Memetic Computing (2020) 12:299–315
https://doi.org/10.1007/s12293-020-00311-8

REGULAR RESEARCH PAPER

An efficient memetic genetic programming framework for symbolic regression

Tiantian Cheng[1] · **Jinghui Zhong**[1]

Received: 7 November 2019 / Accepted: 21 September 2020 / Published online: 13 October 2020

Abstract

Background Symbolic regression is one of the most common applications of genetic programming (GP), which is a popular evolutionary algorithm in automatic computer program generation. Despite existing success of GP on symbolic regression, the accuracy and efficiency of GP can still be improved especially on complicated symbolic regression problems, enabling GP to be applied to more fields.

Purpose This paper proposes a novel memetic GP framework to improve the accuracy and search efficiency of GP on complicated symbolic regression problems. The proposed framework consists of two components: feature construction and feature combination. The first component focuses on constructing diverse features. The second component aims to filter redundant features and linearly combines these independent features.

Methods The first component (feature construction) focuses on constructing polynomial features derived from polynomial functions, and evolves features by a GP solver. In addition, a gradient-based nonlinear least squares algorithm named Levenberg-Marquardt (LM) is embedded in the second component (feature combination) to locally adjust the weights of independent features. A filtering mechanism is put forward to discard redundant features in the second component. Hence, the polynomial features and evolved features can work together in the framework to improve the performance of GP.

Results Experimental results demonstrate that the proposed framework offers enhanced performance compared with several state-of-the-art algorithms in terms of accuracy and search efficiency on nine benchmark regression problems and three real-world regression problems.

Conclusion In this study, a novel memetic genetic programming framework is proposed to improve the performance of GP on symbolic regression. Experimental results demonstrate that the proposed framework can improve the accuracy and search efficiency of GP on complicated symbolic regression problems compared with four state-of-the-art algorithms.

Keywords Evolutionary computation · Genetic programming · Symbolic regression

1 Introduction

Genetic programming (GP) is a popular evolutionary algorithm which has been proved quite effective in automatic computer program generation [13,20]. In GP, computer programs are represented as trees and evolved using genetic operators such as crossover and mutation. In the past decades, GP has been developing rapidly and a number of enhanced GP variants have been proposed such as gene expression programming (GEP) [15,54,55], cartesian genetic programming (CGP) [26], linear genetic programming (LGP) [6], and others [1,33]. Moreover, GP has been applied to a wide range of applications, including but not limited to symbolic regression (SR) [28,55], classification problem [39], job shop scheduling [30], rule discovery [31] and others [14]. Among these applications, symbolic regression is one of the most common applications of GP. It fulfills a regression task which aims to find the relationship between the input variables and responses in the given dataset. Compared with traditional regression tasks, symbolic regression discovers the relationship by combining various mathematical expressions without assuming a model beforehand, which makes it more flexible for practical applications.

✉ Jinghui Zhong
jinghuizhong@gmail.com

Tiantian Cheng
2269130298@qq.com

[1] School of Computer Science and Engineering, South China University of Technology, Guangzhou, China

Despite existing great success, the accuracy and efficiency of GP can still be improved especially on complicated symbolic regression problems, enabling GP to be applied to more fields. To this end, a number of efforts have been made over the past decades. For example, Pawlak et al. [33] proposed a new operator called random desired operator which uses intermediate states determined by semantic backpropagation to decompose the original task into smaller subtasks. Arnaldo et al. [1] combined lasso regression [41] and multi-objective optimization [11] together to boost accuracy and reduce the complexity of GP solution. Moraglio et al. [27] proposed geometric semantic genetic programming (GSGP) to search directly in semantic space. Local search operations based on evolutionary algorithm such as differential evolution (DE) [34] have also been utilized to fine-tune solutions [18,19,42,46,53]. However, existing methods are either time-consuming or unable to offer solutions with satisfying accuracy. Therefore, developing a more accurate and effective GP is still greatly desirable.

For complex optimization problems, memetic algorithms have been proved to be effective to obtain highly accurate solutions [9,12,25,51,52]. Inspired by that, this paper proposes a novel memetic GP framework to improve the accuracy and search efficiency of GP on complicated symbolic regression problems. The proposed framework consists of two components: feature construction and feature combination. The first component focuses on constructing polynomial features derived from polynomial functions, and evolves the features generated by GP or its variants. Diverse features generated by the first component increase the possibility of finding highly accurate solutions. The second component aims to filter redundant features and find proper weights for independent features to form the final solution. Because there may be some correlation between the features passed from the first component, redundant features are filtered in the second component. In addition, a gradient-based nonlinear least squares algorithm named Levenberg–Marquardt (LM) [22] is embedded in the second component to locally adjust the weights of independent features. With the help of LM, accuracies of solutions can be improved greatly. Furthermore, a GP variant named linear imperative programming with differential evolution (LDEP) [16] is integrated into our framework to create evolved features for the first component. This algorithm is called memetic LDEP (MLDEP). Experimental results show that the MLDEP can achieve higher accuracy and more efficient search ability than several state-of-the-art GP variants.

The rest of the paper is organized as follows: the related work and efforts on highly accurate GP are discussed in Sect. 2. The proposed memetic GP framework is described in Sect. 3. Section 4 demonstrates experimental studies. Section 5 draws the conclusion.

2 Preliminaries

In this section, we provide a brief introduction of background knowledge that is relevant to this study. Since the purpose of this paper is to solve the symbolic regression (SR) using GP [32,36,48], a definition of SR is described first to aid readers in better comprehending our method. Next, related work for solving SR using GP is presented.

2.1 Symbolic regression

Regression is a process to identify the relationships between the input variables and the corresponding response. Given a dataset, whose sample is composed of input variables and the corresponding response. The task of SR is to find a function $S(.)$ whose output can fit the response well. The ith sample can be represented as a real-valued vector

$$A_i = [a_{i,1}, a_{i,2}, \ldots, a_{i,d}, y_i], \tag{1}$$

where A_i represents the ith sample, $a_{i,j}$ is the jth input variable of the ith sample, d is the number of input variables and y_i is the corresponding response of the ith sample. Generally speaking, function $S(.)$ is composed of elements selected from the predefined function set and terminal set. The function set can contain common mathematic functions or user-defined functions, while the terminal set is composed of variables and constants. In the beginning, mean square error (MSE) can be adopted as the metric to measure the quality of formula $S(.)$:

$$MSE = \frac{\sum_{i=1}^{n}(y_i - o_i)^2}{n}, \tag{2}$$

where o_i is the output of formula $S(.)$ with respect to the ith sample and n is the number of samples. However, MSE or root mean square error ($RMSE$) is dependent on the range of the response variable. A better measurement is the normalized $RMSE$ ($NRMSE$) [8]

$$F(S(.)) = \frac{\sqrt{MSE}}{\sigma(y)}, \tag{3}$$

where $F(S(.))$ is the objective function to measure the fitness of formula $S(.)$, and $\sigma(y)$ can be the standard deviation of the response variable. In this study, the task of SR is to find a formula $S(.)$ that can minimize $F(S(.))$.

2.2 Related work of GP on SR

Over the past decades, a number of methods have been proposed to improve the accuracy of GP on SR. Generally, these methods can be classified into three groups.

The first group focuses on introducing new symbols to represent constants in the GP tree, so that the algorithm can find highly accurate solutions with constants. For instance, Koza [20] proposed to use a special terminal called ephemeral random constant (ERC) to represent constants in GP. At the initialization step, each ERC is initialized to a random constant of specific data type within a special interval. Then, these ERCs will spread between different trees with the help of mutation or crossover operator. A drawback of this kind of method is that it is not accurate enough in some cases due to the blind random search.

The second group utilizes new operators to improve the search efficiency. Creep mutation and uniform mutation are two kinds of mutation methods proposed at early stages [35]. Recently, semantic-based operators such as semantic-based crossover and mutation operators [4,5,21,29,44,45] have been proposed and attracted increasing attention from researchers. Semantic backpropagation [33] is a powerful operator where semantics of the target response is utilized to improve the search efficiency. In semantic backpropagation, a library of trees is constructed. The heights of trees are restrained so that the number of nodes would not be too large. The semantic backpropagation performs excellently on 1-dimensional problems but mediocre on high dimensional problems. Geometric sematic genetic programming (GSGP) [27] is another GP variant based on semantics which drives the search in the semantics space rather than the syntax space. Recently, Chen et al. [7] proposed an angle-driven selection operator and two angle-driven geometric search operators which induce a unimodal fitness landscape in the semantic space, and offered a theoretical framework for researchers to design geometric semantic operators more efficiently. In general, this kind of method is usually time-consuming.

The third group makes use of optimizers to fine-tune constants to improve the solution accuracy. For example, Zhang et al. [53] proposed a GP variant which utilizes differential evolution [34] to refine the constants. In this method, a special gene called random number generator (RNG) is employed as a kind of terminal to represent constants. All RNGs of an individual form a vector, which is optimized by the differential evolution. In [46], two populations are created for genetic algorithm (GA) and GP respectively, where GA is used to evolve numeric terminals for GP. Other algorithms such as simulated annealing [18], gradient search algorithm [42], estimation of distribution algorithm [40] and nonlinear least squares minimization [19], can also be used as local search operators to fine-tune the constants for GP.

One special cluster of the third group contains several state-of-the-art algorithms. The main idea is to construct the final solution by linearly combining multiple shorter sub-solutions/subexpressions. For example, Arnaldo et al. [1] proposed the multiple regression genetic programming (MRGP) to assign a weight to each node in the solution. In this way, the output of the solution is the linear combination of all nodes in the tree instead of the value stored in the root node. In order to reduce the complexities of solutions, the least absolute shrinkage and selection operator (LASSO) [41] is adopted. So the weight vector of these nodes is regulated using the L_1 norm. However, this method is time-consuming and easily results in over-fitting. Another similar algorithm is evolutionary feature synthesis (EFS) [2] where each feature used in the final solution is simple and independent. Since this method restricts the height of the tree for each feature, each feature maintained in the population is too simple to acquire sufficient expressive ability. GPTIPS [37] adopts the least squares regression to combine GP trees of the current population to form the final solution. Individuals in the population are evolved by multi-gene genetic programming (MGGP) [17]. Fast function extraction (FFX) [23] is a deterministic algorithm based on GP. FFX creates lots of basic functions in advance. Then it employs the regularized linear regression to combine promising functions to form the final model. A shortcoming of FFX is that the basic function set is fixed in advance, which limits its exploration ability.

Our proposed method belongs to the third group. Unlike the above existing works, the proposed framework not only uses complex features evolved by GP but also adopts simple features generated by elementary functions. Hence, our method has superior search ability which increases the chance of finding good solutions. Besides, filtering redundant features and adopting LM optimizer also contribute to improving the accuracy and search efficiency.

3 Proposed framework

3.1 General architecture of the proposed framework

The basic idea of the proposed memetic GP framework is to construct accurate solutions by linearly combining a set of polynomial features and evolved features. As shown in Fig. 1, the proposed framework contains two components: feature construction and feature combination.

The first component focuses on constructing diverse features. Different from previous algorithms, the first component not only uses features generated by polynomial functions but also evolved features generated by GP or its variants. Polynomial functions with low order are applied to each variable to generate polynomial features. Evolved features refer to the subexpressions of the GP trees generated during the process of evolution. Then these features will be passed to the second component.

The second component aims to filter redundant features and linearly combines these independent features. In this component, a filtering mechanism is put forward to discard redundant features. If a feature can be linearly represented by

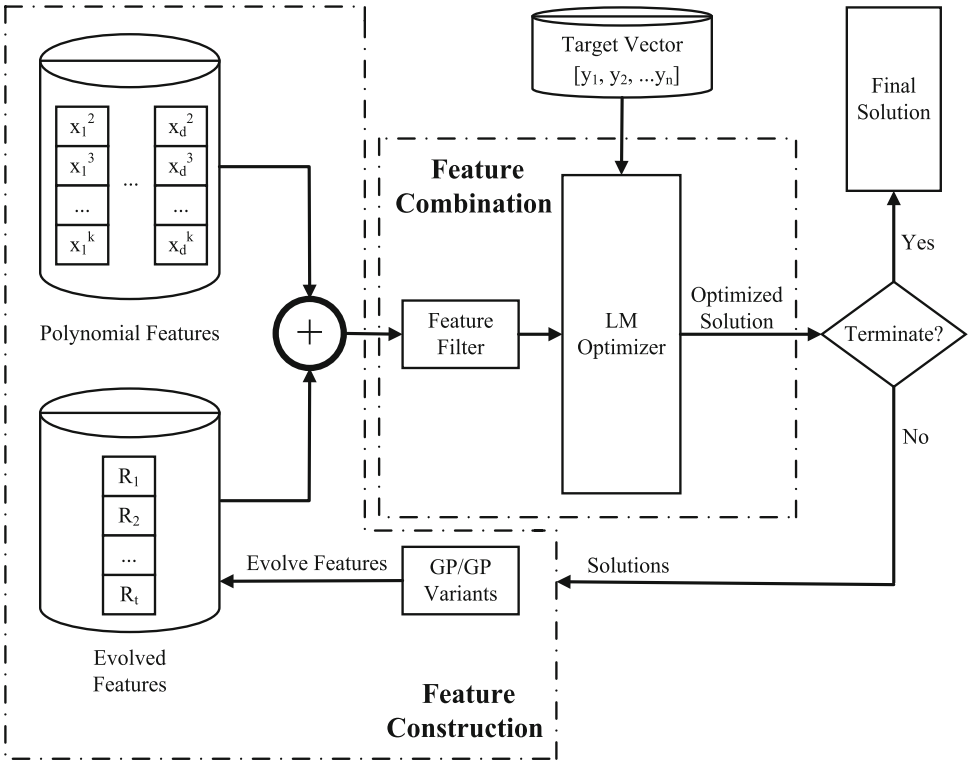

Fig. 1 Flowchart of the memetic GP framework

other features, then this feature will be discarded early before doing the LM optimization. This mechanism can be fulfilled by putting the output of all features into a matrix. Rank of this matrix is the number of independent features that will be left behind. Using this way, all the independent features can be selected. Next, they will be passed to the LM optimizer to construct the final solution. The LM optimizer is adopted to adjust the weight for each independent feature due to its fast convergence speed compared with other nonlinear least square methods [22]. The best individual generated by the second component is outputted as the final solution.

In general, with regard to less complicated problems, polynomial features can be used to approximate them. As for problems containing intricate structures and constants, evolved features with more expressive power are essential to finding accurate solutions. In addition, with the help of the LM optimizer, each feature can get a proper weight, and all features are combined linearly to form the final solution. Hence, the polynomial features and evolved features can work together in the framework to improve the performance of GP.

3.2 Program representation

In order to validate the effectiveness of the proposed framework, a GP solver named LDEP [16] is integrated into our framework to generate evolved features for the first component. This algorithm is called memetic LDEP (MLDEP). The

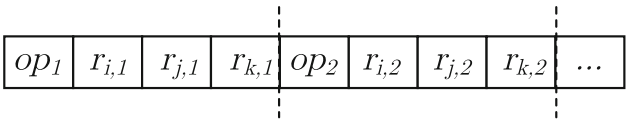

Fig. 2 The schematic diagram of MLDEP

program representation form of MLDEP is based on LDEP and given here to help readers understand MLDEP better.

As shown in Fig. 2, each solution of MLDEP is a sequence of imperative instructions. Every four elements (op, r_i, r_j, r_k) form an instruction. The first element op indicates which operator to use. r_i, r_j, r_k are registers or constants. The instruction (op, r_i, r_j, r_k) represents $r_i = r_j \ op \ r_k$. Several registers are maintained by MLDEP to store the temporary values during the evolution process. For example, an instruction like $r_i = r_j + r_k$ can be reduced to ($+$, r_i, r_j, r_k). r_j and r_k can also be constants. In order to operate these instructions more conveniently, each individual of MLDEP is represented by a vector of floating-point values. Each instruction is represented by 4 floating-point values (v_1, v_2, v_3, v_4), where v_1 encodes the operator, v_2, v_3 and v_4 encode the registers or constants. Each floating-point value v_i can be turned into an integer value to index the operator lists or register lists using the following scheme.

First, a floating point value within [0, 1) is gotten by

$$v_i - \lfloor v_i \rfloor, \tag{4}$$

1.36	2.71	0.59	4.81	5.22	3.76	1.03	9.21

Fig. 3 An example program of MLDEP

where $\lfloor v_i \rfloor$ returns the biggest integer no more than v_i.

Let n_{ops} be the number of operators and n_{regs} be the number of registers, then the operator encoded by v_i can be calculated by

$$\#operator = \lfloor (v_i - \lfloor v_i \rfloor) * n_{ops} \rfloor. \tag{5}$$

When decoding a register or constant, if the value returned by Eq. 4 is bigger than a predefined constant probability c_{rate}, then a register will be used. The register represented by v_i will be decoded similarly

$$\#register = \lfloor (v_i - \lfloor v_i \rfloor) * n_{regs} \rfloor. \tag{6}$$

Otherwise v_i encodes a constant, the constant indicated by v_i is computed using an equation different from the previous one. According to the suggestion described in LDEP, keeping only the fraction part of v_i will cause a strong linkage between the constant probability and the constant register index. Thus, v_i is used directly here to index the constants.

$$\#constant = (\lfloor v_i * C \rfloor) \ mod \ C, \tag{7}$$

where C is the number of constants, mod is the modulo operator.

In the following parts, we give an example to help readers understand the mapping process. Suppose the MLDEP works with 3 registers (r_0 to r_2), 50 constants and 4 following operators

$$0 : + \quad 1 : - \quad 2 : * \quad 3 : / \tag{8}$$

Figure 3 demonstrates a vector of 8 floating-point values which can be decoded to the following two instructions.

Specifically, the first element represents an operator selected from the four operators. To index the operator, the first element is transformed into an integer using Eq. 5, $\lfloor (1.36 - \lfloor 1.36 \rfloor) * 4 \rfloor = 1$. Thus, the operator $-$ is selected. The second value denotes a register. According to Eq. 6, $\lfloor (2.71 - \lfloor 2.71 \rfloor) * 3 \rfloor = 2$, so r_2 is picked. Assume the predefined constant probability equals to 0.05. The third element denotes a register or constant. First a float value in the range [0, 1] is obtained according to Eq. 4. $0.59 - \lfloor 0.59 \rfloor = 0.59$ which is bigger than 0.05, so the third element is decoded as a register. On the basis of Eq. 6, $\lfloor (0.59 - \lfloor 0.59 \rfloor) * 3 \rfloor = 1$, meaning that r_1 is selected. The last element can be decoded in the same way. $4.81 - \lfloor 4.81 \rfloor = 0.81$ which is bigger than 0.05, illustrating a register. Since $\lfloor (0.81 - \lfloor 0.81 \rfloor) * 3 \rfloor = 2$, register r_2 is chosen. After transformation, the first four float-point values make up an instruction

$$r_2 = r_1 - r_2. \tag{9}$$

The next four elements are decoded using the same mapping process except that the seventh element of the vector represents a constant. The seventh element is 1.03, according to Eq. 4 and $1.03 - \lfloor 1.03 \rfloor = 0.03$ that is less than the constant probability 0.05, so this element is turned into a constant. Then using Eq. 7, the index number is $(\lfloor 1.03 * 50 \rfloor) \ mod \ 50 = 1$. Assume the corresponding element of the constant array equals to 7.95, then the last four elements can be mapped as the following instruction

$$r_2 = 7.95 + r_0. \tag{10}$$

Using this representation form, vectors of floating-point values are translated to linear sequences of instructions before evaluation no matter how these vectors are created.

3.3 Algorithm implementation

The procedure of MLDEP is composed of six steps, namely initialization, mutation, crossover, features filtering, weights optimization, and selection. In the following parts, we will introduce the procedure of MLDEP in a step-by-step manner. The pseudo-code of MLDEP is also given in Algorithm 1.

Step 1-initialization During the initialization stage, an array of 50 constants in the range $[-5.0, +5.0]$ is created first. Next, the corresponding polynomial features for each variable are created. For each variable, polynomial functions with order higher than one but no more than MAX_ORDER are applied to produce polynomial features. Thus, there are $MAX\_ORDER - 1$ polynomial features for each variable.

Then, evolved features are generated. Different from traditional LDEP, MLDEP maintains two kinds of registers to store the results of evolved features for different purposes. One kind of registers are unmodifiable which means that these registers can only appear on the right side of the operator. Another kind of registers are modifiable indicating that content of these registers can be changed. Thus, the second kind of registers can appear on both sides of any operators. In order to protect original variables from being overlapped during the evolution process, each input variable is assigned to an unmodifiable separate register to store its value. Each register r_s is initialized with an input variable x_t whose indices satisfy

$$t = s \ mod \ n_{vars}, \tag{11}$$

where n_{vars} is the total number of input variables in the dataset.

To create evolved features, NP programs are created randomly to form the initial population. Each program can be represented by a vector of floating-point values. The ith program H_i can be represented as a vector

$$H_i = [h_{i,1}, h_{i,2}, \ldots, h_{i,j}, \ldots, h_{i,D}], \tag{12}$$

where $i = 1, 2, 3, \ldots, NP$; $j = 1, 2, 3, \cdots, D$; $h_{i,j}$ represents the jth element of H_i. Each $h_{i,j}$ is initialized as a random value chosen from the interval [0, 1], D is the length of the program, which is a multiple of 4. As mentioned before, every consecutive 4 floating-point values in the program represent an instruction. Thus, for a program of length D, it contains $D/4$ instructions to manipulate the registers.

Step 2-mutation In this step, a mutation operator is performed on each target vector to produce a corresponding mutant vector Y_i. In this study, the mutation operator in DE [34] is adopted to generate the linear sequences of imperative instructions, since DE is quite effective and suitable for optimizing non-linear continuous space functions. Thus, the mutant vector is generated by the commonly used "DE/best/2/bin" strategy

$$Y_i = H_{best} + F \cdot (H_{q_1} + H_{q_2} - H_{q_3} - H_{q_4}), \tag{13}$$

where H_{best} is the best individual at the current population, F is the scaling factor which manages the amplification of DE and prevents DE from falling into the local optimum. i, q_1, q_2, q_3, q_4 are five mutually distinct indices that are selected randomly.

Step 3-crossover In the third step, a trial vector U_i is created using H_i and Y_i

$$u_{i,j} = \begin{cases} y_{i,j}, & \text{if } rand(0,1) < CR \text{ or } j = k \\ h_{i,j}, & \text{otherwise,} \end{cases} \tag{14}$$

where CR is the crossover rate, k is a random integer between 1 and D, $u_{i,j}$, $y_{i,j}$ and $h_{i,j}$ are the jth variables of U_i, Y_i and H_i respectively. Similar to the mutation rate, the crossover rate CR is randomly set as $CR = rand(0, 1)$.

Step 4-features filtering After the crossover step, trial vectors representing different programs are generated. Using the same mapping scheme, each trial vector U_i can be decoded to a linear sequence of imperative instructions. After carrying out these instructions one by one, each register of MLDEP can be regarded as an evolved feature. However, one issue arising from previous steps is that features may be linearly dependent. In order to reduce the complexity of final solutions, redundant features will be filtered in this step. If a feature can be linearly represented by other features, then this feature will be discarded. This job can be achieved by putting the semantics of all features (i.e., polynomial features and evolved features) into a matrix $G = [g_1, g_2, \ldots, g_m]$ as follows

$$\begin{bmatrix} g_{1,1} & \cdots & g_{1,m} \\ . & \cdots & . \\ . & \cdots & . \\ . & \cdots & . \\ g_{n,1} & \cdots & g_{n,m} \end{bmatrix} \tag{15}$$

where n is the number of samples, m is the total number of features and $g_{i,j}$ is the ith element of the column vector g_j. Each column represents the semantics of each feature on n samples. Then the rank of this matrix is the number

Algorithm 1: MLDEP

```
Begin:
T1 = clock() /* a function returns the system time */
T2 = clock()
ΔT = T2 − T1
/* initialize the constant pool */
for k = 1 to 50 do
    set constant[k] be a real number between [-5, 5]
/* generate polynomial features */
for each input variable v_i appeared in the dataset do
    for j = 2 to MAX_ORDER do
        v_i^j is generated as one polynomial feature
/* initialize the register */
for s = 1 to n_regs do
    initialize register r_s with corresponding input variable using Eq. 11
/* initialize the population */
for i = 1 to NP do
    for j = 1 to n do
        set h_{i,j} randomly

while ΔT ≤ MAX_TIME do
    for i = 1 to NP do
        /* mutation and crossover */
        F = rand(0, 1); CR = rand(0, 1)
        Randomly choose four individuals: H_{q1}, H_{q2}, H_{q3}, H_{q4}
        and q1 ≠ q2 ≠ q3 ≠ q4 ≠ i from current population
        Set k to be a random integer between 1 and D
        for j = 1 to D do
            if (rand(0,1)<CR or j=k) then
                u_{i,j} = H_{best,j} + F · (H_{q1,j} + H_{q2,j} − H_{q3,j} − H_{q4,j})
            else
                u_{i,j} = h_{i,j}

        /* feature filtering */
        Put the semantics of polynomial features and evolved features
        decoded by U_i into the matrix G = [g1, g2, ..., gm]
        Find the maximally independent features [f1, f2, ..., fz] of G using
        elementary row transformation and leave them behind

        /* weight optimization */
        Assign weights to independent features using the LM Optimizer
        [w0, w1, ..., wz] = LM_Optimizer([1, f1, f2, ..., fz])
        Calculate the formula represented by U_i:
        S(.) = w0 * 1 + w1 * f1 + ··· + wz * fz

        /* selection */
        if F(U_i) ≤ F(H_i) then
            H_i = U_i
            if F(U_i) ≤ F(H_best) then
                H_best = U_i
    T2 = clock()
    ΔT = T2 − T1
End
```

Table 1 Symbolic regression problems for comparison

Problem	Dimension	Objective function	Train	Test
Septic	1	$x^7 - 2x^6 + x^5 - x^4 + x^3 - 2x^2 + x$	$U[-1, 1, 20]$	$U[-1, 1, 20]$
Nguyen-5	1	$sin(x^2) * cos(x) - 1$	$U[-1, 1, 20]$	$U[-1, 1, 20]$
Keijzer-1	1	$0.3xsin(2\pi x)$	$E[-1, 1, 0.1]$	$E[-1, 1, 0.001]$
Korns-11	5	$6.87 + 11cos(7.23{x_1}^3)$	$U[-50, 50, 10{,}000]$	$U[-50, 50, 10{,}000]$
Keijzer-14	2	$8/(2 + {x_1}^2 + {x_2}^2)$	$U[-3, 3, 20]$	$E[-3, 3, 0.01]$
Keijzer-15	2	$0.2{x_1}^3 + 0.5{x_2}^3 - x_1 - x_2$	$U[-3, 3, 20]$	$E[-3, 3, 0.01]$
Vladislavleva-8	2	$\frac{(x_1-3)^4+(x_2-3)^3-(x_2-3)}{(x_2-2)^4+10}$	$U[0.05, 6.05, 50]$	$E[-0.25, 6.35, 0.2]$
Keijzer-5	3	$\frac{30x_1x_3}{(x_1-10){x_2}^2}$	$x_1 : U[-1, 1, 1000]$	$x_1 : U[-1, 1, 10{,}000]$
			$x_2 : U[1, 2, 1000]$	$x_2 : U[1, 2, 10{,}000]$
			$x_3 : U[-1, 1, 1000]$	$x_3 : U[-1, 1, 10{,}000]$
Vladislavleva-5	3	$\frac{30(x_1-1)(x_3-1)}{{x_2}^2(x_1-10)}$	$x_1 : U[0.05, 2, 300]$	$x_1 : E[-0.05, 2.1, 0.15]$
			$x_2 : U[1, 2, 300]$	$x_2 : E[0.95, 2.05, 0.1]$
			$x_3 : U[0.05, 2, 300]$	$x_3 : E[-0.05, 2.1, 0.15]$
ENC	8	N/A	512 samples	256 samples
ENH	8	N/A	512 samples	256 samples
WIR	11	N/A	1066 samples	533 samples

of features that are mutually independent. By using elementary row transformation, the maximally independent column vector group $[f_1, f_2, \ldots, f_z]$ can be found. Thus, the number of features used in the final programs can be cut down to enhance the interpretability of MLDEP solution.

Step 5-weights optimization In this step, all independent features found in the previous step are linked linearly to form the final solution. As shown below, each feature is assigned with a weight.

$$S(.) = w_0 * 1 + w_1 * f_1 + \cdots + w_z * f_z, \tag{16}$$

where w_0 is an intercept, $w_j\ (j = 1, 2, 3, \ldots, z)$ is the weight of jth feature, z is the number of independent features after performing the features filtering step, and $S(.)$ is the formula represented by the trial vector U_i. In order to find a proper weight for each feature, the gradient-based LM algorithm [22] is used in this paper.

Step 6-selection In this step, the fitter solution between each target parent (H_i) and the corresponding mutant vector (U_i) will be selected to survive to the next generation based on their fitness values. On the basis of the above description, each program can be decoded to get a formula $S(.)$. Fitness of a program is defined as the fitness of the corresponding formula, which is computed according to Eq. 3. Thus, we can use Eq. 17 to select the fitter solution.

$$H_i = \begin{matrix} U_i, & \text{if } F(U_i) < F(H_i) \\ H_i, & \text{otherwise,} \end{matrix} \tag{17}$$

where $F(H)$ is the objective function which returns the fitness of the program H.

The last five steps are repeated until the maximal time is reached or a solution with satisfying fitness is found.

To summarize, polynomial features, and evolved features generated by LDEP construct the diversity of features, which increases the chance of finding better solutions. Besides, the filtering mechanism can reduce the search space and improve the search efficiency. At last, the LM optimizer locally fine-tunes weights for features to produce highly accurate solutions.

4 Experiments and comparisons

4.1 Test problems

In this section, MLDEP is tested on nine benchmark regression problems and three real-world regression problems to validate its effectiveness. Table 1 summarizes the regression problems used in this study. In Table 1, the first column is the problem name, the second column lists the dimension of each problem, and the third column shows the objective formulas. The fourth and fifth columns describe the training set and testing set respectively. $E[a, b, c]$ means a grid of points evenly spaced (for this variable) with an interval of c, from a to b inclusive. While $U[a, b, c]$ represents that the c instances selected from interval $[a, b]$ follow a uniform distribution. The function set of all problems is $\{+, -, \times, /, sin, cos\}$.

The nine benchmark regression problems are chosen from [20] and [24], which are commonly used in the literature. The dimension of the nine problems varies from one to five. For problems like Nguyen-5, where no testing set is given in the

literature, the testing set is generated according to uniform distribution within the same interval as the training set.

As for last three real-world problems, the dataset ENC and ENH are selected from [43]. The dataset WIR is selected from [10] where the input variables are physical factors, but the response is a subjective grade evaluated by human from one to ten. For each problem, we reorder the sample data and pick out a certain number of samples as the training set, while the rest samples are considered as the testing set.

4.2 Comparison algorithms and parameter settings

To assess the performance of MLDEP, four state-of-the-art GP variants are used for comparison. These four compared algorithms are described as follows.

Semantic backpropagation for designing search operators In [33], semantic backpropagation is combined with random desired operator (RDO) [50] to improve the accuracy and search efficiency of GP. One feature of this method is that it needs to construct a library of subtrees. According to the original paper, the static library construction scheme (RDO_4) is more stable, so RDO_4 is used in this paper to compete with the proposed MLDEP.

Linear imperative programming with differential evolution (LDEP) Since the proposed MLDEP is designed based on LDEP [16], we compare MLDEP with LDEP to evaluate the effectiveness of the new mechanisms proposed in this study.

Multiple regression genetic programming (MRGP) MRGP is a well known GP variant proposed by Arnaldo et al. [1]. In MRGP, all nodes in the decoded tree of an individual are linearly combined to form the final solution. Besides, the LASSO regression and non-dominated sorting genetic algorithm II (NSGA-II) [11] are used to get a balance between accuracy and complexity.

Evolutionary feature synthesis (EFS) EFS [2] is a recently published GP, which also combines features linearly to generate the final solution. Different from MRGP, EFS has no concept of genotype, cutting down the time of mapping genotype to phenotype. However, due to the height limit imposed on each feature, EFS is difficult to find solutions with complex structures.

Table 2 lists the parameter settings of MLDEP, while the parameters of the other four algorithms are configured according to what have been suggested by the original papers.

Table 2 Parameter settings of MLDEP

Parameters	Values
MAX_TIME	10 min
Stopping condition	Running for 10 min or optimal solution found
Fitness function	Normalized root mean square error ($NRMSE$)
c_{rate}	0.05
D	48
F	0.5
NP	20
CR	0.1
MAX_ORDER	4
Number of registers	Number of input variables plus five
Function set	$+, -, *, /^p$, sin, cos

c_{rate}, the constant probability; D, the program length; F, the scaling factor of the mutation operator; NP, the size of the population; CR, the crossover rate; $/^p$, the protected division; a/b, will return a if b is 0

4.3 Comparison metrics

Three metrics are used for performance evaluation. The first metric is the testing accuracy calculated according to Eq. 3. Generally, this metrics is regarded as the most important one for assessing the performance of an algorithm. The second metric is the success rate (abbreviated as Suc) of reaching perfect hits [3]. Suc is calculated according to the following formula

$$Suc = \frac{R_s}{R} \cdot 100\%, \tag{18}$$

where R_s is the number of runs realizing perfect hit and R is the total number of independent runs. The last metric is adopted to measure the complexity of the compound models evolved by these algorithms. Counting the node numbers and calculating the structural complexity of a tree [38,49] are two alternatives to quantify the complexity of a solution. In this paper, the total node numbers of an expression are computed to measure the complexity of a solution, as done in [33]. It should be noted that for algorithms which assign a weight for each feature, the weights are also included when calculating the total number of nodes.

For the sake of fair comparison, each algorithm is performed for 30 independent runs on each problem and the average results of the 30 runs are used for analysis. Since the computational complexity of these algorithms is different, some algorithms are quite time-consuming, while others run very fast. In order to fairly compare the search efficiency of different algorithms in a reasonable time range, for every independent run, each algorithm is allowed to run on the same computer for 10 min, because 10 min is enough for most algorithms to converge. If an algorithm finds a solution U_i whose fitness $F(U_i) < 10^{-6}$, a perfect hit is assumed, then this algorithm will terminate immediately. For each problem, the Wilcoxon signed-rank test is also done to check whether there are significant differences between MLDEP and its rivals. These experiments are done on a PC with an Intel(R) Core(TM)i7-8700 CPU @3.20GHz and 16.0GB RAM in single thread mode.

4.4 Results for algorithm comparison

4.4.1 Comparison of accuracy

The success rates and $NRMSE$ of the five algorithms are shown in Table 3. The last three rows of Table 3 summarize the results of the Wilcoxon signed-rank test on the problems. The $NRMSE$ is the average testing fitness of the best solution among 30 independent runs. It can be observed that the MLDEP outperforms its rivals on most problems in terms of success rate and $NRMSE$. Next we will analyze experimental results in detail.

The first comparison is among benchmark problems containing one variable. MLDEP achieves perfect hits on problems containing one variable and gets almost zero $NRMSE$ on the testing set. Especially, the success rates of the first two problems are 100%. Among other algorithms, only RDO_4 can achieve successful hits on problem Nguyen-5, but its success rate and testing $NRMSE$ are both worse than MLDEP. MRGP, EFS and LDEP find no perfect hit on all problems. MRGP and LDEP perform better than EFS and RDO_4 obviously, but they are still not as good as MLDEP. Therefore, the winner of this part is MLDEP.

For benchmark problems having more than one input variable, EFS, LDEP and MRGP can't find satisfying solutions, and only MLDEP can achieve successful hits on problem Keijzer-15. Before RDO_4 starting the evolution process, it needs to construct a library of subtrees constrained by a certain height. However, this preparing step is quite slow, so the $NRMSE$ and success rate of RDO_4 are labelled as TLE, which means the preparing step of RDO_4 exceeds the time limit. For problems on which no algorithms can achieve a perfect hit, the proposed MLDEP can always obtain the smallest $NRMSE$ except Korns-11. On problem Korns-11, MLDEP performs competitively to EFS and LDEP. In addition to the problem Korns-11, in terms of $NRMSE$, EFS ranks second on three problems (i.e., Keijzer-14, Keijzer-15, and Vladislavleva-8), while MRGP achieves the second place on two problems (i.e., Keijzer-5 and Vladislavleva-5). LDEP ranks third on Keijzer-5 and Vladislavleva-8, fourth on Keijzer-14 and Vladislavleva-5, and last on Keijzer-15. RDO_4 ranks third only on Keijzer-15, and last on other problems. As for the three real-world regression problems, MLDEP achieves better $NRMSE$ on problems WIR, while EFS performs better than MLDEP on ENC. For problem ENH, MLDEP and EFS get similar results. The results show that MLDEP performs better than or at least competitive to the other algorithms on the three real-world problems.

Generally, the above experimental results demonstrate that the proposed MLDEP outperforms other methods on most problems in terms of $NRMSE$ and Suc.

In addition to comparing with the above four algorithms. We also compares MLDEP with a recently published GP variant which has achieved good results on many datasets. This variant is based on RDO and uses semantic backpropagation synergy with linear scaling to improve the performance of GP [47]. To keep up with the original paper, this GP variant is abbreviated as sLS. Table 4 lists the datasets used to compare MLDEP and sLS. It should be noted that there are ten real-world benchmark datasets in [47], but two of them have been unavailable on the internet. So we use the rest eight datasets, covering different numbers of samples and dimensions. They are commonly used GP datasets and selected from the UCI machine learning repository[1]. For the sake of fairness, we ran MLDEP on the same datasets as sLS and used the same population size and generation number. Since several datasets have lots of samples and dimensions, we make a small modification here. That is if MLDEP can't reach the maximal generation number in an hour for one independent run, it will be terminated early. Except for this, parameter settings and evaluation metrics are the same. 30 independent runs are carried out and Table 5 compares the published results of sLS with MLDEP. It can be observed that MLDEP outperforms sLS on 6 out of 8 problems not only on the training set but also on the testing set. sLS is better than MLDEP mainly on benchmarks (Wr) and (Ww). In summary, MLDEP can find more accurate solutions than sLS on most datasets, which proves its excellent performance.

4.4.2 Comparison of convergence speed

This part compares the convergence speed of the five algorithms. Fig. 4 displays the convergence speed of the best solutions obtained by these algorithms among 30 runs. The horizontal axis represents time and the vertical axis shows the $NRMSE$ of the best individual. It should be noted that the $NRMSE$ is attained on the training set, since evaluating the best individual on testing set every generation is time-consuming for some algorithms especially when we restrict the total running time of each algorithm. Combining Fig. 4 and Table 3, the search efficiency of different algorithms can be analyzed.

The nine benchmark regression problems are studied first. It can be observed that MLDEP converges fastest on seven out of nine problems especially for Septic, Keijzer-1, Keijzer-15, Vladislavleva-8 and Keijzer-5 where MLDEP converges in the early stages of evolution. For problem Korns-11, the convergence speed of MRGP is the fastest but its generalization ability is not good enough and overfits on the testing set. The convergence speed of MLDEP, EFS and LDEP on Korns-11 are close. Actually, the performance of these algorithms has no much difference on Korns-11 in combination with Table 3. MLDEP and MRGP demonstrates similar convergence speed on Vladislavleva-5 and better than other algorithms. Besides

[1] https://archive.ics.uci.edu/ml/index.php.

Table 3 Testing accuracies of RDO_4, MRGP, EFS, LDEP and MLDEP on all problems

Problem	RDO_4		MRGP		EFS		LDEP		MLDEP	
	Suc	*NRMSE*	*Suc* (%)	*NRMSE*	*Suc* (%)	*NRMSE*	*Suc* (%)	*NRMSE*	*Suc* (%)	*NRMSE*
Septic	0%	0.2183 (−)	0	0.0160 (−)	0	0.6493 (−)	0	0.0506 (−)	**100**	**0.0000**
Nguyen-5	33.3%	0.0013 (−)	0	0.0014 (−)	0	0.1065 (−)	0	0.0782 (−)	**100**	**0.0000**
Keijzer-1	0%	0.97 (−)	0	0.0011 (−)	0	0.5691 (−)	0	0.0454 (−)	**56.7**	**0.0000**
Korns-11	TLE	TLE	0	12.6971 (−)	0	0.9991 (≈)	0	1.0186 (≈)	0	**0.9984**
Keijzer-14	0%	112.6 (−)	0	0.7955 (−)	0	0.2761 (≈)	0	27.4872 (−)	0	**0.2102**
Keijzer-15	0%	0.4829 (−)	0	2.2317 (−)	0	0.3725 (−)	0	5.2347 (−)	**100**	**0.0027**
Vladislavlev-8	0%	30.6 (−)	0	13.6824 (−)	0	0.4055 (−)	0	8.6747 (−)	0	**0.0002**
Keijzer-5	TLE	TLE	0	0.0035 (−)	0	0.3789 (−)	0	0.0471 (−)	0	**0.0001**
Vladislavleva-5	TLE	TLE	0	0.0874 (−)	0	0.3776 (−)	0	0.4392 (−)	0	**0.0179**
ENC	TLE	TLE	0	1.3633 (−)	0	**0.1521** (+)	0	0.3085 (−)	0	0.1615
ENH	TLE	TLE	0	0.5386 (−)	0	0.0490 (≈)	0	0.3455 (−)	0	**0.0486**
WIR	TLE	TLE	0	0.9967 (≈)	0	0.8064 (−)	0	0.9190 (−)	0	**0.7955**
−		6		11		8		11		
≈		0		1		3		1		
+		0		0		1		0		

Symbols −, ≈ and + in brackets represent that the competitor is respectively significantly worse than, similar to, and better than MLDEP according to the Wilcoxon signed-rank test at $\alpha = 0.05$. The best results are displayed in bold.

Table 4 Real-world benchmark datasets

(Abbreviation) Name	Examples	Features	Variance (o)
(A) Airfoil	1503	5	4.756×10
(B) Boston housing	506	13	8.442×10
(C) Concrete strength	1030	8	2.788×10^2
(Ec) Energy cooling	768	8	9.039×10
(Eh) Energy heating	768	8	1.017×10^2
(Wr) Wine red	1599	11	7.842×10^{-1}
(Ww) Wine white	4899	11	7.702×10^3
(Y) Yacht hydrodynamics	308	6	2.291×10^2

Table 5 Training and test median normalized MSE for MLDEP and sLS

Problem	Train $NMSE$		Test $NMSE$	
	MLDEP	sLS	MLDEP	sLS
A	**0.06**	0.22	**0.22**	0.29
B	**0.03**	0.14	0.20	**0.16**
C	**0.02**	0.15	**0.14**	0.18
Ec	**0.017**	0.049	**0.027**	0.054
Eh	**0.004**	0.054	**0.0024**	0.071
Wr	0.931	**0.6**	0.65	**0.62**
Ww	0.90	**0.67**	**0.68**	0.70
Y	**0.003**	0.004	**0.005**	0.0061

The better results are displayed in bold

the rapid convergence speed on the training set, the testing fitness of MLDEP is also better than its rivals according to the $NRMSE$ shown in Table 3. As mentioned before, RDO_4 oversteps the time limit at its preparation step, so it is omitted in the following comparison.

As for the real-world regression problems (i.e., ENC, ENH and WIR), LDEP is far behind other algorithms. MRGP converges a little faster than MLDEP and EFS on the training set, but it performs worst than MLDEP and EFS on the testing set. Hence, the generalization ability of MRGP is not as good as MLDEP and EFS. MLDEP and EFS perform similarly on the real-world problems in terms of accuracy and convergence speed. Though their convergence speed is slightly worse than MRGP, their testing accuracies are obviously better than MRGP. Thus, MLDEP and EFS perform best and competitively on real-world regression problems.

In summary, MLDEP converges fastest on seven out of nine benchmark regression problems. With regard to real-world regression problems, MLDEP and EFS are tied for the first place. Therefore, MLDEP wins this comparison of convergence speed. This is a result of using diverse features (i.e., polynomial features and evolved features) and the LM optimizer. With the help of diverse features, MLDEP has more possibilities to find the optimal solution. Given a set of features, the LM optimizer can find the most appropriate coefficient for each feature. These two modules work together in our framework to improve the search efficiency of GP.

4.4.3 Comparison of solution complexity

This section compares the solution complexity of all algorithms. MRGP is a variant of GP which reduces the complexity of solutions by considering accuracy and complexity at the same time. EFS cuts down the complexity of solutions by restricting the height of each subexpression and using the LASSO regression. The shortage of this method is that each subtree is too simple, the linear combination of simple features is not effective enough to solve problems whose objective functions contain complicated expressions. For example, to find function $f(x) = x^3e^{-x}cos(2x)sin(x)log(x)cos(x)$ which isn't the sum of more than one subexpression, each subtree must have sufficient expressive ability. Different from these two algorithms, the total length of each MLDEP program is restricted rather than the height of each subexpression. One advantage is that each subtree of MLDEP can represent more subtle features. In addition to that, the number of registers of MLDEP is also fixed and redundant features in MLDEP programs are filtered.

Table 6 demonstrates the average node numbers of 30 independent runs for all algorithms. It can be observed that the node numbers of different algorithms vary greatly on the same problem. The first algorithm used for comparison is RDO_4. Though RDO_4 uses more nodes than MLDEP on 1-dimensional benchmark problems, its accuracies and convergence speed are still worse than MLDEP. For 2-dimensional benchmark problems, MLDEP employs more nodes than RDO_4, in exchange for a significant increase in accuracies.

The second rival of MLDEP is MRGP, which optimizes accuracy and node numbers at the same time. In Table 6, the average node numbers of the most accurate individual found by MRGP among 30 runs are used for comparison. Compared with MLDEP, MRGP can find more concise but less accurate

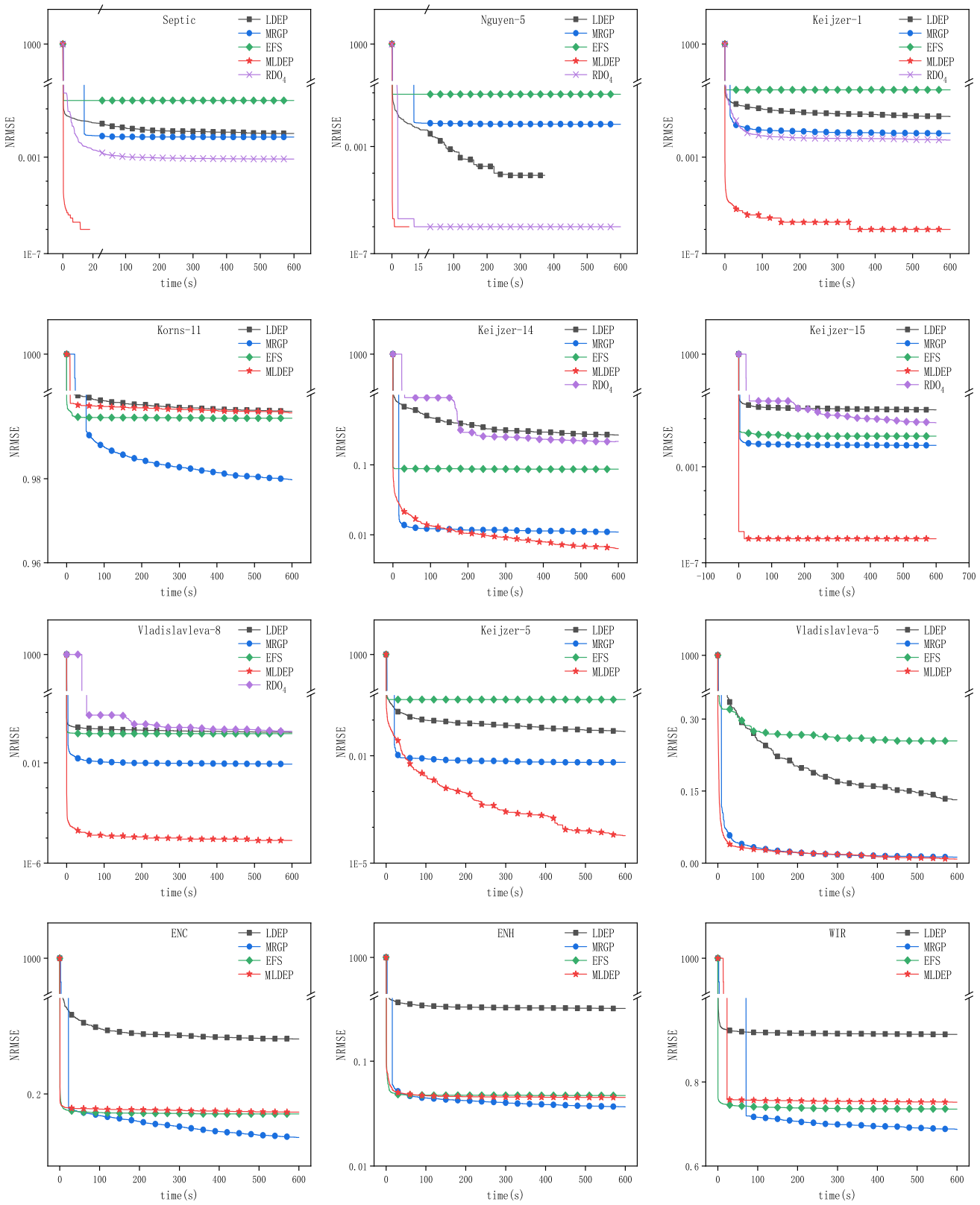

Fig. 4 Convergence trends of the five algorithms on training set with respect to all problems

Table 6 Node numbers of RDO_4, MRGP, EFS, LDEP and MLDEP

Problem	RDO_4	MRGP	EFS	LDEP	MLDEP
Septic	385.9	87.6	**20.8**	310.1	135.8
Nguyen-5	208.3	67.3	**18.1**	474.9	92.2
Keijzer-1	347.0	83.1	**14.1**	161.9	89.8
Korns-11	TLE	11,854.2	**94.3**	150.2	128.2
Keijzer-14	50.7	140.5	**37.8**	53.6	105.3
Keijzer-15	44.2	126.3	**36.7**	61.6	106.4
Vladislavleva-8	55.6	303.8	**37.6**	220.2	92.4
Keijzer-5	TLE	223.3	**51.8**	109.0	118.8
Vladislavleva-5	TLE	1043.0	**57.5**	114.1	124.0
EN	TLE	2998.1	**140.9**	146.3	161.7
ENH	TLE	2960.9	**134.3**	146.9	172.5
WIR	TLE	4449.2	210.5	**117.7**	233.8

The best results are displayed in bold

solutions on 1-dimensional problems. As problem dimensions increase, the node numbers of MRGP increase rapidly and overtake MLDEP obviously. But the testing accuracies of MRGP are still worse than MLDEP on most problems.

The third opponent is EFS. The node numbers of EFS are extraordinarily low due to the height restriction imposed on each feature and the usage of LASSO regression. In reality, the total number of features of EFS is proportional to the number of independent variables. Though the node numbers of EFS are smaller, the accuracies of EFS are the worst one on four out of nine benchmark problems. For the real-world problems, the node numbers of EFS are slightly less than MLDEP. Meanwhile, the accuracies of EFS and MLDEP are also close to each other.

The last competitor is LDEP. It can be observed that there is no obvious relationship between the number of nodes in LDEP and the dimension of the problems. Because LDEP only selects one register as the output register. As the problem dimension gets larger, the total number of registers in LDEP also increases. Though the problems become more complex as the dimension increases, the probability of each register being selected is reduced in the process of evolution. Thus, the node numbers of LDEP may not increase with the dimension of regression problems. For benchmark problems, MLDEP achieves higher accuracies and lower node numbers on 5 out of 9 problems comparing with LDEP. While for real-world regression problems, the node numbers of LDEP are smaller than MLDEP, but its accuracies are worse than MLDEP on all real-world problems.

Generally speaking, MLDEP ranks medium in terms of solution complexity among these algorithms. Considering its high precision and excellent search efficiency, we could fairly claim that MLDEP can find solutions of high accuracies and acceptable solution complexity in comparison with other state-of-the-art algorithms.

4.5 Component analysis

Our framework involves three important components: polynomial features, evolved features and the LM optimizer. In order to investigate the impacts of each module, we design three additional experiments for further explanation. For each experiment, the same Wilcoxon signed-rank test and notations as Table 3 are adopted to check whether the performance of variants of MLDEP is significantly different from that of MLDEP.

4.5.1 Impact of polynomial features

The proposed framework utilizes two kinds of features: polynomial features and evolved features. In this subsection, we investigate whether the polynomial features are necessary and useful to improve the search accuracy and efficiency of our framework. A simplified MLDEP is generated and named MLDEP/PolyF. The only difference between MLDEP/PolyF and MLDEP is that MLDEP/PolyF doesn't contain polynomial features. Table 7 shows the average testing accuracies and node numbers of the best individual among 30 runs of these two algorithms. According to Table 7, the performance of MLDEP and MLDEP/PolyF is similar on the nine benchmark problems since the evolved features are still used in MLDEP/PolyF, and the evolved features have excellent expressive abilities. But there are still differences between MLDEP and MLDEP/PolyF. In terms of $NRMSE$, readers can find that MLDEP outperforms the MLDEP/PolyF on three problems, two of which are complicated real-world regression problems. This shows that with the help of polynomial features, MLDEP can obtain better accuracies on complicated regression problems. As for the node numbers, MLDEP uses more nodes than MLDEP/PolyF on three real-world regression problems. It seems that MLDEP tends to link more features (i.e., polynomial and evolved features) to find more accurate solutions when solving complicated regression problems.

4.5.2 Impact of evolved features

This section explores the effectiveness of the evolved features in our framework. In order to validate it, another simplified algorithm called MLDEP/EvolvedF is generated to compete with the original MLDEP. The only difference between MLDEP/EvolvedF and MLDEP is that MLDEP/EvolvedF doesn't utilize the features evolved by LDEP. It means MLDEP/EvolvedF only combines the polynomial features. In this case, MLDEP/EvolvedF becomes a deterministic algorithm since it doesn't need the evolutionary

Table 7 Accuracies and node numbers of MLDEP and MLDEP/PolyF

Problem	MLDEP		MLDEP/PolyF	
	NRMSE	*Nodenums*	*NRMSE*	*Nodenums*
Septic	**0.0000**	**135.8**	**0.0000** (≈)	136.6 (≈)
Nguyen-5	**0.0000**	**92.2**	**0.0000** (≈)	96.2 (≈)
Keijzer-1	**0.0000**	**89.8**	**0.0000** (≈)	109.0 (≈)
Korns-11	0.9984	**128.2**	**0.9974** (≈)	131.6 (−)
Keijzer-14	**0.2102**	105.3	1.3678 (≈)	**97.3** (≈)
Keijzer-15	**0.0027**	106.4	0.0048 (−)	**97.9** (≈)
Vladislavleva-8	**0.0002**	**92.4**	0.0002 (≈)	92.9 (≈)
Keijzer-5	**0.0001**	**118.8**	**0.0001** (≈)	120.4 (≈)
Vladislavleva-5	0.0179	**124.0**	0.0275 (≈)	129.4 (≈)
ENC	**0.1616**	161.7	0.1670 (−)	**75.2** (+)
ENH	**0.0486**	172.5	0.0609 (−)	**75.3** (+)
WIR	**0.7955**	233.8	0.7959 (≈)	**87.4** (+)
−			3	1
≈			9	8
+			0	3

The better results are displayed in bold

Table 8 Accuracies and node numbers of MLDEP and MLDEP/EvolvedF

Problem	MLDEP		MLDEP/EvolvedF	
	NRMSE	*NodeNums*	*NRMSE*	*NodeNums*
Septic	**0.0000**	135.8	0.6425 (−)	**17.0** (+)
Nguyen-5	**0.0000**	92.2	0.3708 (−)	**17.0** (+)
Keijzer-1	**0.0000**	89.8	0.7057 (−)	**9.0** (+)
Korns-11	**0.9984**	128.2	0.9997 (−)	**73.0** (+)
Keijzer-14	**0.2102**	105.3	0.6831 (−)	**31.0** (+)
Keijzer-15	**0.0027**	106.4	0.2791 (−)	**31.0** (+)
Vladislavleva-8	**0.0002**	92.4	0.5054 (−)	**31.0** (+)
Keijzer-5	**0.0001**	118.8	1.0036 (−)	**45.0** (+)
Vladislavleva-5	**0.0179**	124.0	1.0142 (−)	**45.0** (+)
ENC	**0.1616**	161.7	0.2089 (−)	**107.0** (+)
ENH	**0.0486**	172.5	0.1938 (−)	**107.0** (+)
WIR	**0.7955**	233.8	0.8065 (−)	**157.0** (+)
−			12	0
≈			0	0
+			0	12

The better results are displayed in bold

process. Then running once is enough for MLDEP/EvolvedF. The testing accuracies and node numbers of MLDEP are still the average value of 30 independent runs of the best solution. Table 8 lists the results. It can be observed that the testing accuracies of MLDEP are much better than MLDEP/EvolvedF, which indicates that evolved features play an important role in finding more complex structures. Correspondingly, the node numbers of MLDEP/EvolvedF are smaller than those of MLDEP, which is expected in that MLDEP/EvolvedF only contains the polynomial features. It is worthwhile to improve the accuracies obviously at the cost of a moderate increase of node numbers. Thus, evolved features are meaningful to improve the performance of GP.

4.5.3 Impact of LM optimizer

This section assesses the effectiveness of the LM optimizer in our framework. Similar to the previous two sections, a new variant of MLDEP is created. The variant is called MLDEP/LM. The only difference is that the weight of each

Table 9 Accuracies and node numbers of MLDEP and MLDEP/LM

Problem	MLDEP		MLDEP/LM	
	NRMSE	*NodeNums*	*NRMSE*	*NodeNums*
Septic	**0.0000**	135.8	0.0938 (−)	**103.6** (+)
Nguyen-5	**0.0000**	**92.2**	0.0776 (−)	106.8 (−)
Keijze-1	**0.0000**	**89.8**	0.4935 (−)	94.1 (≈)
Korns-11	0.9984	128.2	11,829.0 (−)	**127.4** (≈)
Keijzer-14	**0.2102**	105.3	735.3 (−)	**91.0** (+)
Keijzer-15	**0.0027**	106.4	1.6240 (−)	**89.6** (+)
Vladislavleva-8	**0.0002**	**92.4**	2.2190 (−)	98.1 (≈)
Keijzer-5	**0.0001**	118.8	1.2139 (−)	**103.1** (+)
Vladislavleva-5	**0.0179**	124.0	1.8160 (−)	**107.2** (+)
ENC	**0.1616**	**161.7**	52,5396.2 (−)	199.5 (−)
ENH	**0.0486**	**172.5**	619,159.2 (−)	191.5 (−)
WIR	**0.7955**	**233.8**	18,185.1 (−)	250.1 (−)
−			12	4
≈			0	3
+			0	5

The better results are displayed in bold

independent feature in MLDEP/LM is set to 1. Redundant features in MLDEP/LM are also filtered. Table 9 shows the average testing accuracies and node numbers of the best individual among 30 independent runs of these two algorithms. From Table 9, it can be observed that MLDEP demonstrates much better performance than MLDEP/LM. MLDEP outperforms its rival on all problems in terms of accuracies in that the weight of each feature reflects the importance of each feature, which is essential to forming good solutions. With regard to node numbers, MLDEP and MLDEP/LM each wins on half of the problems. The above experimental results demonstrate that the LM optimizer is effective to improve the accuracy of GP.

5 Conclusions and future work

In this paper, a novel memetic genetic programming framework is proposed to improve the accuracy and search efficiency of GP on complicated symbolic regression problems. In this framework, a GP solution is regarded as the linear combination of polynomial features and evolved features. In order to improve the simplicity of final solutions, redundant features are filtered before the weight optimization step. A gradient-based solver called LM is adopted to link these features by assigning each feature a weight. A GP variant named LDEP is integrated into this framework to form a new algorithm named MLDEP. Experimental results demonstrate that the proposed MLDEP offers enhanced performance over other four state-of-the-art algorithms in terms of accuracy and search efficiency. There are still several interesting research directions. Some advanced techniques can be used to reduce the node numbers further. Another direction is to explore an adaptive manner to do LM optimization rather than perform it for every individual, so as to save computing resources.

Acknowledgements This work is supported by the Program for Guangdong Introducing Innovative and Entrepreneurial Teams (Grant No. 2017ZT07X183), the Guangdong Natural Science Foundation Research Team (Grant No. 2018B030312003), and the Fundamental Research Funds for the Central Universities (Grant No. D2191200).

References

1. Arnaldo I, Krawiec K, O'Reilly UM (2014) Multiple regression genetic programming. In: Proceedings of the 2014 annual conference on genetic and evolutionary computation. ACM, pp 879–886
2. Arnaldo I, O'Reilly UM, Veeramachaneni K (2015) Building predictive models via feature synthesis. In: Proceedings of the 2015 annual conference on genetic and evolutionary computation. ACM, pp 983–990
3. Barrero, DF (2011) Relibility of performance measures in tree-based genetic programming: a study on Koza's computational effort. Ph.D. thesis, School of Computing of the University of Alcala
4. Beadle L, Johnson CG (2008) Semantically driven crossover in genetic programming. In: IEEE congress on evolutionary computation. IEEE, pp 111–116
5. Beadle L, Johnson CG (2009) Semantically driven mutation in genetic programming. In: IEEE congress on evolutionary computation. IEEE, pp 1336–1342
6. Brameier MF, Banzhaf W (2007) Linear genetic programming. Springer, Berlin
7. Chen Q, Xue B, Zhang M (2018) Improving generalisation of genetic programming for symbolic regression with angle-driven

geometric semantic operators. IEEE Trans Evolut Comput. https://doi.org/10.1109/TEVC.2018.2869621
8. Chen Q, Zhang M, Xue B (2017) Feature selection to improve generalization of genetic programming for high-dimensional symbolic regression. IEEE Trans Evolut Comput 21(5):792–806. https://doi.org/10.1109/TEVC.2017.2683489
9. Chen X, Ong Y, Lim M, Tan KC (2011) A multi-facet survey on memetic computation. IEEE Trans Evolut Comput 15(5):591–607. https://doi.org/10.1109/TEVC.2011.2132725
10. Cortez P, Cerdeira A, Almeida F, Matos T, Reis J (2009) Modeling wine preferences by data mining from physicochemical properties. Decis Support Syst 47(4):547–553
11. Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Trans Evolut Comput 6(2):182–197
12. Eremeev AV, Kovalenko YV (2019) A memetic algorithm with optimal recombination for the asymmetric travelling salesman problem. Memet Comput 12(1):23–36
13. Espejo PG, Ventura S, Herrera F (2010) A survey on the application of genetic programming to classification. IEEE Trans Syst Man Cybern C (Appl Rev) 40(2):121–144
14. Fenton M, Lynch D, Kucera S, Claussen H, O'Neill M (2017) Multilayer optimization of heterogeneous networks using grammatical genetic programming. IEEE Trans Cybern 47(9):2938–2950. https://doi.org/10.1109/TCYB.2017.2688280
15. Ferreira C (2001) Gene expression programming: a new adaptive algorithm for solving problems. arXiv preprint arXiv:cs/0102027
16. Fonlupt C, Robilliard D, Marion-Poty V (2011) Linear imperative programming with differential evolution. In: 2011 IEEE symposium on differential evolution (SDE), pp 1–8. https://doi.org/10.1109/SDE.2011.5952066
17. Hinchliffe M, Hiden H, McKay B, Willis M, Tham M, Barton G (1996) Modelling chemical process systems using a multi-gene genetic programming algorithm. In: Koza JR (ed) Late breaking papers at the genetic programming 1996 conference Stanford University July 28–31, 1996. Stanford Bookstore, Stanford University, CA, USA, pp 56–65
18. Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4598):671–680
19. Kommenda M, Kronberger G, Winkler S, Affenzeller M, Wagner S (2013) 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, pp 1121–1128
20. Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection, vol 1. MIT press, Cambridge
21. Krawiec K, Lichocki P (2009) Approximating geometric crossover in semantic space. In: Proceedings of the 11th annual conference on genetic and evolutionary computation. ACM, pp 987–994
22. Marquardt DW (1963) An algorithm for least-squares estimation of nonlinear parameters. J Soc Ind Appl Math 11(2):431–441
23. McConaghy T (2011) Ffx: fast, scalable, deterministic symbolic regression technology. In: Genetic programming theory and practice IX. Springer, pp 235–260
24. McDermott J, White DR, Luke S, Manzoni L, Castelli M, Vanneschi L, Jaskowski W, Krawiec K, Harper R, De Jong K (2012) Genetic programming needs better benchmarks. In: Proceedings of the 14th annual conference on genetic and evolutionary computation. ACM, pp 791–798
25. Meuth R, Lim MH, Ong YS, Wunsch DC (2009) A proposition on memes and meta-memes in computing for higher-order learning. Memet Comput 1(2):85–100
26. Miller JF (2011) Cartesian genetic programming. Springer, Berlin, pp 17–34
27. Moraglio A, Krawiec K, Johnson CG (2012) Geometric semantic genetic programming. In: International conference on parallel problem solving from nature. Springer, pp 21–31
28. Muñoz L, Trujillo L, Silva S, Castelli M, Vanneschi L (2019) Evolving multidimensional transformations for symbolic regression with m3gp. Memet Comput 11(2):111–126
29. Nguyen QU, Nguyen XH, O'Neill M (2009) Semantic aware crossover for genetic programming: the case for real-valued function regression. In: European conference on genetic programming. Springer, pp 292–302
30. Nguyen S, Zhang M, Johnston M, Tan KC (2015) Automatic programming via iterated local search for dynamic job shop scheduling. IEEE Trans Cybern 45(1):1–14. https://doi.org/10.1109/TCYB.2014.2317488
31. Nguyen S, Zhang M, Tan KC (2017) Surrogate-assisted genetic programming with simplified models for automated design of dispatching rules. IEEE Trans Cybern 47(9):2951–2965. https://doi.org/10.1109/TCYB.2016.2562674
32. Orzechowski P, Cava WL, Moore JH (2018) Where are we now? A large benchmark study of recent symbolic regression methods. CoRR arXiv:1804.09331
33. Pawlak TP, Wieloch B, Krawiec K (2015) Semantic backpropagation for designing search operators in genetic programming. IEEE Trans Evolut Comput 19(3):326–340. https://doi.org/10.1109/TEVC.2014.2321259
34. Price K, Storn R (1995) Differential evolution-a simple and efficient adaptive scheme for global optimization over continuous space. Technical report, International Computer Science Institue, Berkley
35. Ryan C, Keijzer M (2003) An analysis of diversity of constants of genetic programming. In: European conference on genetic programming. Springer, pp 404–413
36. Schmidt M, Lipson H (2009) Distilling free-form natural laws from experimental data. Science 324(5923):81–85. https://doi.org/10.1126/science.1165893
37. Searson DP, Leahy DE, Willis MJ (2010) Gptips: an open source genetic programming toolbox for multigene symbolic regression. In: Proceedings of the international multiconference of engineers and computer scientists, vol 1. Citeseer, pp 77–80
38. Smits GF, Kotanchek M (2005) Pareto-front exploitation in symbolic regression. In: Genetic programming theory and practice II. Springer, pp 283–299
39. Suganuma M, Shirakawa S, Nagao, T (2017) A genetic programming approach to designing convolutional neural network architectures. In: Proceedings of the genetic and evolutionary computation conference, pp 497–504
40. Tan LT, Chen WN, Zhang J (2018) A histogram estimation of distribution algorithm for resource scheduling. In: Proceedings of the genetic and evolutionary computation conference companion. ACM, pp 143–144
41. Tibshirani R (1996) Regression shrinkage and selection via the lasso. J R Stat Soc Ser B (Methodol) 58:267–288
42. Topchy A, Punch WF (2001) Faster genetic programming based on local gradient search of numeric leaf values. In: Proceedings of the 3rd annual conference on genetic and evolutionary computation. Morgan Kaufmann Publishers Inc., pp 155–162
43. Tsanas A, Xifara A (2012) Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. Energy Build 49:560–567
44. Uy NQ, Hoai NX, O'Neill M (2009) Semantics based mutation in genetic programming: the case for real-valued symbolic regression. In: 15th International conference on soft computing, Mendel, vol 9, pp 73–91
45. Uy NQ, Hoai NX, O'Neill M, McKay RI, Galván-López E (2011) Semantically-based crossover in genetic programming: application

to real-valued symbolic regression. Genet Program Evolv Mach 12(2):91–119
46. Vanneschi L, Mauri G, Valsecchi A, Cagnoni S (2006) Heterogeneous cooperative coevolution: strategies of integration between gp and ga. In: Proceedings of the 8th annual conference on genetic and evolutionary computation. ACM, pp 361–368
47. Virgolin M, Alderliesten T, Bosman PAN (2019) Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression. In: Proceedings of the genetic and evolutionary computation conference, GECCO 2019, Prague, Czech Republic, July 13–17, 2019, pp 1084–1092
48. Vladislavleva E, Smits G, Den Hertog D (2010) On the importance of data balancing for symbolic regression. IEEE Trans Evolut Comput 14(2):252–277
49. Vladislavleva EJ, Smits GF, den Hertog D (2009) 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. https://doi.org/10.1109/TEVC.2008.926486
50. Wieloch B, Krawiec K (2013) Running programs backwards: instruction inversion for effective search in semantic spaces. In: Proceedings of the 15th annual conference on genetic and evolutionary computation
51. Ong YS, Keane AJ (2004) Meta-lamarckian learning in memetic algorithms. IEEE Trans Evolut Comput 8(2):99–110. https://doi.org/10.1109/TEVC.2003.819944
52. Ong Y-S, Lim M-H, Zhu N, Wong K-W (2006) Classification of adaptive memetic algorithms: a comparative study. IEEE Tran Syst Man Cybern B (Cybern) 36(1):141–152. https://doi.org/10.1109/TSMCB.2005.856143
53. Zhang Q, Zhou C, Xiao W, Nelson PC (2007) Improving gene expression programming performance by using differential evolution. In: Sixth international conference on machine learning and applications (ICMLA 2007). IEEE, pp 31–37
54. Zhong J, Feng L, Ong Y (2017) Gene expression programming: a survey. IEEE Comput Intell Mag 12(3):54–72. https://doi.org/10.1109/MCI.2017.2708618
55. Zhong J, Ong YS, Cai W (2016) Self-learning gene expression programming. IEEE Trans Evol Comput 20(1):65–80