MRS Communications **9**(3), 793–805 (2019)
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. Journal of Materials Science & Technology **122**, 77–83 (2022)
12. Burlacu, B., Kommenda, M., Kronberger, G., Winkler, S., Affenzeller, M.: Symbolic regression in materials science: Discovering interatomic potentials from data. arXiv preprint arXiv:2206.06422 (2022)
13. Kabliman, E., Kolody, A.H., Kronsteiner, J., Kommenda, M., Kronberger, G.: Application of symbolic regression for constitutive modeling of plastic deformation. Applications in Engineering Science **6**, 100052 (2021)
14. Abdellaoui, I.A., Mehrkanoon, S.: Symbolic regression for scientific discovery: an application to wind speed forecasting. In: 2021 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 01–08 (2021). IEEE
15. Raidl, G.R.: A hybrid gp approach for numerically robust symbolic regression. Genetic Programming, 323–328 (1998)
16. Cerny, B.M., Nelson, P.C., Zhou, C.: Using differential evolution for symbolic regression and numerical constant creation. In: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, pp. 1195–1202 (2008)
17. Schmidt, M.D., Lipson, H.: Age-fitness pareto optimization. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, pp. 543–544 (2010)
18. Karaboga, D., Ozturk, C., Karaboga, N., Gorkemli, B.: Artificial bee colony programming for symbolic regression. Information Sciences **209**, 1–15 (2012)
19. Kommenda, M., Affenzeller, M.: Local optimization and complexity control for symbolic regression. Ph. D. dissertation, Ph. D. thesis (2018)
20. Virgolin, M., Alderliesten, T., Witteveen, C., Bosman, P.A.: Improving model-based genetic programming for symbolic regression of small expressions. Evolutionary computation **29**(2), 211–237 (2021)
21. de França, F.O., Aldeia, G.S.I.: Interaction–transformation evolutionary algorithm for symbolic regression. Evolutionary computation **29**(3), 367–390 (2021)
22. Kantor, D., Von Zuben, F.J., de Franca, F.O.: Simulated annealing for symbolic regression. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 592–599 (2021)
23. Elleuch, S., Jarboui, B., Mladenovic, N., Pei, J.: Variable neighborhood programming for symbolic regression. Optimization Letters, 1–20 (2020)
24. Elleuch, S., Mladenovic, N., Jarboui, B.: Variable Neighborhood Programming: a New Automatic Programming Method in Artificial Intelligence. GERAD HEC Montréal, ??? (2016)
25. Kommenda, M., Burlacu, B., Kronberger, G., Affenzeller, M.: Parameter identification for symbolic regression using nonlinear least squares. Genetic Programming and Evolvable Machines **21**(3), 471–501 (2020)
26. Burlacu, B., Kronberger, G., Kommenda, M.: Operon c++ an efficient genetic programming framework for symbolic regression. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, pp. 1562–1570 (2020)
27. He, B., Lu, Q., Yang, Q., Luo, J., Wang, Z.: Taylor genetic programming for symbolic regression. arXiv preprint arXiv:2205.09751 (2022)
28. Virgolin, M., Alderliesten, T., Bosman, P.A.: Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1084–1092 (2019)
29. Kammerer, L., Kronberger, G., Winkler, S.: Empirical analysis of variance for genetic programming based symbolic regression. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 251–252 (2021)
30. Schmidt, M., Lipson, H.: Distilling free-form natural laws from experimental data. science **324**(5923), 81–85 (2009)
31. Schmidt, M.: Machine science: Automated modeling of deterministic and stochastic dynamical systems (2011)
32. Jin, Y., Fu, W., Kang, J., Guo, J., Guo, J.: Bayesian symbolic regression. arXiv preprint arXiv:1910.08892 (2019)
33. Petersen, B.K., Larma, M.L., Mundhenk, T.N., Santiago, C.P., Kim, S.K., Kim, J.T.: Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. arXiv preprint arXiv:1912.04871 (2019)
34. Landajuela, M., Petersen, B.K., Kim, S.K., Santiago, C.P., Glatt, R., Mundhenk, T.N., Pettit, J.F., Faissol, D.M.: Improving exploration in policy gradient search: Application to symbolic optimization. arXiv preprint arXiv:2107.09158 (2021)
35. Costa, A., Dangovski, R., Dugan, O., Kim, S., Goyal, P., Soljačić, M., Jacobson, J.: Fast neural models for symbolic regression at scale. arXiv preprint arXiv:2007.10784 (2020)
36. Mundhenk, T., Landajuela, M., Glatt, R., Santiago, C.P., Petersen, B.K., *et al.*: Symbolic regression via deep reinforcement learning enhanced genetic programming seeding. Advances in Neural Information Processing Systems **34**, 24912–24923 (2021)
37. La Cava, W., Helmuth, T., Spector, L., Moore, J.H.: A probabilistic and multi-objective analysis of lexicase selection and ε-lexicase selection. Evolutionary Computation **27**(3), 377–402 (2019)
38. La Cava, W., Spector, L., Danai, K.: Epsilon-lexicase selection for regression. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016, pp. 741–748 (2016)
39. McConaghy, T.: Ffx: Fast, scalable, deterministic symbolic regression technology. In: Genetic Programming Theory and Practice IX, pp. 235–260. Springer, ??? (2011)
40. Koza, J.R.: Genetic programming as a means for programming computers by natural selection. Statistics and computing **4**(2), 87–112 (1994)

41. Arnaldo, I., Krawiec, K., O'Reilly, U.-M.: Multiple regression genetic programming. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, pp. 879–886 (2014)
42. La Cava, W., Singh, T.R., Taggart, J., Suri, S., Moore, J.H.: Learning concise representations for regression by evolving networks of trees. arXiv preprint arXiv:1807.00981 (2018)
43. La Cava, W., Orzechowski, P., Burlacu, B., de França, F.O., Virgolin, M., Jin, Y., Kommenda, M., Moore, J.H.: Contemporary symbolic regression methods and their relative performance. arXiv preprint arXiv:2107.14351 (2021)
44. Olson, R.S., La Cava, W., Orzechowski, P., Urbanowicz, R.J., Moore, J.H.: Pmlb: a large benchmark suite for machine learning evaluation and comparison. BioData mining **10**(1), 1–13 (2017)
45. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search. In: Handbook of Metaheuristics, pp. 320–353. Springer, ??? (2003)
46. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search: Framework and applications. In: Handbook of Metaheuristics, pp. 129–168. Springer, ??? (2019)
47. Leng, L., Zhang, T., Kleinman, L., Zhu, W.: Ordinary least square regression, orthogonal regression, geometric mean regression and their applications in aerosol science. In: Journal of Physics: Conference Series, vol. 78, p. 012084 (2007). IOP Publishing
48. Baxter, J.: Local optima avoidance in depot location. Journal of the Operational Research Society **32**(9), 815–819 (1981)
49. Baum, E.: Iterated descent: a better algorithm for local search in combinatorial optimization. Touretzky D. Proc. Neural Information Processing Systems (1998)
50. Martin, O., Otto, S.W., Felten, E.W.: Large-step Markov Chains for the Traveling Salesman Problem. Citeseer, ??? (1991)
51. Martin, O.C., Otto, S.W.: Combining simulated annealing with local search heuristics. Annals of operations research **63**(1), 57–75 (1996)
52. Applegate, D., Cook, W., Rohe, A.: Chained lin-kernighan for large traveling salesman problems. INFORMS Journal on Computing **15**(1), 82–92 (2003)
53. Krishnamoorthy, A., Menon, D.: Matrix inversion using cholesky decomposition. In: 2013 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA), pp. 70–72 (2013). IEEE
54. Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M.W., Pfau, D., Schaul, T., Shillingford, B., De Freitas, N.: Learning to learn by gradient descent by gradient descent. Advances in neural information processing systems **29** (2016)
55. Meurer, A., Smith, C.P., Paprocki, M., Čertík, O., Kirpichev, S.B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J.K., Singh, S., Rathnayake, T., Vig, S., Granger, B.E., Muller, R.P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., Curry, M.J., Terrel, A.R., Roučka, v., Saboo, A., Fernando, I., Kulal, S., Cimrman, R., Scopatz, A.: Sympy: symbolic computing in python. PeerJ Computer Science **3**, 103 (2017). doi:10.7717/peerj-cs.103
56. Glover, F., Laguna, M.: Tabu search. In: Handbook of Combinatorial Optimization, pp. 2093–2229. Springer, ??? (1998)
57. Stephens, T.: Genetic Programming in Python With a Scikit-Learn Inspired API: Gplearn (2016)
58. La Cava, W., Danai, K., Spector, L.: Inference of compact nonlinear dynamic models by epigenetic local search. Engineering Applications of Artificial Intelligence **55**, 292–306 (2016)
59. Pohlert, T.: The pairwise multiple comparison of mean ranks package (pmcmr). R package **27**(2019), 9 (2014)

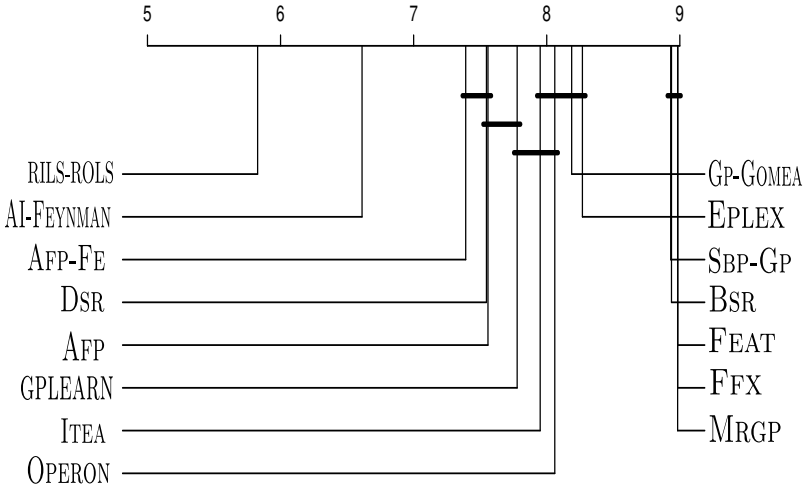
Additional Files

Additional file 1 — Sample additional file title

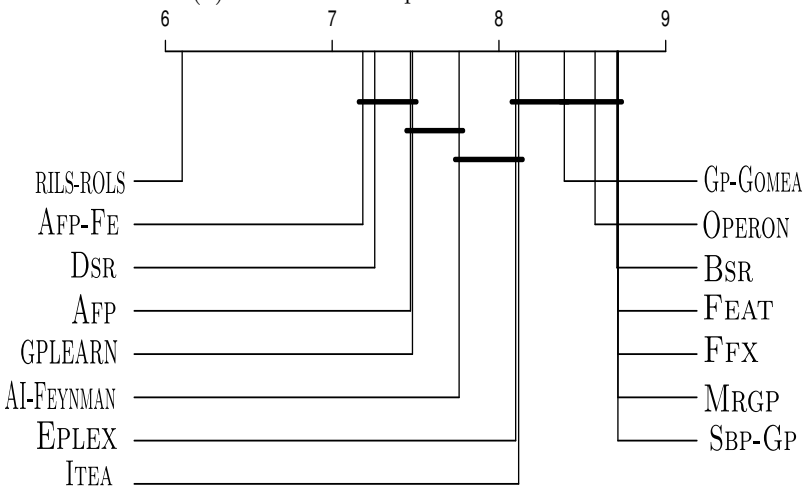
Additional file descriptions text (including details of how to view the file, if it is in a non-standard format or the file extension). This might refer to a multi-page table or a figure...



(a) No noise

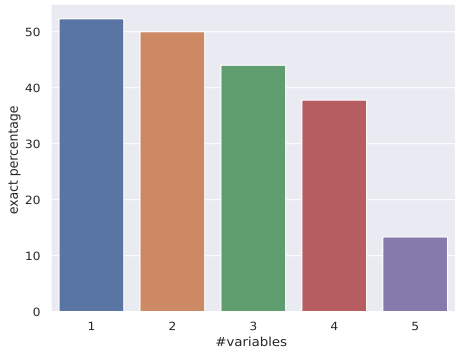


(b) Level of noise equal to 0.001

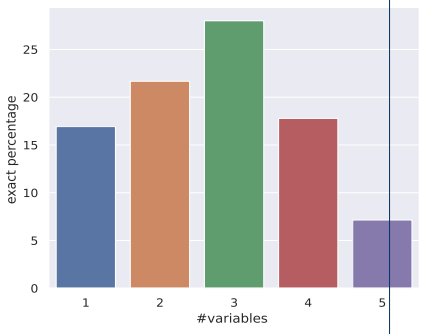


(c) Level of noise equal to 0.01.

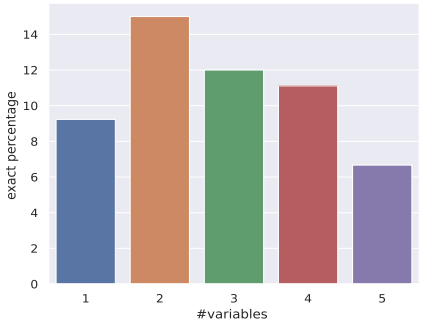
Figure 3: CD plots comparisons in terms of exact percentages over all problem instances from the both ground-truth benchmark sets, varying levels of noise.



(a) No noise

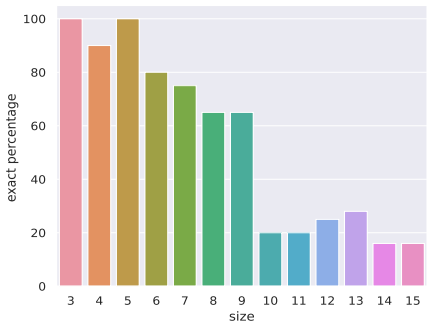


(b) Level of noise equal to 0.001

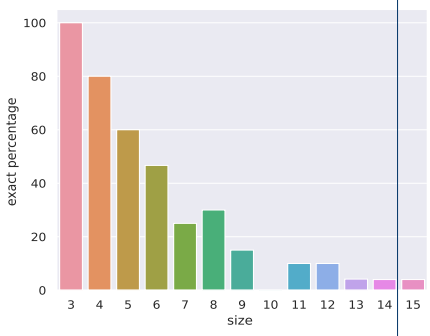


(c) Level of noise equal to 0.01.

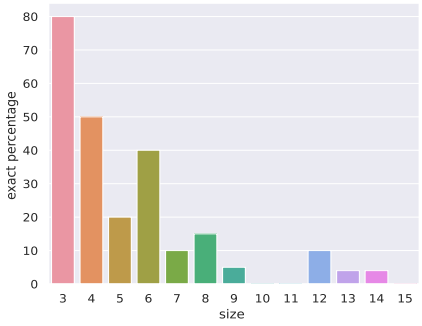
Figure 6: Exact solution percentages for varying levels of noise and variable counts.



(a) No noise

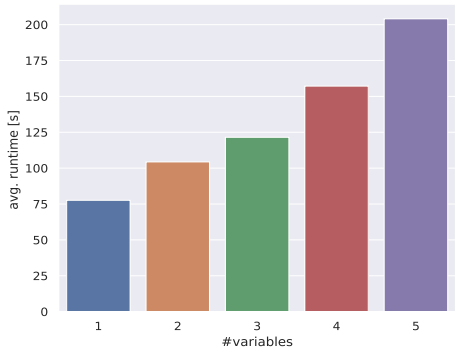


(b) Level of noise equal to 0.001

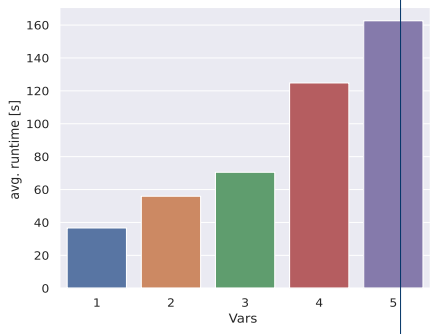


(c) Level of noise equal to 0.01.

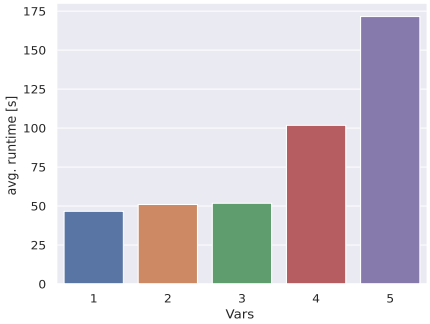
Figure 7: Exact solution percentages for varying levels of noise and exact model sizes.



(a) No noise



(b) Level of noise equal to 0.001



(c) Level of noise equal to 0.01.

Figure 8: Average runtime comparisons for varying levels of noise and variable counts.