RILS-ROLS: Robust Symbolic Regression via Iterated Local Search and Ordinary Least Squares

Aleksandar Kartelj[a], Marko Djukanović[b]

[a]*kartelj@matf.bg.ac.rs,*
Faculty of Mathematics, University of Belgrade, Serbia
[b]*marko.djukanovic@pmf.unibl.org,*
Faculty of Natural Sciences and Mathematics, University of Banja Luka, Bosnia and Herzegovina

Abstract

FiXme: TODO

1. Introduction

The problem of symbolic regression (SR) [6] has attracted many researchers over the last decade to study it intensively. SR can be seen as a generalization of the well known concept of linear regression, i.e., polynomial regression [50]. All regression models in principle have the same task: given a set of a n-dimensional input data and the output data, the aim is to find a mathematical expression (function) consisting of n (input) variables that fits to the output data w.r.t. some in advance known measure. This computationally intensive task is in general provenly NP–hard [55]. When aiming a model to be a linear combination of input variables, it represents the problem of linear regression. However, when there are some nonlinear relations between variables, which are relevant, linear regression models are not enough. This is the point where symbolic regression comes into the play. Unlike linear regression, it allows the search over the space of all possible mathematical formulas in order to find the best fitting ones able to predict the output variable from the input variables. The base of constructing the explicit formula are the basis operations like addition and multiplication, as well as polynomial, trigonometric, exponential, and other elementary functions.

Concrete examples of application of SR include having a less black–box tool, as it may say much more how SR model achieve predictions, that is the coefficients and functions that build the model may indicate on larger importance of some variables over the others. Moreover, in that way we may grasp why the variables are related in the obtained way. As an example, appearance of an exponential function may associate to some physical phenomenon such as intensity of radiation or acceleration over time, see [51]. Additionally, the SR models, if they are correct, may have posses larger extrapolation power than other mathematical models, and especially than neural network–based methods. SR models may maximize obtaining physically more reasonable equations that could serve as insight towards establishing physical theory behind as the final goal. Practical applications of SR in chemical and biological sciences are listed in [58]. In particular, the discovery of a series of new oxide perovskite catalysts with improved activities is presented there. Applications of SR to discovering physical laws from distorted video is studied in [52] by means of a unsupervised learning method. Revealing complex ecological dynamics by SR

is presented in [10], tackled by a machine learning method. Application of SR to model the mutations effects on protein stability, in the domain of fundamental and applied biology, is shown in [34]. One of the recent work studies of auto-discovering conserved quantities using trajectory data from unknown dynamical systems, where SR is solved by a machine learning algorithm, see [33]. Last but not least, we mention the paper [32] discovers the application of SR to model analytic representations of the exciton binding energy. The use of solving the SR in material science is described in [57, 56, 7, 18]. Yet another application to wind speed forecasting is given in [1].

There are many different ways to tackle the SR by achieving the right analytic expressions, most of them are related to machine learning techniques or to the technique of genetic programming (GP) and various approximate methods, such as meta-heuristics. Among the first GP methods to tackle SR is the one of Raidl [45] based on a hybrid variant of genetic programming. A differential evolution algorithm was proposed by Cerny et al. [9]. Age-fitness Pareto Optimization approach is proposed by Smidt and Lipson [48]. Application of artificial bee colony programming to solve SR is proposed by Karaboga et al. [21]. Influence of local–based heuristics on solving SR is reported by Commenda in his Ph.D. thesis [22]. A GP-based approach, the gene-pool optimal mixing evolutionary algorithm (GOMEA) is studied by Virgolin et al. [54]. Another evolutionary algorithm, the interaction-transformation EA (Itea) has been proposed by de Franca et al. [12]. Simulated annealing to solve SR is proposed by Kantor [20]. A Variable neighborhood programming approach to solve SR is proposed by Elleurich et al. [13]; this technique is initially proposed in [14]. Kommenda et al. [23] 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 [8]. The method that uses Taylor polynomial to approximate the symbolic equation that fits the dataset, called Taylor genetic programming, is proposed in [16]. A GP approach that uses the idea of semantic back-propagation (Sbp-Gp) is proposed in [53]. Empirical analysis of variance between many GP–based methods for SR is discussed in [19]. It is also worth to mention the GP–based Eurequa commercial solver [47, 46] that uses age-fitness pareto optimization with co-evolved fitness estimation. This solver is nowadays accepted for the gold standard of symbolic regression.

Machine learning–based method based on Bayesian symbolic regression (Bsr) is proposed in [17], belongs to the family of Markov Chain Monte Carlo algorithms. Deep Symbolic Regression (Dsr), a RNN approach, is proposed in [43] utilizing the policy gradient search. This mechanism of search is further investigated in [30]. A fast neural network approach, called OccamNet, is proposed in [11]. A deep reinforcement learning approach enhanced with genetic programming is given in [41].

Powerful hybrid techniques to solve SR are also well studied; among them, we emphasize the Eplex solver from [26, 29] and AI Feynman algorithm from [51], a physics-inspired divide-and-conquer method that combines neural network fitting. The later one is one of the most efficient method for physically inspired models. We also mention the Fast Function Extraction (Ffx) algorithm developed by McConaghy [39], a non-evolutionary method in combination with a machine learning technique—path-wise regularized learning— which quickly prune a huge set of candidate basis functions down to compact models.

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

Algorithm	Paper/year	Short details
Gp	[24] (1994)	Application of GP to SR
Hybrid-Gp	[45] (1998)	GP–based method; solutions locally optimizes by method of least squares to find optimum coefficients for top-level terms
Dfe	[9] (2008)	Differential evolution algorithm
Apf-Fe	[47, 46] (2009, 2011)	Age-fitness pareto optimization approach using co-evolved fitness estimation
APF	[48] (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	[21] (2012)	Artificial bee colony programming approach
Eplex	[29] (2016)	A parent selection method called ϵ–lexicase selection
Mrgp	[3] (2014)	It decouples and linearly combines a program's subexpressions via multiple regression on the target variable
Local optimization NLS	[22] (2018)	Constants Optimization in GP by Nonlinear Least Square
Feat	[28] (2018)	Features are represented as networks of multi-type expression trees comprised of activation functions; Differentiable features are trained via gradient descent
Sbp-Gp	[53] (2019)	The idea of semantic back-propagation utilized in GP
Bsr	[17] (2019)	ML–based approach; Bayesian symbolic regression
Dsr	[43] (2019)	Deep Symbolic Regression based on a RNN approach further utilizing the policy gradient search
Operon	[23] (2020)	Utilizing nonlinear least squares for parameter identification of SR models with LS
Vnp	[13] (2020)	VNS based GP approach
OccamNet	[11] (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	[51] (2020)	A physics-inspired divide-and-conquer method; it combines neural network fitting
Gomea	[54] (2021)	A model-based EA framework called Gene-pool optimal mixing evolutionary algorithm
Itea	[12] (2021)	EA based approach called the interaction-transformation EA
Sa	[20] (2021)	Simulated annealing approach
Drla	[41] (2021)	A deep reinforcement learning approach enhanced with genetic programming
Taylor-Gp	[16] (2022)	Taylor polynomials approximations

Table 1: Overview of methods to solve SR.

The progress in the SR field suffered from a lack of uniform, robust, and transparent benchmarking standards over the last two decade. Recently, La Cava et al. [27] proposed an open-source, reproducible benchmarking platform for SR, called SRBench. The authors extended PMLB [42], a repository of standardized regression, by 130 SR datasets for which model forms are known; see more in the aforementioned paper. In the extensive experimental evaluation where 14 symbolic regression methods and 7 machine learning methods are compared on the set of 252 diverse regression problems. The general conclusions derived from there are: (*i*) when comes to real-world problem, the best performing algorithms are those which combine genetic algorithm with parameter estimation; (*ii*) when come to the instances with the presence of noise, the GP–based and deep learning techniques are performing similarly.

In this work we present a novel approach to solve SR, which combines the popular iterated local search meta-heuristic (ILS) [35, 36] with ordinary least squares method (OLS) [31]. Briefly, ILS handles combinatorial (discrete) aspects of search space, while OLS helps in the process of coefficient determination, so it handles continuous parts of the search space. As will be shown later, the proposed method seems to be very robust w.r.t. introduced noise, therefore we called the method RILS-ROLS (with letters R corresponding to regression and robust). FiXme: TODO: maybe expand the core of the algorithm...

The main contributions of this work may be summarized as follows:

1. The proposed method outperforms all state-of-the-art methods for ground-truth problems from literature, considered in the SRBench benchmark.

2. It shows high robustness, which is proved by comparison to other algorithms under different levels of Gaussian white noise.

3. The method is very efficient, taking in average around xyz seconds to reach exact solution – when exact solution is found.

4. The new set of unbiased instances is also introduced – randomly generated formula of various sizes and number of input variables. This set was employed to understand the influence of formula size, number of input variables and noise on the solving difficulty.

2. Problem definition and search space

In this section we formally define the SR problem, followed by defining the search space framework of the problem.

Definition 1. *Given is a dataset* $D = \{(x_i, y_i)\}_{i=1}^{n}$*, where* $\boldsymbol{x} \in \mathbb{R}^d$ *represents input variables (features), and* y *target variable. Suppose that there exists an analytical model of the form* $f(\boldsymbol{x}) = g^*(\boldsymbol{x}, \theta^*) + \epsilon$ *that is a generator of all observations from* D*. The goal of SR is to learn a mapping* $\tilde{f}(\boldsymbol{x}) = \tilde{g}(\boldsymbol{x}, \tilde{\theta}) \colon \mathbb{R}^d \mapsto \mathbb{R}$ *estimated by searching through the space of (mathematical) expressions* $\tilde{g}$ *and parameters* $\tilde{\theta}$ *where* ϵ *is the presented white noise using given input data.*

Koza [24] introduced the problem of SR as a specific application of genetic programming. GP deals with the object called programs, which need to be optimized. 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, syntax trees are elements of the 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. How accurate is this mapping, we compute in the basis of the historical data D and the chosen error measure (such as MSE, or R^2). Interestingly, the search space of SR consists of its discrete and continuous part. More precisely, the model we are seeking for might be obtained as a composition of some functions from a (finite) set of mathematical functions. It is common to use the set of the following elementary mathematical functions in the experiments: $\sqrt{x}, x^2$, sin, cos, log, exp, arcsin, arccos, and

a^x. It is allowed to link functions will all four operators: $+$, $-$, $\times$, and $/$. After fixing functions that build models, the set of optimal coefficients that fits to this model must be provided. For the most physical models, these coefficients may receive any real value, that is they are continuous in its nature.

FiXme: TODO: needs for extension...

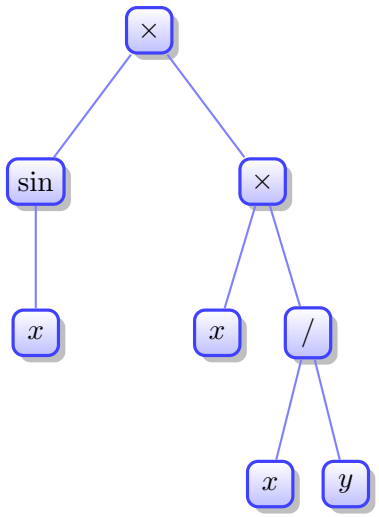


Figure 1: Syntax tree representation for the expression $\sin x \times (x \times x/y) = \frac{x^2 \sin x}{y}$

3. The proposed method

Before we give details on our RILS-ROLS method for solving SR, we will explain the basics of two main techniques incorporated into our algorithm. These are ILS meta-heuristic and OLS methods.

3.1. Iterated local search

Iterated local search (ILS) [35] is an efficient meta-heuristics, whose essential idea is given as follow. Iteratively generate a sequence of solutions produced by the (embedded) heuristic, such as local search 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 appeared in the literature by Baxter [5] 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 [4], large-step Markov chains [37], chained local optimization [38], or, in some cases, combinations of these [2].

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 is perturbed – it is partially randomized. Next, the perturbed solution is improved with LS procedure. Thirdly, the LS outcome is thereafter compared to the incumbent solution and the search us possibly moved to the new solution, with accordance to the chosen acceptance criterion. Sometimes, ILS incorporates a mechanism of tabu list, which prevents the search getting back to already visited solutions.

Algorithm 1 An ILS framework from literature.

1: **Input**: problem instance
2: **Output**: an approximate (feasible) solution
3: $s \leftarrow$ `Initialize`()
4: $s \leftarrow$ `LocalSearch`(s)
5: **while** *stopping criteria is not met* **do**
6: $\quad s' \leftarrow$ `Perturbation`(s)
7: $\quad s' \leftarrow$ `LocalSearch`(s')
8: $\quad s \leftarrow$ `AcceptanceCriterion`(s, s')
9: **end while**
10: **return** s

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 data $(X = (X_1, \ldots, X_d)^T, y)$, the task is to determine linear mapping $\tilde{y} = k\mathbf{x} + b$, that is coefficients (line slope) $k = (k_1, \ldots, k_d)$ and b (intercept), so that $\sum_i |\tilde{y}_i - y_i|^2$ is minimized. This sum is also know as the sum of squared error (SSE). There are many methods to minimize SSE. One of the analytical approaches is the calculus-oriented – it takes into account partial derivation w.r.t. k of the cost function $\sum_i |\tilde{y}_i - y_i|^2$, getting

$$\frac{\partial}{\partial k} \sum_i |\tilde{y}_i - y_i|^2 = \frac{\partial}{\partial k} \sum (kX_i + b - y_i)^2 = \sum 2X_i(kX_i + b - y_i) = \sum -2X_i(y_i - \tilde{y}_i).$$

Taking into account partial derivation w.r.t. b, we have

$$\frac{\partial}{\partial b} \sum_i (\tilde{y}_i - y_i)^2 = \frac{\partial}{\partial b} \sum (kX_i + b - y_i)^2 = \sum -2(y_i - \tilde{y}_i).$$

Thus, we get the system of equations:

$$\begin{aligned} -2X_1(y_1 - \tilde{y}_1) &= 0 \\ &\vdots \\ -2X_d(y_d - \tilde{y}_d) &= 0 \\ \sum -2(y_i - \tilde{y}_i) &= 0. \end{aligned}$$

By solving this system of $(d + 1)$ equations, one can obtain vector of coefficients $k = (k_1, ..., k_d)$ and intercept b. FiXme: TODO: Aca – ovde je pisalo da je to sistem sa dve jednacine i dve nepoznate k i b – medjutim, k je vektor. Ako mozes dopisi jos nesto, tipa koliko je podataka potrebno i slicno – mozda veremenska slozenost.)

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.

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')
      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

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 calculated 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 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. 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. the R^2 and $RMSE$ of the best solution. 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 3.3.2.

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 means of ordinary least squares method, 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 Python library called SymPy [40]. Local search tries to find local optima expressions close to the given p – explained in details in Section 3.3.3. Finally, fitness function value of p is compared to fitness function value of bs. If a new incumbent is found, bs is updated correspondingly and 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. this structure serves as a kind of tabu list, initially proposed in the famous Tablu search algorithm, see [15].

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. In order to avoid this undesired situation, a randomness needs to be triggered. 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 insignificant 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. Indeed, there is a minor chance that these 2-perturbations will have empty set difference with $start_{tried}$ set of starting solutions. Therefore, more expensive 3-perturbations are not considered in our algorithm.

Before returning the final symbolical model, RILS-ROLS peforms the final symbolical simplification and rounding of model coefficients. The later is sensitive to control parameter $tolerance_{error}$ – smaller value means information loss during rounding is smaller, i.e. 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.

FiXme: Dati neku smislenu procjenu kompleksnosti algoritma, ako to nije pretesko.

3.3.1. Fitness function

Symbolic regression objective is to determine the expression governing available data. It is also allowed to obtain some equivalent expression, keeping in mind that 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, simply because the goal expression is not known at that point as only target values for some of the input features are known. Thus, different 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 mean squared error (MSE) or root mean squared error (RMSE). The important aspect of the solution quality might also be the solution expression complexity. This follows the Occam's razor principle [11] 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.

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

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 $1 + R^2(s)$ 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, we award solutions that are more simple with its representation.

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

3.3.2. 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 thus keeping them within a reasonable size. This reduction is reasonable, since normalization preserves the essential functional form. (Note that coefficients get tuned later during the OLS phase and 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 $(x + y^3) \cdot x/\sqrt{y}$. The purpose of simplification is removing unnecessary coefficients, but in general it can also be 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 (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 $\{(x + y^3) \cdot x/\sqrt{y}, (x + y^3), x/\sqrt{y}, x, y^3, \sqrt{y}, y\}$. This set of

subtrees is obtained if left subtree of the whole expression is $(x + y^3)$ while the right one is $x/\sqrt{y}$. For each subtree n, the set of perturbations around n is generated (lines 6-9 of Algorithm 3).

Algorithm 4 shows how perturbations around given subtree (node) are generated.

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. 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) $(x + y^3) + x/\sqrt{y}$.

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 the 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. Subtree ss 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:

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

FiXme: Nekako zbunjuje u Algoritmu 4, pojava NodeChanges)() i Replace(). Mislim, razumijem iz opisa sta one rade, ali nisam siguran da treba imenovati metode, ako ih barem u kontekstu opisa ne pomenemo direktno po nazivu.

FiXme: Dati kompleksnost velicine skupa perturbacija u odnosu na velicinu drveta i dopustene promjene

3.3.3. Local search

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

For a given perturbation p, local search systematically explores all 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 that the actual best *bp*, the OLS coefficient fitting (`FitOLS`) is performed.

4. Experimental evaluation

Our Rils–Rols algorithm is implemented in Python version 3.x. All experiments are performed on a machine xx with xx GHz and a memory limit of xx GB in single-threaded mode. The maximum computation time allowed for each run was set to 1h minutes, i.e., 3600 seconds.

The following 15 algorithms are compared: AI-Feynman, GOMEA, Afp-Fe, Itea, Afp, Dsr, Operon, the Gplearn from python package gplearn [49], Sbp-Gp, Eplex, Bsr, Feat, Ffx, Mrgp and our Rils-Rols algorithm.

4.1. Datasets and SRBench

SRBench is an open-source benchmarking project which merge a large set of diverse benchmark datasets, contemporary SR methods as well as ML methods around a shared model evaluation and the environment for analysis, see [27]. It brought researchers an easy handled environment to evaluation, compare, and analyze their methods. Among all benchmark sets, we involve two real-world inspired benchmark sets into our computational evaluation:

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**
      $n_{candidates} \leftarrow$ `All1PerturbationsAroundNode`(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

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

Table 2: Some Feynman equations.

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 problems.

- FEYNMAN benchmark set, inspired from physics and formulas/models that correspond to various natural laws. It consists of 10^5 samples, each described in [51]. Some exact models (equations) of problem instances from this benchmark are shown in Table 2.

- STROGATZ benchmark set from [25]. Each dataset is represented by a 2-state system of first-order, ordinary differential equations. The aim of each problem is to predict rate of change of the subsequent state w.r.t the current two states on which it depends. These equations describes various non-linear dynamics also exhibiting chaos. The equations for some of the datasets that belong to this benchmark are shown in Table 3.

The above-mentioned benchmark sets may be biased as models which describe physical laws usually have symmetries, periodicity w.r.t. some variables, internal separability on some variables etc. In that way, the search process and the search space can be biased and in-advance significantly reduced. Thus, some methods may get significant help with this regards, while some others may be in the subordinate situation. In order to permit any kind of favoritism, we generate a set of randomly generated instances, called RANDOM. These instances are generated according the following procedure. FiXme: Opisati proceduru generisanja instanci + kombinaciju parametara za instance, itd.

All (15) competitor algorithms were run once per each instance.

4.2. Parameter tuning

The RILS-ROLS algorithms, apart of the most other methods for SR, employs just two parameters that need to be tuned, $penalty_{size}$ and $tolerance_{error}$. The first parameter is set to 0.001(?), according to the preliminary experimental evaluation. The second parameter is tuned monitoring the two errors the algorithm produced on the training set; we choose $tolerance_{error} =?$ for which a largest average R^2 and the smallest average RMSE error over all instances from the training set are produced. Note that the training set is chosen randomly with a cardinalty of xx instances.

FiXme: drugi parametar manje vise zavistan od ocekivanog nivoa suma – ne znam kako obrazloziti njegov tuning (mozda prema greski na trening skupu pa izabrati onaj za koji su R2 i RMSE bolji...).

4.3. Comparison with other methods on real–world benchark sets

In this section, we evaluate the performance of RILS-ROLS algorithm with 15 other competitors from the literature that solve the problem of Symbolic regression. The results are reported in terms of three numerical Tables 4–6, where two real-world benchmark sets, FEYNMAN and STORGATZ are considered, utilizing different values of the noise levels: 0.0 (no noise), 0.001 (a low-level noise), and 0.01 (a high-level noise), respectively. Concerning the added noise, the White Gaussian noise was added to the target (y-axis values) as a fraction of the signal RMS value, also applied in [27], i.e. for target noise level α, we have

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

Each table consist of three blocks. The first block reports the algorithm whose performances are reported. The second block reports, in two columns, percentages of instances where the model which matches to the exact one has been found by respective algorithms on both, FEYNMAN and STROGATZ, benchmark sets. The third column of this block provides the overall summary percentage of success over all instances. The third block displays the results of the respective algorithm in the presence of R^2 error which can indicate on accuracy, i.e. in our cases when the algorithm at least reached the accuracy (R^2 score) of 0.999. Note that the termination of our algorithm may not necessary be triggered by the error equal to 1. These results are of interest as it may point out the existence of an over-fitting. This block consists of three rows that indicate the percentage of instances where respective algorithm is able to reach the desired accuracy (fulfilling condition $R^2 \geq 0.999$ before a termination due exceeding the time limit) on the both real–world benchmark sets separately, and the overall percentage of success rate (over all considered instances) w.r.t. the same criterion of success.

In short, the exact results, thus the number of instances (in percentages) for which the output of respective algorithm match to the exact model, are displayed in the second block of each of the tables. The numbers of instances (in percentages) for which each algorithms deliver a precise enough model (i.e. satisfying $R^2 \geq 0.999$ upon its termination) are displayed in the third block of each of the tables.

The following conclusions may be drawn from the numerical results:

- Concerning the exact results in case no noise is utilized in the instance problems, the best performing algorithm is our RILS–ROLS, able to find the right model on 57.84% problems instances of the benchmark set FEYNMAN; the same algorithm was even more impressive in solving problem instances from benchmark set STROGATZ, begin successful in 83.57% cases. The second best performing approach is AI-FEYNMAN, successful in solving 55.78% and just 27.14% of all instances of the benchmark set FEYNMAN and STROGATZ, respectively. All other approaches are performing significantly worse, and none of them is able to solve more than 30% instances of any of the two considered benchmark sets.

- Concerning the exact results when the target noise is presented with a level of 0.001, our Rils–Rols is still performing reasonably well, having found an exact model in more than 42% cases considering the problems from the both benchmark sets. Interestingly, AI-Feynman is quite sensible w.r.t. the increase in the target noise level, as its performances rapidly drop down – it is successful on just 2.81% of the all problem instances. Better from AI-Feynman are in this case the Dsr, Gplearn, and Afp-Fe in this order of success, whose percentages of success rate range from ≈15–17%.

- Concerning the exact results when the presence of noise is significant, that is with a level of 0.01, our Rils–Rols is still performing reasonably well, having found an exact model for 34.77% of the all problem instances. The second best is Gplearn which percentage of success is just 12.75%, followed by Dsr which is successful for just 9.23% of the all problem instances.

- When comparing performances of the algorithms in terms of number of instances for which the algorithm's termination led to an accurate model, that is $R^2 \geq 0.999$, in case when no noise is integrated, our Rils-Rols is successful with the rate of success of 83.38%. However, in contrast with the exact model percentages comparisons from above, there are a few methods reaching even better percentages of success than ours: Mrgp, and Operon, with 92.69%, and 86.93%. On the other hand, as the exact model percentage rates of these two approaches are rather low (0% and 16%, respectively) these indicate the presence of a high level over-fitting of the later two algorithms. Thus, they are able to obtain a model that is accurate enough on the training data, but most of the produced models are in essence wrong.

- When comparing the performances of the competitors in terms of number of instances for which the termination led to an accurate enough model, in the presence of the noise level of 0.001, our Rils–Rols method has 78.62% of a success rate, concerning all considered instances. The Operon solver only has a better percentage of success among the competitors, which is 83.08%. However, the over-fitting is conspicuous as this algorithm is rarely able to obtain the model which matches to the exact one (which is successful for 0.31% cases).

- When comparing the performances of the competitors in terms of number of instances for which the termination led to an accurate enough model, in the presence of the high–level noise of 0.01, our Rils–Rols method is still highly accurate, delivering 62.92% of a success rate. Interestingly, it is the best performing in terms of the percentage rate among other competitors. Just for the information, the second best approach is Dsr, with a 16.92% success rate. It can be concluded that our Rils–Rols method does not pay a big price in presence of a higher noise, like the other methods do, and seem to be very sensible w.r.t. the noise increase. Our Rils–Rols method still delivers high-accuracy models for the noisy input data, but also many of them match to the exact solutions (models).

- From the above conclusions, our Rils-Rols algorithm is more robust than the other approaches, as shown by the produced results w.r.t. different levels of noise

in the input data. Moreover, this is a new state-of-the-art algorithms when one concerns the problem of SR.

Table 4: 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	52.69	78.51	35.71	73.9
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	26.93
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**	86.93
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	11.93
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**

Table 5: 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
Dsr	16.81	18.75	17.02	26.29	14.29	25
gplearn	15.84	8.93	15.1	24.14	7.14	22.31
Afp-Fe	15.95	6.43	14.92	57.76	28.57	54.62
Afp	14.87	7.14	14.04	44.83	25	42.69
Eplex	10.02	2.68	9.23	48.71	21.43	45.77
AI-Feynman	1.49	13.73	2.81	9.73	35.71	12.53
Itea	1.94	4.46	2.21	26.72	21.43	26.15
Gomea	1.62	0.89	1.54	72.84	71.43	72.69
Operon	0.34	0	0.31	**81.9**	**92.86**	**83.08**
Ffx	0.11	0	0.1	19.4	14.29	18.85
Feat	0	0.89	0.1	13.79	50	17.69
Sbp-Gp	0	0	0	62.07	53.57	61.15
Bsr	0	0	0	7.76	14.29	8.46
Mrgp	0	0	0	0.86	17.86	2.69

4.4. Scalability of Rils-Rols *algorithm*

In this section we study scalability of our method in terms of different noise level in the input data as well as different complexity of the exact solutions (models).

Table 6: 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**
gplearn	13.21	8.93	12.75	14.22	7.14	13.46
Dsr	8.41	16.07	9.23	17.24	14.29	16.92
Eplex	8.77	0	7.83	14.22	3.57	13.07
Afp	7.11	0.89	6.44	8.19	7.14	8.08
Afp-Fe	6.12	3.57	5.85	5.6	0	5
Itea	0.11	0.89	0.19	1.72	10.71	2.69
AI-Feynman	0	0.79	0.09	0	0	0
Operon	0.09	0	0.08	0	0	0
Gomea	0	0	0	0	0	0
Sbp-Gp	0	0	0	0	0	0
Bsr	0	0	0	0	0	0
Feat	0	0	0	0	14.29	1.54
Ffx	0	0	0	0	0	0
Mrgp	0	0	0	0	0	0

First, we divide our benchmark sets into three parts as follows.

- Small–sized instances: all instances from Random benchmark whose the corresponding exact models tree representations have from 3 to 7 nodes belong to this sub-set.
- Middle–sized instances: all instances from Random benchmark whose the corresponding exact models tree representations have from 8 to 11 nodes belong to this sub-set.
- Large–sized instances: all instances from Random benchmark whose the corresponding exact models tree representations have from 12 to 15 nodes belong to this sub-set.

The results are displayed in Figure 2. They are grouped with accordance to the (three) formed sub-groups (x-axis) and for each of them the exact percentage rate of success of Rils–Rols algorithm is shown (y-axis). The following conclusions may be drawn from here:

- As expected, the success rate is the highest for the small–sized instance problems; it is sightly bellow 90% for the clean (no-noise) data. For more noisy data, the success rate of the algorithm gets decreased. For example, on the small-sized instances with the presence of the noise of level 0.0001, and 0.01, the rate of success of Rils–Rols is still reasonably well, about 55% and 35 %, respectively.
- For middle-sized instance problems, the rate of success is slightly bellow 50%. It also decreases by increasing the level of noise in the target data; for the highest level of chosen noise (of 0.01), the success of rate is about 6%.

- For the large-sized instance problems without the presence of a noise, the rate fo success of RILS–ROLS algorithm is at about 25%, which shows that increase in the size of exact models affects the algorithm's performance, but within reasonable expectations.

- We conclude that the size of exact models (solutions) and the increase of the noise level evenly contribute to overall performance of our RILS–ROLS method. The success rate of the algorithm (i.e. the exact percentage) seems decreasing linearly with the average growing size of the exact models, regardless of values of the noise level.

- Concerning the high–accuracy of the RILS–ROLS algorithm – that is when termination of the algorithm yielded the error $R^2 \geq 0.999$ – for the input data with the zero noise level on the small–sized random instance problems, it almost reaches the perfect score (i.e. ≈100%). In the presence of a low noise level (equals to 0.001), the accuracy delivered by the algorithm is still high, at about 95%. Just when a high level of noise (equals to 0.01) is presented in the input data, the accuracy stagnates, but still on the respectable 66% small-sized instances it reaches a high–precision of $R^2 = 0.999$ upon termination. However, as we said, a half of them math to the respectable exact models.

- Concerning the high–accuracy of the RILS–ROLS algorithm on the middle-sized instance problems, it is obtained for 65% cases in the case when the input data are free of the noise. The noisy the data are, the accuracy of the algorithms, as expected, drops off. However, not so dramatically . For the level of noise of 0.01, the high–accuracy is reported on ≈30% cases.

- Concerning the high–accuracy of the RILS–ROLS algorithm on the large-sized instance problems, the high–accuracy in the data free of the noise presence is reported for 60% cases. Not so surprisingly, the hardest case to obtain the desired high–accuracy (that is, ensuring n accuracy of $R^2 \geq 0.999$ upon termination) is in the presence of a high level of noise (of 0.01), where the algorithm was successful on about 20% of the considered large–sized instance problems.

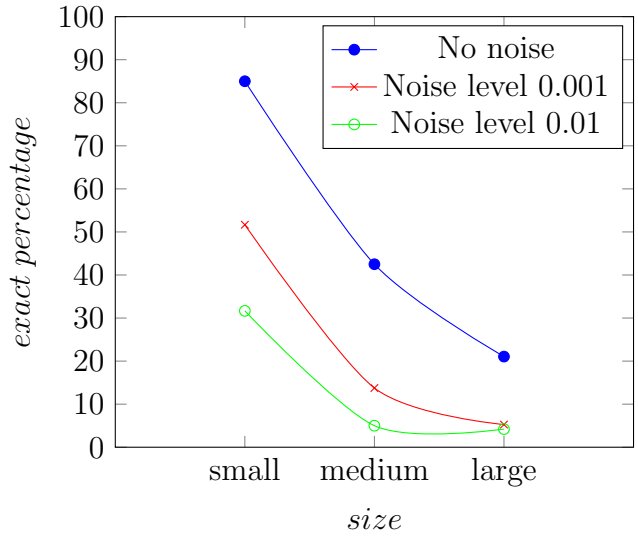


Figure 2: Exact solution percentages for varying levels of noise and formulae sizes

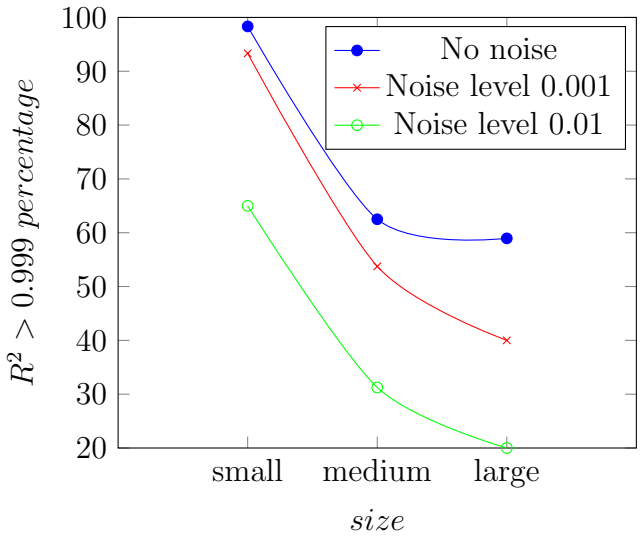
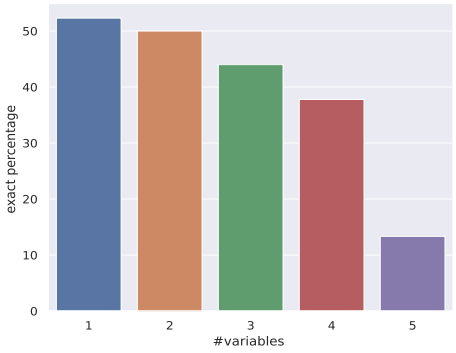


Figure 3: Percentages of solutions having $R^2 > 0.999$ for varying levels of noise and formulae sizes

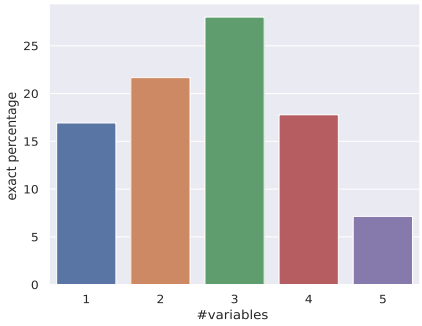
In this section we investigate the scalability of our RILS–ROLS algorithm in terms of the number of variables and sizes of corresponding exact models of the problem instances varying levels of noise. There is one line plot shown per each level of noise. These are shown by Figures 4 and 5. The instances are grouped according to the different variable count/size of respective exact models (x-axis). For each group, the exact percentages of success rate of RILS–ROLS algorithm is shown (y-axis).

The following conclusions can be drawn from Figure 4.

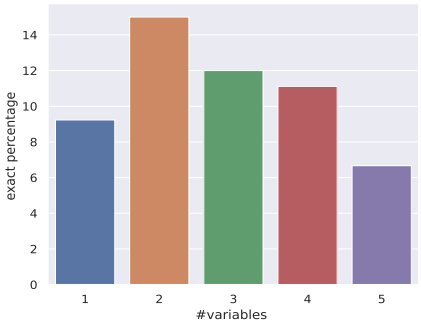
- The rate of success of RILS–ROLS algorithm slightly decrease as the expected number of variables gets larger in case of the data with no noise.
- The success rate percentages drop off with the presence of noise. The higher the noise level, the smaller the success rate percentages.
- Interestingly, in the presence of noise, the highest exact percentages are not for the instances whose exact models have one variable, but two or three. We argue it by the fact that the early phase of the algorithm will more frequently include additional variables in the solution at the cost of increasing model accuracy to ensure the fitness value improvements over iterations. In the presence of the zero noise, OLS method efficiently finds a smaller model (on $\approx 50\%$ cases) of a high accuracy that fits to the input data. However, this is no the case when the data is noisy and the decision of adding more variables is forced. By a progression of the algorithms, it gets harder improving the fitness value of the current best solution by adding variables, and other operations are more frequently involved.



(a) No noise



(b) Level of noise equal to 0.001

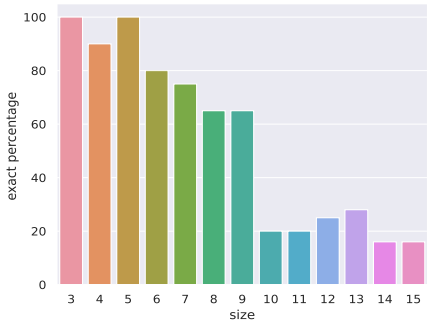


(c) Level of noise equal to 0.01.

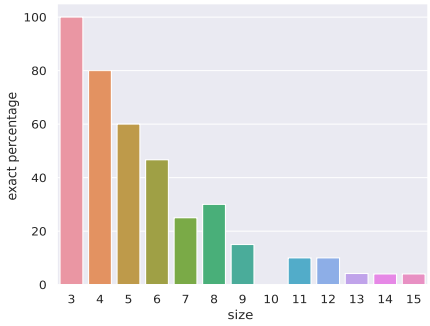
Figure 4: Exact solution percentages for varying levels of noise and variable counts.

The following conclusions may be drawn from Figure 5.

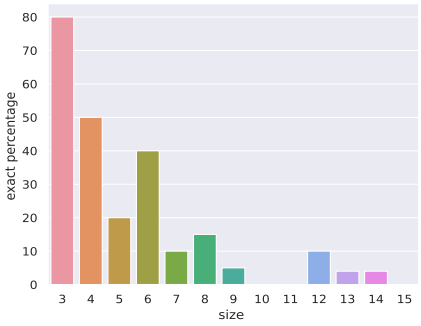
- The exact percentage rates are decreased as the size of exact models increase, in case of any level of noise is utilized within the input data.
- The highest exact percentages, higher than 70%, are noticed for the instances with a small-to-middle size (3–7) exact models. However, for the same sizes, for noisy data, these percentages are significantly lower; for example, in case of models of size 7 and the level of noise of 0.001 and 0.01, the exact percentages obtained are 20% and 10%, respectively.
- For the largest sizes of (exact) models, for the no noisy data the exact percentages are never lower from 20%. The other way around holds for noisy data.



(a) No noise



(b) Level of noise equal to 0.001

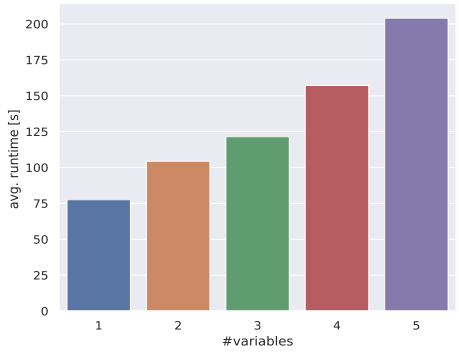


(c) Level of noise equal to 0.01.

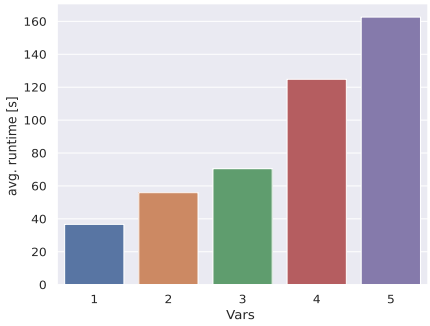
Figure 5: Exact solution percentages for varying levels of noise and exact model sizes.

When one concerns of runtime analysis of the RILS–ROLS algorithm on the RANDOM benchmark set w.r.t number of variables, the results are shown in Figure 6. The instances are grouped w.r.t. number of variables included into corresponding exact model. The following conclusions may be drawn from there.

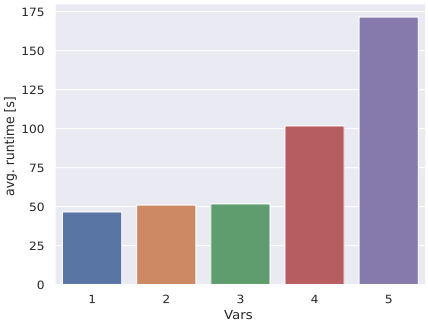
- As expected, average runtime is getting increased with the increase of the number of variables.
- Note that the average runtime in case of the instances whose exact models have the largest number of variables (5) is about 200s.
- The two above-mentioned points indicate efficacy of our method especially in case of expected exact models of a smaller size on the data with small–to–middle levels of noise. In these cases, the algorithm is still able to deliver the reasonable exact percentages, which are larger than 50%. Note that the models with these characteristics are desirable in the areas of natural sciences due to their interpret-ability more than models represented by complex expressions.



(a) No noise



(b) Level of noise equal to 0.001



(c) Level of noise equal to 0.01.

Figure 6: Average runtime comparisons for varying levels of noise and variable counts.

4.5. Statistical evaluation

In order to check the statistical significance of the obtained exact percentages and their differences between the studied approaches, we employed the statistical methodology which is outlined below. This concerns the 15 approaches on two real-world benchmark sets, FEYNMAN and STROGATZ. 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 results of the competitor approaches) pairwise comparisons are further performed by using the Nemenyi post-hoc test [44]. The outcome is represented by means of critical difference (CD) plots. In a CD plot, the 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 displayed. FiXme: Marko: continue

5. Conclusions and future work

Appendix A. Overview of RILS-ROLS python package

References

[1] I. A. Abdellaoui and S. Mehrkanoon. Symbolic regression for scientific discovery: an application to wind speed forecasting. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 01–08. IEEE, 2021.

[2] D. Applegate, W. Cook, and A. Rohe. Chained lin-kernighan for large traveling salesman problems. *INFORMS Journal on Computing*, 15(1):82–92, 2003.

[3] I. Arnaldo, K. Krawiec, and U.-M. O'Reilly. Multiple regression genetic programming. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 879–886, 2014.

[4] E. Baum. Iterated descent: a better algorithm for local search in combinatorial optimization, 1998.

[5] J. Baxter. Local optima avoidance in depot location. *Journal of the Operational Research Society*, 32(9):815–819, 1981.

[6] L. Billard and E. Diday. Symbolic regression analysis. In *Classification, clustering, and data analysis*, pages 281–288. Springer, 2002.

[7] B. Burlacu, M. Kommenda, G. Kronberger, S. Winkler, and M. Affenzeller. Symbolic regression in materials science: Discovering interatomic potentials from data. *arXiv preprint arXiv:2206.06422*, 2022.

[8] B. Burlacu, G. Kronberger, and M. Kommenda. Operon c++ an efficient genetic programming framework for symbolic regression. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, pages 1562–1570, 2020.

[9] B. M. Cerny, P. C. Nelson, and C. Zhou. Using differential evolution for symbolic regression and numerical constant creation. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1195–1202, 2008.

[10] Y. Chen, M. T. Angulo, and Y.-Y. Liu. Revealing complex ecological dynamics via symbolic regression. *BioEssays*, 41(12):1900069, 2019.

[11] A. Costa, R. Dangovski, O. Dugan, S. Kim, P. Goyal, M. Soljačić, and J. Jacobson. Fast neural models for symbolic regression at scale. *arXiv preprint arXiv:2007.10784*, 2020.

[12] F. O. de França and G. S. I. Aldeia. Interaction–transformation evolutionary algorithm for symbolic regression. *Evolutionary computation*, 29(3):367–390, 2021.

[13] S. Elleuch, B. Jarboui, N. Mladenovic, and J. Pei. Variable neighborhood programming for symbolic regression. *Optimization Letters*, pages 1–20, 2020.

[14] S. Elleuch, N. Mladenovic, and B. Jarboui. *Variable neighborhood programming: a new automatic programming method in artificial intelligence*. GERAD HEC Montréal, 2016.

[15] F. Glover and M. Laguna. Tabu search. In *Handbook of combinatorial optimization*, pages 2093–2229. Springer, 1998.

[16] B. He, Q. Lu, Q. Yang, J. Luo, and Z. Wang. Taylor genetic programming for symbolic regression. *arXiv preprint arXiv:2205.09751*, 2022.

[17] Y. Jin, W. Fu, J. Kang, J. Guo, and J. Guo. Bayesian symbolic regression. *arXiv preprint arXiv:1910.08892*, 2019.

[18] E. Kabliman, A. H. Kolody, J. Kronsteiner, M. Kommenda, and G. Kronberger. Application of symbolic regression for constitutive modeling of plastic deformation. *Applications in Engineering Science*, 6:100052, 2021.

[19] L. Kammerer, G. Kronberger, and S. Winkler. Empirical analysis of variance for genetic programming based symbolic regression. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 251–252, 2021.

[20] D. Kantor, F. J. Von Zuben, and F. O. de Franca. Simulated annealing for symbolic regression. In *proceedings of the genetic and evolutionary computation conference*, pages 592–599, 2021.

[21] D. Karaboga, C. Ozturk, N. Karaboga, and B. Gorkemli. Artificial bee colony programming for symbolic regression. *Information Sciences*, 209:1–15, 2012.

[22] M. Kommenda and M. Affenzeller. Local optimization and complexity control for symbolic regression. *Ph. D. dissertation, Ph. D. thesis*, 2018.

[23] M. Kommenda, B. Burlacu, G. Kronberger, and M. Affenzeller. Parameter identification for symbolic regression using nonlinear least squares. *Genetic Programming and Evolvable Machines*, 21(3):471–501, 2020.

[24] J. R. Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4(2):87–112, 1994.

[25] W. La Cava, K. Danai, and L. Spector. Inference of compact nonlinear dynamic models by epigenetic local search. *Engineering Applications of Artificial Intelligence*, 55:292–306, 2016.

[26] W. La Cava, T. Helmuth, L. Spector, and J. H. Moore. A probabilistic and multi-objective analysis of lexicase selection and ε-lexicase selection. *Evolutionary Computation*, 27(3):377–402, 2019.

[27] W. La Cava, P. Orzechowski, B. Burlacu, F. O. de França, M. Virgolin, Y. Jin, M. Kommenda, and J. H. Moore. Contemporary symbolic regression methods and their relative performance. *arXiv preprint arXiv:2107.14351*, 2021.

[28] W. La Cava, T. R. Singh, J. Taggart, S. Suri, and J. H. Moore. Learning concise representations for regression by evolving networks of trees. *arXiv preprint arXiv:1807.00981*, 2018.

[29] W. La Cava, L. Spector, and K. Danai. Epsilon-lexicase selection for regression. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 741–748, 2016.

[30] M. Landajuela, B. K. Petersen, S. K. Kim, C. P. Santiago, R. Glatt, T. N. Mundhenk, J. F. Pettit, and D. M. Faissol. Improving exploration in policy gradient search: Application to symbolic optimization. *arXiv preprint arXiv:2107.09158*, 2021.

[31] L. Leng, T. Zhang, L. Kleinman, and W. Zhu. Ordinary least square regression, orthogonal regression, geometric mean regression and their applications in aerosol science. In *Journal of Physics: Conference Series*, volume 78, page 012084. IOP Publishing, 2007.

[32] J. Liang and X. Zhu. Phillips-inspired machine learning for band gap and exciton binding energy prediction. *The journal of physical chemistry letters*, 10(18):5640–5646, 2019.

[33] Z. Liu and M. Tegmark. Machine learning conservation laws from trajectories. *Physical Review Letters*, 126(18):180604, 2021.

[34] B. B. Louis and L. A. Abriata. 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.

[35] H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search. In *Handbook of metaheuristics*, pages 320–353. Springer, 2003.

[36] H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search: Framework and applications. In *Handbook of metaheuristics*, pages 129–168. Springer, 2019.

[37] O. Martin, S. W. Otto, and E. W. Felten. *Large-step Markov chains for the traveling salesman problem*. Citeseer, 1991.

[38] O. C. Martin and S. W. Otto. Combining simulated annealing with local search heuristics. *Annals of operations research*, 63(1):57–75, 1996.

[39] T. McConaghy. Ffx: Fast, scalable, deterministic symbolic regression technology. In *Genetic Programming Theory and Practice IX*, pages 235–260. Springer, 2011.

[40] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, v. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, and A. Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, Jan. 2017.

[41] T. Mundhenk, M. Landajuela, R. Glatt, C. P. Santiago, B. K. Petersen, et al. Symbolic regression via deep reinforcement learning enhanced genetic programming seeding. *Advances in Neural Information Processing Systems*, 34:24912–24923, 2021.

[42] R. S. Olson, W. La Cava, P. Orzechowski, R. J. Urbanowicz, and J. H. Moore. Pmlb: a large benchmark suite for machine learning evaluation and comparison. *BioData mining*, 10(1):1–13, 2017.

[43] B. K. Petersen, M. L. Larma, T. N. Mundhenk, C. P. Santiago, S. K. Kim, and J. T. Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. *arXiv preprint arXiv:1912.04871*, 2019.

[44] T. Pohlert. The pairwise multiple comparison of mean ranks package (pmcmr). *R package*, 27(2019):9, 2014.

[45] G. R. Raidl. A hybrid gp approach for numerically robust symbolic regression. *Genetic Programming*, pages 323–328, 1998.

[46] M. Schmidt. Machine science: Automated modeling of deterministic and stochastic dynamical systems. 2011.

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

[48] M. D. Schmidt and H. Lipson. Age-fitness pareto optimization. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 543–544, 2010.

[49] T. Stephens. Genetic programming in python with a scikit-learn inspired api: Gplearn, 2016.

[50] J. A. Stimson, E. G. Carmines, and R. A. Zeller. Interpreting polynomial regression. *Sociological Methods & Research*, 6(4):515–524, 1978.

[51] S.-M. Udrescu and M. Tegmark. Ai feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, 2020.

[52] S.-M. Udrescu and M. Tegmark. Symbolic pregression: Discovering physical laws from distorted video. *Physical Review E*, 103(4):043307, 2021.

[53] M. Virgolin, T. Alderliesten, and P. A. Bosman. Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression. In *Proceedings of the genetic and evolutionary computation conference*, pages 1084–1092, 2019.

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

[55] M. Virgolin and S. P. Pissis. Symbolic regression is np-hard. *arXiv preprint arXiv:2207.01018*, 2022.

[56] C. Wang, Y. Zhang, C. Wen, M. Yang, T. Lookman, Y. Su, and T.-Y. Zhang. Symbolic regression in materials science via dimension-synchronous-computation. *Journal of Materials Science & Technology*, 122:77–83, 2022.

[57] Y. Wang, N. Wagner, and J. M. Rondinelli. Symbolic regression in materials science. *MRS Communications*, 9(3):793–805, 2019.

[58] B. Weng, Z. Song, R. Zhu, Q. Yan, Q. Sun, C. G. Grice, Y. Yan, and W.-J. Yin. Simple descriptor derived from symbolic regression accelerating the discovery of new perovskite catalysts. *Nature communications*, 11(1):1–8, 2020.