Genetic Programming and Evolvable Machines (2020) 21:471–501
https://doi.org/10.1007/s10710-019-09371-3

Parameter identification for symbolic regression using nonlinear least squares

Michael Kommenda[1] · Bogdan Burlacu[1] · Gabriel Kronberger[1] · Michael Affenzeller[1,2]

Received: 20 January 2019 / Revised: 4 November 2019 / Published online: 10 December 2019

Abstract

In this paper we analyze the effects of using nonlinear least squares for parameter identification of symbolic regression models and integrate it as local search mechanism in tree-based genetic programming. We employ the Levenberg–Marquardt algorithm for parameter optimization and calculate gradients via automatic differentiation. We provide examples where the parameter identification succeeds and fails and highlight its computational overhead. Using an extensive suite of symbolic regression benchmark problems we demonstrate the increased performance when incorporating nonlinear least squares within genetic programming. Our results are compared with recently published results obtained by several genetic programming variants and state of the art machine learning algorithms. Genetic programming with nonlinear least squares performs among the best on the defined benchmark suite and the local search can be easily integrated in different genetic programming algorithms as long as only differentiable functions are used within the models.

Keywords Genetic programming · Symbolic regression · Parameter identification · Nonlinear least squares · Automatic differentiation

The authors gratefully acknowledge the financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development within the Josef Ressel Center for Symbolic Regression.

✉ Michael Kommenda
michael.kommenda@fh-hagenberg.at

[1] Heuristic and Evolutionary Algorithm Laboratory, Josef Ressel Center for Symbolic Regression, University of Applied Sciences Upper Austria, Softwarepark 11, 4232 Hagenberg, Austria

[2] Institute for Formal Models and Verification, Johannes Kepler University, Altenberger Straße 69, 4040 Linz, Austria

1 Introduction

Symbolic regression is the task of finding a mathematical model that best explains the relationship between one or more independent variables and one dependent variable. The ability to simultaneously search the space of possible model structures and their parameters (in terms of appropriate numerical coefficients) makes genetic programming (GP) [21, 34] a popular approach for symbolic regression.

However, as a biologically-inspired approach guided by fitness-based selection, the GP search process for symbolic regression is characterized by a loose coupling between fitness, expressed as an error measure with respect to the target variable, and variation operators | subordinate search heuristics in solution space that generate new models in each generation.

Consequently, it is difficult to foresee the effects on model output when variation operators perform changes on the model structure, often leading to situations where promising model structures are ignored by the algorithm due to low fitness caused by ill-fitting parameters [44]. In some cases, this can lead to necessary building blocks becoming extinct in the population before they are combined in a solution and thus recognized by the algorithm.

Generally speaking, achieving high-quality solutions in GP-based symbolic regression requires solving three interrelated subtasks:

1. Selection of the appropriate subset of variables (feature selection)
2. Discovery of the best suited model structure containing these variables
3. Determination of optimal parameter values of the model

Each of these subtasks depends on the results of the previous subtask to generate optimal solutions, therefore improvements of one task can lead to improvements to the whole algorithm. Although these three subtasks have to be solved in any case, most algorithms create solutions without explicitly addressing the necessary steps.

Symbolic regression problems can be solved by tree-based GP that evolves individuals which capture all characteristics of a solution such as the appropriate subset of variables, model structure, and parameters. Individuals are in general manipulated as a whole (by crossover and/or mutation), which results in difficulties for generating good solutions. The reason is that such evolutionary variations partially change the model structure and occurring variables which necessitates a re-fitting of all parameters of the model.

In this context, hybridization with local search methods improves algorithm effectiveness by shifting the burden of finding appropriate numerical coefficients to subordinate algorithms or heuristics that become part of the evolutionary process at the same level with crossover and mutation operators. This division of tasks is particularly appropriate since genetic programming is already well suited for variable selection [6, 41].

Memetic algorithms combine such refinement methods with global optimization methods, often population-based, evolutionary algorithms [7]. In its initial

definition, a memetic algorithm [30] is a genetic algorithm [11] hybridized with hill climbing, but hybridization generally works with many other algorithms:

- Metaheuristics such as genetic algorithms [13], differential evolution [52], evolution strategies [2, 49], or simulated annealing [40].
- Machine learning algorithms such as linear regression [14, 25, 35]
- Numerical optimization algorithms such as multipoint approximation [43] or gradient descent [5, 42, 50, 51].

In this contribution we study the hybridization of tree-based GP with nonlinear least squares optimization of numeric parameters and discuss aspects ranging from implementation to runtime performance and solution quality. Our methodology for parameter identification in symbolic regression combines linear scaling, automatic differentiation and gradient-based optimization through the Levenberg–Marquardt (LM) algorithm. A novel contribution is the comparison of results produced by our proposed approach to results of a number of similar approaches that integrate local search mechanisms as well and other non-evolutionary regression techniques as reported in [33].

1.1 Numerical parameters in symbolic regression

When performing symbolic regression, numerical parameters are of prime importance. Numerical parameters, also referred to as constants, form together with variables the terminal set $\mathscr{T}$. Every new leaf node in a symbolic expression tree is initialized with elements from the terminal set $\mathscr{T}$. Internal nodes are initialized with elements from the function set $\mathscr{F}$. Together, the two sets define the primitive set $\mathscr{P}$ used by the GP system to generate new symbolic expressions.

Constants in $\mathscr{T}$ can be either explicitly stated (as *predefined* and *immutable* numerical constants), or they can be defined as *ephemeral random constants* (ERC) [21]. In the first case, the terminal set may contain different numerical values alongside variables, such as $\mathscr{T} = \{X, 1.0, 2.0, \pi\}$ containing the variable X and three predefined numerical constants. Consequently, random uniform initialization would result in a 75% probability for a constant to be selected during tree creation when a leaf node is initialized. This disparate ratio between constants and variables could be altered by introducing a selection bias for terminal symbols.

In the second case, the special symbol $\mathscr{R}$ is added to the terminal set and every time the ERC symbol $\mathscr{R}$ is selected during tree creation a new constant value is sampled from a predefined distribution. ERCs provide greater flexibility as it is possible to create a greater variety of real-valued constants, compared to including constants directly in the terminal set. Whether immutable or ephemeral constants are more suitable largely depends on the problem at hand and both approaches might even be combined.

The numerical constants potentially reachable through the variation of solutions depend solely on the initial constant values during tree creation. To allow new constants to be discovered, GP literature recommends adding special manipulation

operators to the algorithm that alter the numerical constants of a solution, for example as described in Schoenauer et al. [38], where a random Gaussian variable is added to the constant, which is inspired by mutation in evolution strategies [39]. A similar technique by Gaussian mutation is detailed in Ryan and Keijzer [37]. Another possibility inspired by simulated annealing [18] is to replace all numeric constants with new values, sampled from a uniform distribution and adapted by a temperature factor [10]. Another alternative for adapting numerical values in symbolic regression is the inclusion of local search in GP.

1.2 Literature review

Local search aims to find a local optimum starting from a single solution. The best solution among a neighboring set of solutions is selected by applying a local move and through iterative application of such moves a local optimum is reached. Local search algorithms are often employed as subordinate heuristics for solution refinement within a higher-level metaheuristic framework. In the context of symbolic regression, local search refers to a further improvement of existing solutions towards a local optimum.

Overall, the hybridization of GP with local search methods represents a good fit. The inherent disadvantage of local search methods of converging toward the next attracting local optimum (depending on initial starting conditions) is reduced by GP, because it is likely that multiple, differently parameterized instances of the same model structure are present in the population, thus providing different starting conditions for the local search.

Another helpful technique to escape such local optima is to keep random mutation enabled, although its role and significance are reduced as the identification of appropriate numerical parameters is often performed by local search methods. As a result mutation is mostly responsible for introducing variations during the search for symbolic regression solutions.

Krawiec [22] integrates a hill-climbing algorithm into GP for symbolic classification and applies it to the best solution in each generation. The author reports a statistically significant improvement over standard GP in terms of classification accuracy. This result shows that even a small amount of local optimization can provide substantial benefits.

Topchy and Punch [42] use gradient-descent for parameter adaptation in GP for symbolic regression. They perform 100 gradient-descent iterations for each individual and show that this successfully prevents the loss of good structures by comparing the results with and without gradient-descent. Furthermore, they show that a bias towards model structures that are more readily adaptable is introduced and that their approach outperforms standard GP.

Wang et al. [48] apply a set of representation-specific local search operators to decision trees, for GP-based classification. The algorithm called Memetic GP (MGP) achieves better training quality but is more prone to overfitting due to the high intensity local search.

Lane et al. [26] use a different approach where they change the functions of internal tree nodes until fitness is improved. They test different strategies, according to several tree parsimony measures, when to apply this form of tuning and to which internal nodes. They report significant improvements in test quality over standard GP. These results are verified by Azad and Ryan [3] when integrating the tuning of internal tree nodes in GP.

Z-Flores et al. [50] use an alternative parameterization of GP trees, where a weight coefficient is assigned to each function node. They employ gradient descent [8] and test different strategies for the integration in GP. They find that best results are obtained when local search is applied to all individuals and that optimizing only a subset of the best individuals also represents a viable strategy. The same approach of adding and tuning weight coefficients of internal nodes is also evaluated in GP for binary classification [51], where the classification accuracy improved on all tested problems.

Juarez et al. [15] test the benefits of integrating local search in GP and neat-GP. While the performance in terms of quality does not increase significantly, they were able to produce consistently smaller and easier to interpret solutions. Trujillo et al. [44] also use gradient descent to optimize a percentage of individuals in each generation. They apply local search stochastically based on a probability given by tree size. They report substantial improvements in terms of solution quality, convergence speed and program size.

La Cava et al. [24] add an "epigenetic layer" as a mechanism for local search and test its effectiveness in solving symbolic regression and program synthesis problems. They attach a corresponding epigenome to each individual, which determines which genes are active in its structure. Epigenomes are altered by mutation and only changes that improve fitness or program complexity are accepted. The proposed approach is able to outperform GP in terms of fitness, exact solutions, and program sizes.

Castelli et al. [4] integrate local search in Geometric Semantic GP and test it on a number of real-world symbolic regression problems. They find that the resulting algorithm severely overfits and propose an alternative approach where local search is only applied in the first 50 generations of the evolutionary run.

Table 1 offers a detailed summary of previous approaches of local search in genetic programming. Gradient descent and hill climbing are prevalent local search methods, while Lamarckian evolution is the preferred local learning behavior. In the context of learning behavior, Lamarckian and Baldwinian learning refer to the way GP handles the information obtained via local search, which can be coded back into the genotype (Lamarckian) or not (Baldwinian). In both cases, local search affects the selection process and changes the behavior of the algorithm.

1.3 Scope of this study

Only few other works [15, 42, 50] referenced in our survey use gradient-based local search with GP for symbolic regression. We therefore consider it opportune to revisit

Table 1 Overview of local search methods used in GP

References	Local search method	Learning behavior	Experimental settings
Topchy and Punch [42]	Gradient descent	Baldwinian	Symbolic regression
			5 problems, 20 fitness cases
			100 individuals, 10 elites
			100 gradient descent iterations
			3×10^3 fitness evaluations
Krawiec et al. [22]	Greedy hill climbing	Lamarckian	Symbolic classification
			MNIST data
			50 individuals
			1 LS step applied to best individual
			20, 40, 60, 80, 100 generations
Wang et al. [48]	Hill climbing	Lamarckian	Symbolic classification
			10 UCI datasets
			100 individuals
			150 generations, dynamic termination
Lane et al. [26]	Hill climbing (probabilistic, exhaustive)	Lamarckian	Symbolic regression
			11 test problems
			500 individuals (with LS)
			1500 individuals (without LS)
			1.6×10^6 processed nodes limit
Azad and Ryan [3]	Hill climbing (probabilistic, exhaustive)	Lamarckian	Symbolic regression
			4 synthetic, 4 real-world problems
			500 individuals
			1.2×10^6 processed nodes limit
			7.5×10^4 fitness evaluations limit

Table 1 (continued)

References	Local search method	Learning behavior	Experimental settings
Z-Flores et al. [50]	Gradient descent (trust region)	Lamarckian	Symbolic regression (Matlab/GPLAB) 5 synthetic, 1 real-world problem 500 individuals 400 LS iterations 2×10^6 fitness evaluations limit (not counting LS iterations)
Juaréz-Smith and Trujillo [15]	Gradient descent (trust region)	Lamarckian	Symbolic regression (Python/neat-GP) 1 real-world problem Employs speciation and fitness sharing 100 individuals 10^5 fitness evaluations limit LS applied probabilistically ($p = 0.5$)
La Cava et al. [24]	Epigenetic hill climbing	Lamarckian	Symbolic regression (Age-fitness Pareto GP) Program synthesis (PushGP) 4 synthetic, 1 real-world problem 1000 individuals 2.5×10^{10} fitness evaluations limit
Castelli et al. [4]	Linear fit using SVD	Lamarckian	Symbolic regression (GSGP) 4 real-world test problems 250 individuals, 500 generations
Kommenda et al. (this work)	Gradient descent (trust region)	Lamarckian	Symbolic regression (GP and OSGP) 94 test problems 10^5 fitness evaluations limit (not counting local search iterations)

the topic and investigate additional aspects such as the effectiveness of local search, its convergent behavior and runtime impact on the evolutionary algorithm.

The main aim of this work is to provide a detailed treatment of gradient-based numerical optimization in the context of GP local search for symbolic regression. We follow the Lamarckian model where numerical coefficients optimized via nonlinear least squares are written back to the genotype. We investigate the benefits of nonlinear least squares optimization for two GP flavors (Standard GP and Offspring Selection GP [1]) and analyze performance in comparison with several other state-of-the-art methods on a large selection of benchmark problems [33].

2 Methodology

Our approach for parameter identification in symbolic regression combines automatic differentiation with gradient-based optimization [19, 20]. This approach is integrated as a local search mechanism in genetic programming and has been implemented in the open source framework for heuristic optimization HeuristicLab [47].

2.1 Mathematical formulation

For simplicity, we assume no implicit feature selection is performed and each GP model uses the entire set of input variables.

1. Let $X \in \mathbb{R}^{n \times m}$ be an $n \times m$ matrix where each column $x_i \in \mathbb{R}^n, i = 1, \ldots, m$ is an n-dimensional input variable and each row $s_j \in \mathbb{R}^m, j = 1, \ldots, n$ is an m-dimensional training sample. In what follows we take X as training data for the algorithm.
2. Let $y \in \mathbb{R}^n$ be the target vector for the regression problem.
3. Let $\mathscr{P}$ be the GP primitive set and S the syntactic search space defined by it.
4. Let $\boldsymbol{\Phi}$ be the space of possible expressions and their parameters. That is, the set of all tuples (E, θ), where $E \in S$ is a symbolic expression and $\theta \in \mathbb{R}^p$ a parameter vector of length p corresponding to the numerical parameters of E. Let us call a tuple (E, θ) a *symbolic expression model* $M_{E,\theta} \in \boldsymbol{\Phi}$.
5. Let $G : \boldsymbol{\Phi} \times \mathbb{R}^{n \times m} \rightarrow \mathbb{R}^n$ be a function that evaluates a model $M_{E,\theta} \in \boldsymbol{\Phi}$ on training data X and returns an n-dimensional output vector $\hat{y} \in \mathbb{R}^n$:

$$\hat{y} = G(M_{E,\theta}, X)$$

The overall symbolic regression goal is to find the optimal model $M_{opt} = (E_{opt}, \theta_{opt})$

$$M_{opt} = \underset{M_{E,\theta} \in \boldsymbol{\Phi}}{\arg\min} \frac{1}{2} \left\| G(M_{E,\theta}, X) - y \right\|^2 \tag{1}$$

During local search the model structure E remains fixed for a model $M_{E,\theta}$ while the parameter vector $\theta \in \mathbb{R}^p$ is subject to optimization. Thus, for the sake of simplicity we define the local search residual $H : \mathbb{R}^p \rightarrow \mathbb{R}^n$ as a function that evaluates

parameter vector θ and returns the difference between the model output and the actual target:

$$H(\theta) = G(M_{E,\theta}, X) - y$$

We formulate the goal of local search as a minimization problem:

$$\arg\min_{\theta \in \mathbb{R}^p} \frac{1}{2} \|H(\theta)\|^2 \tag{2}$$

To optimize θ over n training samples we consider the Jacobian $J(\theta)$ of $H(\theta)$:

$$J(\theta) = \begin{bmatrix} \frac{\partial h_1}{\partial \theta_1} & \cdots & \frac{\partial h_1}{\partial \theta_p} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_n}{\partial \theta_1} & \cdots & \frac{\partial h_n}{\partial \theta_p} \end{bmatrix} \tag{3}$$

For nonlinear least squares, at each iteration of the gradient descent algorithm we use the linearization

$$H(\theta + \Delta\theta) \approx H(\theta) + J(\theta)\Delta\theta \tag{4}$$

which leads to the following linear least squares problem:

$$\min_{\Delta\theta} \frac{1}{2} \|H(\theta) + J(\theta)\Delta\theta\|^2 \tag{5}$$

The problem described by Eq. (5) can be efficiently solved using trust region or line search methods [31].

2.2 Local search algorithm

Our implementation uses a trust region gradient descent approach, namely the Levenberg–Marquardt (LM) algorithm [27, 29] with an iteration limit as stopping criterion. Per default the LM algorithm is stopped after ten iterations.

Derivatives are calculated using automatic differentiation [12, 36], since the structure of the evaluated expressions is known and contains only arithmetic operations and elementary functions that can be derived using the chain rule.

We integrate linear scaling [16, 17] with local search in order to bring each estimate $\hat{y}$ in range with the target values y. The reason is to eliminate the need to find the correct offset and scale for the estimates, allowing GP to focus on finding the correct model structure.

Linear scaling is achieved in practice by introducing a structural extension such that a model $M_{E,\theta}$ is added as an input to a linear transformation block as illustrated in Fig. 1. The resulting expression will contain four additional nodes and its parameter vector θ will include two additional coefficients (the slope and intercept of the linear transformation).

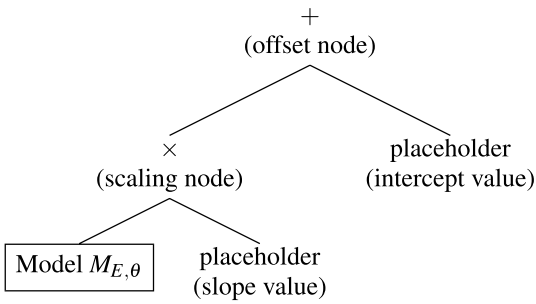

Fig. 1 Model $M_{E,\theta}$ extended with linear scaling nodes

These individual components linear scaling, gradient calculation, and nonlinear least squares optimization are the building blocks for parameter identification of symbolic regression models and the whole method is given as pseudo-code in Algorithm 1.

Algorithm 1: Parameter identification for symbolic regression.

```
   input : A symbolic expression tree T
   /* Prepare model and extract parameter vector                    */
1  M_{E,θ} ← prepare T for automatic differentiation and extract numerical coefficients;
2  Extend M_{E,θ} with linear transformation block (see Figure 1);
   /* Start Levenberg-Marquardt algorithm                           */
3  while Stopping criterion of LM not reached do
       /* Perform LM iteration                                      */
4      Compute residual H(θ);
5      Compute J(θ) by automatic differentiation;
6      Solve Equation 5;
7      Update parameter vector θ;
8  Write optimized numerical coefficients θ back to the corresponding nodes in T;
9  Calculate fitness of symbolic expression tree T;
```

Implementation-wise we prepare a model $M_{E,\theta}$ from a tree-based symbolic expression by first converting it to a data structure with which automatic differentiation can operate. The initial values for θ are extracted from the leaf nodes of the tree. At each iteration the LM algorithm computes the residuals $H(\theta)$ and the Jacobian $J(\theta)$ and updates θ accordingly.

Then the fitness of the symbolic expression tree is calculated, according to the objective function used by the algorithm solving the symbolic regression problem. This way, although the LM algorithm optimizes the mean squared error, another objective function (for example the coefficient of determination R^2 or the mean relative error) could be used to assess the fitness of the generated model. Finally, the optimized numerical values are written back to the symbolic expression tree according to the Lamarckian learning model.

Table 2 Progression of parameter values during optimization

	θ_1	θ_2	θ_3	θ_4
Iteration 0	− 74.544	− 52.262	− 12.887	79.309
Iteration 5	− 48.101	− 2.164	0.771	99.143
Iteration 10	− 31.826	1.516	1.766	106.164
Iteration 15	1.697	1.753	1.923	114.243
Iteration 20	81.793	1.245	0.990	93.486
Iteration 25	128.814	0.973	1.000	39.151
Iteration 30	29.671	1.000	1.000	9.903
Iteration 35	30.000	1.000	1.000	10.000

3 Analysis

We analyze the adaptation of numeric parameter values with respect to achieved improvement, convergence behavior and runtime overhead. We provide a detailed breakdown of measured execution time in relation with the number of training samples and the number of local optimization iterations.

First, we demonstrate the behavior of optimizing the parameters of a predefined model structure that matches the data generating function. The model structure (Eq. 6) contains three variables x_i and four parameters θ_i. The training data is generated using Eq. (6) with $\theta = \{30.0, 1.0, 1.0, 10.0\}$ by randomly sampling 300 data points in the interval [0.05, 2] for x_1 and x_3 and [1, 2] for x_2. This data generating procedure corresponds to the original problem definition given in [45]. We have chosen this particular synthetic problem because the parameters have a nonlinear effect in the model and several iterative steps are required to find the optimal values when using the LM algorithm. The elements of the initial vector θ are sampled uniformly from the interval $[-100, 100]$. Then, θ is iteratively optimized until no further improvement can be achieved.

$$M_{E,\theta} = \frac{\theta_1(x_1 - \theta_2)(x_3 - \theta_3)}{{x_2}^2(x_1 - \theta_4)} \tag{6}$$

Table 2 illustrates a successful application of the LM algorithm, where the parameters of the model are correctly identified. The first row 'Iteration 0' states the initial parameter values and each subsequent row the updated parameter vector θ. It can be observed that all parameters are simultaneously adapted according to the current gradient information. Furthermore, it could happen that the values deviate more from the target values compared to previous iterations (see for example θ_1 between Iteration 15 and 20), but in this example the correct values are identified after 35 iterations .

Appropriate starting values play a crucial role in the success of the LM algorithm. It requires a starting point within the basin of attraction for the global optimum otherwise it converges to a local optimum. Thus, the algorithm might fail to identify the optimal model parameters even when an optimal model structure is

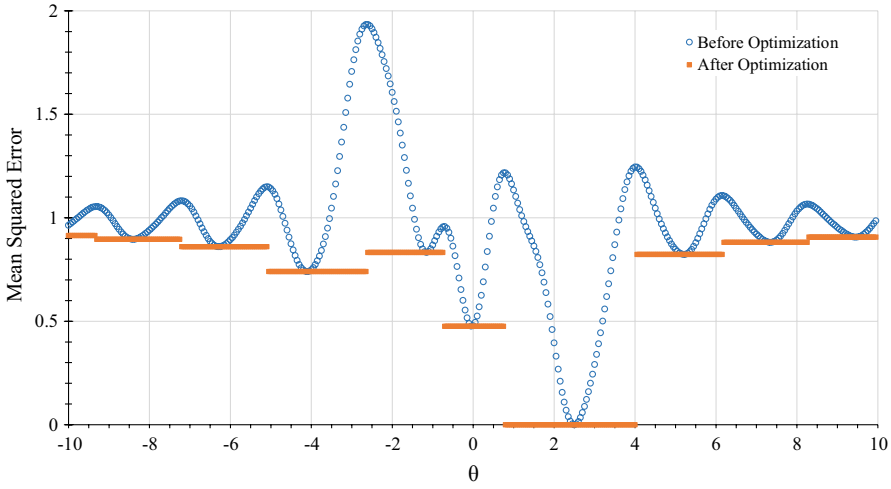

Fig. 2 Mean squared errors before and after LM optimization for $sin(\theta x)$ (Color figure online)

identified by GP. Model structures with multiple optima are particularly vulnerable to this phenomenon.

We illustrate the importance of good starting values by optimizing the model structure $M_{E,\theta} = sin(\theta x)$. In contrast with the previous example where θ was a parameter vector, here θ is a scalar value. This corresponds to identification of the frequency of a sinusoidal function. We generate a dataset containing 6000 samples for x ranging from -3.0 to 3.0 with a step size of 0.001 and a frequency $\theta = 2.5$.

We test different starting values of θ in the interval $[-10, 10]$ with a step size of 0.05 and plot the mean squared error between the generated data and the model outputs $sin(\theta x)$ before and after LM optimization. The results illustrated in Fig. 2 show that LM always improves the mean squared error. However, the global optimum is only reached if the starting value for θ lies in the interval [0.8, 4] and thus within the basin of attraction of the global optimum. Otherwise, the optimization converges towards a local optimum and a mean squared error of almost zero cannot be reached.

The largest drawback of the methodology is that it is computationally expensive to perform multiple gradient and function evaluations for each model structure. The LM algorithm has an asymptotic runtime complexity of $O(nd^2)$ with n the number of training samples and d the number of parameters for a fixed number of iterations. We empirically analyze its impact on the execution performance of GP using randomly created expression trees. Figure 3 shows the growth in runtime for the LM algorithm with an increasing number of maximum iterations and training samples. For this experiment we created 1000 symbolic expression trees with the probabilistic

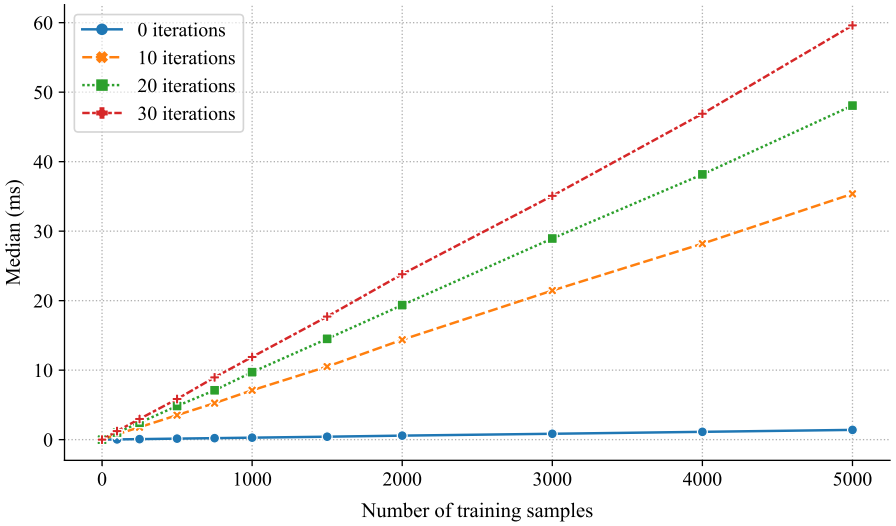

Fig. 3 Median execution time of parameter identification per tree with varying number of training samples and LM iterations ('0 iterations' indicates no parameter identification) (Color figure online)

tree creator (PTC 2) [28] and report the median execution time of the local search for an individual tree.[1]

As expected, the runtime increases linearly with the number of training samples for the different tested LM iterations. The number of training samples has the largest influence, as the training error has to be recalculated after each iteration of the LM algorithm. The number of iterations has a smaller influence on runtime since the LM algorithm may stop early when no more improvement can be achieved.

Compared to tree evaluation without local search (0 iterations), the accumulated overhead exceeds one order of magnitude for training data containing more than 1000 training samples. This effect (observed in numerous other publications) can be alleviated by applying parameter identification to smaller segments of the population, selected either probabilistically or according to fitness, or by using fewer training samples for parameter identification.

4 Experiments and results

We use the Penn Machine Learning Benchmarks (PMLB) collection of benchmark regression problems developed by Olson et al. [32] with the same choice of problems as in Orzechowski et al. [33] for evaluating the performance of local search

[1] Actual performance measurements are acquired using BenchmarkDotNet, a benchmarking framework providing an automated environment for performance analysis (https://benchmarkdotnet.org). Each test configuration has been run in an isolated project on an Intel® Core™ i7-8700 processor.

Table 3 Algorithm configuration of *GP* and *OSGP*

Parameter	Value
Function set	Unary functions (square, sqrt, exp, log)
	Binary functions ($+, -, \times$, unprotected $\div$)
Terminal set	*constant*, *weight* · *variable*
Max. tree length	* Selected via parameter tuning *
Max. evaluated solutions	100, 000
Tree initialization	Probabilistic tree creator (PTC2) [28]
Population size	* Selected via parameter tuning *
Selection GP	Tournament group size 2
Selection OSGP	Gender specific selection (proportional and random) [46]
Max. selection pressure OSGP	100
Crossover probability	100%
Crossover operator	Subtree crossover
Mutation probability	25%
Mutation operator	Change symbol, single-point, remove branch, replace branch
Fitness function	Pearson's R^2 correlation with the target
Evaluation budget	10^5 fitness evaluations
Local optimization iterations	10
Linear scaling	Enabled for all variants

within GP. The authors from [33] only consider problems with fewer than 3000 training samples, leaving a total of 94 problems in their benchmark suite.

We test two GP flavors: Standard GP and Offspring Selection GP [1] in both their standard version using linear scaling and hybridized with nonlinear least squares (NLS) parameter identification. This results in four configurations: *GP*, *GP NLS*, *OSGP* and *OSG NLS*. For each problem we have tuned the following parameters using grid search and fivefold cross-validation:

- Maximum tree length: 10, 25, 50, 75 nodes
- Population size: 100, 500, 1000 individuals
- Maximum generations: 100, 200, 1000 generations

A fivefold cross-validation is performed on the training data and repeated five times to account for stochastic effects. Hence, we have created 25 symbolic regression models (five times fivefold cross validation) that are trained on 4/5 folds of the training data and evaluated on the remaining fold. Afterwards, we select the best parameter settings for each algorithm and problem based on the average mean squared error on the fold not used for training obtained by these 25 generated models. Using the identified parameter settings we perform 50 repetitions of each algorithm on the whole training partition. The parameter settings for GP and OSGP, which have not been tuned, are detailed in Table 3.

This results are illustrated graphically in Fig. 4 for GP and Fig. 5 for OSGP, where each chart column (ordered ascending by the number of training samples)

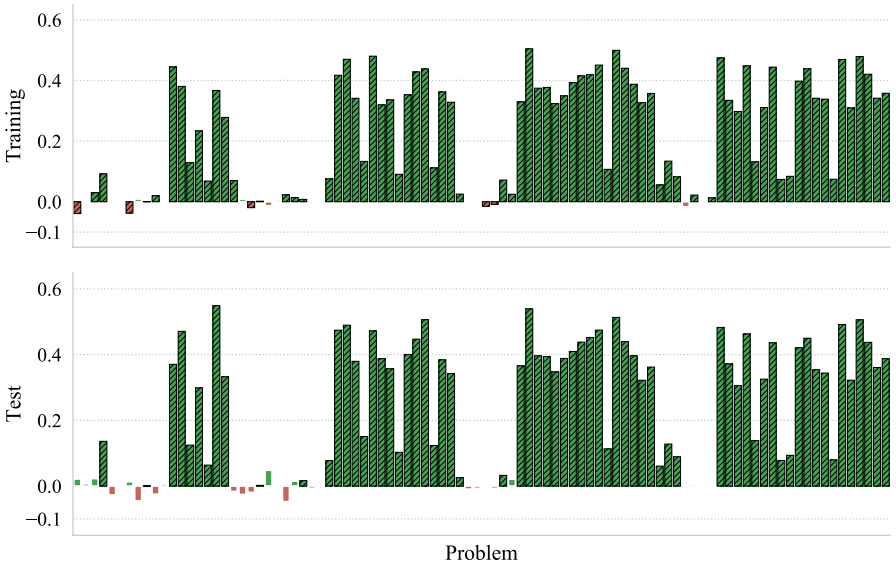

Fig. 4 Difference in median R^2 on the training and test set between *GP NLS* and *GP* on all problems sorted by number of samples. A positive difference indicates that *GP NLS* performs better and is highlighted in green and vice versa a negative difference is highlighted in red (Color figure online)

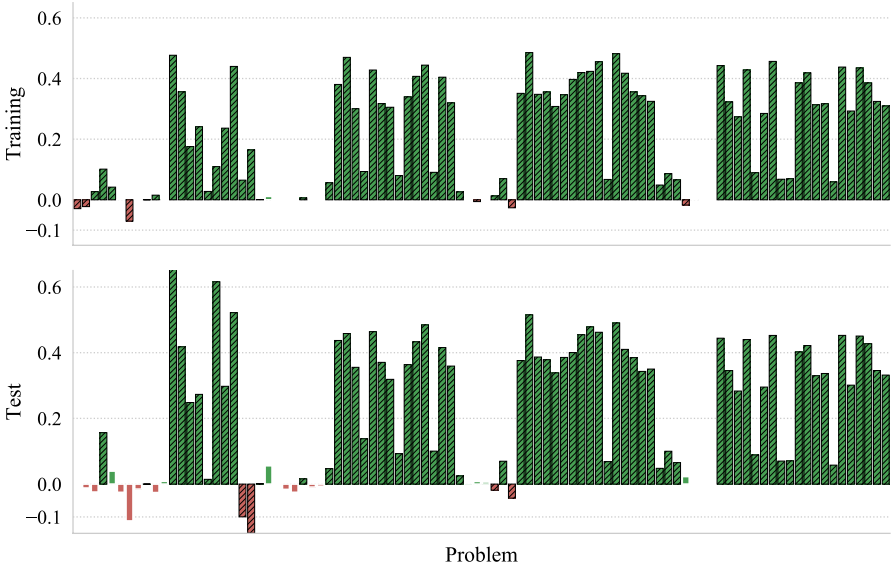

Fig. 5 Difference in median R^2 on the training and test set between *OSGP NLS* and *OSGP* on all problems sorted by number of samples. A positive difference indicates that *OSGP NLS* performs better and is highlighted in green and vice versa a negative difference is highlighted in red (Color figure online)

Table 4 Performance summary of *GP NLS* compared to *GP* and *OSGP NLS* compared to *OSGP*

	Training			Test		
	+	−	≈	+	−	≈
GP NLS	75	5	14	67	0	27
OSGP NLS	74	6	14	68	4	22

Table 5 Machine learning methods used for comparison taken from Orzechowski et al. [33]

Name	Description
GSGP	Geometric Semantic Genetic Programming
AFP	Age-fitness Pareto Optimization
MRGP	Multiple Regression Genetic Programming
EPLEX	ε-Lexicase Selection
XGBoost	Extreme Gradient Boosting
GradBoost	Gradient Boosting Regression
MLP	Multilayer Perceptrons (Neural Network) Regression
RF	Random Forest Regression
KernelRidge	Kernel Ridge Regression
AdaBoost	Adaptive Boosting Regression
Lasso	Least-angle Regression with Lasso
LinSVR	Liner Support Vector Regression
LinReg	Linear Regression
SGD	Stochastic Gradient Descent Regression

corresponds to a tested problem. These figures show the difference in the median R^2 between the local search variant and the standard genetic programming variant. The color green represents a positive difference in favor of the *NLS* hybridization while the color red represents a negative difference. Column hatching indicates statistical significance calculated using a two-sided Wilcoxon rank-sum test ($\alpha = 0.05$). Overall it can be observed that hybridization with local search drastically improves the modeling accuracy.

A summary of the overall results is given in Table 4, where it can be seen that both *GP NLS* and *OSGP NLS* produce statistically-better results for the majority of problems, for both training and test data. For an easier comparison we use symbols $+, -$ to denote a statistically-significant performance difference between the *NLS* hybridization and the baseline variant.

The detailed raw data containing the results of each algorithm repetition is available for download at the HeuristicLab homepage.[2] Aggregated results for each problem are given in Appendix Tables 6 and 7 as median $R^2 \pm$ interquartile range on the training and test data. The last column in each table illustrates how *GP NLS* and *OSGP NLS* compare against their standard counterparts.

We then perform a large-scale comparison between our GP variants and several symbolic regression methods tested by Orzechowski et al. [33] and summarized in

[2] https://dev.heuristiclab.com/trac.fcgi/wiki/AdditionalMaterial.

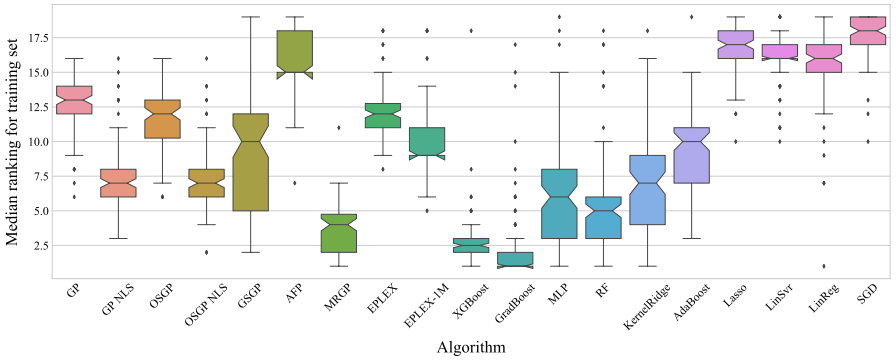

Fig. 6 Algorithm ranking based on the MSE scores on the training set (Color figure online)

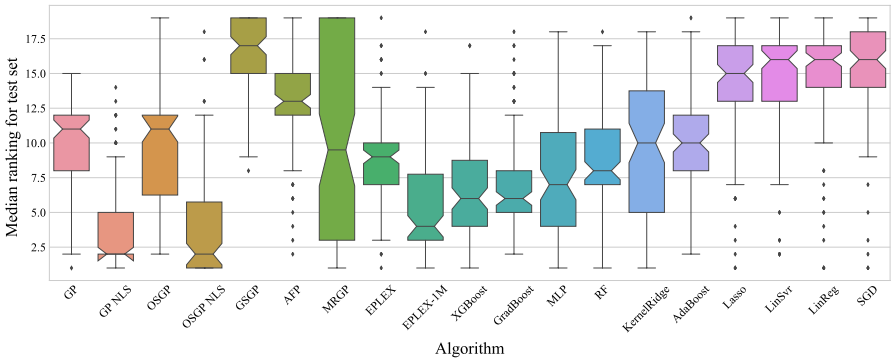

Fig. 7 Algorithm ranking based on the MSE scores on the test set (Color figure online)

Table 5. We follow the original methodology in [33] and calculate algorithm ranks on the tested problems based on median mean squared error.

The experimental setup has been intentionally chosen as similar as possible to the benchmarking study [33] so that we can compare the results of our methods with already published ones. Therefore, we ranked the performance of all algorithms based on the median mean squared errors. However, we only created new results for *GP*, *GP NLS*, *OSGP* and *OSPG NLS* and reused the publicly available results and scripts for analysis and visualization provided by the Epistasis Lab at the University of Pennsylvania.[3] Box plots showing the rank distribution for each algorithm across all problems are shown in Fig. 6 for the training set and in Fig. 7 for the test set.

We assess performance on the basis of training and test median ranks, and we compare the significance of the problem rankings obtained by each algorithm using the Kruskal test [23] to ascertain that the calculated rank medians are statistically different and Dunn's test [9] with Holm–Bonferroni adjustments, to ascertain whether the pairwise rank differences between the tested algorithms are

[3] https://github.com/EpistasisLab/regression-benchmark.

statistically significant. The calculated p values are shown in Appendix Tables 8 and 9.

In terms of training performance, both *GP* and *OSGP* algorithms fall behind all other GP variants except *AFP*. At the same time, neither *GP NLS* and *OSGP NLS* distinguish themselves as top performers on this benchmark set as they are outranked by *MRGP*, *XGBoost* and *GradBoost*.

However, in terms of generalization ability (test performance), the results show that *GP NLS* is tied with *OSGP NLS*, *EPLEX-1M* and *XGBoost* and outperforms all other methods. In turn, *OSGP NLS* is tied with *GP NLS* and *EPLEX-1M* and outperforms all other methods. Overall, the proposed local search hybridization is able to significantly improve the generalization ability of algorithms.

5 Conclusion

Hybridization with local search represents a particularly effective approach for improving GP performance on a wide array of symbolic regression and symbolic classification. Many studies in the literature report substantial benefits in terms of solution quality, convergence speed and model size, with the added cost of increased running time caused by additional evaluations required by the local improvement procedure.

In this work we described in detail one such hybridization using GP and Offspring Selection GP for the evolution of model structure and the Levenberg–Marquardt algorithm for optimization of numerical parameters for symbolic regression. Implementation-wise, it is straight-forward to integrate linear scaling by extending the model parameter vector with two additional coefficients for scale and intercept. In general the approach works remarkably well on the suite of benchmark problems that we used for testing regardless of the GP variant employed. However, local search may get trapped in local optima, as we exemplified on the problem of frequency detection for a trigonometric function.

Our testing indicates improved generalization as the main benefit of this approach. In terms of training quality *GP NLS* and *OSGP NLS* rank behind MRGP (Multiple Regression Genetic Programming), However, on the test set *OSGP NLS* produces on average the best predictions among all GP variants.

As future work in this area we plan to investigate in more detail the correspondence between local search effort (in terms of gradient descent iterations) and algorithm performance within the Lamarckian framework. We conjecture that a more efficient search model can be attained by tuning the balance between local and global search effort.

Acknowledgements Open access funding provided by University of Applied Sciences Upper Austria. We would especially like to thank the researchers of the Epistasis Lab at the University of Pennsylvania for collecting the benchmark suite and providing their results for comparison as well as the accompanying analysis scripts.

Appendix

See Tables 6, 7, 8 and 9.

Table 6 GP result comparison

Problem	GP		GP NLS		Comparison	
	R^2 (train)	R^2 (test)	R^2 (train)	R^2 (test)	Train	Test
1089_USCrime	0.925 ± 0.028	0.728 ± 0.262	0.887 ± 0.038	0.749 ± 0.166	−	≈
659_sleuth_ex1714	0.884 ± 0.022	0.618 ± 0.297	0.880 ± 0.025	0.624 ± 0.385	≈	≈
485_analcatdata_vehicle	0.689 ± 0.042	0.591 ± 0.153	0.719 ± 0.032	0.614 ± 0.127	+	≈
1096_FacultySalaries	0.897 ± 0.021	0.834 ± 0.180	0.989 ± 0.012	0.970 ± 0.111	+	+
192_vineyard	0.784 ± 0.060	0.541 ± 0.601	0.785 ± 0.072	0.514 ± 0.527	≈	≈
228_elusage	0.847 ± 0.034	0.781 ± 0.120	0.848 ± 0.036	0.779 ± 0.130	≈	≈
542_pollution	0.747 ± 0.040	0.399 ± 0.379	0.709 ± 0.061	0.412 ± 0.282	−	≈
687_sleuth_ex1605	0.626 ± 0.052	0.569 ± 0.156	0.635 ± 0.048	0.524 ± 0.207	≈	≈
527_analcatdata_election2000	1.000 ± 0.000	0.999 ± 0.001	1.000 ± 0.000	1.000 ± 0.000	+	+
706_sleuth_case1202	0.750 ± 0.038	0.682 ± 0.137	0.770 ± 0.040	0.657 ± 0.154	+	≈
523_analcatdata_neavote	0.954 ± 0.012	0.935 ± 0.051	0.954 ± 0.012	0.940 ± 0.039	≈	≈
591_fri_c1_100_10	0.418 ± 0.040	0.322 ± 0.289	0.862 ± 0.066	0.693 ± 0.234	+	+
594_fri_c2_100_5	0.603 ± 0.032	0.433 ± 0.253	0.984 ± 0.006	0.904 ± 0.128	+	+
611_fri_c3_100_5	0.769 ± 0.038	0.677 ± 0.132	0.898 ± 0.049	0.802 ± 0.115	+	+
621_fri_c0_100_10	0.739 ± 0.045	0.636 ± 0.169	0.973 ± 0.005	0.935 ± 0.032	+	+
624_fri_c0_100_5	0.905 ± 0.030	0.868 ± 0.073	0.974 ± 0.005	0.932 ± 0.031	+	+
634_fri_c2_100_10	0.495 ± 0.041	0.130 ± 0.313	0.862 ± 0.066	0.679 ± 0.242	+	+
651_fri_c0_100_25	0.698 ± 0.051	0.600 ± 0.186	0.976 ± 0.005	0.932 ± 0.071	+	+
656_fri_c1_100_5	0.446 ± 0.045	0.335 ± 0.267	0.516 ± 0.083	0.318 ± 0.309	+	≈
210_cloud	0.890 ± 0.031	0.821 ± 0.243	0.899 ± 0.041	0.796 ± 0.280	≈	≈
678_visualizing_environmental	0.452 ± 0.058	0.272 ± 0.218	0.432 ± 0.050	0.253 ± 0.190	−	≈
663_rabe_266	0.998 ± 0.001	0.997 ± 0.002	1.000 ± 0.000	0.999 ± 0.002	+	+
665_sleuth_case2002	0.432 ± 0.080	0.290 ± 0.334	0.419 ± 0.058	0.338 ± 0.248	≈	≈
195_auto_price	0.877 ± 0.022	0.821 ± 0.120	0.882 ± 0.022	0.824 ± 0.141	≈	≈
207_autoPrice	0.862 ± 0.022	0.825 ± 0.087	0.885 ± 0.024	0.778 ± 0.161	+	≈
230_machine_cpu	0.928 ± 0.022	0.854 ± 0.136	0.942 ± 0.012	0.869 ± 0.124	+	≈
561_cpu	0.992 ± 0.005	0.979 ± 0.039	0.999 ± 0.000	0.995 ± 0.011	+	+
712_chscase_geyser1	0.785 ± 0.019	0.770 ± 0.053	0.787 ± 0.014	0.763 ± 0.060	≈	≈

Table 6 (continued)

Problem	GP		GP NLS		Comparison	
	R^2 (train)	R^2 (test)	R^2 (train)	R^2 (test)	Train	Test
695_chatfield_4	0.887 ± 0.015	0.859 ± 0.047	0.888 ± 0.014	0.855 ± 0.051	≈	≈
579_fri_c0_250_5	0.889 ± 0.020	0.872 ± 0.050	0.964 ± 0.004	0.949 ± 0.017	+	+
596_fri_c2_250_5	0.552 ± 0.025	0.478 ± 0.167	0.969 ± 0.015	0.952 ± 0.038	+	+
601_fri_c1_250_5	0.514 ± 0.027	0.482 ± 0.201	0.984 ± 0.003	0.972 ± 0.009	+	+
602_fri_c3_250_10	0.641 ± 0.036	0.583 ± 0.137	0.982 ± 0.005	0.963 ± 0.013	+	+
603_fri_c0_250_50	0.836 ± 0.042	0.797 ± 0.097	0.969 ± 0.003	0.947 ± 0.017	+	+
605_fri_c2_250_25	0.492 ± 0.023	0.459 ± 0.165	0.972 ± 0.013	0.931 ± 0.043	+	+
613_fri_c3_250_5	0.659 ± 0.030	0.569 ± 0.161	0.978 ± 0.003	0.956 ± 0.020	+	+
615_fri_c4_250_10	0.642 ± 0.042	0.586 ± 0.158	0.978 ± 0.009	0.943 ± 0.056	+	+
635_fri_c0_250_10	0.872 ± 0.041	0.847 ± 0.095	0.962 ± 0.004	0.949 ± 0.010	+	+
644_fri_c4_250_25	0.620 ± 0.041	0.547 ± 0.144	0.972 ± 0.012	0.947 ± 0.026	+	+
647_fri_c1_250_10	0.550 ± 0.031	0.515 ± 0.142	0.978 ± 0.010	0.962 ± 0.018	+	+
648_fri_c1_250_50	0.536 ± 0.040	0.446 ± 0.229	0.973 ± 0.010	0.952 ± 0.018	+	+
653_fri_c0_250_25	0.847 ± 0.062	0.820 ± 0.070	0.958 ± 0.005	0.944 ± 0.015	+	+
657_fri_c2_250_10	0.603 ± 0.024	0.558 ± 0.118	0.965 ± 0.017	0.942 ± 0.039	+	+
658_fri_c3_250_25	0.641 ± 0.027	0.583 ± 0.146	0.969 ± 0.016	0.925 ± 0.144	+	+
690_visualizing_galaxy	0.954 ± 0.022	0.947 ± 0.030	0.979 ± 0.003	0.973 ± 0.011	+	+
519_vinnie	0.759 ± 0.015	0.726 ± 0.056	0.763 ± 0.015	0.717 ± 0.050	≈	≈
556_analcatdata_apnea2	0.900 ± 0.020	0.878 ± 0.056	0.904 ± 0.022	0.871 ± 0.072	≈	≈
557_analcatdata_apnea1	0.923 ± 0.014	0.861 ± 0.115	0.908 ± 0.016	0.864 ± 0.081	−	≈
1027_ESL	0.883 ± 0.008	0.850 ± 0.033	0.874 ± 0.011	0.843 ± 0.036	−	≈
522_pm10	0.252 ± 0.035	0.189 ± 0.072	0.324 ± 0.045	0.221 ± 0.111	+	+
547_no2	0.542 ± 0.034	0.485 ± 0.084	0.567 ± 0.023	0.506 ± 0.077	+	≈
581_fri_c3_500_25	0.637 ± 0.021	0.593 ± 0.094	0.967 ± 0.020	0.959 ± 0.024	+	+
582_fri_c1_500_25	0.475 ± 0.031	0.436 ± 0.124	0.979 ± 0.015	0.975 ± 0.020	+	+
584_fri_c4_500_25	0.602 ± 0.050	0.569 ± 0.110	0.977 ± 0.017	0.965 ± 0.029	+	+
597_fri_c2_500_5	0.603 ± 0.030	0.579 ± 0.075	0.980 ± 0.004	0.973 ± 0.011	+	+
604_fri_c4_500_10	0.654 ± 0.022	0.623 ± 0.075	0.978 ± 0.006	0.971 ± 0.012	+	+
616_fri_c4_500_50	0.615 ± 0.029	0.557 ± 0.106	0.964 ± 0.022	0.946 ± 0.043	+	+
617_fri_c3_500_5	0.586 ± 0.032	0.562 ± 0.094	0.979 ± 0.005	0.972 ± 0.007	+	+
626_fri_c2_500_50	0.561 ± 0.030	0.526 ± 0.104	0.976 ± 0.017	0.964 ± 0.020	+	+
627_fri_c2_500_10	0.559 ± 0.025	0.512 ± 0.133	0.978 ± 0.008	0.964 ± 0.043	+	+
631_fri_c1_500_5	0.529 ± 0.044	0.497 ± 0.113	0.980 ± 0.004	0.971 ± 0.008	+	+
633_fri_c0_500_25	0.856 ± 0.071	0.844 ± 0.064	0.963 ± 0.003	0.956 ± 0.010	+	+
637_fri_c1_500_50	0.467 ± 0.034	0.443 ± 0.092	0.966 ± 0.018	0.956 ± 0.022	+	+
641_fri_c1_500_10	0.539 ± 0.025	0.535 ± 0.115	0.979 ± 0.006	0.974 ± 0.009	+	+
643_fri_c2_500_25	0.588 ± 0.032	0.570 ± 0.086	0.975 ± 0.011	0.967 ± 0.013	+	+
645_fri_c3_500_50	0.624 ± 0.043	0.620 ± 0.099	0.951 ± 0.026	0.942 ± 0.038	+	+
646_fri_c3_500_10	0.623 ± 0.031	0.611 ± 0.098	0.980 ± 0.008	0.973 ± 0.010	+	+
649_fri_c0_500_5	0.907 ± 0.013	0.894 ± 0.022	0.962 ± 0.002	0.955 ± 0.008	+	+

Table 6 (continued)

Problem	GP		GP NLS		Comparison	
	R^2 (train)	R^2 (test)	R^2 (train)	R^2 (test)	Train	Test
650_fri_c0_500_50	0.827 ± 0.039	0.827 ± 0.062	0.961 ± 0.004	0.955 ± 0.008	+	+
654_fri_c0_500_10	0.876 ± 0.026	0.863 ± 0.031	0.958 ± 0.003	0.951 ± 0.011	+	+
666_rmftsa_ladata	0.685 ± 0.044	0.675 ± 0.140	0.669 ± 0.049	0.675 ± 0.145	≈	≈
1028_SWD	0.437 ± 0.020	0.380 ± 0.045	0.459 ± 0.014	0.385 ± 0.060	+	≈
1029_LEV	0.569 ± 0.026	0.558 ± 0.083	0.569 ± 0.026	0.559 ± 0.083	≈	≈
1030_ERA	0.391 ± 0.019	0.360 ± 0.049	0.403 ± 0.016	0.363 ± 0.053	+	≈
583_fri_c1_1000_50	0.501 ± 0.031	0.483 ± 0.063	0.975 ± 0.017	0.966 ± 0.024	+	+
586_fri_c3_1000_25	0.644 ± 0.027	0.601 ± 0.089	0.978 ± 0.007	0.973 ± 0.011	+	+
588_fri_c4_1000_100	0.679 ± 0.034	0.665 ± 0.069	0.976 ± 0.012	0.971 ± 0.017	+	+
589_fri_c2_1000_25	0.531 ± 0.031	0.513 ± 0.090	0.979 ± 0.006	0.976 ± 0.011	+	+
590_fri_c0_1000_50	0.833 ± 0.074	0.821 ± 0.077	0.964 ± 0.002	0.959 ± 0.006	+	+
592_fri_c4_1000_25	0.657 ± 0.029	0.637 ± 0.060	0.968 ± 0.015	0.962 ± 0.020	+	+
593_fri_c1_1000_10	0.535 ± 0.031	0.541 ± 0.055	0.979 ± 0.005	0.977 ± 0.009	+	+
595_fri_c0_1000_10	0.883 ± 0.010	0.875 ± 0.030	0.956 ± 0.002	0.953 ± 0.006	+	+
598_fri_c0_1000_25	0.880 ± 0.069	0.867 ± 0.077	0.964 ± 0.001	0.960 ± 0.004	+	+
599_fri_c2_1000_5	0.582 ± 0.021	0.556 ± 0.056	0.980 ± 0.004	0.977 ± 0.007	+	+
606_fri_c2_1000_10	0.536 ± 0.021	0.521 ± 0.056	0.974 ± 0.007	0.971 ± 0.011	+	+
607_fri_c4_1000_50	0.635 ± 0.033	0.617 ± 0.075	0.977 ± 0.012	0.971 ± 0.012	+	+
608_fri_c3_1000_10	0.640 ± 0.024	0.630 ± 0.055	0.979 ± 0.006	0.974 ± 0.010	+	+
609_fri_c0_1000_5	0.888 ± 0.009	0.881 ± 0.022	0.963 ± 0.001	0.960 ± 0.004	+	+
612_fri_c1_1000_5	0.509 ± 0.022	0.483 ± 0.057	0.979 ± 0.005	0.974 ± 0.005	+	+
618_fri_c3_1000_50	0.665 ± 0.035	0.648 ± 0.068	0.974 ± 0.012	0.970 ± 0.017	+	+
620_fri_c1_1000_25	0.496 ± 0.024	0.464 ± 0.077	0.975 ± 0.013	0.970 ± 0.014	+	+
622_fri_c2_1000_50	0.555 ± 0.031	0.532 ± 0.062	0.976 ± 0.015	0.969 ± 0.020	+	+
623_fri_c4_1000_10	0.637 ± 0.027	0.612 ± 0.101	0.978 ± 0.007	0.973 ± 0.009	+	+
628_fri_c3_1000_5	0.621 ± 0.024	0.587 ± 0.069	0.979 ± 0.007	0.974 ± 0.012	+	+

R^2 values reported as median ± interquartile range. Symbols $+, -, \approx$ denote statistically-better, statistically-worse and statistically-insignificant differences between *GP NLS* and *GP* using a two-sided Wilcoxon rank-sum test ($\alpha = 0.05$)

Table 7 OSGP result comparison

Problem	OSGP		OSGP NLS		Comparison	
	R^2 (train)	R^2 (test)	R^2 (train)	R^2 (test)	Train	Test
1089_USCrime	0.927 ± 0.025	0.772 ± 0.176	0.898 ± 0.029	0.769 ± 0.180	−	≈
659_sleuth_ex1714	0.915 ± 0.019	0.642 ± 0.509	0.893 ± 0.029	0.630 ± 0.217	−	≈
485_analcatdata_vehicle	0.701 ± 0.049	0.609 ± 0.135	0.728 ± 0.035	0.585 ± 0.134	+	≈
1096_FacultySalaries	0.899 ± 0.023	0.828 ± 0.187	1.000 ± 0.000	0.984 ± 0.297	+	+
192_vineyard	0.795 ± 0.044	0.477 ± 0.697	0.837 ± 0.056	0.517 ± 0.531	+	≈
228_elusage	0.849 ± 0.034	0.780 ± 0.135	0.849 ± 0.034	0.755 ± 0.135	≈	≈
542_pollution	0.801 ± 0.049	0.436 ± 0.422	0.731 ± 0.047	0.324 ± 0.370	−	≈
687_sleuth_ex1605	0.635 ± 0.049	0.562 ± 0.173	0.640 ± 0.051	0.547 ± 0.171	≈	≈
527_analcatdata_election2000	1.000 ± 0.000	1.000 ± 0.003	1.000 ± 0.000	1.000 ± 0.000	+	+
706_sleuth_case1202	0.757 ± 0.042	0.685 ± 0.135	0.773 ± 0.036	0.659 ± 0.160	+	≈
523_analcatdata_neavote	0.954 ± 0.012	0.933 ± 0.061	0.954 ± 0.012	0.942 ± 0.046	≈	≈
591_fri_c1_100_10	0.462 ± 0.027	0.205 ± 0.375	0.938 ± 0.020	0.859 ± 0.131	+	+
594_fri_c2_100_5	0.613 ± 0.020	0.477 ± 0.241	0.969 ± 0.011	0.895 ± 0.153	+	+
611_fri_c3_100_5	0.781 ± 0.028	0.660 ± 0.172	0.956 ± 0.011	0.909 ± 0.060	+	+
621_fri_c0_100_10	0.742 ± 0.040	0.633 ± 0.176	0.983 ± 0.005	0.906 ± 0.061	+	+
624_fri_c0_100_5	0.925 ± 0.016	0.890 ± 0.041	0.953 ± 0.011	0.905 ± 0.051	+	+
634_fri_c2_100_10	0.565 ± 0.024	-0.092 ± 0.987	0.674 ± 0.065	0.524 ± 0.249	+	+
651_fri_c0_100_25	0.710 ± 0.046	0.600 ± 0.218	0.947 ± 0.018	0.898 ± 0.072	+	+
656_fri_c1_100_5	0.487 ± 0.059	0.315 ± 0.347	0.927 ± 0.024	0.837 ± 0.104	+	+
210_cloud	0.890 ± 0.036	0.818 ± 0.215	0.955 ± 0.010	0.718 ± 0.433	+	−
678_visualizing_environmental	0.425 ± 0.048	0.274 ± 0.209	0.590 ± 0.041	0.125 ± 0.521	+	−
663_rabe_266	0.998 ± 0.000	0.998 ± 0.001	1.000 ± 0.000	0.999 ± 0.000	+	+
665_sleuth_case2002	0.431 ± 0.084	0.230 ± 3.251	0.442 ± 0.059	0.286 ± 0.235	≈	≈
195_auto_price	0.864 ± 0.023	0.829 ± 0.076	0.863 ± 0.023	0.828 ± 0.100	≈	≈
207_autoPrice	0.866 ± 0.023	0.841 ± 0.121	0.865 ± 0.018	0.824 ± 0.094	≈	≈
230_machine_cpu	0.944 ± 0.016	0.893 ± 0.089	0.944 ± 0.010	0.868 ± 0.127	≈	≈
561_cpu	0.993 ± 0.003	0.981 ± 0.021	0.999 ± 0.000	0.997 ± 0.017	+	+
712_chscase_geyser1	0.786 ± 0.017	0.770 ± 0.060	0.788 ± 0.018	0.761 ± 0.068	≈	≈
695_chatfield_4	0.889 ± 0.016	0.857 ± 0.059	0.890 ± 0.013	0.851 ± 0.055	≈	≈
579_fri_c0_250_5	0.907 ± 0.032	0.902 ± 0.047	0.963 ± 0.004	0.949 ± 0.016	+	+
596_fri_c2_250_5	0.604 ± 0.027	0.539 ± 0.155	0.985 ± 0.003	0.976 ± 0.013	+	+
601_fri_c1_250_5	0.509 ± 0.021	0.509 ± 0.166	0.979 ± 0.005	0.968 ± 0.015	+	+
602_fri_c3_250_10	0.679 ± 0.038	0.609 ± 0.134	0.980 ± 0.004	0.965 ± 0.015	+	+
603_fri_c0_250_50	0.874 ± 0.063	0.812 ± 0.101	0.967 ± 0.003	0.950 ± 0.012	+	+
605_fri_c2_250_25	0.533 ± 0.038	0.459 ± 0.164	0.961 ± 0.016	0.923 ± 0.028	+	+
613_fri_c3_250_5	0.662 ± 0.030	0.594 ± 0.134	0.979 ± 0.003	0.965 ± 0.018	+	+
615_fri_c4_250_10	0.654 ± 0.038	0.601 ± 0.122	0.960 ± 0.017	0.920 ± 0.052	+	+
635_fri_c0_250_10	0.883 ± 0.019	0.855 ± 0.040	0.963 ± 0.003	0.948 ± 0.012	+	+
644_fri_c4_250_25	0.618 ± 0.037	0.567 ± 0.100	0.958 ± 0.019	0.931 ± 0.034	+	+
647_fri_c1_250_10	0.560 ± 0.026	0.523 ± 0.165	0.968 ± 0.015	0.956 ± 0.025	+	+

Table 7 (continued)

Problem	OSGP		OSGP NLS		Comparison	
	R^2 (train)	R^2 (test)	R^2 (train)	R^2 (test)	Train	Test
648_fri_c1_250_50	0.520 ± 0.023	0.458 ± 0.206	0.965 ± 0.011	0.943 ± 0.034	+	+
653_fri_c0_250_25	0.871 ± 0.047	0.843 ± 0.063	0.962 ± 0.004	0.944 ± 0.014	+	+
657_fri_c2_250_10	0.575 ± 0.023	0.550 ± 0.166	0.979 ± 0.005	0.966 ± 0.012	+	+
658_fri_c3_250_25	0.641 ± 0.027	0.569 ± 0.185	0.961 ± 0.014	0.928 ± 0.058	+	+
690_visualizing_galaxy	0.953 ± 0.013	0.949 ± 0.015	0.979 ± 0.003	0.974 ± 0.007	+	+
519_vinnie	0.762 ± 0.015	0.721 ± 0.055	0.763 ± 0.016	0.722 ± 0.058	≈	≈
556_analcatdata_apnea2	0.905 ± 0.020	0.876 ± 0.075	0.899 ± 0.015	0.884 ± 0.058	−	≈
557_analcatdata_apnea1	0.910 ± 0.019	0.878 ± 0.066	0.907 ± 0.015	0.885 ± 0.069	≈	≈
1027_ESL	0.884 ± 0.007	0.854 ± 0.026	0.896 ± 0.007	0.836 ± 0.074	+	−
522_pm10	0.275 ± 0.030	0.207 ± 0.092	0.344 ± 0.027	0.277 ± 0.110	+	+
547_no2	0.542 ± 0.026	0.492 ± 0.069	0.516 ± 0.032	0.449 ± 0.099	−	−
581_fri_c3_500_25	0.623 ± 0.022	0.588 ± 0.068	0.974 ± 0.015	0.965 ± 0.023	+	+
582_fri_c1_500_25	0.490 ± 0.025	0.456 ± 0.091	0.975 ± 0.010	0.971 ± 0.011	+	+
584_fri_c4_500_25	0.625 ± 0.031	0.581 ± 0.069	0.973 ± 0.009	0.968 ± 0.016	+	+
597_fri_c2_500_5	0.627 ± 0.020	0.600 ± 0.094	0.982 ± 0.002	0.978 ± 0.009	+	+
604_fri_c4_500_10	0.673 ± 0.021	0.635 ± 0.107	0.981 ± 0.004	0.973 ± 0.012	+	+
616_fri_c4_500_50	0.610 ± 0.032	0.553 ± 0.073	0.957 ± 0.029	0.938 ± 0.039	+	+
617_fri_c3_500_5	0.582 ± 0.031	0.574 ± 0.070	0.979 ± 0.003	0.974 ± 0.008	+	+
626_fri_c2_500_50	0.556 ± 0.023	0.514 ± 0.129	0.976 ± 0.007	0.969 ± 0.014	+	+
627_fri_c2_500_10	0.559 ± 0.028	0.498 ± 0.138	0.983 ± 0.002	0.977 ± 0.016	+	+
631_fri_c1_500_5	0.525 ± 0.020	0.512 ± 0.099	0.980 ± 0.002	0.975 ± 0.004	+	+
633_fri_c0_500_25	0.896 ± 0.042	0.888 ± 0.046	0.963 ± 0.002	0.956 ± 0.009	+	+
637_fri_c1_500_50	0.489 ± 0.026	0.476 ± 0.106	0.971 ± 0.014	0.966 ± 0.017	+	+
641_fri_c1_500_10	0.566 ± 0.031	0.568 ± 0.096	0.983 ± 0.004	0.978 ± 0.008	+	+
643_fri_c2_500_25	0.618 ± 0.025	0.579 ± 0.120	0.974 ± 0.009	0.964 ± 0.015	+	+
645_fri_c3_500_50	0.612 ± 0.028	0.595 ± 0.075	0.955 ± 0.031	0.938 ± 0.037	+	+
646_fri_c3_500_10	0.656 ± 0.021	0.624 ± 0.103	0.981 ± 0.005	0.974 ± 0.010	+	+
649_fri_c0_500_5	0.913 ± 0.022	0.907 ± 0.029	0.962 ± 0.002	0.955 ± 0.007	+	+
650_fri_c0_500_50	0.876 ± 0.068	0.856 ± 0.075	0.962 ± 0.002	0.956 ± 0.009	+	+
654_fri_c0_500_10	0.892 ± 0.031	0.886 ± 0.037	0.958 ± 0.003	0.951 ± 0.011	+	+
666_rmftsa_ladata	0.684 ± 0.045	0.647 ± 0.179	0.667 ± 0.047	0.669 ± 0.143	−	≈
1028_SWD	0.449 ± 0.013	0.383 ± 0.045	0.449 ± 0.017	0.384 ± 0.048	≈	≈
1029_LEV	0.568 ± 0.027	0.558 ± 0.081	0.569 ± 0.026	0.559 ± 0.086	≈	≈
1030_ERA	0.395 ± 0.018	0.363 ± 0.056	0.389 ± 0.018	0.359 ± 0.053	≈	≈
583_fri_c1_1000_50	0.519 ± 0.038	0.512 ± 0.084	0.961 ± 0.022	0.956 ± 0.025	+	+
586_fri_c3_1000_25	0.658 ± 0.025	0.632 ± 0.070	0.981 ± 0.003	0.977 ± 0.007	+	+
588_fri_c4_1000_100	0.687 ± 0.023	0.674 ± 0.069	0.961 ± 0.028	0.958 ± 0.027	+	+
589_fri_c2_1000_25	0.554 ± 0.028	0.538 ± 0.077	0.982 ± 0.003	0.978 ± 0.007	+	+
590_fri_c0_1000_50	0.874 ± 0.082	0.871 ± 0.071	0.963 ± 0.002	0.959 ± 0.006	+	+
592_fri_c4_1000_25	0.692 ± 0.026	0.677 ± 0.064	0.977 ± 0.007	0.972 ± 0.009	+	+
593_fri_c1_1000_10	0.524 ± 0.023	0.526 ± 0.053	0.981 ± 0.003	0.979 ± 0.005	+	+
595_fri_c0_1000_10	0.888 ± 0.029	0.884 ± 0.038	0.956 ± 0.002	0.954 ± 0.008	+	+

Table 7 (continued)

Problem	OSGP		OSGP NLS		Comparison	
	R^2 (train)	R^2 (test)	R^2 (train)	R^2 (test)	Train	Test
598_fri_c0_1000_25	0.895 ± 0.028	0.890 ± 0.041	0.964 ± 0.001	0.960 ± 0.004	+	+
599_fri_c2_1000_5	0.598 ± 0.026	0.579 ± 0.057	0.984 ± 0.002	0.981 ± 0.003	+	+
606_fri_c2_1000_10	0.559 ± 0.021	0.552 ± 0.063	0.978 ± 0.004	0.974 ± 0.006	+	+
607_fri_c4_1000_50	0.666 ± 0.038	0.645 ± 0.051	0.979 ± 0.007	0.975 ± 0.009	+	+
608_fri_c3_1000_10	0.664 ± 0.031	0.640 ± 0.059	0.980 ± 0.003	0.977 ± 0.008	+	+
609_fri_c0_1000_5	0.903 ± 0.035	0.903 ± 0.037	0.963 ± 0.002	0.960 ± 0.005	+	+
612_fri_c1_1000_5	0.544 ± 0.029	0.525 ± 0.073	0.982 ± 0.002	0.978 ± 0.005	+	+
618_fri_c3_1000_50	0.686 ± 0.023	0.673 ± 0.063	0.979 ± 0.006	0.974 ± 0.009	+	+
620_fri_c1_1000_25	0.543 ± 0.028	0.524 ± 0.057	0.978 ± 0.004	0.974 ± 0.007	+	+
622_fri_c2_1000_50	0.594 ± 0.030	0.548 ± 0.067	0.980 ± 0.007	0.975 ± 0.011	+	+
623_fri_c4_1000_10	0.656 ± 0.028	0.631 ± 0.072	0.980 ± 0.005	0.977 ± 0.006	+	+
628_fri_c3_1000_5	0.670 ± 0.029	0.644 ± 0.079	0.979 ± 0.004	0.976 ± 0.004	+	+

R^2 values reported as median ± interquartile range. Symbols $+, -, \approx$ denote statistically-better, statistically-worse and statistically-insignificant differences between *OSGP NLS* and *OSGP* using a two-sided Wilcoxon rank-sum test ($\alpha = 0.05$)

Table 8 Holm–Bonferroni adjusted p values using a post hoc pairwise test (Dunn's test) on training data

	GP	GP NLS	OSGP	OSGP NLS	GSGP	AFP	MRGP	EPLEX	EPLEX-1M
GP NLS	**5.3e−09**	–	–	–	–	–	–	–	–
OSGP	1.0e+00	**9.4e−06**	–	–	–	–	–	–	–
OSGP NLS	**2.9e−10**	1.0e+00	**9.6e−07**	–	–	–	–	–	–
GSGP	**5.9e−05**	1.0e+00	**1.4e−02**	1.0e+00	–	–	–	–	–
AFP	**2.9e−03**	**4.5e−24**	**7.8e−06**	**4.6e−26**	**4.4e−17**	–	–	–	–
MRGP	**4.3e−29**	**2.6e−05**	**5.3e−23**	**2.0e−04**	**1.7e−09**	**4.6e−53**	–	–	–
EPLEX	1.0e+00	**2.4e−06**	1.0e+00	**2.1e−07**	**5.1e−03**	**2.9e−05**	**3.5e−24**	–	–
EPLEX-1M	**1.6e−02**	1.1e−01	6.6e−01	**3.2e−02**	1.0e+00	**2.7e−12**	**9.0e−14**	3.6e−01	–
XGBoost	**2.5e−32**	**9.5e−07**	**6.9e−26**	**9.3e−06**	**2.0e−11**	**2.2e−57**	1.0e+00	**3.8e−27**	**4.8e−16**
GradBoost	**8.2e−36**	**2.0e−08**	**5.2e−29**	**2.7e−07**	**1.5e−13**	**5.4e−62**	1.0e+00	**2.5e−30**	**1.5e−18**
MLP	**5.7e−12**	1.0e+00	**3.7e−08**	1.0e+00	3.2e−01	**1.0e−28**	**2.1e−03**	**7.3e−09**	**4.3e−03**
RF	**2.1e−18**	2.4e−01	**1.4e−13**	6.6e−01	**7.4e−04**	**3.2e−38**	6.0e−01	**1.6e−14**	**9.6e−07**
KernelRidge	**2.5e−11**	1.0e+00	**1.3e−07**	1.0e+00	5.3e−01	**1.0e−27**	**9.4e−04**	**2.6e−08**	**9.3e−03**
AdaBoost	**9.8e−04**	6.9e−01	1.0e−01	2.7e−01	1.0e+00	**8.7e−15**	**2.3e−11**	**4.7e−02**	1.0e+00
Lasso	**9.4e−06**	**4.6e−30**	**5.3e−09**	**2.7e−32**	**3.2e−22**	1.0e+00	**8.5e−62**	**2.8e−08**	**1.0e−16**
LinSvr	**1.1e−03**	**3.5e−25**	**2.1e−06**	**3.2e−27**	**5.0e−18**	1.0e+00	**1.0e−54**	**8.6e−06**	**4.2e−13**
LinReg	5.6e−02	**1.8e−20**	**4.1e−04**	**2.5e−22**	**4.6e−14**	1.0e+00	**1.1e−47**	**1.2e−03**	**9.6e−10**
SGD	**2.6e−08**	**1.4e−35**	**3.6e−12**	**5.2e−38**	**5.2e−27**	6.8e−01	**1.3e−69**	**2.5e−11**	**6.4e−21**
	XGBoost	GradBoost	MLP	RF	KernelRidge	AdaBoost	Lasso	LinSvr	LinReg
GradBoost	1.0e+00	–	–	–	–	–	–	–	–
MLP	**1.4e−04**	**5.7e−06**	–	–	–	–	–	–	–
RF	1.1e−01	**1.5e−02**	1.0e+00	–	–	–	–	–	–
KernelRidge	**5.2e−05**	**1.9e−06**	1.0e+00	1.0e+00	–	–	–	–	–
AdaBoost	**1.9e−13**	**1.0e−15**	5.6e−02	**4.2e−05**	1.0e−01	–	–	–	–

Table 8 (continued)

	XGBoost	GradBoost	MLP	RF	KernelRidge	AdaBoost	Lasso	LinSvr	LinReg
Lasso	**1.9e−66**	**2.1e−71**	**3.1e−35**	**1.1e−45**	**3.9e−34**	**1.3e−19**	–	–	–
LinSvr	**4.2e−59**	**9.0e−64**	**6.4e−30**	**1.3e−39**	**6.5e−29**	**1.1e−15**	1.0*e*+00	–	–
LinReg	**8.7e−52**	**3.7e−56**	**8.9e−25**	**1.1e−33**	**7.3e−24**	**5.4e−12**	1.0*e*+00	1.0*e*+00	–
SGD	**1.5e−74**	**9.0e−80**	**3.4e−41**	**1.8e−52**	**5.3e−40**	**3.9e−24**	1.0*e*+00	1.0*e*+00	8.9*e*−02

Bold values indicate statistical significance ($p < 0.05$)

Table 9 Holm–Bonferroni adjusted p values using a post hoc pairwise test (Dunn's test) on test data

	GP	GP NLS	OSGP	OSGP NLS	GSGP	AFP	MRGP	EPLEX	EPLEX-1M
GP NLS	**1.8e−12**	–	–	–	–	–	–	–	–
OSGP	1.0e+00	**2.0e−10**	–	–	–	–	–	–	–
OSGP NLS	**2.2e−13**	1.0e+00	**2.9e−11**	–	–	–	–	–	–
GSGP	**3.6e−12**	**1.2e−50**	**2.3e−14**	**2.1e−52**	–	–	–	–	–
AFP	**4.6e−02**	**2.8e−26**	**4.2e−03**	**1.4e−27**	**1.7e−03**	–	–	–	–
MRGP	1.0e+00	**7.0e−15**	1.0e+00	**7.2e−16**	**5.3e−10**	3.6e−01	–	–	–
EPLEX	1.0e+00	**3.4e−08**	1.0e+00	**6.2e−09**	**2.9e−17**	**1.4e−04**	1.0e+00	–	–
EPLEX-1M	**9.4e−06**	7.8e−01	**2.3e−04**	4.0e−01	**5.3e−36**	**5.2e−16**	**1.9e−07**	**6.3e−03**	–
XGBoost	**7.4e−04**	6.3e−02	**9.9e−03**	**2.6e−02**	**4.1e−31**	**8.4e−13**	**2.8e−05**	1.4e−01	1.0e+00
GradBoost	**2.8e−02**	**2.0e−03**	2.1e−01	**6.7e−04**	**1.6e−26**	**6.6e−10**	**1.9e−03**	1.0e+00	1.0e+00
MLP	7.3e−01	**1.1e−05**	1.0e+00	**2.7e−06**	**1.9e−21**	**7.0e−07**	1.1e−01	1.0e+00	2.0e−01
RF	1.0e+00	**4.2e−08**	1.0e+00	**8.0e−09**	**2.0e−17**	**1.2e−04**	1.0e+00	1.0e+00	**7.2e−03**
KernelRidge	1.0e+00	**3.2e−09**	1.0e+00	**5.3e−10**	**7.2e−16**	**7.6e−04**	1.0e+00	1.0e+00	**1.4e−03**
AdaBoost	1.0e+00	**2.0e−13**	1.0e+00	**2.3e−14**	**2.9e−11**	1.2e−01	1.0e+00	1.0e+00	**2.0e−06**
Lasso	**9.8e−05**	**5.7e−34**	**3.6e−06**	**2.0e−35**	3.1e−01	1.0e+00	**2.1e−03**	**3.8e−08**	**3.7e−22**
LinSvr	**2.1e−05**	**1.1e−35**	**6.4e−07**	**3.7e−37**	6.9e−01	1.0e+00	**5.8e−04**	**5.4e−09**	**1.5e−23**
LinReg	**2.0e−06**	**3.7e−38**	**4.5e−08**	**1.1e−39**	1.0e+00	1.0e+00	**7.5e−05**	**2.7e−10**	**1.4e−25**
SGD	**1.4e−08**	**4.3e−43**	**1.9e−10**	**9.9e−45**	1.0e+00	1.3e−01	**9.0e−07**	**6.0e−13**	**1.1e−29**
	XGBoost	GradBoost	MLP	RF	KernelRidge	AdaBoost	Lasso	LinSvr	LinReg
GradBoost	1.0e+00	–	–	–	–	–	–	–	–
MLP	1.0e+00	1.0e+00	–	–	–	–	–	–	–
RF	1.6e−01	1.0e+00	1.0e+00	–	–	–	–	–	–
KernelRidge	**4.5e−02**	6.5e−01	1.0e+00	1.0e+00	–	–	–	–	–
AdaBoost	**2.1e−04**	**9.8e−03**	3.6e−01	1.0e+00	1.0e+00	–	–	–	–

Table 9 (continued)

	XGBoost	GradBoost	MLP	RF	KernelRidge	AdaBoost	Lasso	LinSvr	LinReg
Lasso	**2.4e−18**	**7.5e−15**	**4.2e−11**	**3.0e−08**	**3.5e−07**	**3.6e−04**	–	–	–
LinSvr	**1.3e−19**	**5.5e−16**	**4.3e−12**	**4.2e−09**	**5.4e−08**	**8.7e−05**	1.0*e*+00	–	–
LinReg	**1.8e−21**	**1.1e−17**	**1.4e−13**	**2.1e−10**	**3.3e−09**	**9.1e−06**	1.0*e*+00	1.0*e*+00	–
SGD	**3.0e−25**	**4.0e−21**	**1.3e−16**	**4.4e−13**	**9.8e−12**	**7.8e−08**	1.0*e*+00	1.0*e*+00	1.0*e*+00

Bold values indicate statistical significance ($p < 0.05$)

References

1. M. Affenzeller, S. Wagner, Offspring selection: a new self-adaptive selection scheme for genetic algorithms, in *Adaptive and Natural Computing Algorithms, Springer Computer Science*, ed. by B. Ribeiro, R.F. Albrecht, A. Dobnikar, D.W. Pearson, N.C. Steele (Springer, Berlin, 2005), pp. 218–221
2. C.L. Alonso, J.L. Montana, C.E. Borges, Evolution strategies for constants optimization in genetic programming, in *21st International Conference on Tools with Artificial Intelligence. ICTAI '09* (2009), pp. 703–707
3. R.M.A. Azad, C. Ryan, A simple approach to lifetime learning in genetic programming-based symbolic regression. Evol. Comput. **22**(2), 287–317 (2014)
4. M. Castelli, L. Trujillo, L. Vanneschi, S. Silva, et al. Geometric semantic genetic programming with local search, in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation* (ACM, 2015), pp. 999–1006
5. Q. Chen, B. Xue, M. Zhang, Generalisation and domain adaptation in GP with gradient descent for symbolic regression, in *2015 IEEE Congress on Evolutionary Computation (CEC)* (IEEE, 2015), pp. 1137–1144
6. Q. Chen, M. Zhang, B. Xue, Feature selection to improve generalisation of genetic programming for high-dimensional symbolic regression. IEEE Trans. Evol. Comput. **21**, 792–806 (2017)
7. X. Chen, Y.S. Ong, M.H. Lim, K.C. Tan, A multi-facet survey on memetic computation. IEEE Trans. Evol. Comput. **15**(5), 591–607 (2011)
8. A. Conn, N. Gould, P. Toint, *Trust Region Methods. MPS-SIAM Series on Optimization* (Society for Industrial and Applied Mathematics, Philadelphia, 2000)
9. O.J. Dunn, Multiple comparisons using rank sums. Technometrics **6**(3), 241–252 (1964)
10. T. Fernandez, M. Evett, Numeric mutation as an improvement to symbolic regression in genetic programming, in *Evolutionary Programming VII*, ed. by V.W. Porto, N. Saravanan, D. Waagen, A.E. Eiben (Springer, Berlin, 1998), pp. 251–260
11. D.E. Goldberg et al., *Genetic Algorithms in Search Optimization and Machine Learning*, vol. 412 (Addison-Wesley Reading, Menlo Park, 1989)
12. A. Griewank, A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation* (SIAM, Philadelphia, 2008)
13. L.M. Howard, D.J. D'Angelo, The GA-P: a genetic algorithm and genetic programming hybrid. IEEE Expert **10**(3), 11–15 (1995)
14. I. Icke, J.C. Bongard, Improving genetic programming based symbolic regression using deterministic machine learning, in *2013 IEEE Congress on Evolutionary Computation (CEC)* (IEEE, 2013), pp. 1763–1770
15. P. Juárez-Smith, L. Trujillo, Integrating local search within neat-GP, in *Proceedings of the 2016 Genetic and Evolutionary Computation Conference Companion* (ACM, 2016), pp. 993–996
16. M. Keijzer, Improving symbolic regression with interval arithmetic and linear scaling, in *Proceedings of the 6th European Conference on Genetic Programming, EuroGP 2003. LNCS*, vol. 2610, ed. by C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, E. Costa (Springer, Berlin, 2003), pp. 70–82
17. M. Keijzer, Scaled symbolic regression. Genet. Program. Evolvable Mach. **5**(3), 259–269 (2004)
18. S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi et al., Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
19. M. Kommenda, M. Affenzeller, G. Kronberger, S.M. Winkler, Nonlinear least squares optimization of constants in symbolic regression, in *International Conference on Computer Aided Systems Theory* (Springer, 2013), pp. 420–427
20. M. Kommenda, G. Kronberger, S. Winkler, M. Affenzeller, S. Wagner, Effects of constant optimization by nonlinear least squares minimization in symbolic regression, in *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation* (ACM, 2013), pp. 1121–1128
21. J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (MIT Press, Cambridge, MA, 1992)
22. K. Krawiec, Genetic programming with local improvement for visual learning from examples, in *Computer Analysis of Images and Patterns*, ed. by W. Skarbek (Springer, Berlin, 2001), pp. 209–216
23. W.H. Kruskal, W.A. Wallis, Use of ranks in one-criterion variance analysis. J. Am. Stat. Assoc. **47**(260), 583–621 (1952)

24. W. La Cava, T. Helmuth, L. Spector, K. Danai, Genetic programming with epigenetic local search, in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation* (ACM, 2015), pp. 1055–1062
25. W. La Cava, J.H. Moore, Semantic variation operators for multidimensional genetic programming, in *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '19* (ACM, New York, NY, 2019), pp. 1056–1064
26. F. Lane, R.M.A. Azad, C. Ryan, On effective and inexpensive local search techniques in genetic programming regression, in *International Conference on Parallel Problem Solving from Nature* (Springer, 2014), pp. 444–453
27. K. Levenberg, A method for the solution of certain non-linear problems in least squares. Q. J. Appl. Math. **II**(2), 164–168 (1944)
28. S. Luke, Two fast tree-creation algorithms for genetic programming. IEEE Trans. Evol. Comput. **4**(3), 274–283 (2000)
29. D.W. Marquardt, An algorithm for least-squares estimation of nonlinear parameters. J. Soc. Ind. Appl. Math. **11**(2), 431–441 (1963)
30. P. Moscato et al., On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms. Caltech concurrent computation program. C3P Report 826, 1989 (1989)
31. J. Nocedal, S.J. Wright, *Numerical Optimization*, 2nd edn. (Springer, New York, NY, 2006)
32. R.S. Olson, W. La Cava, P. Orzechowski, R.J. Urbanowicz, J.H. Moore, PMLB: a large benchmark suite for machine learning evaluation and comparison. BioData Min. **10**(1), 36 (2017)
33. P. Orzechowski, W. La Cava, J.H. Moore, Where are we now?: A large benchmark study of recent symbolic regression methods, in *GECCO '18: Proceedings of the Genetic and Evolutionary Computation Conference* (ACM, New York, NY, 2018), pp. 1183–1190
34. R. Poli, W.B. Langdon, N.F. McPhee, A field guide to genetic programming. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk (2008). Accessed 2 Dec 2019
35. J.R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D.B. Fogel, M.H. Garzon, D.E. Goldberg, H. Iba, R.L. Riolo (eds.), in *Genetic Programming 1998: Proceedings of the Third Annual Conference*, July 22–25, 1998, University of Wisconsin, Madison, Wisconsin (San Francisco, CA, Morgan Kaufmann)
36. L.B. Rall, *Automatic Differentiation: Techniques and Applications. Lecture Notes in Computer Science*, vol. 120 (Springer, Berlin, 1981)
37. C. Ryan, M. Keijzer, An analysis of diversity of constants of genetic programming, in *Genetic Programming. Proceedings of EuroGP'2003, LNCS*, vol. 2610, ed. by C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, E. Costa (Springer, Berlin, 2003), pp. 404–413
38. M. Schoenauer, M. Sebag, F. Jouve, B. Lamy, H. Maitournam, Evolutionary identification of macro-mechanical models, in *Advances in Genetic Programming 2*, ed. by P.J. Angeline, K.E. Kinnear Jr. (MIT Press, Cambridge, MA, 1996), pp. 467–488
39. H.P. Schwefel, *Numerical Optimization of Computer Models* (Wiley, Hoboken, 1981)
40. K.C. Sharman, A.I. Esparcia Alcazar, Y. Li, Evolving signal processing algorithms by genetic programming, in *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA*, vol. 414, ed. by A.M.S. Zalzala (IEE, Sheffield, 1995), pp. 473–480
41. S. Stijven, W. Minnebo, K. Vladislavleva, Separating the wheat from the chaff: on feature selection and feature importance in regression random forests and symbolic regression, in *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation*, ed. by S. Gustafson, E. Vladislavleva (ACM, Dublin, 2011), pp. 623–630
42. A. Topchy, W.F. Punch, Faster genetic programming based on local gradient search of numeric leaf values, in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, ed. by L. Spector, E.D. Goodman, A. Wu, W.B. Langdon, H.M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M.H. Garzon, E. Burke (Morgan Kaufmann, San Francisco, CA, 2001), pp. 155–162
43. V.V. Toropov, L.F. Alvarez, Application of genetic programming to the choice of a structure of multipoint approximations, in *1st ISSMO/NASA International Conference on Approximations and Fast Reanalysis in Engineering Optimization* (1998)
44. L. Trujillo, Z. Emigdio, P.S. Juárez-Smith, P. Legrand, S. Silva, M. Castelli, L. Vanneschi, O. Schütze, L. Muñoz et al., Local search is underused in genetic programming, in *Genetic Programming Theory and Practice XIV*, ed. by R. Riolo, B. Worzel, B. Goldman, B. Tozier (Springer, Cham, 2018), pp. 119–137

45. E.J. Vladislavleva, G.F. Smits, D. Den Hertog, Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. IEEE Trans. Evolut. Comput. **13**(2), 333–349 (2009)
46. S. Wagner, M. Affenzeller, Sexualga: gender-specific selection for genetic algorithms, in *Proceedings of the 9th World Multi-Conference on Systemics, Cybernetics and Informatics (WMSCI)*, vol. 4 (2005), pp. 76–81
47. S. Wagner, G. Kronberger, A. Beham, M. Kommenda, A. Scheibenpflug, E. Pitzer, S. Vonolfen, M. Kofler, S. Winkler, V. Dorfer et al., Architecture and design of the HeuristicLab optimization environment, in *Advanced Methods and Applications in Computational Intelligence*, ed. by R. Klempous, J. Nikodem, W. Jacak, Z. Chaczko (Springer, Heidelberg, 2014), pp. 197–261
48. P. Wang, K. Tang, E.P. Tsang, X. Yao, A memetic genetic programming with decision tree-based local search for classification problems, in *2011 IEEE Congress on Evolutionary Computation (CEC)* (IEEE, 2011), pp. 917–924
49. S.M. Winkler, Evolutionary system identification—modern concepts and practical applications. Ph.D. thesis, Institute for Formal Models and Verification, Johannes Kepler University, Linz, Austria (2008)
50. E. Z-Flores, L. Trujillo, O. Schütze, P. Legrand et al., Evaluating the effects of local search in genetic programming, in *EVOLVE—A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation V*, ed. by A.A. Tantar, et al. (Springer, Cham, 2014), pp. 213–228
51. E. Z-Flores, L. Trujillo, O. Schütze, P. Legrand, et al., A local search approach to genetic programming for binary classification, in *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference-GECCO'15* (2015)
52. Q. Zhang, C. Zhou, W. Xiao, P.C. Nelson, Improving gene expression programming performance by using differential evolution, in *Sixth International Conference on Machine Learning and Applications, ICMLA 2007* (IEEE, Cincinnati, OH, 2007), pp. 31–37