

A Recommender System for Metaheuristic Algorithms for Continuous Optimization Based on Deep Recurrent Neural Networks

Ye Tian, Shichen Peng, Xingyi Zhang, *Senior Member, IEEE*, Tobias Rodemann, Kay Chen Tan, *Fellow, IEEE*, and Yaochu Jin, *Fellow, IEEE*

Abstract—As revealed by the no free lunch theorem, no single algorithm can outperform any others on all classes of optimization problems. To tackle this issue, methods for recommending an existing algorithm for solving given problems have been proposed. However, existing recommendation methods for continuous optimization suffer from low practicability and transferability, mainly due to the difficulty in extracting features that can effectively describe the problem structure and lack of data for training a recommendation model. This work proposes a generic recommender system to address the above two challenges. First, a novel method is proposed to represent an analytic objective function of a continuous optimization problem as a tree, which is directly used as the features of the problem. For black-box optimization problems whose objective function is unknown, a symbolic regressor is adopted to estimate the tree structure. Second, a large number of benchmark problems are randomly created based on the proposed tree representation, providing an abundant amount of training data with various levels of difficulty. By employing a deep recurrent neural network, a recommendation model is trained to recommend a most suitable metaheuristic algorithm for white- or black-

box optimization, making a significant step forward towards fully automated algorithm recommendation for continuous optimization. Experimental results on 100,000 benchmark problems show that the proposed recommendation model achieves considerably better performance than existing ones, and exhibits high transferability to real-world problems.

Impact Statement—Real-world optimization problems such as aerodynamic design of turbine engines and automated trading have been successfully solved by metaheuristics. However, practitioners are confronted with the challenge of how to choose an appropriate metaheuristic algorithm to solve a particular instance of these problems. This paper proposes a recommender system that can automatically select a best-suited metaheuristic algorithm without trial and error on a given problem. The proposed method develops a generic tree-like data structure for representing the difficulties of optimization problems and then trains a deep recurrent neural network to learn to choose the best metaheuristic algorithm, making automated algorithm recommendation practical for real-world problem-solving. The method will make metaheuristic optimization techniques accessible to industrial practitioners, policy makers, and other stakeholders who have no knowledge in metaheuristic algorithms.

Index Terms—Metaheuristics, continuous optimization problem, algorithm recommendation, deep recurrent neural network, symbolic regression, decision tree.

Manuscript received -. This work was supported in part by National Key R&D Program of China under Grant 2018AAA0100100, in part by the National Natural Science Foundation of China under Grant 61672033, Grant 61822301, Grant 61876123, Grant 61906001, Grant 61590922, and Grant U1804262, in part by the Hong Kong Scholars Program under Grant XJ2019035, in part by the Anhui Provincial Natural Science Foundation under Grant 1808085J06 and Grant 1908085QF271, in part by the State Key Laboratory of Synthetical Automation for Process Industries under Grant PAL-N201805, in part by the Research Grants Council of the Hong Kong Special Administrative Region, China under Grant CityU11202418 and Grant CityU11209219, and in part by a Royal Society International Exchanges Program under Grant IEC\NSFC\170279. Ye Tian acknowledges the financial support from Honda Research Institute Europe. (Corresponding authors: Xingyi Zhang and Yaochu Jin.)

Y. Tian is with the Key Laboratory of Intelligent Computing and Signal Processing of Ministry of Education, Institutes of Physical Science and Information Technology, Anhui University, Hefei 230601, China (email: field910921@gmail.com).

S. Peng and X. Zhang are with the Key Laboratory of Intelligent Computing and Signal Processing of Ministry of Education, School of Computer Science and Technology, Anhui University, Hefei 230601, China (email: severus.peng@gmail.com; xyzhanghust@gmail.com).

T. Rodemann is with the Honda Research Institute Europe, 63073 Offenbach, Germany (tobias.rodemann@honda-ri.de).

K. C. Tan is with the Department of Computer Science, City University of Hong Kong, Kowloon Tong, Hong Kong SAR (email: kaytan@cityu.edu.hk).

Y. Jin is with the Department of Computer Science, University of Surrey, Guildford, Surrey, GU2 7XH, U.K., and also with State Key Laboratory of Synthetical Automation for Process Industries, Northeastern University, Shenyang 110819, China (email: yaochu.jin@surrey.ac.uk).

I. INTRODUCTION

INSPIRED by the biological evolution mechanisms and swarm behaviors in nature, many metaheuristics have gained extensive development and usage over the past decades, such as genetic algorithms [1], particle swarm optimization [2], differential evolution [3], ant colony optimization [4], and among many others [5]–[7]. These algorithms are high-level methodologies that do not rely on the specific characteristics of problems, having shown appealing competitiveness in solving various complex optimization problems.

As revealed by the no free lunch theorem, however, there does not exist a single algorithm that can outperform any others on all classes of optimization problems [8]. Hence, much effort has been made to develop algorithms tailored for specific types of problems [9]–[12]. Since the design of dedicated algorithms may require specific domain knowledge, it is difficult for engineers to customize new algorithms for different applications; instead, existing algorithms are often selected on the

basis of empirical comparisons [13]. In practice, it is unrealistic to perform a large number of experiments to choose a best existing solver, since many industrial optimization problems are computationally expensive [14]–[16]. This leads to a new research field aiming at performance analysis of metaheuristics [17]. There are several ideas for identifying the relationship between algorithm performance and problem difficulty, including predicting the performance of algorithms on problems with specific settings [18], estimating the running time when algorithms can converge to the global optimum of specific problems [19], tuning the parameters of algorithms for achieving the best performance [20], and controlling the parameters of algorithms during the search process [21].

Since the effective use of these methods requires good understanding of the algorithms, it is difficult for engineers who are not an expert in metaheuristics [22]. A more realistic idea is to select potentially best-performing algorithms for a given problem from multiple candidate algorithms, where the methods based on this idea can be divided into online and offline ones. The online approaches belong to hybrid metaheuristics, where multiple metaheuristics are simultaneously used to solve the problem and the ones generating better offspring are given a higher priority [23]. For example, one idea is to run each metaheuristic for a generation and adjusts the resources for each metaheuristic according to its performance [24]. In [25], the components of multi-objective evolutionary algorithm are automatically configured during the search process, while in [26] a stochastic online decision process is proposed to select metaheuristics based on dynamic multi-armed bandits.

On the other hand, the offline methods aim to select a single best-performing algorithm before solving the problem, which are known as algorithm recommendation [27]. As illustrated in Fig. 1, algorithm recommendation can be regarded as a classification task, where each sample consists of the features representing the difficulty or structure of an optimization problem and the corresponding label is the index of the best suited optimization algorithm for the problem. Obviously, extracting effective features for representing the characteristics of optimization problems and building a high-performance classifier for learning the mapping between the features and the label (the recommended optimization algorithm) are two pivotal components for the success of a recommendation method. In the last decade, some methods have been developed for algorithm recommendation on various combinatorial optimization problems including propositional satisfiability problem [28], automated planning [29], quadratic assignment problem [30], traveling salesman problem [31], and *pmnk*-landscape [18]. By contrast, the research on algorithm recommendation for continuous optimization problems is still in its infancy, and only a small number of methods have been developed based on a few algorithms and problem classes [32]–[34]. The difficulties of algorithm recommendation

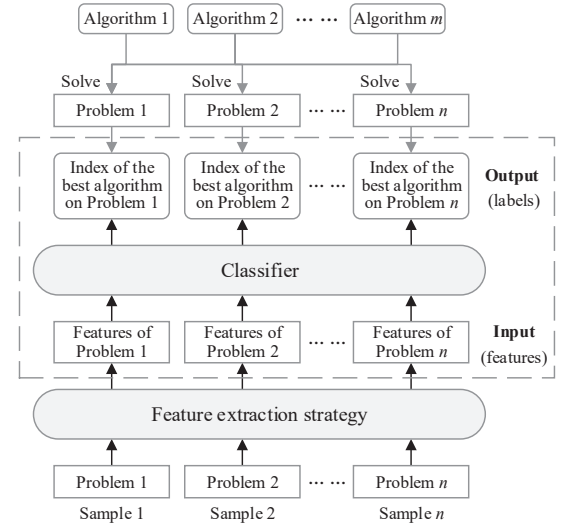


Fig. 1. A generic framework for algorithm recommendation.

for continuous optimization problems lie in the following two aspects:

1) *Feature extraction*: Due to the complexity of the functions in continuous optimization problems, most existing methods consider them as black-box problems and extract features to represent the landscape characteristics. Specifically, a number of decision vectors are sampled in the decision space of the problem by Latin hypercube sampling [35], and their objective values are calculated. Then, various features are calculated based on these objective values. It is believed that the algorithm performance is highly dependent on the landscapes of problems [33], hence these landscape-related features can build a bridge between algorithm performance and problem difficulty. This idea is straightforward and reasonable, but it suffers from the curse of dimensionality since a^d solutions should be sampled in theory for a divisions on each of the d decision variables. Since only a small number of function evaluations are allowed from real-world problems, the learned characterization of the landscape is very likely to be inaccurate for complex problems.

2) *Sample generation*: In contrast to combinatorial optimization problems whose datasets can be automatically synthesized [18], [31], continuous optimization problems are generally represented by complex functions that are difficult to be arbitrarily generated. Hence, existing recommendation methods for continuous optimization problems typically adopt the benchmark problems for training the recommendation model. For instance, 24 BBOB problems [36] were involved in [32], 12 WFG problems [37] were involved in [33], and 33 problems from the CEC competition [38] were involved in [34]. Although these benchmark problems are sufficient for assessing the performance of metaheuristics, it is difficult to train an effective classifier based on such a small number of samples. More importantly, existing bench-

mark problems consist of a limited number of manually designed mapping functions, which cannot provide adequately diverse difficulties for different metaheuristics. That is, a single metaheuristic good at handling these mapping functions can perform the best on most benchmark problems, thus leading to a highly unbalanced training set that is harmful for the training of the recommendation model.

As a consequence, algorithm recommendation still remains a promising but challenging topic, and many existing methods emphasize on analyzing landscape-related features [18], [32] rather than enhancing the performance of the recommendation model. To take a step forward, this work proposes a novel recommender system that represents an optimization problem using a tree structure and then feeds it into a deep recurrent neural network for training, improving the accuracy, explainability, and transferability of algorithm recommendation for continuous optimization problems. It addresses the limitations of existing recommendation methods since it does not need to extract landscape-related features and can generate sufficient benchmark problems as training samples. The main components of the proposed method include the following two aspects:

- 1) A function representation strategy is proposed for extracting features from continuous optimization problems. Based on a novel tree structure, the proposed method defines seven operands as the leaf nodes and 20 operators as the non-leaf nodes of the tree. By doing so, an analytic objective function can be represented as a tree and the nodes in the tree are regarded as the features. For black-box problems whose objective function is unknown, a number of solutions are sampled and their objective values on the problem are calculated; then, a symbolic regressor is adopted to estimate the tree for learning the mappings between the solutions and their objective values. This way, the proposed method offers a unified and explainable approach to representing white- and black-box optimization problems, which uses the operands and operators instead of the landscape-related features.
- 2) A deep learning based classifier is proposed to recommend a most suited metaheuristics for continuous optimization problems. Given the tree representation of a problem, the proposed method converts it into a reverse Polish expression. Then, the expression is regarded as a sentence and fed into a deep recurrent neural network used in natural language processing. To train the neural network, a large number of benchmark problems are randomly created with the help of the tree structure, and their labels are obtained by testing several metaheuristics on each problem. The proposed method makes full advantage of deep learning with an elegant and practical training data collection strategy, effectively learning the relations

between algorithm performance and problem difficulty.

To summarize, the proposed recommender system distinguishes itself from existing work [30], [32]–[34] in the following three aspects:

- 1) *Feature extraction strategy.* Existing methods sample a number of solutions from a given problem, and then extract the features of the problem by calculating some indicators based on the sampled solutions. By contrast, the proposed method suggests a novel tree structure to represent the given problem, and then converts the tree into a reverse Polish expression as the features of the problem. In comparison to existing methods, the proposed feature extraction strategy does not suffer from the curse of dimensionality and can better characterize the problem.
- 2) *Training sample collection strategy.* Existing methods rely on a limited number of benchmark problems to generate training samples, whilst the proposed method suggests a tree based strategy that can generate an arbitrary number of benchmark problems with various levels of difficulty. In comparison to existing methods, the proposed training sample collection strategy can provide a vast amount of data for training a high-performance recommendation model.
- 3) *Recommendation model.* Most existing methods employ conventional machine learning algorithms as the recommendation model. This work adopts a deep recurrent neural network as the recommendation model, which is more powerful than those used in existing methods.

In the empirical studies, we test ten popular metaheuristics on 100,000 created benchmark problems as the dataset. The results show that the proposed method achieves a prediction accuracy of 92.15%, whilst state-of-the-art recommendation methods can achieve an accuracy of up to 67.57%. In addition, the proposed method exhibits an appealingly high transferability on two types of real-world problems.

The rest of this paper is organized as follows. Section II details the function representation strategies for both white- and black-box optimization problems, Section III describes data collection and training of the deep learning based classifier, Section IV reports the experimental results, and Section V concludes this paper with discussions on promising future work.

II. TREE-BASED FUNCTION REPRESENTATION FOR FEATURE EXTRACTION FROM OPTIMIZATION PROBLEMS

A. A Tree Structure for Representing Continuous Optimization Problems

The continuous optimization problems considered in this work can be mathematically defined as follows:

$$\begin{aligned} &\text{Minimize} && f(\mathbf{x}) \\ &\text{s. t.} && \mathbf{x} \in \Omega \subseteq \mathbb{R}^d, \end{aligned} \quad (1)$$

where f denotes the objective function, $\mathbf{x} = (x_1, \dots, x_d)$ denotes the decision vector, Ω is the decision space, and d is the number of decision variables. The purpose of the proposed tree structure is to represent various function f with diverse difficulties, while the decision space and the number of decision variables can be arbitrarily specified for the created function.

In this work, we hypothesize that the difficulties of an objective function are mainly determined by the included operators and operands. For example, a multimodal landscape can be characterized by $\sin(x)$ or $\cos(x)$, and a linkage between decision variables can be captured by $(x_i - x_{i+1})^2$. Hence, seven operands and 20 operators are defined in the tree structure, as listed in Table I, where operands are stored in the leaf nodes and operators are stored in the non-leaf nodes of a tree. The operands and operators can be classified into the following five categories:

Numbers. Real numbers are the most common operands in the functions of continuous optimization problems. The proposed method uses the notation `a` to represent a real constant. Note that we do not use the notations like `1`, `2.2`, and `3.14` to represent particular constants since we aim to define a finite number of operands as the features, which constitute a finite vocabulary and can be fed into the recurrent neural network shown in Fig. 3. Besides, a notation `rand` is considered to represent a random number, whose value is changed in each function evaluation for representing noise.

Decision variables. The decision vector \mathbf{x} is indispensable in the functions, which is represented by the notation `x` in the tree. While using `x` can only represent fully separable functions, additional notations are also designed to represent non-separable functions like many popular benchmark problems, including the first variable `x1` providing linkages between all the variables and a single one [39], the translated decision vector `xt` providing linkages between each two continuous variables [40], the rotated decision vector `xr` providing complex linkages between all the variables [41], and the index vector `index` providing different optimal values to all the variables [42].

Binary operators. Four basic binary operators are considered in the tree, namely, addition, subtraction, multiplication, and division. Note that we do not notate some other binary operators like a^x and $\log_a x$ since the unary operators e^x and $\ln x$ are considered, where a^x is equivalent to e^{bx} and $\log_a x$ is equivalent to $\frac{\ln x}{b}$, where a denotes a positive number and $b = \ln a$.

Unary operators. Eleven unary operators are considered in the tree. As listed in Table I, the unary operators `neg`, `rec`, and `multen` can change the ranges of real constants and decision variables, the unary operators `square`, `sqrt`, `abs`, `log`, and `exp` can provide unimodal landscapes, the unary operators `sin` and `cos` can provide multimodal landscapes, and the unary operator `round` can provide flat landscapes.

Vector-oriented operators. Since the functions of contin-

TABLE I
OPERANDS AND OPERATORS USED IN THE TREE STRUCTURE.

| Notation | Meaning | Syntax |
|---------------------------|----------------------------|-----------------------------------------------|
| Numbers | | |
| <code>a</code> | A real constant | a |
| <code>rand</code> | A random number | $rand$ |
| Decision variables | | |
| <code>x</code> | Decision vector | (x_1, \dots, x_d) |
| <code>x1</code> | First variable | x_1 |
| <code>xt</code> | Translated decision vector | $(x_2, \dots, x_d, 0)$ |
| <code>xr</code> | Rotated decision vector | \mathbf{xr} |
| <code>index</code> | Index vector | $(1, \dots, d)$ |
| Binary operators | | |
| <code>add</code> | Addition | $a + x$ |
| <code>sub</code> | Subtraction | $a - x$ |
| <code>mul</code> | Multiplication | $a \cdot x$ |
| <code>div</code> | Division | a/x |
| Unary operators | | |
| <code>neg</code> | Negative | $-x$ |
| <code>rec</code> | Reciprocal | $1/x$ |
| <code>multen</code> | Multiplying by ten | $10x$ |
| <code>square</code> | Square | x^2 |
| <code>sqrt</code> | Square root | $\sqrt{ x }$ |
| <code>abs</code> | Absolute value | $ x $ |
| <code>exp</code> | Exponent | e^x |
| <code>log</code> | Logarithm | $\ln x $ |
| <code>sin</code> | Sine | $\sin(2\pi x)$ |
| <code>cos</code> | Cosine | $\cos(2\pi x)$ |
| <code>round</code> | Rounded value | $[x]$ |
| Vector-oriented operators | | |
| <code>sum</code> | Sum of vector | $\sum_{i=1}^d x_i$ |
| <code>mean</code> | Mean of vector | $\frac{1}{d} \sum_{i=1}^d x_i$ |
| <code>cum</code> | Cumulative sum of vector | $(\sum_{i=1}^1 x_i, \dots, \sum_{i=1}^d x_i)$ |
| <code>prod</code> | Product of vector | $\prod_{i=1}^d x_i$ |
| <code>max</code> | Maximum value of vector | $\max_{i=1, \dots, d} x_i$ |

uous optimization problems are calculated based on a decision vector rather than a single decision variable, a vector-oriented operator is required to map the multi-dimensional decision vector to a one-dimensional objective value. For this purpose, five vector-oriented operators are considered in the tree, including calculating the sum, mean, cumulative sum, product, and maximum value of all the variables in the decision vector.

After defining all the operands and operators, as plotted in Fig. 2, the function of a continuous optimization problem can be represented by the operands and operators and converted into a tree. The tree is then converted into a reverse Polish expression and used as the input of the classifier introduced in Section III-A, while no additional features need to be extracted. Besides, the proposed tree structure can facilitate the estimation of the expressions of black-box problems and the generation of training data, where these strategies are detailed in Sections II-B and III-B, respectively.

B. Estimated Tree Representation of Black-Box Problems Using Symbolic Regression

The proposed tree structure is very efficient for feature extraction since it does not need to calculate the

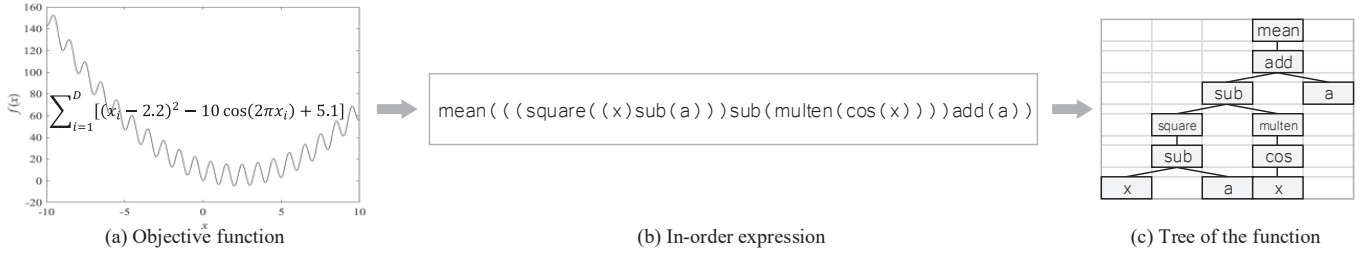


Fig. 2. Illustrative example of representing a continuous function as a tree.

Algorithm 1: *symbolic_regression(f)*

Input: f (a black-box problem)
Output: *tree* (the estimated tree)

- 1 $X \leftarrow$ Sample a number of solutions by Latin hypercube sampling;
- 2 **for each** $x \in X$ **do**
- 3 Calculate $f(x)$;
- 4 **for each benchmark problem** g **do**
- 5 $Sim(f, g) \leftarrow$ Calculate the similarity between functions f and g by (2);
- 6 $P \leftarrow$ Trees of the N most similar benchmark problems to f ;
- 7 **for each tree** $\in P$ **do**
- 8 Optimize the constants in *tree* by random search.
- 9 Calculate the fitness of each tree in P by (2);
- 10 **for generation** = 1 to G **do**
- 11 $P' \leftarrow$ Select N trees from P by binary tournament selection according to the fitness;
- 12 $O \leftarrow recombination(P')$;
- 13 **for each tree** $\in O$ **do**
- 14 Optimize the constants in *tree* by random search.
- 15 Calculate the fitness of each tree in O by (2);
- 16 $P \leftarrow P \cup O$;
- 17 $P \leftarrow$ Remain the N best trees in P ;
- 18 *tree* \leftarrow The best tree in P ;
- 19 **return** *tree*;

objective value of any solution on the problem. However, it is not applicable to real-world optimization problems whose objective function is not analytically known. For instance, the evaluation of the objective function is based on computational fluid dynamics simulations [43] or a large dataset [44]. Therefore, the operands and operators of the function of a black-box problem must be estimated based on a number of solutions sampled from the problem.

Since the proposed tree structure contains different types of decision vectors and vector-oriented operators, this work adopts a symbolic regressor to estimate the operands and operators included in a black-box problem, which is slightly different from existing methods based on genetic programming [45]. The procedure of

the proposed symbolic regressor is presented in Algorithm 1. To begin with, a number of solutions X are uniformly sampled in the decision space by Latin hypercube sampling, and their objective values on the black-box problem f are calculated. Then, the objective values of the solutions on each created benchmark problem g are also calculated, and the similarity (approximation error) between f and g is measured according to the following loss function:

$$Sim(f, g) = \sqrt{\sum_{x \in X} (f(x) - g(x))^2}. \quad (2)$$

Based on the similarity between f and all the benchmark problems, the trees of the N most similar benchmark problems to f are selected as the individuals in the initial population, so that the symbolic regressor does not need to search from scratch. It is worth noting that the proposed tree structure stores the locations of constants but does not specify their values. Hence, random search is adopted to optimize the constants in each tree, where ten random values are assigned to each constant and the value leading to the best fitness is kept. The fitness of a tree is defined as the similarity between f and the function determined by the tree, so that the tree with the best approximation of the mappings between all the $x \in X$ and $f(x)$ can be preserved.

At each generation, N parents are selected from P by binary tournament selection according to the fitness, and they are used to generate N offspring by crossover and mutation operators. After optimizing the constants and calculating the fitness values of the offspring, the parent population is combined with the offspring and N individuals having better fitness values are selected as the parents of the next generation. Algorithm 2 gives the procedure of the crossover and mutation operators. Regarding the crossover operator, two trees are picked up from the parent population each time, and two subtrees are randomly selected from them and exchanged to generate two new trees. Regarding the mutation operator, each new tree is mutated by one of the following three operators with a predefined probability: a) randomly select a leaf node from the tree and extend the node by a randomly selected operator; b) randomly select a subtree from the tree and replace the subtree with one of its subtrees; c) randomly select a node from the tree and randomly modify its notation. The offspring

Algorithm 2: recombination(P)

Input: P (parent population)
Output: O (offspring population)

```

1  $O \leftarrow \emptyset$ ;
  //Crossover operator
2 while  $P \neq \emptyset$  do
3    $[tree_1, tree_2] \leftarrow$  Pick up two trees from  $P$ ;
4    $P \leftarrow P \setminus \{tree_1, tree_2\}$ ;
5    $t_1 \leftarrow$  Randomly select a subtree from  $tree_1$ ;
6    $t_2 \leftarrow$  Randomly select a subtree from  $tree_2$ ;
7   Exchange  $t_1$  and  $t_2$ ;
8    $O \leftarrow O \cup \{tree_1, tree_2\}$ ;
  //Mutation operator
9 for each  $tree \in O$  do
10  if  $rand < mutation\_probability$  then
11     $r \leftarrow rand$ ;
12    if  $r < \frac{1}{3}$  then
13       $t \leftarrow$  Randomly select a leaf node from  $tree$ ;
14      Extend  $t$  by a randomly selected operator;
15    else if  $r < \frac{2}{3}$  then
16       $t \leftarrow$  Randomly select a subtree from  $tree$ ;
17       $t' \leftarrow$  Randomly select a subtree from  $t$ ;
18      Replace  $t$  with  $t'$  in  $tree$ ;
19    else
20       $t \leftarrow$  Randomly select a node from  $tree$ ;
21       $t' \leftarrow$  Randomly generate a node with the same category to  $t$ ;
22      Replace  $t$  with  $t'$  in  $tree$ ;
23 return  $O$ ;
```

population consists of all the trees generated by the crossover operator and tuned by the mutation operator.

Consequently, the proposed method is able to find a tree structure representation for both white- and black-box continuous optimization problems, where white-box problems are directly represented by the tree structure according to their functions, and the trees of black-box problems are estimated by the proposed symbolic regressor. Therefore, the recommendation model will be applicable to both white- and black-box optimization problems. In the next section, the construction of deep learning based classifier for algorithm recommendation will be elaborated.

III. A DEEP LEARNING BASED CLASSIFIER FOR AUTOMATED ALGORITHM RECOMMENDATION

A. A Deep Recurrent Neural Network for Algorithm Recommendation

The proposed method uses the components of a tree representation for a continuous optimization problem as the input attributes of the classifier. Specifically,

Algorithm 3: tree_to_expr($tree$)

Input: $tree$ (the constructed tree)
Output: $expr$ (the reverse Polish expression)

```

1  $expr \leftarrow tree\_to\_expr(tree.left) \cup$   

    $tree\_to\_expr(tree.right) \cup tree.value$ ;
2 return  $expr$ ;
```

Algorithm 4: expr_to_func($expr$)

Input: $expr$ (the reverse Polish expression)
Output: $func$ (the created function)

```

1  $func \leftarrow \emptyset$ ;
2 for  $i = 1$  to  $|expr|$  do
3    $note \leftarrow expr(i)$ ; //  $i$ -th notation
4   if  $note$  is an operand then
5      $func \leftarrow func \cup note$ ;
6   else if  $note$  is a unary/vector-oriented operator then
7      $note2 \leftarrow func(end)$ ; // Last notation
8     Delete  $note2$  from  $func$ ;
9      $func \leftarrow func \cup (note \cup note2)$ ;
10  else if  $note$  is a binary operator then
11     $note2 \leftarrow func(end)$ ; // Last notation
12     $note3 \leftarrow func(end - 1)$ ; // Second last notation
13    Delete  $note2$  and  $note3$  from  $func$ ;
14     $func \leftarrow func \cup (note2 \cup note \cup note3)$ ;
15 return  $func$ ;
```

it converts the tree into a reverse Polish expression [46], as shown in Algorithm 3, where $tree$ denotes the root of the tree, $tree.value$ denotes the notation of the node, and $tree.left$ and $tree.right$ denote the left and right children of the node, respectively. It is worth noting that the function of a problem can also be represented as an in-order expression as shown in Fig. 2, however, it is unreasonable to use the in-order expression as the input attributes to the classifier since the expression contains many parentheses for determining the order of operations. Taking the function $\sum_{i=1}^d (10 \sin(2\pi x_i) - 3)$ as an example, the in-order expression is $sum((multen(\sin(x))) sub(a))$ and the reverse Polish expression is $x \sin multen a sub sum$. Obviously, the reverse Polish expression does not contain any parenthesis and can be correctly converted back into the function by Algorithm 4. Note that a post-order expression generally cannot be converted into an in-order expression, while the reverse Polish expression can be correctly converted into a function, since there are two different types of nodes (i.e., operand and operator) in the tree, where the operands are always leaf nodes and the operators are always non-leaf nodes.

The reverse Polish expression is in fact a variable-length sentence with a finite vocabulary, hence a deep

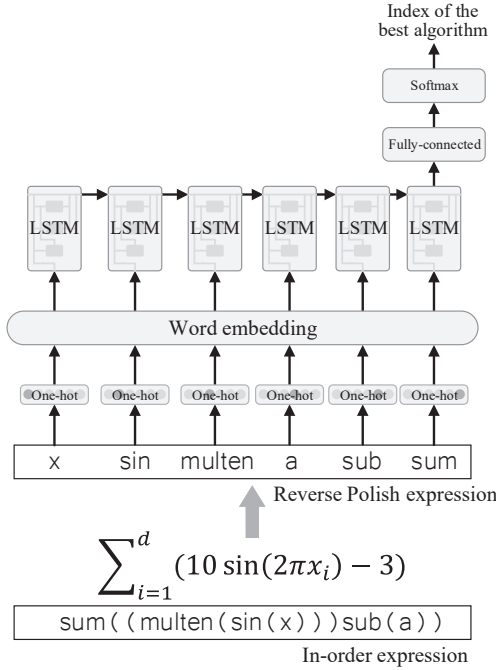


Fig. 3. The deep recurrent neural network used in the proposed method.

recurrent neural network is adopted as the classifier, i.e., the recommendation of algorithms is regarded as a natural language processing task. As depicted in Fig. 3, the reverse Polish expression is first tokenized by encoding each notation in a one-hot representation [47], i.e., a binary vector with one dimension being 1 and the rest being 0. Then, the one-hot representation of each notation is fed into a word embedding layer [48] to obtain a word vector. Afterwards, the word vectors in the same expression are fed into an LSTM layer [49] in sequence, which is tailed by a fully-connected layer and a softmax layer for predicting the index of the best algorithm. The length of the one-hot representation is 27 since there are 27 notations defined in Table I, the size of the word embedding layer and LSTM layer is set to 30, and the size of the fully-connected layer and softmax layer is equal to the number of candidate algorithms. The whole model is trained by Adam [50] with a learning rate of 0.01, a mini-batch size of 128, and 20 epochs, for minimizing the following cross entropy loss:

$$CE = \frac{1}{L} \sum_{i=1}^L \sum_{j=1}^K -y_{ij} \log(\hat{y}_{ij}), \quad (3)$$

where L denotes the number of training samples (i.e., problems), K denotes the number of categories (i.e., algorithms), y_{ij} is a binary variable denoting whether the j -th algorithm performs the best on the i -th problem, and \hat{y}_{ij} is the j -th output of the model on the i -th sample.

B. A Tree based Strategy for Training Data Generation

Last but not the least, a large number of samples must be obtained to train the deep recurrent neural network

Algorithm 5: *generate_tree(D)*

Input: D (number of operations in the tree)

Output: *tree* (the constructed tree)

```

1 tree.value  $\leftarrow$  'mean'; //Notation of the
  node
2 tree.left.value  $\leftarrow$   $\emptyset$ ; //Left child
3 tree.right.value  $\leftarrow$  'x'; //Right child
4 for  $i = 2$  to  $D$  do
    //Randomly reach a leaf node
5    $t \leftarrow$  tree;
6   while  $t.right \neq \emptyset$  do
7     if  $t.left == \emptyset$  or  $\text{rand} < 0.5$  then
8        $t \leftarrow t.right$ ;
9     else
10       $t \leftarrow t.left$ ;
    //Extend the leaf node
11   operator  $\leftarrow$  Randomly select an operator;
12   if operator is a binary operator then
13     operand  $\leftarrow$  Randomly select an operand;
14     if  $\text{rand} < 0.5$  then
15        $t.left.value \leftarrow t.value$ ;
16        $t.right.value \leftarrow$  operand;
17     else
18        $t.left.value \leftarrow$  operand;
19        $t.right.value \leftarrow t.value$ ;
20      $t.value \leftarrow$  operator;
21   else
22      $t.right.value \leftarrow t.value$ ;
23      $t.value \leftarrow$  operator;
24 return tree;

```

in advance. Since existing benchmark problems in the literature cannot provide sufficient training samples, a training data collection strategy is proposed to randomly generate benchmark problems. It is worth noting that randomly adding operators and operands to the decision vector \mathbf{x} in a bottom-up manner is likely to create a long monomial rather than an arbitrary function. For instance, it is easy to create monomials like $10 \sin(2\pi x)$ but difficult to create more complex functions like $e^x + x^2 + 2$. Therefore, the proposed training data collection strategy adopts the proposed tree structure to create complex functions.

Fig. 4 and Algorithm 5 present the detailed procedure of constructing a random tree. As can be seen, the initial nodes of the tree are always *mean* and *x*, which can ensure all the decision variables are involved and the value of the function is a scalar. Afterwards, the following steps are repeated for a predefined number of times to construct a tree: Randomly selecting a leaf node, extending this node by a new operator, and randomly selecting a new operand if the new operator is a binary operator. It is obvious that the proposed method can cre-

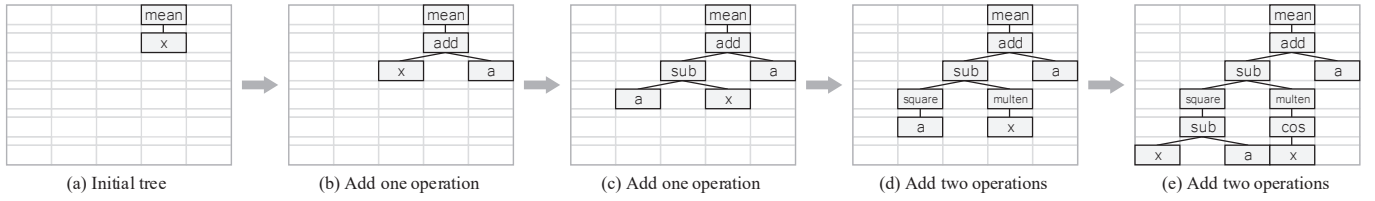


Fig. 4. Procedure of creating a random tree.

TABLE II
ALL THE DIFFICULTIES TO BE INJECTED INTO THE FUNCTION *func*.

| | Difficulty | Operation | Probability of being selected |
|---|------------------------------------------------------|---------------------------------------------------------------------------|-------------------------------|
| 1 | Noise | Replace <i>func</i> by <i>func</i> · <i>rand</i> | 0.05 |
| 2 | Flat landscape | Replace <i>func</i> by $\lceil \text{func} \rceil$ | 0.05 |
| 3 | Multimodal landscape | Replace <i>func</i> by $\text{func} + \sin(2\pi \cdot \text{func})$ | 0.1 |
| 4 | Highly multimodal landscape | Replace <i>func</i> by $\text{func} + 10 \sin(2\pi \cdot \text{func})$ | 0.05 |
| 5 | Linkages between all the variables and the first one | Replace (x_1, \dots, x_d) by $(x_1, \dots, x_d) - x_1$ | 0.05 |
| 6 | Linkages between each two contiguous variables | Replace (x_1, \dots, x_d) by $(x_1, \dots, x_d) - (x_2, \dots, x_d, 0)$ | 0.05 |
| 7 | Complex linkages between all the variables | Replace (x_1, \dots, x_d) by xr | 0.05 |
| 8 | Different optimal values of the variables | Replace (x_1, \dots, x_d) by $(1 \cdot x_1, \dots, d \cdot x_d)$ | 0.05 |

ate complex functions rather than monomials. However, the created functions may have very simple landscapes or invalid objective values. To address this issue, we introduce in the following three operations to modify the constructed trees.

1) *Difficulty injection*: In practice, the optimization problems in real-world applications usually have various difficulties or difficulties such as noise [51], flat landscapes [52], multimodal landscapes [11], and complex linkages between variables [53]. Therefore, a difficulty injection operation is designed here to inject particular difficulties into the functions. As listed in Table II, eight difficulties are designed by extending the created function *func* or decision vector **x**, including noise, flat landscape, multimodal landscape, highly multimodal landscape, linkages between variables, and different optimal values of variables. The pseudocode of this operation is presented in Supplementary Materials I.

2) *Tree cleaning*: Since the operators in the created functions are randomly selected, there usually exist some redundant operators that may be harmful to the training of classifier, e.g., $-x - (-a)$ can be simplified into $a - x$ and $\prod_{i=1}^d \sum_{i=1}^d x_i$ can be simplified into $\sum_{i=1}^d x_i$. The tree cleaning operation simplifies 17 scenarios as shown

TABLE III
ALL THE SCENARIOS TO BE SIMPLIFIED.

| | Scenario | Example (original) | Example (Simplified) |
|----|-------------------------------------------|----------------------------------|----------------------|
| 1 | Vector-oriented operators acted on scalar | $\prod_{i=1}^d \sum_{i=1}^d x_i$ | $\sum_{i=1}^d x_i$ |
| 2 | Redundant abs | $ \sqrt{ x } $ | $\sqrt{ x }$ |
| 3 | Successive square and sqrt | $\sqrt{x^2}$ | x |
| 4 | Successive exp and log | $\ln e^x$ | x |
| 5 | Successive two constants | $a + b$ | a |
| 6 | Successive two neg | $-(-x)$ | x |
| 7 | Successive neg and sub | $-(a - x)$ | $x - a$ |
| 8 | Successive add and neg | $a + (-x)$ | $a - x$ |
| 9 | Successive sub and neg | $a - (-x)$ | $a + x$ |
| 10 | Successive two add/sub | $(a + x) - b$ | $a + x$ |
| 11 | Successive three add/sub | $(a + x_1) - (b + x_2)$ | $a + x_1 - x_2$ |
| 12 | Successive two rec | $1/(1/x)$ | x |
| 13 | Successive rec and div | $1/(a/x)$ | x/a |
| 14 | Successive mul and rec | $a \cdot (1/x)$ | a/x |
| 15 | Successive div and rec | $a/(1/x)$ | $a \cdot x$ |
| 16 | Successive two mul/div | $(a \cdot x)/b$ | $a \cdot x$ |
| 17 | Successive three mul/div | $(a \cdot x_1)/(b \cdot x_2)$ | $a \cdot x_1/x_2$ |

in Table III, including the vector-oriented operators acted on scalar, the redundant abs, the successive square and sqrt, the successive exp and log, the successive add, sub, and neg, and the successive mul, div, and rec. Note that two successive constants like $a + b$ are also simplified into a single one a , since a particular setting of the constant can make them equivalent. The pseudocode of this operation is presented in Supplementary Materials II.

3) *Invalid problem elimination*: It should be noted that the combinations of some operands and operators may lead to invalid or extreme objective values, such as $1/x$ with $x = 0$ and e^{e^x} with $x > 3$. Hence, we directly

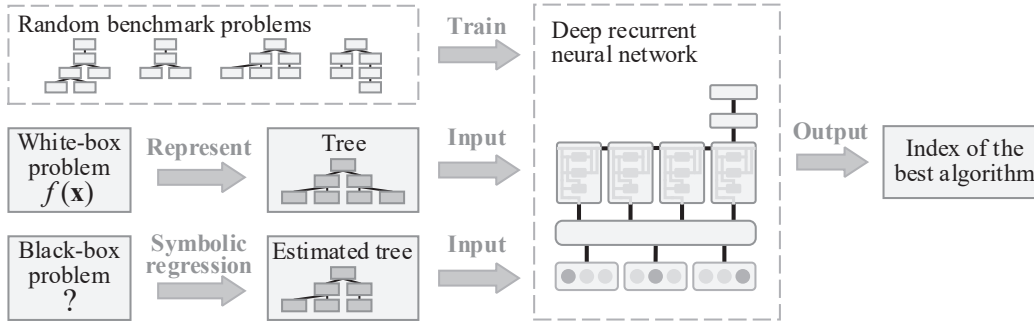


Fig. 5. Overall framework of the proposed recommender system.

eliminate the problem if an invalid or extreme objective value (i.e., larger than 10^{10} or smaller than -10^{10}) is generated when testing algorithms on it. In addition, to make the label of each problem unique, only one algorithm can have the best performance on the problem. For this purpose, we test each algorithm on each problem for multiple independent runs, then eliminate the problem if the best algorithm has a similar result to any other algorithms statistically in terms of the Wilcoxon rank sum test [54] with a significance level of 0.05.

C. Overall Framework of the Proposed Method

The overall framework of the proposed method is illustrated in Fig. 5. Firstly, a large number benchmark problems are randomly created by the training data collection strategy introduced in Section III-B, and their labels are obtained by testing several metaheuristics on them. Note that the values of constants a are set to random values within $[1, 10]$ and the random numbers rand vary within $[1, 2]$. Then, these problems are used to train the deep recurrent neural network introduced in Section III-A. For a white-box continuous optimization problem, it is represented by the tree structure introduced in Section II-A and then fed into the neural network. For a black-box continuous optimization problem, the tree is estimated by the symbolic regressor introduced in Section II-B and then fed into the neural network based classifier.

IV. EMPIRICAL STUDIES

A. Data Sample Generation

We randomly create 100,000 benchmark problems with 8 to 12 operations by using the proposed training data collection strategy. Then, we test ten metaheuristics on each problem to get the label of each sample, including artificial bee colony algorithm (ABC) [55], ant colony optimization (ACO) [4], covariance matrix adaptation evolution strategy (CMA-ES) [56], competitive swarm optimizer (CSO) [57], differential evolution (DE) with $\text{rand}/1/\text{bin}$ [3], fast evolutionary programming (FEP) [58], genetic algorithm (GA) with simulated binary crossover [59] and polynomial mutation [60], particle swarm optimization (PSO) [2], simulated annealing (SA)

[61], and random search (Rand). Most of these algorithms have shown promising performance on real-world applications by using different search strategies, covering most existing paradigms of metaheuristics.

Each algorithm is executed on each problem for 20 independent runs, and the best algorithm on each problem is determined by comparing the mean of the minimum objective values found over 20 runs. For all the problems, the number of decision variables is set to 10, the upper bound of each decision variable is set to 10, and the lower bound of each decision variable is set to -10 . For all the algorithms, the population size is set to 100 and the maximum number of function evaluations is set to 10,000, which is empirically confirmed to be enough for the best algorithm to converge. It is important to note that the parameter settings of all the algorithms are set to the same to those in their original literatures, and we have not tuned them for better performance. This is because the purpose of testing these algorithms is to obtain the label of each problem, rather than solve all the problems as much as possible. If one algorithm is tuned to have the best performance on most problems, the label of most problems will be the same and it will be meaningless to do algorithm recommendation, since we can just select the algorithm having the best overall performance for all the problems.

B. Analysis of Data Samples

Before applying the proposed recommender system, the performance of the ten metaheuristics on the benchmark problems is analyzed. Fig. 6 depicts the ratio of problems where each metaheuristic performs the best. It can be observed that GA obtains the most best results, which is followed by CMA-ES and PSO. This is intuitive since these three algorithms have shown high performance on many real-world applications [22], [62], [63]. Note that the experimental results only indicate the performance of the ten algorithms on the 100,000 problems created by the proposed systematic method, while they should perform the best on the same ratio of problems among all problems according to the no free lunch theorem.

In order to study the relationship between algorithm performance and problem difficulty, an indicator is de-

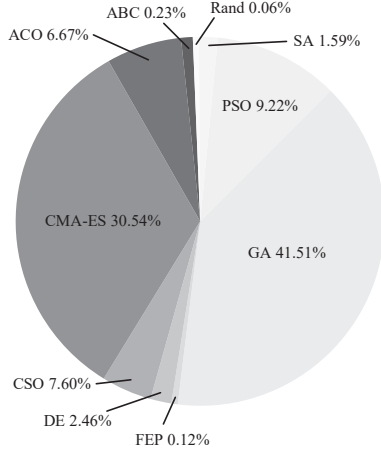


Fig. 6. Ratio of problems where each algorithm performs the best among 100,000 randomly created problems.

finied for each operand and operator on each algorithm, termed r -value. Formally, the r -value of an operand or operator on an algorithm is defined by

$$r = \frac{O' \cdot L}{L' \cdot O}, \quad (4)$$

where L and L' count the total numbers of all the problems and the problems where the algorithm performs the best, respectively, while O and O' count the numbers of the operand or operator in all the problems and the problems where the algorithm performs the best, respectively. In short, the r -value calculates the ratio of the frequency of the operand or operator in the problems where the algorithm performs the best to the frequency of the operand or operator in all the problems. Hence, $r > 1$ means that the algorithm is good at handling the operand or operator while $r < 1$ means that the algorithm is bad at handling the operand or operator.

As plotted in Fig. 7, the r -values of all the operands and operators on GA and CMA-ES are quite different. On the one hand, the r -values of \sin and \cos on GA are obviously larger than 1, which means that GA is good at handling multimodal landscapes; by contrast, the r -value of xr on GA is obviously smaller than 1, indicating that GA cannot handle complex linkages between variables well. On the other hand, CMA-ES is good at handling noise, but it is bad at handling complex linkages and multimodal landscapes. These observations are consistent with some existing work, where GA is good at solving multimodal problems owing to the mutation operator [60], [64] and CMA-ES is good at handling unimodal problems with noise due to the usage of Gaussian distribution model [65], [66]. According to Fig. 8, it can be seen that the problem on which GA performs the best is quite rugged while the problem where CMA-ES performs the best is relatively smooth.

The r -values in Fig. 7 indicate that both GA and CMA-ES are bad at handling complex linkages between variables. In contrast, Fig. 9 shows the r -values of all the operands and operators on DE and CSO, where DE is

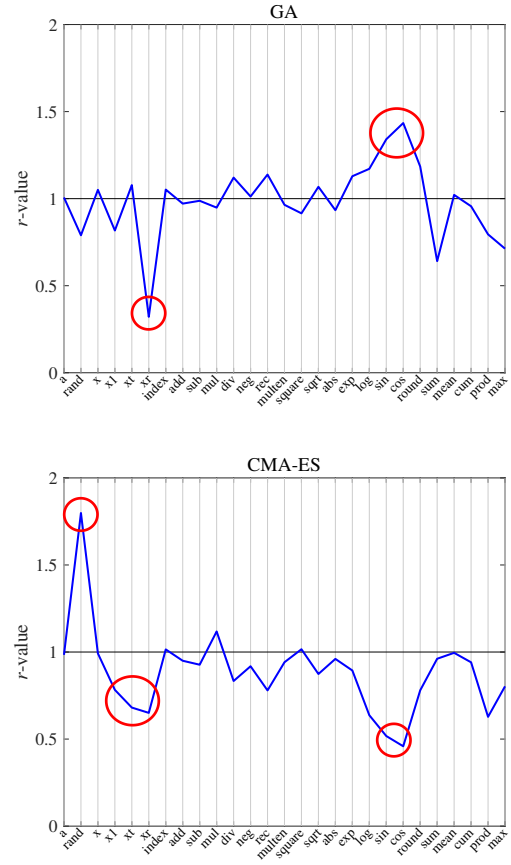


Fig. 7. r -values of all the operands and operators on GA and CMA-ES.

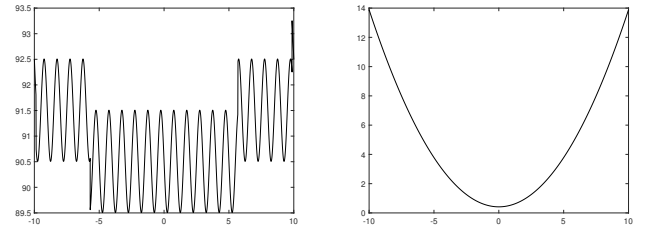


Fig. 8. A problem $f(x) = \lceil \frac{x}{8.1} \rceil^2 - \sin(2\pi x) + 81.9$ where GA performs the best (left) and a problem $f(x) = \frac{x^2 + 3.1}{7.4}$ where CMA-ES performs the best (right).

good at handling translated decision vector xt and CSO is good at handling rotated decision vector xr , which indicates that these two algorithms are more suited for problems with complex linkages. In fact, both DE and CSO have shown high performance in solving many challenging problems with very complex linkages in the literature [67], [68].

In addition, it is interesting to investigate when random search can perform better than the others. According to the r -values shown in Fig. 10, random search is likely to better handle xt , neg , rec , $prod$, and max , all of which provide extreme objective values and complex landscapes. As shown in Fig. 10, the problem where

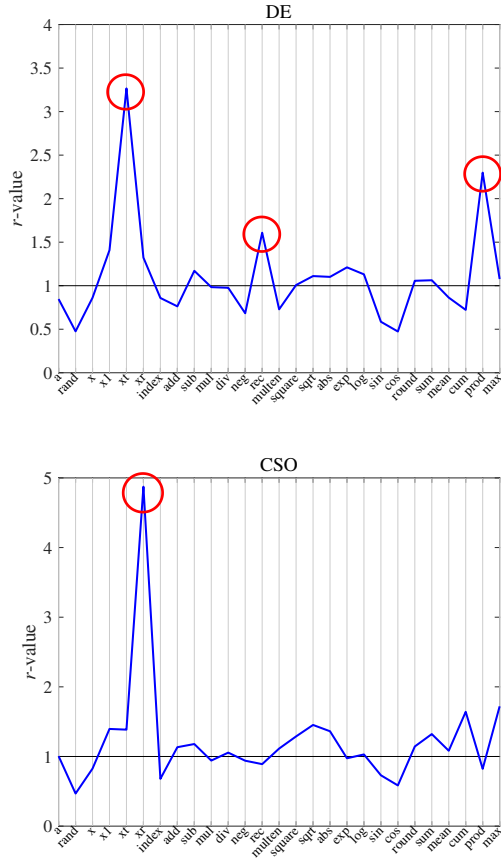


Fig. 9. r -values of all the operands and operators on DE and CSO.

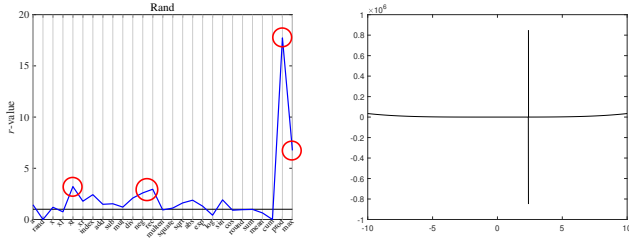


Fig. 10. r -values of all the operands and operators on random search (left) and a problem $f(x) = \frac{8.4931}{x-2.3911} + (1.8571x^2)^2$ where random search performs the best (right).

random search performs the best looks quite unusual and difficult to optimize. Random search can indeed show competitiveness on some real-world applications such as neural architecture search [69], [70], in which the mapping between decision variables (e.g., architecture of deep neural network) and the objective value (e.g., classification error) is very complex.

C. Performance of the Proposed Method

According to the above analysis, the performance of each metaheuristic is highly related to the operands and operators in the problem, hence it is reasonable to use the operands and operators as features for algorithm recommendation. To study the performance of

the proposed method, two recently proposed algorithm recommendation methods are adopted as baselines here. The first method (denoted as Baseline1) [33] samples a number of solutions on each problem, and directly uses the objective values of all the solutions as the features. The second method (denoted as Baseline2) [34] also samples a number of solutions on each problem, while a number of landscape-related features are extracted from the objective values of all the solutions. Besides, a naive method (denoted as Baseline0) is also involved, which always recommends the algorithm having the best overall performance (i.e., GA) on all the problems.

Two variants of the proposed method are involved in the experiment, where one variant (termed AR-WB) is meant for white-box problems and the other (termed AR-BB) for black-box problems. Besides, five classifiers are adopted in Baseline1 and Baseline2, including decision tree (with Gini's diversity index) [71], k -nearest neighbor (k -NN, with $k = 3, 9, 99$) [72], naive Bayes (with Gaussian kernel) [73], multi-layer perceptron (MLP, with sigmoid function) [74], and support vector machine (SVM, with Gaussian kernel) [75]. Note that these classifiers cannot be used in the proposed method since the input (i.e., reverse Polish expression) has a variable length. For a fair comparison, the total number of sampled solutions on each sample for Baseline1, Baseline2, and AR-BB is set to 500; the population size N , the number of generations G , and the mutation probability in the symbolic regressor of AR-BB are set to 20, 10, and 0.1, respectively.

Table IV lists the prediction accuracy of Baseline0, Baseline1, Baseline2, and the proposed method on the 100,000 created benchmark problems with five-fold cross-validation. On the one hand, the proposed AR-WB gains the best accuracy of 92.15%, which is much higher than all the other methods. On the other hand, the proposed AR-BB also performs better than the baselines in terms of all the classifiers, having achieved a test accuracy of 73.26%. Table V presents the average true ranking of the algorithms selected by the compared methods. It can be observed that the results are consistent with those in Table IV, where AR-WB performs the best and is followed by AR-BB. The average true rankings of AR-WB and AR-BB are 1.251 and 2.057, respectively, which means that they can generally select the best or second best algorithm for all the problems.

Furthermore, the influence of the number of training samples and the number of sampled solutions on each training sample to the test accuracy is studied. Fig. 11 shows the test accuracy of Baseline1, Baseline2, and AR-BB with different numbers of training samples and sampled solutions. On the one hand, the test accuracy reduces rapidly with the decreased number of training samples, which indicates that a large number of random problems created by the proposed method are necessary for training an effective classifier. On the other hand, the test accuracy slightly degrades with the decreased number of sampled solutions, hence it is possible to sample

TABLE IV
TEST ACCURACY OBTAINED BY BASELINE0, BASELINE1, BASELINE2, AR-WB, AND AR-BB ON THE CREATED BENCHMARK PROBLEMS. THE BEST RESULT IS HIGHLIGHTED.

| Method | Decision tree | k -NN ($k=3$) | k -NN ($k=9$) | k -NN ($k=99$) | MLP | Naive Bayes | SVM |
|-----------|---------------|-------------------|-------------------|--------------------|--------|-------------|--------|
| Baseline0 | | | | 41.27% | | | |
| Baseline1 | 54.73% | 63.80% | 59.95% | 55.95% | 50.88% | 16.93% | 41.28% |
| Baseline2 | 62.98% | 66.22% | 67.57% | 62.20% | 54.46% | 47.28% | 60.68% |
| AR-WB | | | | 92.15% | | | |
| AR-BB | | | | 73.26% | | | |

TABLE V
TRUE RANKING OBTAINED BY BASELINE0, BASELINE1, BASELINE2, AR-WB, AND AR-BB ON THE CREATED BENCHMARK PROBLEMS. THE BEST RESULT IS HIGHLIGHTED.

| Method | Decision tree | k -NN ($k=3$) | k -NN ($k=9$) | k -NN ($k=99$) | MLP | Naive Bayes | SVM |
|-----------|---------------|-------------------|-------------------|--------------------|-------|-------------|-------|
| Baseline0 | | | | 2.347 | | | |
| Baseline1 | 2.441 | 2.314 | 2.448 | 2.496 | 2.492 | 4.689 | 2.353 |
| Baseline2 | 2.109 | 2.119 | 2.021 | 2.193 | 2.435 | 2.650 | 2.264 |
| AR-WB | | | | 1.251 | | | |
| AR-BB | | | | 2.057 | | | |

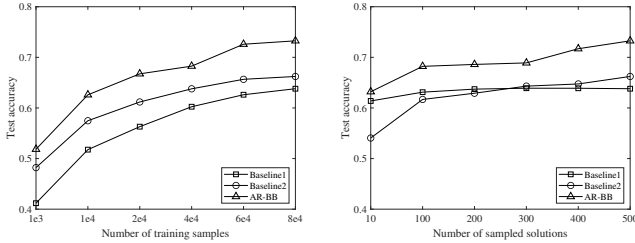


Fig. 11. Test accuracy of Baseline1, Baseline2, and AR-BB with different numbers of training samples (left) and sampled solutions (right).

fewer solutions on the problems with computationally expensive functions.

D. Transferability of the Proposed Method

The transferability of the proposed method is studied here on some real-world problems. The time series forecast problem [76] and the portfolio optimization problem [44] are adopted in the experiments, not only because they are very important and challenging tasks in quantitative finance, but also because their datasets can be automatically synthesized to create a sufficient number of test samples [77]. The detailed definitions of these two problems can be found in Supplementary Materials III.

Table VI presents the prediction accuracy of the compared methods on 10,000 time series forecast problems and 10,000 portfolio optimization problems with ten metaheuristics, where Baseline0 always recommends GA, Baseline1 and Baseline2 use the models trained by k -NN with $k=3$ on the 100,000 created benchmark

TABLE VI
TEST ACCURACY AND TRUE RANKING OBTAINED BY BASELINE0, BASELINE1, BASELINE2, AND AR-BB ON REAL-WORLD PROBLEMS. THE BEST RESULTS ARE HIGHLIGHTED.

| Method | Time series forecast | |
|-----------|------------------------|--------------|
| | Test accuracy | True ranking |
| Baseline0 | 35.45% | 3.039 |
| Baseline1 | 42.53% | 3.580 |
| Baseline2 | 52.68% | 2.785 |
| AR-BB | 61.84% | 2.728 |
| Method | Portfolio optimization | |
| | Test accuracy | True ranking |
| Baseline0 | 41.49% | 2.675 |
| Baseline1 | 54.89% | 3.219 |
| Baseline2 | 63.04% | 2.542 |
| AR-BB | 68.86% | 2.290 |

problems, and the deep recurrent neural network in AR-BB is also trained with the 100,000 created benchmark problems. The proposed AR-WB is not involved since the functions of the real-world problems cannot be directly represented as trees. According to the experimental results in Table VI, AR-BB obtains the best test accuracy and true ranking on both the two types of problems, which is followed by Baseline2, Baseline1, and Baseline0. In conclusion, the proposed AR-BB has better overall performance than the compared methods on real-world problems, and the high transferability of the proposed method can be verified.

V. CONCLUSIONS

This paper presents a recommender system for selecting metaheuristic algorithms for solving continuous optimization problems. In contrast to existing methods that extract landscape-related features on a small number of benchmark problems, the proposed method uses the operands and operators as features and generates a huge number of benchmark problems as training samples, thereby significantly enhancing the performance of the recommendation model. The proposed representation strategy cannot only represent a white-box function in the form of a tree structure consisting of operands and operators, it can also be used to estimate the tree representation of a black-box problem by means of symbolic regression. The tree of a problem is then converted into a reverse Polish expression and fed into a deep recurrent neural network, which is trained with a large number of benchmark problems created by a training data collection strategy.

According to the experimental results of ten metaheuristics on 100,000 problems created by the proposed training data collection strategy, instructive observations can be made to illustrate the strengths and weaknesses of each algorithm for continuous optimization. Although our results show the promises of the proposed recommender system, many questions remain open. For example, it is highly desired to further improve the recommendation performance on black-box problems without increasing the computational budget. Meanwhile, the proposed strategy for training data collection can be extended to other types of problems such as combinatorial optimization problems [18] and multi-objective optimization problems [33] as a tool for performance analysis. Finally, it is of great importance to find multiple and more informative features for representing and learning the difficulties in solving complex optimization problems.

REFERENCES

- [1] J. H. Holland, *Adaptation in Natural and Artificial Systems*. MIT Press, 1992.
- [2] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proceedings of the 6th International Symposium on Micro Machine and Human Science*, 1995, pp. 39–43.
- [3] R. Storn and K. Price, "Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.
- [4] K. Socha and M. Dorigo, "Ant colony optimization for continuous domains," *European Journal of Operational Research*, vol. 185, no. 3, pp. 1155–1173, 2008.
- [5] G. Hong and Z. Mao, "Immune algorithm," in *Proceedings of the 4th World Congress on Intelligent Control and Automation*, 2002, pp. 1784–1788.
- [6] X. Li, "A new intelligent optimization - artificial fish school algorithm," Ph.D. dissertation, Zhejiang University, China, 2003.
- [7] M. Eusuff, K. Lansey, and F. Pasha, "Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization," *Engineering Optimization*, vol. 38, no. 2, pp. 129–154, 2006.
- [8] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.
- [9] C. A. C. Coello, "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art," *Computer Methods in Applied Mechanics and Engineering*, vol. 191, no. 11–12, pp. 1245–1287, 2002.
- [10] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang, "Multiobjective evolutionary algorithms: A survey of the state of the art," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 32–49, 2011.
- [11] X. Li, M. G. Epitropakis, K. Deb, and A. Engelbrecht, "Seeking multiple solutions: An updated survey on niching methods and their applications," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 4, pp. 518–538, 2016.
- [12] Y. Jin, H. Wang, T. Chugh, D. Guo, and K. Miettinen, "Data-driven evolutionary optimization: An overview and case studies," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 3, pp. 442–458, 2019.
- [13] T. M. Lindauer, H. H. Hoos, F. Hutter, and T. Schaub, "AutoFolio: An automatically configured algorithm selector," *Journal of Artificial Intelligence Research*, vol. 53, pp. 745–778, 2015.
- [14] G. Endicott, M. Olhofer, T. Arima, and T. Sonoda, "A novel transonic fan swept outlet guide vane using 3D design optimization," in *Proceedings of the ASME Turbo Expo*, 2014.
- [15] T. Rodemann, K. Narukawa, M. Fischer, and M. Awada, "Many-objective optimization of a hybrid car controller," in *Applications of Evolutionary Computation*. Springer International Publishing, 2015, pp. 593–603.
- [16] N. Aulig and M. Olhofer, "State-based representation for structural topology optimization and application to crashworthiness," in *Proceedings of the 2016 IEEE Congress on Evolutionary Computation*, 2016.
- [17] M. A. Muñoz Acosta and K. A. Smith-Miles, "Performance analysis of continuous blackbox optimization algorithms via footprints in instance space," *Evolutionary Computation*, vol. 25, no. 4, pp. 529–554, 2017.
- [18] A. Liefvooghe, F. Daolio, S. Verel, B. Derbel, H. Aguirre, and K. Tanaka, "Landscape-aware performance prediction for evolutionary multi-objective optimization," *IEEE Transactions on Evolutionary Computation*, 2019, in press.
- [19] H. Huang, J. Su, Y. Zhang, and Z. Hao, "An experimental method to estimate running time of evolutionary algorithms for continuous optimization," *IEEE Transactions on Evolutionary Computation*, 2019, in press.
- [20] C. Huang, Y. Li, and X. Yao, "A survey of automatic parameter tuning methods for metaheuristics," *IEEE Transactions on Evolutionary Computation*, 2019, in press.
- [21] G. Karafotias, M. Hoogendoorn, and A. E. Eiben, "Parameter control in evolutionary algorithms: Trends and challenges," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 2, pp. 167–187, 2015.
- [22] T. Rodemann, "Industrial portfolio management for many-objective optimization algorithms," in *Proceedings of the 2018 IEEE Congress on Evolutionary Computation*, 2018.
- [23] C. Blum, J. Puchinger, G. R. Raidl, and A. Roli, "Hybrid metaheuristics in combinatorial optimization: A survey," *Applied Soft Computing*, vol. 11, no. 6, pp. 4135–4151, 2011.
- [24] A. LaTorre, S. Muelas, and J. M. Pena, "Multiple offspring sampling in large scale global optimization," in *Proceedings of the 2012 IEEE Congress on Evolutionary Computation*, 2012.
- [25] L. C. Bezerra, M. López-Ibáñez, and T. Stützle, "Automatic component-wise design of multiobjective evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 3, pp. 403–417, 2016.
- [26] M. Zhao, H. Ge, Y. Lian, and C. L. P. Chen, "Online decisioning meta-heuristic framework for large scale black-box optimization," *arXiv preprint arXiv:1812.06585v1*, 2018.
- [27] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann, "Automated algorithm selection: Survey and perspectives," *Evolutionary Computation*, vol. 27, no. 1, pp. 3–45, 2018.
- [28] E. Nudelman, K. Leyton-Brown, H. H. Hoos, A. Devkar, and Y. Shoham, "Understanding random SAT: Beyond the clauses-to-variables ratio," in *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming*, 2004, pp. 438–452.
- [29] M. Roberts, A. E. Howe, B. Wilson, and M. desJardins, "What makes planners predictable?" in *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling*, 2008, pp. 288–295.

- [30] K. A. Smith-Miles, "Towards insightful algorithm selection for optimisation using metalearning concepts," in *Proceedings of the IEEE International Joint Conference on Neural Networks*, 2008, pp. 4118–4124.
- [31] P. Kerschke, L. Kotthoff, J. Bossek, H. H. Hoos, and H. Trautmann, "Leveraging TSP solver complementarity through machine learning," *Evolutionary Computation*, vol. 26, no. 4, pp. 597–620, 2018.
- [32] P. Kerschke and H. Trautmann, "Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning," *Evolutionary Computation*, vol. 27, no. 1, pp. 99–127, 2019.
- [33] Y. Tian, S. Peng, T. Rodemann, X. Zhang, and Y. Jin, "Automated selection of evolutionary multi-objective optimization algorithms," in *Proceedings of the 2019 IEEE Symposium Series on Computational Intelligence*, 2019, pp. 3225–3232.
- [34] X. Chu, F. Cai, C. Cui, M. Hu, L. Li, and Q. Qin, "Adaptive recommendation model using meta-learning for population-based algorithms," *Information Sciences*, vol. 476, pp. 192–210, 2019.
- [35] M. Stein, "Large sample properties of simulations using Latin hypercube sampling," *Technometrics*, vol. 29, no. 2, pp. 143–151, 1987.
- [36] N. Hansen, S. Finck, R. Ros, and A. Auger, "Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions," INRIA, Tech. Rep. RR-6829, 2009.
- [37] L. B. S. Huband, P. Hingston and L. While, "A review of multi-objective test problems and a scalable test problem toolkit," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 5, pp. 477–506, 2006.
- [38] J. Liang, B. Qu, P. Suganthan, and Q. Chen, "Problem definitions and evaluation criteria for the CEC 2015 competition on learning-based real-parameter single objective optimization," Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Tech. Rep. 201411A, 2014.
- [39] Q. Zhang, A. Zhou, S. Zhao, P. N. Suganthan, W. Liu, and S. Tiwari, "Multiobjective optimization test instances for the CEC 2009 special session and competition," University of Essex, Colchester, UK and Nanyang technological University, Tech. Rep. CES-487, 2008.
- [40] M. Gürbüzbalaban and M. L. Overton, "On Nesterov's non-smooth Chebyshev–Rosenbrock functions," *Nonlinear Analysis: Theory, Methods & Applications*, vol. 75, no. 3, pp. 1282–1289, 2012.
- [41] X. Li, K. Tang, M. N. Omidvar, Z. Yang, and K. Qin, "Benchmark functions for the CEC'2013 special session and competition on large-scale global optimization," *Gene*, vol. 7, no. 33, 2013.
- [42] H. Cho, F. Olivera, and S. D. Guikema, "A derivation of the number of minima of the Griewank function," *Applied Mathematics and Computation*, vol. 204, no. 2, pp. 694–701, 2008.
- [43] K. Foli, T. Okabe, M. Olhofer, Y. Jin, and B. Sendhoff, "Optimization of micro heat exchanger: CFD, analytical approach and multi-objective evolutionary algorithms," *International Journal of Heat and Mass Transfer*, pp. 1090–1099, 2006.
- [44] A. Ponsich, A. L. Jaimes, and C. A. C. Coello, "A survey on multi-objective evolutionary algorithms for the solution of the portfolio optimization problem and other finance and economics applications," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 3, pp. 321–344, 2013.
- [45] R. Poli, W. B. Langdon, and N. F. McPhee, *A field guide to genetic programming*. <http://lulu.com>, 2008.
- [46] P. V. Krtolica and P. S. Stanimirović, "On some properties of reverse Polish notation," *Filomat*, pp. 157–172, 1999.
- [47] T. Mokolov, M. Karafiát, L. Burget, J. Černocký, and S. Khudanpur, "Recurrent neural network based language model," in *Proceedings of the Eleventh Annual Conference of the International Speech Communication Association*, 2010.
- [48] T. Mokolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *ICLR Workshop*, 2013.
- [49] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [50] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
- [51] R. Pasupathy and S. G. Henderson, "A testbed of simulation-optimization problems," in *Proceedings of the 2006 Winter Simulation Conference*, 2006, pp. 255–263.
- [52] M. G. Resende, R. Martí, M. Gallego, and A. Duarte, "GRASP and path relinking for the max–min diversity problem," *Computers & Operations Research*, vol. 37, no. 3, pp. 498–508, 2010.
- [53] C. He, R. Cheng, C. Zhang, Y. Tian, H. Li, and X. Yao, "Multi-objective formulation and analysis of the time-varying ratio error estimation," *IEEE Transactions on Evolutionary Computation*, 2020, in press.
- [54] J. Derrac, S. Garcia, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 3–18, 2011.
- [55] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," Erciyes University, Tech. Rep. tr06, 2005.
- [56] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.
- [57] R. Cheng and Y. Jin, "A competitive swarm optimizer for large scale optimization," *IEEE Transactions on Cybernetics*, vol. 45, no. 2, pp. 191–204, 2014.
- [58] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, pp. 82–102, 1999.
- [59] K. Deb and R. B. Agrawal, "Simulated binary crossover for continuous search space," *Complex Systems*, vol. 9, no. 4, pp. 115–148, 1995.
- [60] K. Deb and M. Goyal, "A combined genetic adaptive search (GeneAS) for engineering design," *Computer Science and Informatics*, vol. 26, no. 4, pp. 30–45, 1996.
- [61] D. Bertsimas, J. Tsitsiklis *et al.*, "Simulated annealing," *Statistical Science*, vol. 8, no. 1, pp. 10–15, 1993.
- [62] Y. Su, N. Guo, Y. Tian, and X. Zhang, "A non-revisiting genetic algorithm based on a novel binary space partition tree," *Information Sciences*, vol. 512, pp. 661–674, 2020.
- [63] B. Xue, M. Zhang, and W. N. Browne, "Particle swarm optimization for feature selection in classification: A multi-objective approach," *IEEE Transactions on Cybernetics*, vol. 43, no. 6, pp. 1656–1671, 2013.
- [64] Y. Tian, R. Cheng, X. Zhang, Y. Su, and Y. Jin, "A strengthened dominance relation considering convergence and diversity for evolutionary many-objective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 2, pp. 331–345, 2019.
- [65] M. Hellwig and H.-G. Beyer, "Evolution under strong noise: A self-adaptive evolution strategy can reach the lower performance bound - the pCMTSA-ES," in *Proceedings of the 2016 Parallel Problem Solving from Nature*, 2016, pp. 26–36.
- [66] H. Li, Q. Zhang, and J. Deng, "Biased multiobjective optimization and decomposition algorithm," *IEEE Transactions on Cybernetics*, vol. 47, no. 1, pp. 52–66, 2017.
- [67] Q. Zhang, W. Liu, and H. Li, "The performance of a new version of MOEA/D on CEC09 unconstrained MOP test instances," in *Proceedings of the 2009 IEEE Congress on Evolutionary Computation*, vol. 1, 2009, pp. 203–208.
- [68] Y. Tian, X. Zheng, X. Zhang, and Y. Jin, "Efficient large-scale multi-objective optimization based on a competitive swarm optimizer," *IEEE Transactions on Cybernetics*, 2019, in press.
- [69] C. Zhang, M. Ren, and R. Urtasun, "Graph hypernetworks for neural architecture search," *arXiv preprint arXiv:1810.05749*, 2018.
- [70] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "SMASH: One-shot model architecture search through hypernetworks," in *Proceedings of the 2018 International Conference on Learning Representations*, 2018.
- [71] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. CRC Press, 1984.
- [72] Y. Wu, K. Ianakiev, and V. Govindaraju, "Improved k -nearest neighbor classification," *Pattern Recognition*, vol. 35, no. 10, pp. 2311–2318, 2002.
- [73] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer, 2008.
- [74] I. D. Longstaff and J. F. Cross, "A pattern recognition approach to understanding the multi-layer perceptron," *Pattern Recognition Letters*, vol. 5, no. 5, pp. 315–319, 1987.
- [75] N. Christianini and J. C. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, 2000.
- [76] M. P. Clements and H. M. Krolzig, "A comparison of the forecast performance of Markov-switching and threshold autoregressive models of US GNP," *Econometrics Journal*, vol. 1, pp. C47–C75, 1998.

- [77] Y. Tian, C. Lu, X. Zhang, K. C. Tan, and Y. Jin, "Solving large-scale multi-objective optimization problems with sparse optimal solutions via unsupervised neural networks," *IEEE Transactions on Cybernetics*, 2020, in press.



Ye Tian received the B.Sc., M.Sc., and Ph.D. degrees from Anhui University, Hefei, China, in 2012, 2015, and 2018, respectively.

He is currently an Associate Professor with the Institutes of Physical Science and Information Technology, Anhui University, Hefei, China, and also a Postdoctoral Research Fellow with the Department of Computer Science, City University of Hong Kong, Hong Kong. His current research interests include multi-objective optimization methods and their applications.

He is the recipient of the 2018 and 2021 IEEE Transactions on Evolutionary Computation Outstanding Paper Award and the 2020 IEEE Computational Intelligence Magazine Outstanding Paper Award.



Shichen Peng received the B.Sc. degree from Anhui University, Hefei, China, in 2018, where he is currently pursuing the M.Sc. degree with the School of Computer Science and Technology, Anhui University, Hefei, China.

His current research interests include evolutionary optimization and deep learning.



Xingyi Zhang (SM'18) received the B.Sc. degree from Fuyang Normal College, Fuyang, China, in 2003, and the M.Sc. and Ph.D. degrees from Huazhong University of Science and Technology, Wuhan, China, in 2006 and 2009, respectively.

He is currently a Professor with the School of Computer Science and Technology, Anhui University, Hefei, China. His current research interests include unconventional models and algorithms of computation, multi-objective optimization, and membrane computing.

He is the recipient of the 2018 and 2021 IEEE Transactions on Evolutionary Computation Outstanding Paper Award and the 2020 IEEE Computational Intelligence Magazine Outstanding Paper Award.



Tobias Rodemann studied physics and neuroinformatics at the Ruhr Universität Bochum, Germany, and received his Dipl.-Phys. degree from the Universität Bochum in 1998 and a Ph.D. degree from the Technische Universität Bielefeld, Germany in 2003.

In 1998 he joined the Future Technology Research Division of Honda R&D Europe in Offenbach, Germany, where he worked on various research topics like biologically-inspired vision systems, computational neuroscience, and auditory processing.

Since 2011 he is working at the Honda Research Institute Europe as a principal scientist on system modeling and the application of many-objective optimization methods for energy management.



Kay Chen Tan (SM'08-F'14) received the B.Eng. (First Class Hons.) degree in electronics and electrical engineering and the Ph.D. degree from the University of Glasgow, Glasgow, U.K., in 1994 and 1997, respectively.

He is a Full Professor with the Department of Computer Science, City University of Hong Kong, Hong Kong. He has published over 200 refereed articles and six books.

Prof. Tan is the Editor-in-Chief of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, was the Editor-in-Chief of the IEEE Computational Intelligence Magazine from 2010 to 2013, and currently serves as the

Editorial Board Member of over 10 journals. He is an elected member of the IEEE CIS AdCom from 2017 to 2019.



Yaochu Jin (SM'02-F'16) received the B.Sc., M.Sc., and Ph.D. degrees from Zhejiang University, Hangzhou, China, in 1988, 1991, and 1996, respectively, and the Dr.-Ing. degree from Ruhr University Bochum, Bochum, Germany, in 2001.

He is currently a Distinguished Chair Professor in Computational Intelligence, Department of Computer Science, University of Surrey, Guildford, U.K., where he heads the Nature Inspired Computing and Engineering Group.

He was a Finland Distinguished Professor and a Changjiang Distinguished Visiting Professor. He has (co)authored over 300 peer-reviewed journal and conference papers and been granted eight patents on evolutionary optimization.

He is the Co-Editor-in-Chief of the IEEE Transactions on Cognitive and Developmental Systems and Complex & Intelligent Systems. He is also an Associate Editor or Editorial Board Member of the IEEE Transactions on Evolutionary Computation, IEEE Transactions on Cybernetics, IEEE Transactions on Nanobioscience, Evolutionary Computation, BioSystems, Soft Computing, and Natural Computing. He is an IEEE Distinguished Lecturer (2017-2019). He is the recipient of the 2014, 2016, and 2020 IEEE Computational Intelligence Magazine Outstanding Paper Award, the 2018 and 2021 IEEE Transactions on Evolutionary Computation Outstanding Paper Award, and the Best Paper Award of the 2010 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology. He has been named a Highly Cited Researcher for 2019 by the Web of Science group. He is a Fellow of IEEE.