IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 23, 2020, accepted April 12, 2020, date of publication April 17, 2020, date of current version May 4, 2020.

*Digital Object Identifier 10.1109/ACCESS.2020.2988587*

# Information Fusion in Offspring Generation: A Case Study in Gene Expression Programming

**TONGLIN LIU[1], HENGZHE ZHANG[2], HU ZHANG[1], AND AIMIN ZHOU[2], (Senior Member, IEEE)**

[1]Science and Technology on Complex System Control and Intelligent Agent Cooperation Laboratory, Beijing Electro-Mechanical Engineering Institute, Beijing 100074, China
[2]Shanghai Key Laboratory of Multidimensional Information Processing, School of Computer Science and Technology, East China Normal University, Shanghai 200062, China

Corresponding author: Hu Zhang (jxzhanghu@126.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 61703382, Grant 51875053, and Grant 61673180, in part by the China Ministry of Science and Technology Key Research and Development Program under Grant 2018YFC1903101, in part by the Science Foundation of Science and Technology on Complex System Control and Intelligent Agent Cooperative Laboratory under Grant 181601, and in part by the Science and Technology Commission of Shanghai Municipality under Grant 19511120600.

**ABSTRACT** Gene expression programming (GEP), which is a variant of genetic programming (GP) with a fixed-length linear model, has been applied in many domains. Typically, GEP uses genetic operators to generate offspring. In recent years, the estimation of distribution algorithm (EDA) has also been proven to be efficient for offspring generation. Genetic operators such as crossover and mutation generate offspring from an implicit model by using the individual information. By contrast, EDA operators generate offspring from an explicit model by using the population distribution information. Since both the individual and population distribution information are useful in offspring generation, it is natural to hybrid EDA and genetic operators to improve the search efficiency. To this end, we propose a hybrid offspring generation strategy for GEP by using a univariate categorical distribution based EDA operator and its original genetic operators. To evaluate the performance of the new hybrid algorithm, we apply the algorithm to ten regression tasks using various parameters and strategies. The experimental results demonstrate that the new algorithm is a promising approach for solving regression problems efficiently. The GEP with hybrid operators outperforms the original GEP that uses genetic operators on eight out of ten benchmark datasets.

**INDEX TERMS** Genetic programming, gene expression programming, estimation of distribution algorithm, offspring generation.

## I. INTRODUCTION

Gene expression programming (GEP) [1] is an evolution-based algorithm for solving symbolic regression problems. Comparing with traditional machine learning algorithms, such as the neural network, GEP evolves interpretable symbolic expressions without calculating gradients. Although lack of pervasive theoretical performance studies, the distinct features of GEP also draw extensive attention from multiple disciplines to build an explainable model, such as in energy management [2], economic modeling [3], and power distribution [4] domains. GEP is similar to the tree-based genetic programming algorithm (GP) [5], which both generate a symbolic expression under the framework of evolutionary algorithms (EAs). However, in contrast with GP, GEP has a fixed-length gene sequence and an unconstrained search space through a genotype-phenotype mapping mechanism [6]. This advantage enables GEP to produce more satisfactory results in diverse domains. For example, in the water resources management domain, substantial improvement has founded by replacing GP with GEP [7]. In other domains, such as in rock mechanics [8] and data mining [9] domains, similar results have also been obtained.

GEP uses a linear representation to represent a variety of programs. The linear gene in GEP is composed of a head and a tail. The head contains function primitives and terminal primitives, and the tail only contains terminal primitives. Fig.1 presents an example of the GEP expression.

In GEP, computer program functions are usually considered as function primitives, which can accept several

The associate editor coordinating the review of this manuscript and approving it for publication was Li Zhang.

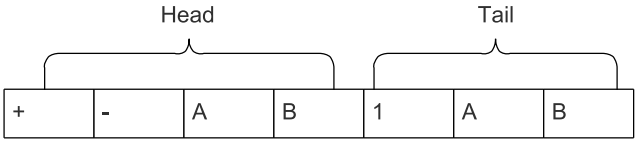



**FIGURE 1.** An example of a gene expression in GEP.

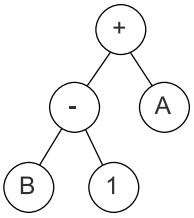



**FIGURE 2.** A valid expression tree converted by a gene expression.

parameters and return a result after operation. Variables and constants are regarded as terminal primitives. Hence, all function primitives are located at non-leaf nodes in the gene expression tree, and all terminal primitives are located at leaf nodes in the gene expression tree.

In GEP, two parameters, the head length and the tail length, are used to determine the size of a gene expression. The head length can be set to any natural number $h \in \mathbb{N}$. However, the tail length $t$ can not be set freely because it is calculated from the head length as (1):

$$t = h * (a - 1) + 1 \tag{1}$$

where $t$ represents the tail length, $h$ represents the head length, and $a$ represents the maximum arity of the function primitives. In Fig. 1, the maximum arity functions are binary operators "+" and "−". Hence, under this circumstance, the maximum arity is 2.

By this mechanism, each symbolic expression tree can be converted to a linear gene expression, and each linear gene expression can be converted to a valid symbolic expression tree. For example, the gene expression in Fig.1 can be converted to an expression tree as illustrated in Fig.2.

One of the main contributions of GEP is its representation, which uses a fixed-length gene expression to represent a variety of symbolic expressions. This representation directly leads to an advantage in the offspring generation. As representation with a fixed-length gene expression, many offspring generation operators can be applied. Basically, there are two kinds of operators that have widely used in GEP.

The first one is genetic operators, such as crossover and mutation operators. These kinds of operators choose one or two parent individuals, exchange the gene components between parent individuals or vary the gene components in a single parent individual. In traditional GEPs [10], various mutation operators and crossover operators are proposed for generating offspring. These operators increase the diversity of the population. A random numerical constant (RNC) array has also been proposed for improving the constant fitting efficiency. However, genetic operators are fixed during the evolution process. In real practice, an adaptive operator to accommodate specific problems is more desired to achieve better performance than traditional genetic operators. EDA operator is suitable for this requirement [11].

Therefore, the second one is the EDA operator. This kind of operator builds a probabilistic model to extract the population distribution information from the population, and sample new offspring solutions from the probabilistic model. In the community of GP, EDA has been widely used. Probabilistic incremental program evolution (PIPE) [12] is the first algorithm that uses a probabilistic model to generate solutions. Subsequently, extended compact genetic programming (ECGP) [13] was proposed, which incorporates the strategy of extended compact genetic algorithm (ECGA) [14] into GP. The Bayesian network [15] has also been used to integrate with the traditional genetic algorithm in the GP algorithm. The experimental results proved that using the integrated approach outperforms using both algorithms separately [16]. However, it needs much effort to construct a Bayesian network, especially when the structure of the Bayesian network is needed to learn from data. Therefore, in the EDA domain, traditional statistical models are still widely used [17], [18]. Recently, scholars have tried to apply EDA to GEP to realize improved performance. Probabilistic developmental program evolution (PDPE) [19] has been proposed, which uses a probabilistic model to generate offspring. However, [19] only explores the effects of a probabilistic model based GEP offspring generation algorithm, and the effects of the hybrid algorithm based on the genetic algorithm (GA) and the estimation of distribution algorithm (EDA) has not been discussed. Although the EDA operator reveals many impressive results, some drawbacks have been founded in real practice. A significant shortcoming of the EDA algorithm is that it tends to be trapped in a local minimum [20]. Another problem that might be encountered in EDA-GP is the stochastic sampling drift problem, which impedes the algorithm to achieve superior performance due to the loss of the population diversity [21].

Both genetic operators and EDA operators show their advantages and disadvantages in GEP offspring generation. Therefore, it is natural to combine both of these two kinds of operators in GEP. By integrating the EDA operator and genetic operators, the genetic algorithm will generate diverse solutions [20], and EDA can fully leverage the information in the evolutionary process to intensification the evolution process. These characteristics of the hybrid algorithm are important for the optimization process in the genetic programming domain, due to the multimodal landscape property of genetic programming.

As discussed in our previous work [17], the key issue is how to hybrid the two different kinds of offspring generation operators. There are three ways to do so, i.e., hybrid on the population level, the individual level, or the chromosome level. In this paper, we propose a hybrid offspring generation strategy on the individual level to give full play to the advantages of those two kinds of operators. In the new

approach, new offspring are partly sampled from a categorical distribution and partly generated via genetic operators. The new approach has been applied to ten PMLB regression datasets, and the experimental results demonstrate that the new approach outperforms the traditional GEP algorithm.

The main contributions of this paper are as follows:

- We hybrid EDA and GA offspring generation operators to enhance the performance of GEP. Our experiments demonstrate that the hybrid algorithm outperforms both EDA and GA based GEP algorithms.
- We design two sampling strategies and a standardization strategy for the EDA operator. By conducting experiments on ten benchmark datasets, these strategies exhibit different characteristics, and the best policy outperforms GA-GEP on eight out of ten datasets.

The remainder of this paper is organized as follows: Section II introduces the algorithm framework and the details of the individual evaluation, the probabilistic model, and the offspring generation. Section III presents some experiments on ten datasets for evaluating the performance of our proposed approach. We also discuss the impacts of the parameters on algorithm performance. Section IV summarizes the paper and discusses future works.

## II. ALGORITHM

In this section, we introduce the main algorithm, which involves two offspring generation strategies. The main components, including the individual evaluation, model definition, model initialization, model updating, and model sampling, are presented in detail.

### A. ALGORITHMIC FRAMEWORK

In this section, we propose a hybrid GEP algorithm ($GEP_H$), which fuses information at the individual level. In other words, in each generation, some offspring are sampled from the probabilistic model, and the others are generated by applying genetic operators to some selected individuals. The pseudo-code of the $GEP_H$ algorithm is presented in Algorithm 1, and the flowchart is shown in Fig.3.

- Population initialization: In line 1, the population $X$ is randomly initialized by randomly sampling in the search space.
- Model initialization: In line 2, the probabilistic model $P$ is initialized as a uniform distribution.
- Individual evaluation: In line 3 and 21, the individual fitness $f_x$ is evaluated in terms of the mean squared error.
- Model updating: In line 5, each value $p$ in the EDA model $P$ is updated according to the fitness values.
- EDA offspring generation: In line 7-10, EDA offspring $X_e$ are generated by sampling from the probabilistic model $P$. $|X_e|$ represents the number of individuals generated by the EDA model.
- Parent selection: In line 11, GA uses the tournament selection operator [22] to select promising individuals for generating the population of the next generation. $|X_g|$ represents the number of individuals generated by genetic operators.
- GA offspring generation: In line 12-19, offspring $X_g$ are generated by applying crossover and mutation operators to the selected promising individuals.
- Offspring merge: In line 20, EDA offspring and GA offspring are merged to generate a new population $X$.

**Algorithm 1** Hybrid GEP Algorithm

1: $X \leftarrow$ population_initialization()
2: $P \leftarrow$ categorical_distribution initialization()
3: $\{f_x \mid x \in X\} \leftarrow$ evaluation(X)
4: **while** iteration $<$ max_iteration **do**
5: $\quad P \leftarrow$ updating$(P, \{f_x \mid x \in X\})$ ▷ Model updating
6: $\quad X_e = \emptyset$
7: $\quad$ **for** $i = 1, \cdots, |X_e|$ **do** ▷ EDA generation
8: $\quad\quad x \leftarrow$ sampling$(P)$
9: $\quad\quad X_e \leftarrow X_e \cup \{x\}$
10: $\quad$ **end for**
11: $\quad X_g \leftarrow$ selection$(X, |X_g|)$ ▷ Parent selection
12: $\quad$ **for** $i = 1, \cdots, |X_g|$ **do** ▷ GA generation
13: $\quad\quad$ **if** $i\%2 == 0$ & $rand() < pm$ **then**
14: $\quad\quad\quad X_g^{i+1}, X_g^i \leftarrow$ crossover$(X_g^i, X_g^{i+1})$
15: $\quad\quad$ **end if**
16: $\quad\quad$ **if** $rand() < pc$ **then**
17: $\quad\quad\quad X_g^i \leftarrow$ mutation$(X_g^i)$
18: $\quad\quad$ **end if**
19: $\quad$ **end for**
20: $\quad X \leftarrow X_e \cup X_g$
21: $\quad \{f_x \mid x \in X\} \leftarrow$ evaluation(X)
22: **end while**
23: **return** X

### B. INDIVIDUAL EVALUATION

#### 1) INDIVIDUAL EVALUATION

This section introduces the individual evaluation process. This evaluation process is the same as the traditional evaluation process. For a k-dimensional input and a one-dimensional output dataset $D = (z, y; z \in R^k, y \in R)$, the basic evaluation function of an individual $x$ is defined as:

$$f_x = -Loss(E(z), Y) \tag{2}$$

This function is similar to the traditional machine learning evaluation function. We use a negative sign before the loss value because our target is to maximize the fitness value. $E$ represents the symbolic expression of $x$, and $E(z)$ output a scalar value presents the predicted value of the specific task. *Loss* represents the loss function. For the regression problem, the mean absolute error (MAE) or the mean square error (MSE) can be used as a reasonable loss function [23].

In the remaining paragraphs, MSE is used as the evaluation metric. MSE is defined as follow:

$$MSE(E(z), Y) = (E(z) - Y)^2 \tag{3}$$

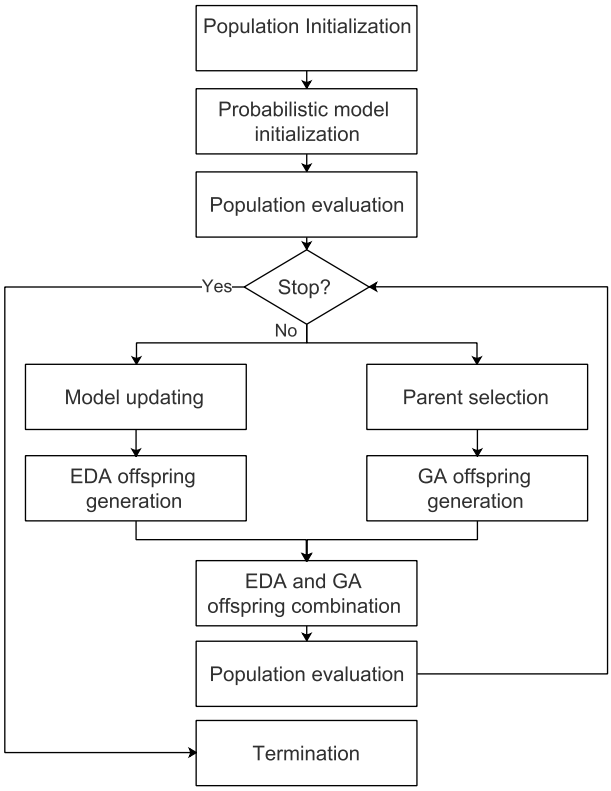



**FIGURE 3.** The flowchart of the hybrid GEP algorithm.

### 2) FITNESS STANDARDIZATION

Since the value of the loss function depends on the training data, the magnitude of the training data will influence the fitness value. This will cause the updating scale of the probabilistic model becomes uncontrollable. Therefore, a fitness value standardization function can be applied to the fitness value. The standardization function is defined as below:

$$f_x := \begin{cases} \frac{f_x - \mu_f}{\sigma_f} & \text{if } \sigma_f > 0 \\ 0 & \text{if } \sigma_f == 0 \end{cases} \tag{4}$$

where $\mu_f$ represents the mean value of fitness values, i.e., $\mu_f = \frac{\sum_{x \in X} f_x}{|X|}$. $\sigma_f$ represents the variance value of fitness values, i.e., $\sigma_f = \frac{\sum_{x \in X} (f_x - \mu_f)^2}{|X|}$.

When the standard deviation of fitness values is zero, the fitness values will not be standardized again. All fitness values should be set as zero in this scenario to avoid updating the probabilistic model. After the standardization, the standardized fitness values satisfy a Gaussian distribution $fitness \sim \mathcal{N}(0,\ 1)$ .

Fitness standardization is not an essential part of the model update process. The effects of fitness standardization are empirically studied in the following section.

## C. PROBABILISTIC MODEL

### 1) MODEL DEFINITION

The algorithm uses a group of univariate categorical distributions as the distribution model of programs. The group of categorical distributions is defined as $Cat(P)$, and $P$ is the probability matrix, which represents the likelihood of primitives on gene points. In the probability matrix $P$, $p_{ij}$ represents the likelihood to realize excellent performance when applying the j-th primitive at the i-th point.

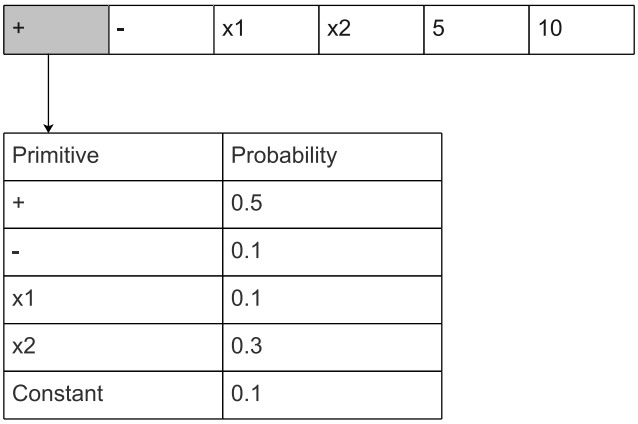



**FIGURE 4.** An illustration of gene points sampled from a categorical distribution in $GEP_H$.

Applying the univariate categorical distribution as the probabilistic model means that we do not consider the dependency between gene points. Due to each gene expression has several gene points, we use a group of univariate categorical distributions as the distribution model. For example, the categorical distribution can be represented as Fig.4. Each gene point is sampled from a categorical distribution, and multiple categorical distributions constitute an estimate of the distribution model.

### 2) MODEL INITIALIZATION

The categorical distribution is initialized as a uniform distribution since we have no prior knowledge of the target problems. For example, if the number of primitives is 5, then we can initialize the categorical distribution as in Fig.5.

The initialization of the probability distribution can be expressed as:

$$p_{ij} = \frac{1}{primitive\_amount} \tag{5}$$

where *primitive_amount* represents the number of primitives.

### 3) MODEL UPDATING

In the model updating process, the probability distribution is updated according to the fitness values. The update process is described as the below formula:

$$p_{ij} := p_{ij} + \sum_{i=1}^{n} \eta * f_x \tag{6}$$

where $\eta$ represents the learning rate of the updating process, and $f_x$ is the standardized fitness value.

After updating the probabilistic model, some of the probability values might be less than zero. This will cause trouble in

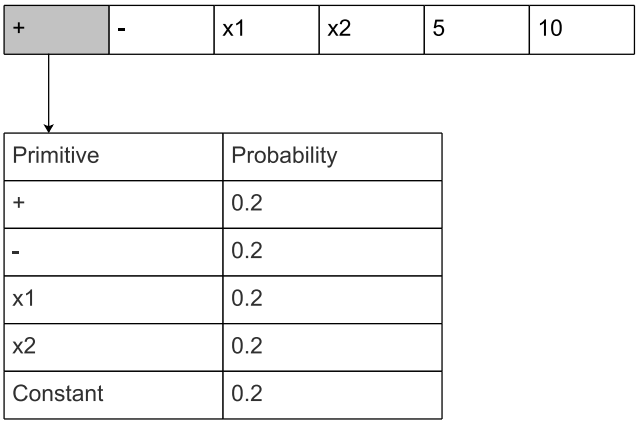



**FIGURE 5.** An illustration of initialization of the categorical distribution in $GEP_H$.

the offspring generation process. Therefore, repair is applied to ensure that the probability is larger than zero.

$$p_{ij} := p_{ij} - \min_{j=1}^{m} p_{ij} \tag{7}$$

In the following sections, $p_{ij}$ represents the repaired fitness value. After this step, each fitness value is guaranteed to be greater than zero.

### D. OFFSPRING GENERATION

After model initialization and model updating, offspring can be generated by applying the genetic operators and the EDA operators.

#### 1) GA OFFSPRING GENERATION

In the genetic algorithm, crossover and mutation operators are used to generate offspring. Suppose there are two gene expressions $x^1 = (x_1^1, x_2^1, x_3^1 ... x_n^1)$ and $x^2 = (x_1^2, x_2^2, x_3^2 ... x_n^2)$, and a random integer $r \in [1, n]$ represents the crossover or the mutation point location. A typical crossover operation used in GEP is the single point crossover. By applying this operator, the gene points before the location $r$ are exchanged, and the result is $x^1 = (x_1^2, x_2^2, x_3^2 ... x_r^2, x_{r+1}^1 ... x_n^1)$ $x^2 = (x_1^1, x_2^1, x_3^1 ... x_r^1, x_{r+1}^2 ... x_n^2)$. A typical mutation operation used in GEP is the single point mutation. By applying this operator, the gene point at the location $r$ is mutated to a new value $x'^1_r$. Then, the new gene expression is $x^1 = (x_1^1, x_2^1, x_3^1 ... x'^1_r ... x_n^1)$. The crossover process of the single point crossover operator and the mutation process of the single point mutation operator are illustrated in Fig.6 and Fig.7.

#### 2) EDA OFFSPRING GENERATION

Since the new gene expression is composed of multiple primitives, generating a new offspring is equivalent to sampling new primitives from the categorical distributions.

Suppose a primitive $s_i$ is sampled from a categorical distribution $s_i \sim p_i$. Then, this primitive should be placed at location $i$ in the gene expression. For example, suppose the categorical distribution at the first point is plotted as Fig.4.

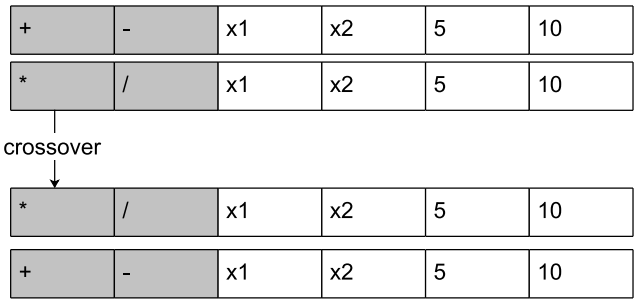



**FIGURE 6.** An illustration of single point crossover used in GEP.


**FIGURE 7.** An illustration of single point muation used in GEP.

Then, the primitive "+" may be sampled from this distribution and become the first gene point of this linear model.

Two strategies can be applied in the sampling process:

1) Probabilistic sampling strategy ($GEP_{HP}$): sampling a primitive from the categorical distribution.
2) Max sampling strategy ($GEP_{HM}$): choosing the primitive that corresponds to the maximum probability.

When using the probabilistic sampling strategy, one must ensure that $\sum_{j=1}^{n} p_{ij}^k = 1$ is satisfied. Hence, a normalization process should be applied. The normalization process is described as the following formula:

$$p_{ij}^k = \frac{p_{ij}^k}{\sum_{j=1}^{m} p_{ij}^k} \tag{8}$$

The algorithm 2 shows the process of EDA offspring generation.

**Algorithm 2** EDA Offspring Generation

```
1: for i = 1, ··· , n do
2:     for j = 1, ··· , m do
3:         p_ij ← p_ij / Σ_{j=1}^{n} p_ij        ▷ Normalization
4:     end for
5:     if strategy == GEP_HP then
6:         x_i ← sample(p_i)
7:     else if strategy == GEP_MP then
8:         x_i ← arg max_i p_i
9:     end if
10: end for
11: return x
```

- In line 3, the probability value is normalized to make sure the sum of probability values is equal to one.
- In line 5-9, a new gene point $x_i$ is sampled from the categorical distribution $p_i$.

**TABLE 1. Parameter settings.**

| parameter name | parameter value |
|---|---|
| population size | 20 |
| generation | 200 |
| head length | 50 |
| RNC range | [-25,25] |
| RNC length | 15 |
| max arity | 3 |
| selection algorithm | Tournament |
| tournament size | 5 |
| random seed | [0,50) |
| learning rate | 0.4 |
| EDA individual ratio | 0.1 |

**TABLE 2. Operators.**

| operator name | math formula |
|---|---|
| multiply | x*y |
| divide | $\begin{cases} \frac{x}{y} & \text{if } x > 10^{-6} \\ \frac{x}{10^{10}} & \text{if } x <= 10^{-6} \end{cases}$ |
| add | x+y |
| sub | x-y |
| ternary_add | x+y+z |
| ternary_sub | x-y-z |

## III. EXPERIMENTS

### A. EXPERIMENT ENVIRONMENT

#### 1) PARAMETER SETTINGS

Unless otherwise stated, the experiments are conducted using the parameters in TABLE 1.

- The population size is 20, and all algorithms stop after 200 iterations.
- Each chromosome has only one gene expression. The head length $h$ is set as 50, and RNC length is set as 15. The RNC range is set as $[-25,25]$. The learning rate and the EDA individual ratio are determined by experiments in the following section.
- The function operators that are used in GEP are introduced in TABLE 2. The max arity $a$ corresponds to these operators is 3. The tournament selection algorithm [22] is adopted with a tournament size of 5.
- The fitness standardized version of $GEP_H$ with the probabilistic sampling strategy ($GEP_{HSP}$) is used as the default $GEP_H$ algorithm.
- Every experiment is conducted on each dataset with 50 distinct random seeds. To ensure reproducibility, we set the random seed of the pseudo-random generator of Numpy [24] with a range from 0 to 49.

#### 2) EXPERIMENTAL DATASET

In order to evaluate the universality of the algorithm, ten datasets from the PMLB dataset [25] were used to evaluate the performance of our algorithm. In order to reveal the general applicability of the proposed algorithm, the size of benchmark datasets is ranged from 60 to 8192, and the number of features is spanned from 4 to 22. Detailed information of these datasets is presented in TABLE 3.

**TABLE 3. Experimental datasets.**

| dataset_name | instances | features |
|---|---|---|
| 195_auto_price | 159 | 16 |
| 197_cpu_act | 8192 | 22 |
| 227_cpu_small | 8192 | 13 |
| 527_analcatdata_election2000 | 67 | 15 |
| 529_pollen | 3848 | 5 |
| 542_pollution | 60 | 16 |
| 557_analcatdata_apnea1 | 475 | 4 |
| 560_bodyfat | 252 | 15 |
| 561_cpu | 209 | 8 |
| 666_rmftsa_ladata | 508 | 11 |

In the hyper-parameter study part, we use ''195_auto_price'' and ''197_cpu_act'' to study the impacts of various hyperparameter values.

### B. CONTROL PARAMETER SENSITIVITY

When introducing the EDA algorithm into GEP, two parameters are also introduced: the learning rate $\eta$ and the EDA offspring ratio, i.e., $\frac{X_e}{(X_e+X_g)}$. Like many other machine-learning hyperparameters, it is hard to give a proper value beforehand. The most suitable value must be determined based on data.

In industrial applications, algorithms that are insensitive to the hyperparameter or for which the hyperparameter is easy to determine based on simple principles are popular because these algorithms require low computation resources to yield a satisfactory result. Therefore, the influence of various hyperparameters is the focus of this section.

#### 1) LEARNING RATE

The learning rate can control the convergence speed of the EDA probabilistic model. If a low learning rate is chosen, the probabilistic model will cost much time to converge to the optimal distribution. If a high learning rate is chosen, the probabilistic model will converge quickly. However, a high learning rate may also impede the model converge to the optimal model due to it can not fine-tune the probability distribution. Therefore, the selection of an appropriate learning rate is challenging many machine learning engineers. An algorithm that can easy to choose a reasonable learning rate is more likely to be applied in industrial scenarios.

The current experiment focuses on identifying the relationship between algorithm performance and the learning rate. The ''195_auto_price'' dataset has been chosen with a learning rate from 0.1 to 1 with a step interval of 0.1.

Fig.8 and Fig.9 present the experiment results of the ''195_auto_price'' dataset. The ''Baseline'' presents the performance of GEP without EDA. The experimental results reveal that the learning rate of 0.3 is the most suitable learning rate. Although results are unstable with different learning rates, they universally outperform the GA-GEP algorithm. However, a careful choice of the learning rate will yield a much better result.

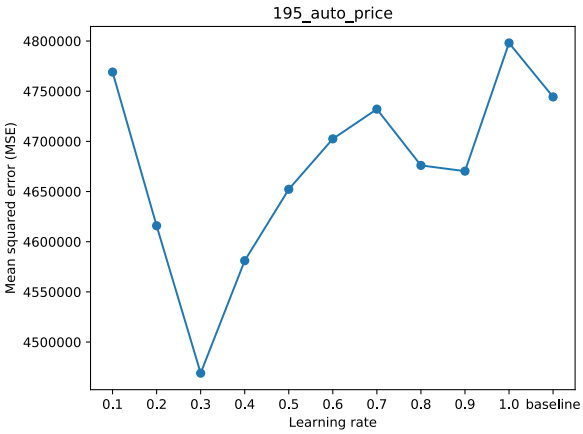



**FIGURE 8.** MSE versus learning rate on "195_auto_price".

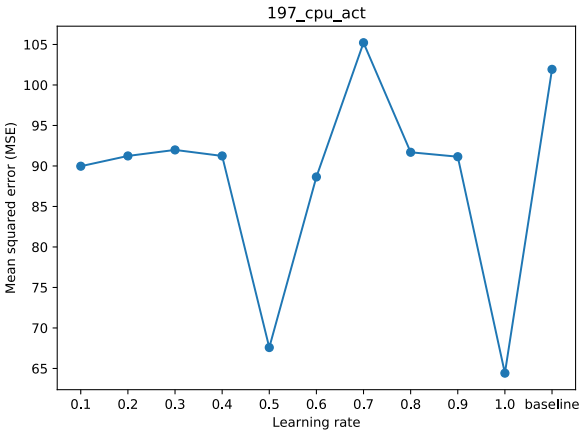



**FIGURE 10.** MSE versus learning rate on "197_cpu_act".

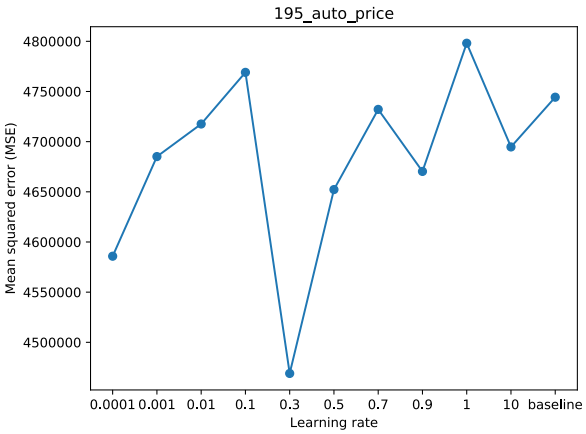



**FIGURE 9.** MSE versus learning rate on "195_auto_price".

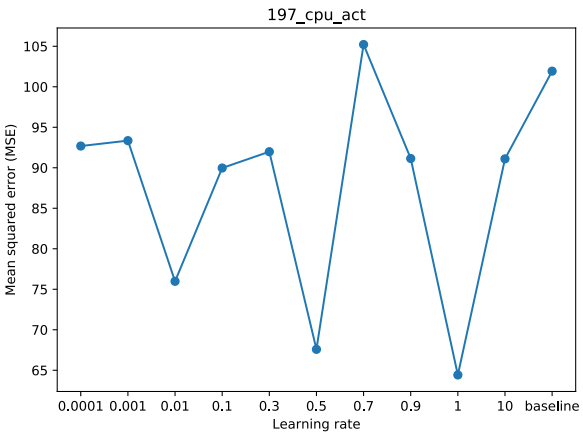



**FIGURE 11.** MSE versus learning rate on "197_cpu_act".

Fig.10 and Fig.11 present another experiment on the "197_cpu_act" dataset with a 0.1 EDA individual ratio. The results demonstrate that a learning rate of 1.0 yields the best result.

From the above two experiments, we conclude that: $GEP_H$ can not find a clear rule for determining the best learning rate. However, even though a worse learning rate has been chosen, it also achieves better results than the $GEP_G$ algorithm.

### 2) EDA OFFSPRING RATIO

In Section II, we proposed a hybrid EDA and GP algorithm that integrates the advantages of these two algorithms. However, this algorithm introduces a new parameter, the EDA offspring ratio, which must be specified based on the specific task.

Therefore, an experiment is conducted to investigate the effects of various values of the EDA offspring ratio. Our experiment considers offspring ratios from 0 to 1, with a step interval of 0.1. If the ratio is 0, it is equivalent to the GA-GEP algorithm. If the ratio is 1, it is equivalent to the EDA-GEP algorithm.

Fig.12 plots MSEs with various offspring ratios of the "195_auto_price" dataset. This figure demonstrates that the best performance of the hybrid algorithm achieves when the EDA individual ratio equals to 0.1. In this experiment, the hybrid algorithm reveals better performance than the GA-GEP algorithm and the EDA-GEP algorithm. Moreover, although the hybrid algorithm performs much better with the EDA offspring ratio increases, after reaching a specific threshold, the performance of the hybrid algorithm is decreased with the EDA offspring ratio increases. This phenomenon proves the necessity of the hybrid algorithm.

Fig.13 presents another experiment on the "197_cpu_act" dataset, which is based on a learning rate of 0.4. The result is similar to the experimental result that is presented above, too high or too low of the ratio will cause the final result to deteriorate, and the EDA individual ratio of 0.2 achieves the best result.

Considering both the above two experiment results, we can conclude that: a reasonable offspring ratio is close to 0.1. The more precisely value must be determined based on the specific task.

**TABLE 4.** Experimental results obtained by four comparison algorithms.

| | $GEP_G$ | $GEP_{HM}$ | $GEP_{HSP}$ | $GEP_{HP}$ |
|---|---|---|---|---|
| **195_auto_price** | 6272555 | 5976027 | 6055014 | **5891680** |
| **197_cpu_act** | 109.7618 | **97.40095** | 98.33862 | 98.50997 |
| **227_cpu_small** | 55.64986 | 73.70299 | **54.99544** | 59.25873 |
| **527_analcatdata_election2000** | 14138.55 | 15927.73 | 14510.95 | **13472.3** |
| **529_pollen** | 2.414881 | 2.414881 | **2.318949** | 2.349986 |
| **542_pollution** | 1309.173 | **1229.237** | 1294.513 | 1279.198 |
| **557_analcatdata_apnea1** | 6717743 | 5673621 | **4230696** | 6316790 |
| **560_bodyfat** | 0.411286 | 0.411286 | **0.406398** | 0.411286 |
| **561_cpu** | 1304.742 | 1385.123 | 1416.459 | **1258.414** |
| **666_rmftsa_ladata** | 3.657152 | **3.566582** | 3.639322 | 3.698454 |

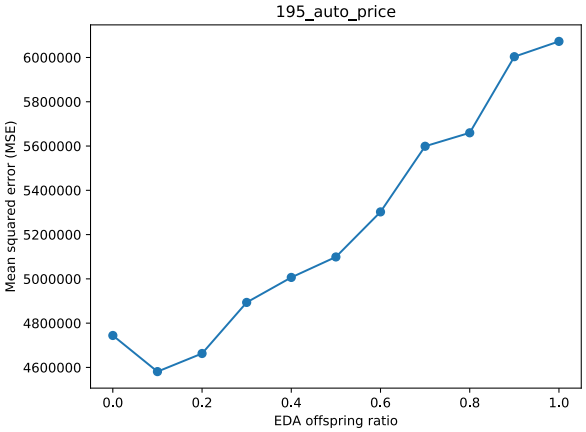



**FIGURE 12.** MSE versus EDA offspring ratio on "195_auto_price".

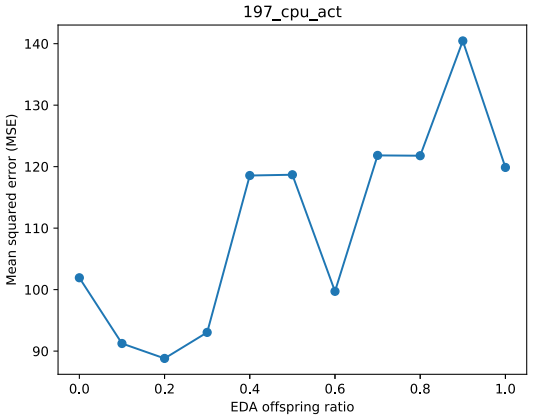



**FIGURE 13.** MSE versus EDA offspring ratio on "197_cpu_act".

### C. A COMPARISON STUDY ON THE PMLB DATASET

#### 1) EXPERIMENT CONFIGURATION

In order to have a fair comparison of GA-GEP ($GEP_G$), max sampling strategy $GEP_H$ ($GEP_{HM}$), and probabilistic sampling strategy $GEP_H$ ($GEP_{HP}$), experiments are conducted with the same parameters. Details on the parameters are presented in TABLE 1. We also conduct an experiment with standardized fitness values based $GEP_{HP}$ algorithm ($GEP_{HSP}$) for comparison with the unstandardized version of the algorithm.

For the sake of reproducibility, the pseudo-random generator is used again. The learning rate and the EDA generation individual ratio are only applied to the hybrid algorithm.

#### 2) EXPERIMENTAL RESULTS

Fig.14 uses a box plot to present the experimental results, and TABLE 4 gives the median values of 50 experiment results with various random seeds. The final results demonstrate that $GEP_{HSP}$ exhibits satisfactory performance on these datasets.

Experimental results show that $GEP_{HSP}$ outperforms $GEP_G$ on eight tasks. $GEP_{HM}$ and $GEP_{HP}$ are the second-best ones because they outperform $GEP_G$ on seven tasks. $GEP_{HSP}$ and $GEP_{HM}$ both perform poorly on the "527_analcatdata_election2000" and "561_cpu" datasets. On the contrary, $GEP_{HP}$ performs well on these datasets. Therefore, it is reasonable to consider $GEP_{HP}$ as a complementary strategy, which can be applied when $GEP_{HSP}$ does not perform well.

Although $GEP_{HM}$ performs poorly on three datasets and does not remedy the shortcomings of $GEP_{HSP}$, the algorithm can still have a place in some scenarios. For example, $GEP_{HSP}$ realizes the best performance on the "666_rmftsa_ladata" dataset. Therefore, we can consider $GEP_{HM}$ as an alternative strategy to pursue superior performance.

In essence, $GEP_{HP}$ and $GEP_{HSP}$ are equivalent in a way because we can regard the scale of loss value in $GEP_{HP}$ as the scale of the learning rate. Section III shows that there is no universal principle for selecting an appropriate learning rate. Therefore, it is difficult to explain the different characteristics between $GEP_{HSP}$ and $GEP_{HP}$. However, due to the substantial differences and complementary property of different learning rate strategies, an adaptive learning rate algorithm that can combine the benefits of $GEP_{HP}$ and $GEP_{HSP}$ may effectively improve the performance of $GEP_H$ algorithms, and it deserves further investigation in further works.

Fig.15 presents the training processes of ten datasets. This figure supports our conclusion that the hybrid strategy can generate a better model than the traditional strategy. From the training curves, we find that $GEP_{HM}$ can speed up the training process on some datasets, such as on the "666_rmftsa_ladata" dataset. In the training process of "666_rmftsa_ladata", $GEP_{HM}$ may find some inherent patterns which lead it performs much better than other

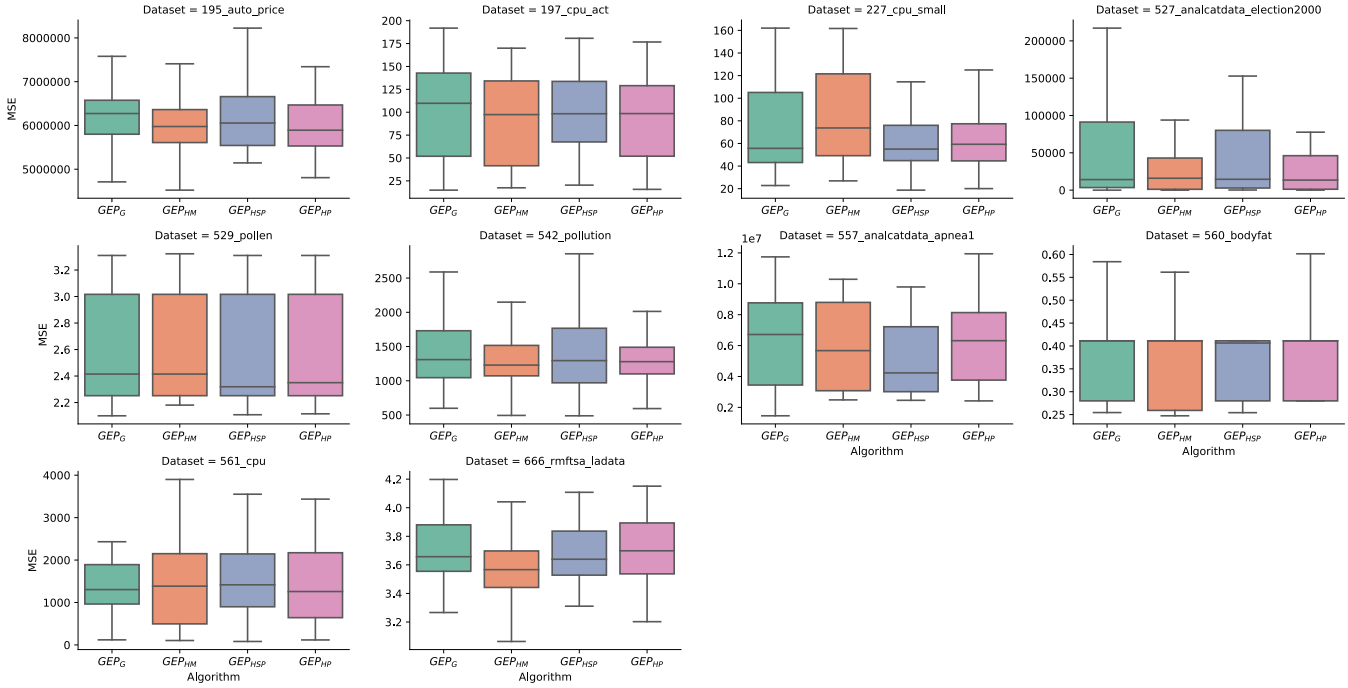



**FIGURE 14.** Experimental results obtained by four algorithms on ten benchmark datasets.

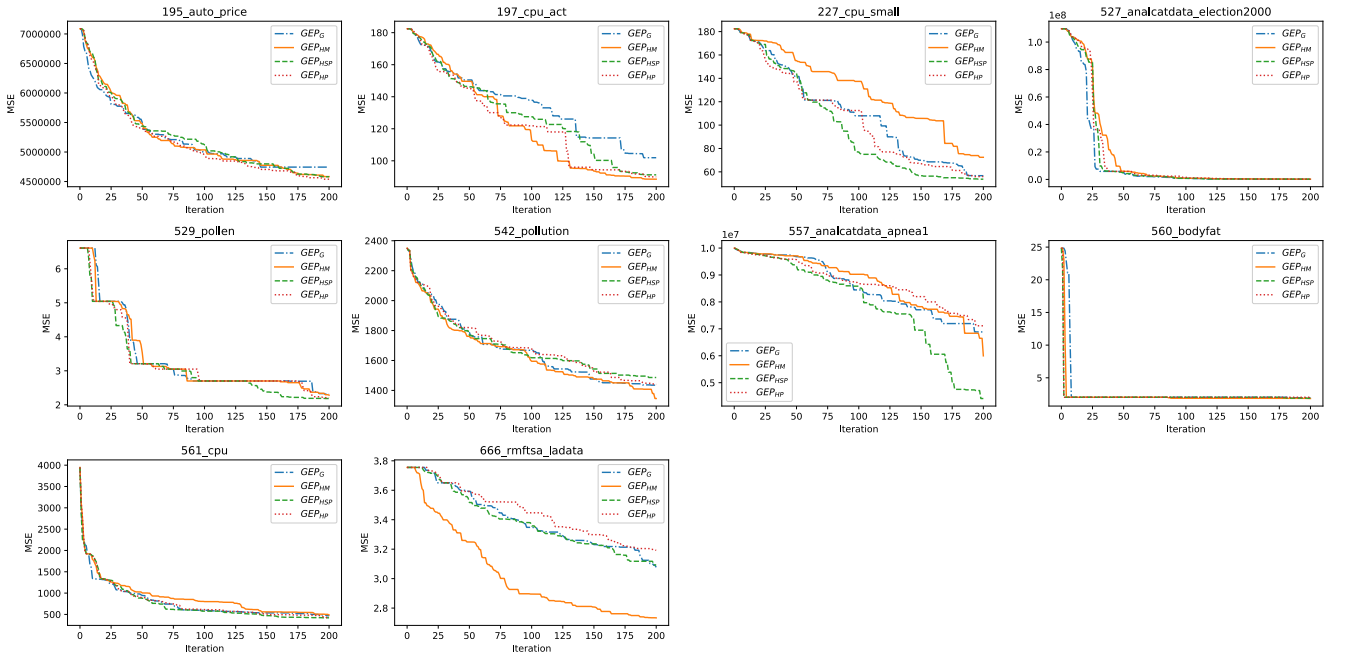



**FIGURE 15.** Training curves obtained by four algorithms on ten benchmark datasets.

strategies. Finally, $GEP_{HM}$ achieves the lowest training loss on "666_rmftsa_ladata", and also achieves the best generalization performance among other strategies as shown in TABLE 4. Although $GEP_{HM}$ performs much better than other strategies on "666_rmftsa_ladata", it should be pointed out that $GEP_{HM}$ performs much inferior to other strategies on the "227_cpu_small" dataset. Therefore, $GEP_{HM}$ should be used cautiously in real practice. By contrast, $GEP_{HSP}$ reveals superior performance on "557_analcatdata_apnea1" and "227_cpu_small" datasets comparing with other strategies,

**TABLE 5. List of abbreviations.**

| Abbreviation | Full name |
|---|---|
| $GA$ | Genetic Algorithm |
| $GP$ | Genetic Programming |
| $PIPE$ | Probabilistic Incremental Program Evolution |
| $PDPE$ | Probabilistic Developmental Program Evolution |
| $ECGA$ | Extended Compact Genetic Programming |
| $ECGP$ | Extended Compact Genetic Algorithm |
| $GEP$ | Gene Expression Programming |
| $EDA$ | Estimation of distribution |
| $GEP_G$ | GA based GEP |
| $GEP_H$ | GA-EDA Hybrid GEP |
| $GEP_{HM}$ | GA-EDA Hybrid GEP with Maximum Sampling Strategy |
| $GEP_{HP}$ | GA-EDA Hybrid GEP with Probabilistic Sampling Strategy |
| $GEP_{HSP}$ | GA-EDA Hybrid GEP with Probabilistic Sampling Strategy and Fitness Standardization |

and also with competent performance on other datasets, which confirms that $GEP_{HSP}$ is a relatively good strategy among other strategies.

## IV. CONCLUSION AND FUTURE WORK

In this paper, we considered utilizing both genetic and EDA operators for the offspring generation. To this end, a hybrid offspring generation algorithm was proposed for GEP in which some of the solutions are generated by genetic operators, and some of the solutions are generated by EDA operators. The EDA operator constructs a univariate categorical distribution and then samples the offspring randomly from that categorical distribution. In this way, the individual information and the population information are combined. The disadvantages faced by genetic operators and EDA operators are thus solved.

The new algorithm was applied to ten datasets, and the experimental results demonstrate that this new hybrid algorithm can outperform the GA algorithm for symbolic regression problems. The effects of two offspring generation strategies and the fitness standardization strategy were empirically studied via experiments. The experimental results demonstrate that the standardized probabilistic sample strategy $GEP_H$ ($GEP_{HSP}$) algorithm is the most suitable algorithm for most regression tasks. The effect of hyperparameters, such as the effect of different EDA learning rates, and the effect of different EDA individual ratios, had also been empirically studied. The results from these experiments demonstrate that the worse learning rate can also perform better than the GA based algorithm ($GEP_G$), and an appropriate EDA individual ratio is approximately 0.1.

The work reported in this paper is preliminary. Several issues should be considered in the future. For example, although the hybrid model can achieve substantial improvements, a carefully selected learning rate is also important to achieve superior performance. Therefore, an adaptive learning rate method merits future investigation. Moreover, designing a more fine-grained hybrid algorithm is also a promising direction to further improve the performance of the hybrid GEP algorithm.

## LIST OF ABBREVIATIONS

A list of abbreviations used in this paper is given in TABLE 5.

## REFERENCES

[1] C. Ferreira, "Gene expression programming: A new adaptive algorithm for solving problems," *Complex Syst.*, vol. 13, no. 2, pp. 87–129, Mar. 2001.
[2] G. Landeras, J. J. López, O. Kisi, and J. Shiri, "Comparison of gene expression programming with neuro-fuzzy and neural network computing techniques in estimating daily incoming solar radiation in the Basque country (Northern Spain)," *Energy Convers. Manage.*, vol. 62, pp. 1–13, Oct. 2012.
[3] M. M. Mostafa and A. A. El-Masry, "Oil price forecasting using gene expression programming and artificial neural networks," *Econ. Model.*, vol. 54, pp. 40–53, Apr. 2016.
[4] S. Deng, C. Yuan, J. Yang, and A. Zhou, "Distributed mining for content filtering function based on simulated annealing and gene expression programming in active distribution network," *IEEE Access*, vol. 5, pp. 2319–2328, 2017, doi: 10.1109/ACCESS.2017.2669106.
[5] J. R. Koza and J. R. Koza, *Genetic Programming as a Means for Programming Computers by Natural Selection*, vol. 1. Cambridge, MA, USA: MIT Press, 1992.
[6] C. Ferreira, "Gene expression programming in problem solving," in *Soft Computing and Industry*. London, U.K.: Springer, 2002, pp. 635–653.
[7] H. M. Azamathulla, A. A. Ghani, C. S. Leow, C. K. Chang, and N. A. Zakaria, "Gene-expression programming for the development of a stage-discharge curve of the pahang river," *Water Resour. Manage.*, vol. 25, no. 11, pp. 2901–2916, Sep. 2011.
[8] R. S. Faradonbeh, D. J. Armaghani, M. Monjezi, and E. T. Mohamad, "Genetic programming and gene expression programming for flyrock assessment due to mine blasting," *Int. J. Rock Mech. Mining Sci.*, vol. 88, pp. 254–264, Oct. 2016.
[9] C. Zhou, W. Xiao, T. M. Tirpak, and P. C. Nelson, "Evolving accurate and compact classification rules with gene expression programming," *IEEE Trans. Evol. Comput.*, vol. 7, no. 6, pp. 519–531, Dec. 2003.
[10] C. Ferreira, *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence*. Berlin, Germany: Springer, 2006, vol. 21.
[11] M. Hauschild and M. Pelikan, "An introduction and survey of estimation of distribution algorithms," *Swarm Evol. Comput.*, vol. 1, no. 3, pp. 111–128, Sep. 2011.
[12] R. Salustowicz and J. Schmidhuber, "Probabilistic incremental program evolution," *Evol. Comput.*, vol. 5, no. 2, pp. 123–141, Jun. 1997.
[13] K. Sastry and D. E. Goldberg, "Probabilistic model building and competent genetic programming," in *Genetic Programming Theory and Practice*. Boston, MA, USA: Springer, 2003, pp. 205–220.
[14] G. Harik, "Linkage learning via probabilistic modeling in the ECGA," Illinois Genetic Algorithms Lab., Univ. Illinois Urbana-Champaign, Champaign, IL, USA, Tech. Rep. 99010, 1999.
[15] D. Heckerman, "A tutorial on learning with Bayesian networks," in *Innovations in Bayesian Networks*. Berlin, Germany: Springer, 2008, pp. 33–82.
[16] K. Yanai and H. Iba, "Program evolution by integrating EDP and GP," in *Proc. Genetic Evol. Comput. Conf.* Berlin, Germany: Springer, 2004, pp. 774–785.
[17] H. Fang, A. Zhou, and H. Zhang, "Information fusion in offspring generation: A case study in DE and EDA," *Swarm Evol. Comput.*, vol. 42, pp. 99–108, Oct. 2018.
[18] Y. Xue, Z. Rui, X. Yu, X. Sang, and W. Liu, "Estimation of distribution evolution memetic algorithm for the unrelated parallel-machine green scheduling problem," *Memetic Comput.*, vol. 11, no. 4, pp. 423–437, Dec. 2019.
[19] E. Ghoulbeigi and M. V. dos Santos, "Probabilistic developmental program evolution," in *Proc. ACM Symp. Appl. Comput. (SAC)*, 2010, pp. 1138–1142.
[20] V. Robles, J. M. Peña, M. S. Pérez, P. Herrero, and O. Cubo, "Extending the GA-EDA hybrid algorithm to study diversification and intensification in GAs and EDAs," in *Proc. Int. Symp. Intell. Data Anal.* Berlin, Germany: Springer, 2005, pp. 339–350.
[21] K. Kim and R. I. McKay, "Stochastic diversity loss and scalability in estimation of distribution genetic programming," *IEEE Trans. Evol. Comput.*, vol. 17, no. 3, pp. 301–320, Jun. 2013.

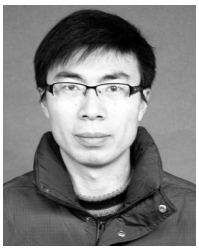
[22] B. L. Miller and D. E. Goldberg, "Genetic algorithms, tournament selection, and the effects of noise," *Complex Syst.*, vol. 9, no. 3, pp. 193–212, 1995.
[23] M. Hajihassani, S. S. Abdullah, P. G. Asteris, and D. J. Armaghani, "A gene expression programming model for predicting tunnel convergence," *Appl. Sci.*, vol. 9, no. 21, p. 4650, 2019.
[24] S. van der Walt, S. C. Colbert, and G. Varoquaux, "The NumPy array: A structure for efficient numerical computation," *Comput. Sci. Eng.*, vol. 13, no. 2, pp. 22–30, Mar. 2011.
[25] R. S. Olson, W. La Cava, P. Orzechowski, R. J. Urbanowicz, and J. H. Moore, "PMLB: A large benchmark suite for machine learning evaluation and comparison," *BioData Mining*, vol. 10, no. 1, p. 36, Dec. 2017.

**TONGLIN LIU** received the Ph.D. degree from the Shenyang Institute of Automation, Chinese Academy of Science, Shenyang, China, in 2010.

From 2012 to 2014, he carried out Postdoctoral Research with the Institute of Aeronautical Technology, China Aerospace Science and Industry Corporation, Beijing, China. He is currently a Senior Engineer with the Beijing Electro-Mechanical Engineering Institute, Beijing. His current research interests include complex system design, modeling, assessment, and optimization.

**HENGZHE ZHANG** received the Bc. degree in software engineering from Xiangtan University, Hunan, China, in 2019. He is currently pursuing the master's degree with East China Normal University, Shanghai, China.

His current research interests include symbolic regression, genetic programming, evolution computation, and statistical machine learning.



**HU ZHANG** received the M.Sc. degree in mechanical design and theory from China Three Gorges University, Hubei, China, in 2012, and the Ph.D. degree from the Center for Control Theory and Guidance Technology, Harbin Institute of Technology, Harbin, China, in 2016.

He is currently a Senior Engineer with the Beijing Electro-Mechanical Engineering Institute, Beijing, China. His current research interests include evolutionary computation, statistical machine learning, optimal design on system of systems, and operation assessment.

**AIMIN ZHOU** (Senior Member, IEEE) received the B.S. and M.S. degrees in computer science from Wuhan University, Wuhan, China, in 2001 and 2003, respectively, and the Ph.D. degree in computer science from the University of Essex, Colchester, U.K., in 2009.

He is currently a Professor with the School of Computer Science and Technology, East China Normal University, Shanghai, China. He has published more than 70 peer-reviewed articles on evolutionary computation and image processing. His current research interests include evolutionary computation, machine learning, image processing, and their applications.