Honda Research Institute EU
Honda Research Institute Europe GmbH
https://www.honda-ri.de/

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

Ye Tian, Shichen Peng, Xingyi Zhang, Tobias Rodemann, Kay Chen Tan, Yaochu Jin

2020

Preprint:

This is an accepted article published in IEEE Transactions on Artificial Intelligence . The final authenticated version is available online at: https://doi.org/10.1109/TAI.2020.3022339 Copyright 2020 IEEE

Algorithm Recommendation on Continuous Optimization Problems: Sample Generation and Feature Extraction

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 the optimization problems. One common idea thus is to design effective algorithms for different types of problems. While developing new algorithms is challenging, some recommendation methods have been proposed to apply existing algorithms for finding the best solutions on specific problems. However, existing recommendation methods for continuous optimization problems suffer from the drawbacks of low practicability and transferability, mainly due to the lack of diverse benchmark problems as training samples. To address the issue, this paper proposes a systematic method to create random continuous optimization problems by using a novel tree structure. The systematic method provides sufficient benchmark problems with various difficulties, giving birth to an effective algorithm recommendation method. For a black-box continuous optimization problem, the proposed recommendation method first estimates the most suitable tree by symbolic regression, then feeds the tree to a deep recurrent neural network to predict the best algorithm for the problem, where the deep recurrent neural network is trained with the benchmark problems created by the systematic method. The experiments on 100,000 benchmark problems show that the proposed recommendation method achieves considerably better prediction accuracy than existing recommendation methods, and it also exhibits high transferability on problems in real-world applications.**

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

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 algorithm [1], particle swarm optimization [2], differential evolution [3], ant colony optimization [4], and so on [5]–[7]. These algorithms are high-level methodologies that do not rely on specific characteristics of problems, having shown undoubted competitiveness in solving various complex optimization problems.

As revealed by the no free lunch theorem, there does not exist a single algorithm outperforming any others on all the optimization problems [8]. Hence, many algorithms have been showed to give a better performance on 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]. However, it is not practical to perform time-consuming experiments on various computationally expensive problems in the industry [14]–[16], which gives rise to a new subfield on performance analysis of metaheuristics [17].

There are several ideas to study the relation between algorithm performance and problem difficulty. For example, some work predicts the performance of algorithms on problems with specific settings [18], some estimates the running time when algorithms can converge to the global optimum of specific problems [19], some tunes the parameters of algorithms for achieving the best performance [20], some controls the parameters of algorithms during the search process [21], and some suggests hybrid metaheuristics that integrate the components of multiple algorithms [22].

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, 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. (email: yaochu.jin@surrey.ac.uk).

0000–0000/00$00.00 © 0000 IEEE

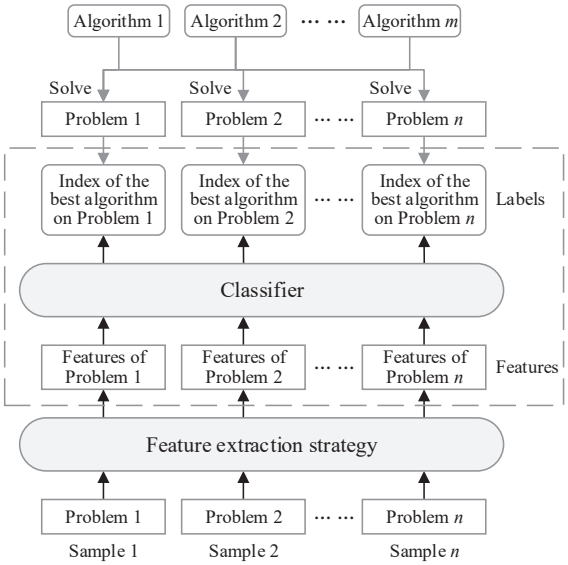


Fig. 1. General framework of algorithm recommendation.

Since the usage of these methods requires some understanding of the algorithms, engineers may find it difficult to use these methods as they may not necessarily be experts in metaheuristics [23]. Alternatively, the most practical and efficient solution would be algorithm recommendation, which directly selects the best-performing algorithm among multiple candidate algorithms for a given problem [24]. As illustrated in Fig. 1, algorithm recommendation can be regarded as a classification task, where each sample consists of the features of a problem and the corresponding label is the index of the best algorithm on the problem. Obviously, the core components of algorithm recommendation are the feature extraction strategy for representing the characteristics of problems and the classifier for learning the mapping between the features and the label, while no knowledge of metaheuristics is required. In the last decade, some methods have been tailored for algorithm recommendation on various combinatorial optimization problems including propositional satisfiability problem [25], automated planning [26], quadratic assignment problem [27], traveling salesman problem [28], and ρmnk-landscape [18]. However, they are difficult to be applied to each other since the encoding schemes and data structures of the combinatorial optimization problems are different. On the other hand, the study of algorithm recommendation on continuous optimization problems is still in the infancy, where only several methods have been developed based on a few algorithms and problems [29]–[31]. The difficulties of algorithm recommendation on continuous optimization problems mainly lie in the following two aspects:

1) Sample generation: In contrast to combinatorial optimization problems whose datasets can be automatically synthesized [18], [28], continuous optimization problems are generally represented by complex functions that are difficult to be arbitrarily generated. Hence, existing recommendation methods for continuous optimization problems can only adopt existing benchmark problems. For instance, 24 BBOB problems [32] were involved in [29], 12 WFG problems [33] were involved in [30], and 33 problems from the CEC competition [34] were involved in [31]. Although these benchmark problems are enough 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 benchmark problems consist of a limited number of manually designed mapping functions, which cannot provide sufficient diverse difficulties to meet all the candidate 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 to the training of recommendation model. Besides, it is very difficult to analyze the merits and demerits of different metaheuristics based on such a small number of difficulties introduced by the mapping functions.

2) Feature extraction: Due to the complexity of the functions in continuous optimization problems, existing methods regarded them as totally black-box problems and extracted 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 related to the landscapes of problems [30], hence these 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. While only a small number of function evaluations are available from real-world problems, the characterization of the landscape is inaccurate for complex problems.

As a consequence, algorithm recommendation still remains a promising but challenging topic, and many existing works are focused on analyzing the effectiveness of different features [18], [29] rather than pursuing a better performance of the recommendation model. To take a step forward, this work proposes a series of methods to improve the accuracy, interpretability, and transferability of algorithm recommendation on continuous optimization problems, where the main contributions include the following three aspects:

1) A systematic method is proposed to create continuous optimization problems as training samples. Based on a novel tree structure, the proposed method defines 7 operands as the left nodes and 20 operators as the non-leaf nodes of the tree. By extending an empty tree with randomly selected operands and operators, complex functions can be quickly created to construct many continu-

ous optimization problems with diverse difficulties. Moreover, three operations (i.e., difficulty injection, tree cleaning, and invalid problem elimination) are performed to further embellish the created problems. We test the performance of ten popular metaheuristics on 100,000 problems created by the proposed method. Based on a novel indicator, some meaningful observations are made according to the experimental results, shedding some light on the relations between problem difficulties and the performance of different algorithms. In short, the proposed systematic method not only provides sufficient diverse samples for training an effective recommendation model for continuous optimization problems, but also provides a tool for analyzing the merits and demerits of metaheuristics.

2) We find that the performance of metaheuristics is highly related to the operands and operators included in the problem. Motivated by this, an algorithm recommendation method is proposed by regarding the operands and operators as features. This method represents the function of a continuous optimization problem by the proposed tree structure and converts it to a reverse Polish expression. Then, the expression is regarded as a sentence and directly fed to a deep recurrent neural network used in natural language processing. Experiments on the 100,000 created problems show that this method achieves a prediction accuracy of 92.15%, as compared to some existing recommendation methods of up to 67.57% accuracy.
3) Moreover, the proposed recommendation method is extended to black-box problems whose functions are unknown. This method samples a number of solutions and calculates their objective values on the problem. Then, a symbolic regressor is used to find the tree for learning the mappings between the solutions and their objective values. The tree can thus be converted to a reverse Polish expression and fed to the deep recurrent neural network trained before. Experimental results indicate that this method achieves a prediction accuracy of 74.82% on the created benchmark problems, and it also exhibits high transferability on two types of real-world problems.

The rest of this paper is organized as follows. Section II presents the proposed systematic method for creating random continuous optimization problems and analyzes the performance of ten metaheuristics on 100,000 problems created by the systematic method. Section III details the proposed algorithm recommendation method and compares it to existing recommendation methods. Section IV presents the proposed algorithm recommendation method for black-box problems and reports its prediction accuracy on the created problems and real-world problems. Lastly, Section V concludes this work with some discussions on future work.

II. The Proposed Systematic Method

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

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

where f denotes the objective function, $\mathbf{x} = (x_1, \ldots, x_d)$ denotes the decision vector, Ω is the decision space, and d is the number of decision variables. The main idea of the proposed systematic method is to randomly create the function f with diverse difficulties, while the decision space and the number of decision variables can be arbitrarily specified for the created function.

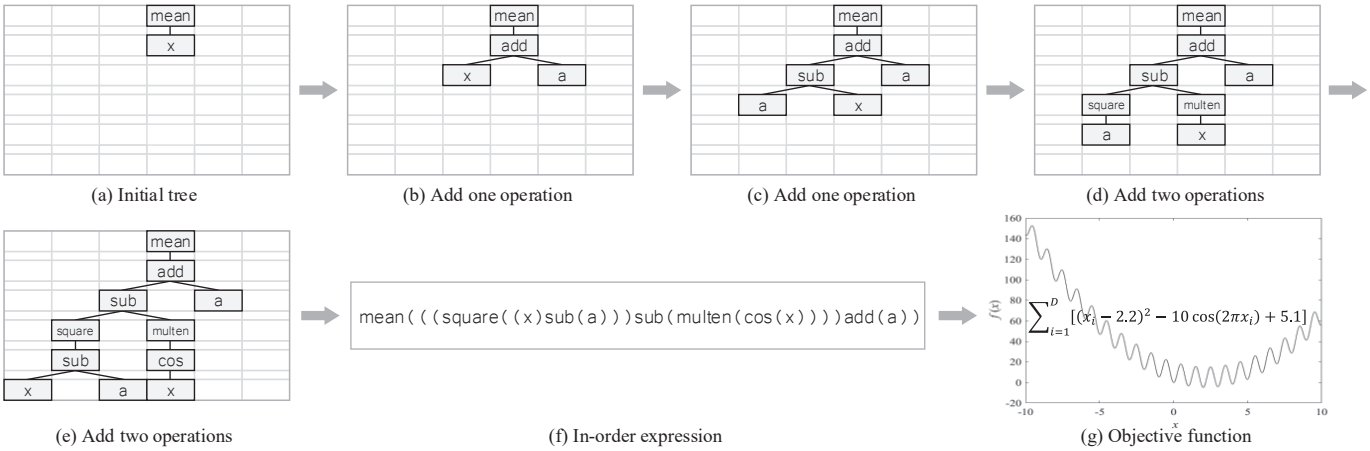
The difficulties of a function are determined by the included operators and operands. For example, a multimodal landscape can be provided by $\sin(x)$ or $\cos(x)$, and a linkage between decision variables can be provided by $(x_i - x_{i+1})^2$. However, randomly adding operators and operands to the 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, a novel tree structure is designed for creating complex functions in the proposed systematic method, the details of which are elaborated in the following.

A. Tree Construction

The proposed method encodes the function of a continuous optimization problem in a tree structure, in which each leaf node represents an operand and each non-leaf node represents an operator. Table I lists all the 7 operands and 20 operators used in the tree structure, which 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 within $[1, 10]$, whose value is randomly determined in the decoding operation. Note that we do not use the notations like `1`, `2.2`, and `3.14` to represent particular constants since we aim to create arbitrary functions rather than represent a particular one, and using a single operand `a` can highly decrease the total number of operands and facilitate the feature extraction operation. Besides, a notation `rand` is considered to represent a random number, whose value is changed in each function evaluation for providing 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 $\mathbf{x}$ can only create fully-separable functions, additional notations are also designed to create non-separable functions like many popular benchmark problems, including the first variable `x1` providing linkages between all the variables and a single one [36], the translated decision vector `xt` providing linkages between each two contiguous



(a) Initial tree (b) Add one operation (c) Add one operation (d) Add two operations

(e) Add two operations (f) In-order expression (g) Objective function

Fig. 2. Procedure of constructing a tree by the proposed systematic method.

TABLE I
ALL THE OPERANDS AND OPERATORS USED IN THE TREE STRUCTURE.

Notation	Meaning	Syntax	Probability of being selected
	Numbers		
a	A real constant in $[1, 10]$	a	0.76
rand	A random number in $[1, 1.1]$	$rand$	0.00
	Decision variables		
x	Decision vector	$(x_1, \ldots, x_d)$	0.12
x1	First variable	x_1	0.03
xt	Translated decision vector	$(x_2, \ldots, x_d, 0)$	0.03
xr	Rotated decision vector	$\mathbf{xr}$	0.03
index	Index vector	$(1, \ldots, d)$	0.03
	Binary operators		
add	Addition	$a + x$	0.16
sub	Subtraction	$a - x$	0.16
mul	Multiplication	$a \cdot x$	0.10
div	Division	a/x	0.10
	Unary operators		
neg	Negative	$-x$	0.03
rec	Reciprocal	$1/x$	0.03
multen	Multiplying by ten	$10x$	0.03
square	Square	x^2	0.06
sqrt	Square root	$\sqrt{\lvert x\rvert}$	0.06
abs	Absolute value	$\lvert x\rvert$	0.03
exp	Exponent	e^x	0.03
log	Logarithm	$\ln \lvert x\rvert$	0.03
sin	Sine	$\sin(2\pi x)$	0.03
cos	Cosine	$\cos(2\pi x)$	0.03
round	Rounded value	$\lceil x \rfloor$	0.03
	Vector-oriented operators		
sum	Sum of vector	$\sum_{i=1}^{d} x_i$	0.03
mean	Mean of vector	$\frac{1}{d}\sum_{i=1}^{d} x_i$	0.03
cum	Cumulative sum of vector	$(\sum_{i=1}^{1} x_i, \ldots, \sum_{i=1}^{d} x_i)$	0.01
prod	Product of vector	$\prod_{i=1}^{d} x_i$	0.01
max	Maximum value of vector	$\max_{i=1,\ldots,d} x_i$	0.01

variables [37], the rotated decision vector `xr` providing complex linkages between all the variables [38], and the index vector `index` providing different optimal values to all the variables [39].

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, $b = \ln a$, and e denotes the Euler's number.

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 continuous 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, a tree can be constructed by randomly selecting the operands and operators one after another. It is worth mentioning that the operands and operators are given different probabilities to be selected, since some operands and operators (e.g., `xt` and `round`) should be more rare than the others (e.g., `a` and `add`). Besides, the probabilities of selecting some operands and operators are set to very small values or even zero, since they will be selected with higher probabilities in the difficulty injection operation. The probabilities are assigned by experience, which can be modified to create functions with other biases.

Fig. 2 and Algorithm 1 present the detailed procedure of constructing a 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

Algorithm 1: $generate_tree(D)$

Input: D (number of operations in the tree)
Output: $tree$ (the constructed tree)

$tree.value \leftarrow$ 'mean'; //Notation of the node
$tree.left.value \leftarrow \emptyset$; //Left child
$tree.right.value \leftarrow$ 'x'; //Right child
for $i = 2$ *to* D **do**
  //Randomly reach a leaf node
  $t \leftarrow tree$;
  while $t.right \neq \emptyset$ **do**
    if $t.left == \emptyset$ *or* $rand < 0.5$ **then**
      $t \leftarrow t.right$;
    else
      $t \leftarrow t.left$;
  //Extend the leaf node
  $operator \leftarrow$ Randomly select an operator;
  if *operator is a binary operator* **then**
    $operand \leftarrow$ Randomly select an operand;
    if $rand < 0.5$ **then**
      $t.left.value \leftarrow t.value$;
      $t.right.value \leftarrow operand$;
    else
      $t.left.value \leftarrow operand$;
      $t.right.value \leftarrow t.value$;
    $t.value \leftarrow operator$;
  else
    $t.right.value \leftarrow t.value$;
    $t.value \leftarrow operator$;
return $tree$;

Algorithm 2: $tree_to_func(tree)$

Input: $tree$ (the constructed tree)
Output: $func$ (the created function)

$func \leftarrow (tree_to_func(tree.left)) \cup tree.value \cup (tree_to_func(tree.right))$;
return $func$;

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. Lastly, the constructed tree can be recursively converted to a function by the in-order traversal as shown in Algorithm 2.

B. Embellishment of Trees

It is obvious that the proposed method can create complex functions rather than monomials. However, the created functions may have very simple landscapes or invalid objective values. To address this issue, the following introduces three operations to modify the constructed trees.

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

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

1) Difficulty injection: In practice, the optimization problems in real-world applications usually have various difficulties such as noise [40], flat landscapes [41], multimodal landscapes [11], and complex linkages between variables [42]. 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 $\mathbf{x}$, including noise, flat landscape, multimodal landscape, highly multimodal landscape, linkages between variables, and different optimal values of variables. In the proposed method, the difficulty injection operation is performed before converting the tree to a function, where 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 feature extraction operation, 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 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 tree cleaning operation is also performed before converting the tree to a function, and the pseudocode of this operation is presented in

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`	$\lvert\sqrt{\lvert x\rvert}\rvert$	$\sqrt{\lvert x\rvert}$
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`	$(a\cdot x_1)/(b\cdot x_2)$	$a\cdot x_1/x_2$

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 $x>3$. To address this issue, we directly 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 aim, we test each algorithm on each problem for multiple runs, then eliminate the problem if the best algorithm has similar result to any other algorithms statistically in terms of the Wilcoxon rank sum test [43] with a significance level of 0.05.

C. Data Sample Generation

We randomly create 100,000 continuous optimization problems with 8 to 12 operations. Then, we test ten algorithms on each problem to get the label of each sample, including artificial bee colony algorithm (ABC) [44], ant colony optimization (ACO) [4], covariance matrix adaptation evolution strategy (CMA-ES) [45], competitive swarm optimizer (CSO) [46], differential evolution (DE) with rand/1/bin [3], fast evolutionary programming (FEP) [47], genetic algorithm (GA) with simulated binary crossover [48] and polynomial mutation [49], particle swarm optimization (PSO) [2], simulated annealing (SA) [50], 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.

D. Analysis of Data Samples

Fig. 3 depicts the ratio of problems where each algorithm 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 showed high performance on many real-world applications [23], [51], [52]. 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 further examine the relation between algorithm performance and problem difficulty, an indicator is defined 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

$$r=\frac{O'\cdot L}{L'\cdot O}, \tag{2}$$

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

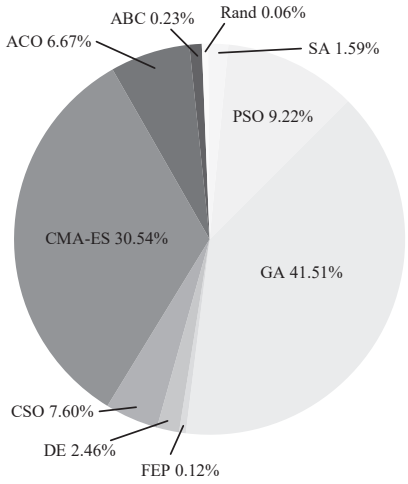


Fig. 3. Ratio of problems where each algorithm performs the best among 100,000 randomly created problems.

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. 4, 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 [49], [53] and CMA-ES is good at handling unimodal problems with noise due to the usage of Gaussian distribution model [54], [55]. According to Fig. 5, it can be seen that the problem where GA performs the best is quite rugged while the problem where CMA-ES performs the best is relatively smooth.

The r-values in Fig. 4 indicate that both GA and CMA-ES are bad at handling complex linkages between variables. In contrast, Fig. 6 shows the r-values of all the operands and operators on DE and CSO, where DE is 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 showed high performance in solving many challenging problems with very complex linkages in the literature [56], [57].

In addition, it is interesting to study when the random

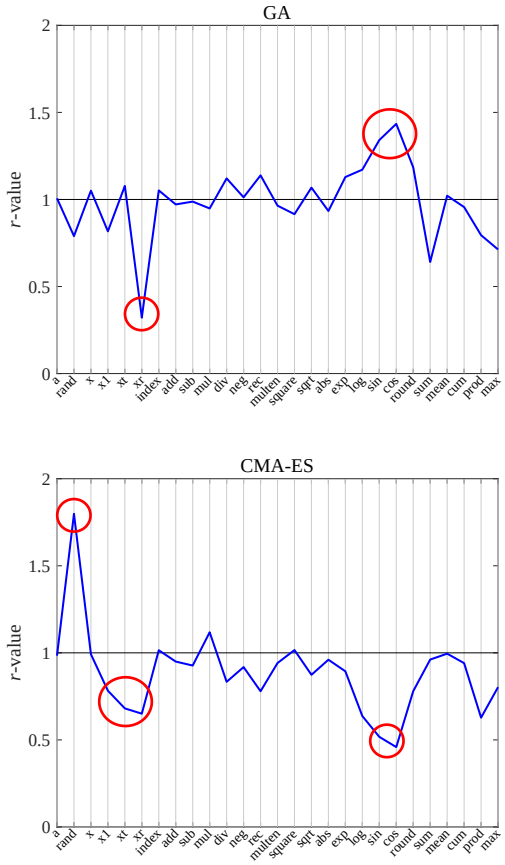


Fig. 4. r-values of all the operands and operators on GA and CMA-ES.

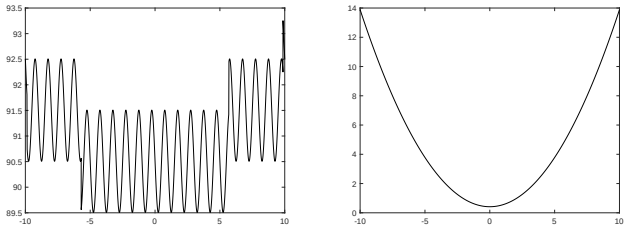


Fig. 5. A problem $f(x) = \lceil \frac{x}{8.1} \rceil^2 - \sin(2\pi x) + 81.9$ where GA performs the best (left) and a problem $f(x) = \frac{x^2+3.1}{7.4}$ where CMA-ES performs the best (right).

search can perform better than the others. According to the r-values shown in Fig. 7, the 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. 7, the problem where random search performs the best looks quite unusual and difficult to be optimized. The random search can indeed show competitiveness on some real-world applications such as neural architecture search [58], [59], in which the mapping between decision variables (e.g., architecture of deep neural network) and objective value (e.g., classification error) is very complex.

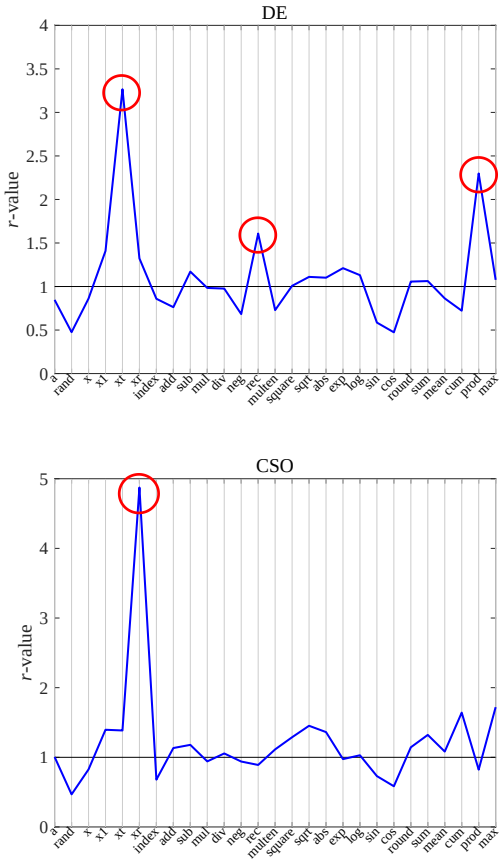


Fig. 6. r-values of all the operands and operators on DE and CSO.

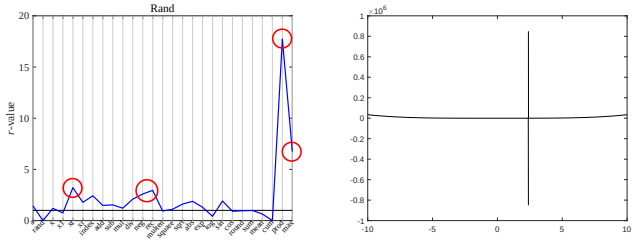


Fig. 7. r-values of all the operands and operators on random search (left) and a problem $f(x) = \frac{8.4931}{x-2.3911} + (1.8571x^2)^2$ where random search performs the best (right).

III. The Proposed Algorithm Recommendation Method

As can be observed from Figs. 4–7, the performance of an algorithm is highly related to the operands and operators included in the problem. Hence, the operands and operators can be used as features for predicting the best algorithm on the problem. In this section, an algorithm recommendation method based on the techniques in natural language processing is proposed, termed AR-NLP.

Algorithm 3: $tree_to_expr(tree)$

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

$expr \leftarrow tree_to_func(tree.left) \cup tree_to_func(tree.right) \cup tree.value$;
return $expr$;

Algorithm 4: $expr_to_func(expr)$

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

$func \leftarrow \emptyset$;
for $i = 1$ *to* $|expr|$ **do**
  $note \leftarrow expr(i)$; `//`i`-th notation`
  if $note$ *is an operand* **then**
    $func \leftarrow func \cup note$;
  else if $note$ *is a unary/vector-oriented operator* **then**
    $note2 \leftarrow func(end)$; `//Last notation`
    Delete $note2$ from $func$;
    $func \leftarrow func \cup (note \cup note2)$;
  else if $note$ *is a binary operator* **then**
    $note2 \leftarrow func(end)$; `//Last notation`
    $note3 \leftarrow func(end-1)$; `//Second last notation`
    Delete $note2$ and $note3$ from $func$;
    $func \leftarrow func \cup (note2 \cup note \cup note3)$;
return $func$;

A. Procedure of the Proposed AR-NLP

By taking advantage of the mathematical definition of the problem, the proposed AR-NLP represents the function as a tree and uses the notations in the tree as the features. For example, the function $\sum_{i=1}^{d}(10\sin(2\pi x_i) - 3)$ can be represented as `sum((multen(sin(x)))sub(a))` by the in-order traversal given in Algorithm 2. However, many parentheses should be added to determine the order of operations. To eliminate these parentheses for a compact representation, the proposed AR-NLP adopts the reverse Polish expression [60] to rearrange the notations. More specifically, as shown in Algorithm 3, the function is first represented as a tree, then the reverse Polish expression can be obtained by the post-order traversal. For examples, the reverse Polish expression of $\sum_{i=1}^{d}(10\sin(2\pi x_i)-3)$ is `x sin multen a sub sum`. Obviously, the reverse Polish expression does not contain any parenthesis and can be converted back to the function by Algorithm 4. Note that a post-order expression generally cannot be converted to an in-order expression, while the reverse Polish expression can be correctly converted to a function since there are two different types of nodes (i.e., operand and operator) in the tree.

Since the reverse Polish expression is a variable-length sentence with a finite vocabulary, the proposed AR-

TABLE IV
TEST ACCURACY OBTAINED BY BASELINE0, BASELINE1, BASELINE2, AND AR-NLP ON THE CREATED BENCHMARK PROBLEMS. 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-NLP				92.15%			

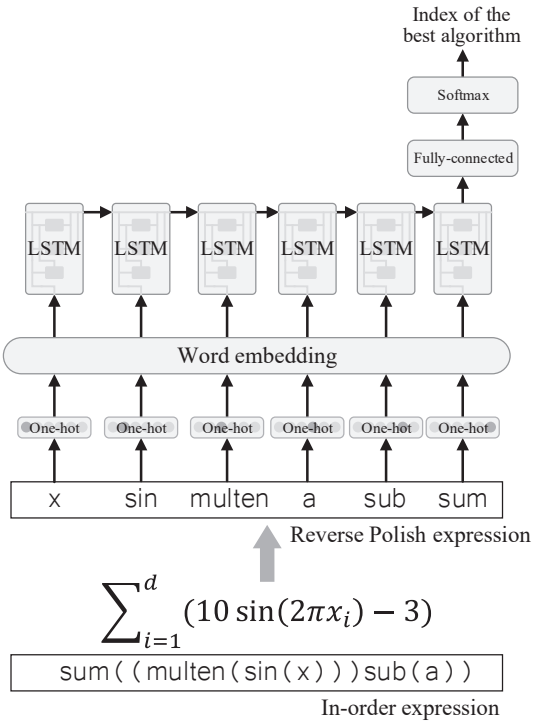


Fig. 8. The deep recurrent neural network used in AR-NLP.

NLP uses a deep recurrent neural network to handle it. As depicted in Fig. 8, the reverse Polish expression is first tokenized by encoding each notation in a one-hot representation [61], 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 to a word embedding layer [62] to obtain a word vector. Afterwards, the word vectors in the same expression are fed to an LSTM layer [63] 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 [64] with a learning rate of 0.01, a mini-batch size of 128, and 20 epochs, for minimizing the following cross entropy loss:

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

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. Comparison Between AR-NLP and Existing Methods

To compare the performance of the proposed algorithm recommendation method with existing ones, two recently proposed algorithm recommendation methods are adopted as baselines here. The first method (denoted as Baseline1) [30] 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) [31] also samples a number of solutions on each problem, while a number of landscape 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.

Table IV lists the prediction accuracy of Baseline0, Baseline1, Baseline2, and the proposed AR-NLP on the 100,000 created benchmark problems with five-fold cross-validation. For a fair comparison, the total number of sampled solutions on each sample for Baseline1 and Baseline2 is set to 500, and five classifiers are adopted in the two methods, including decision tree (with Gini's diversity index) [65], k-nearest neighbor (k-NN, with $k = 3, 9, 99$) [66], naive Bayes (with Gaussian kernel) [67], multi-layer perception (MLP, with sigmoid function) [68], and support vector machine (SVM, with Gaussian kernel) [69]. On the one hand, Baseline1 and Baseline2 outperform Baseline0 and obtain the best performance when using k-NN with k=3 and k=9, respectively. On the other hand, the proposed AR-NLP gains a test accuracy of 92.15%, which is much higher than all the other methods. Table V presents the average true ranking of the algorithms predicted by the compared methods. It can be observed that the results are consistent with those in Table IV, where AR-NLP performs significantly better

TABLE V
TRUE RANKING OBTAINED BY BASELINE0, BASELINE1, BASELINE2, AND AR-NLP 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-NLP				1.251			

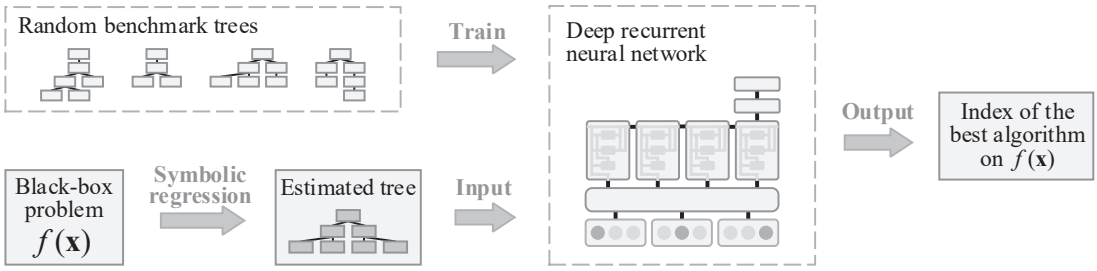


Fig. 9. Procedure of AR-NLP+SR for predicting the best algorithm for a black-box problem $f(\mathbf{x})$.

than the other methods. As a consequence, it is confirmed that the operands and operators are superior over landscape features for the algorithm recommendation on continuous optimization problems.

IV. THE PROPOSED ALGORITHM RECOMMENDATION METHOD FOR BLACK-BOX PROBLEMS

The proposed algorithm recommendation method is very efficient since it does not need to calculate the objective value of any solution on the problem. However, it is unsuitable for some real-world optimization problems whose operands and operators are unknown (e.g., the function is based on computational fluid dynamics [70] or a large dataset [71]). Therefore, this section extends the proposed method to those black-box problems by estimating the operands and operators via symbolic regression, termed AR-NLP+SR.

A. Procedure of the Proposed AR-NLP+SR

As illustrated in Fig. 9, the proposed AR-NLP+SR estimates the tree of the black-box problem and converts the tree to a reverse Polish expression, hence the best algorithm can be predicted by feeding the expression to the deep recurrent neural network trained in AR-NLP. Thus, the core task is to accurately estimate the tree of the black-box problem. Since the proposed tree structure contains different types of decision vectors and vector-oriented operators, we design a new symbolic regressor to estimate the operands and operators included in the problem efficiently, rather than use existing genetic programming method [72].

The procedure of the proposed symbolic regressor is presented in Algorithm 5. 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 between

Algorithm 5: $symbolic_regression(f)$

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

$X \leftarrow$ Sample a number of solutions by Latin hypercube sampling;
for *each* $\mathbf{x} \in X$ **do**
  Calculate $f(\mathbf{x})$;
for *each benchmark problem* g **do**
  $Sim(f, g) \leftarrow$ Calculate the similarity between functions f and g by (4);
$P \leftarrow$ Trees of the N most similar benchmark problems to f;
for *each* $tree \in P$ **do**
  Optimize the constants in $tree$ by random search.
Calculate the fitness of each tree in P by (4);
for $generation = 1$ *to* G **do**
  $P' \leftarrow$ Select N trees from P by binary tournament selection according to the fitness;
  $O \leftarrow recombination(P')$;
  for *each* $tree \in O$ **do**
    Optimize the constants in $tree$ by random search.
  Calculate the fitness of each tree in O by (4);
  $P \leftarrow P \cup O$;
  $P \leftarrow$ Remain the N best trees in P;
$tree \leftarrow$ The best tree in P;
return $tree$;

Algorithm 6: $recombination(P)$

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

$O \leftarrow \emptyset$;
//Crossover operator
while $P \neq \emptyset$ **do**
  $[tree_1, tree_2] \leftarrow$ Pick up two trees from P;
  $P \leftarrow P \setminus \{tree_1, tree_2\}$;
  $t_1 \leftarrow$ Randomly select a subtree from $tree_1$;
  $t_2 \leftarrow$ Randomly select a subtree from $tree_2$;
  Exchange t_1 and t_2;
  $O \leftarrow O \cup \{tree_1, tree_2\}$;
//Mutation operator
for *each* $tree \in O$ **do**
  if $rand < mutation_probability$ **then**
    $r \leftarrow rand$;
    if $r < \frac{1}{3}$ **then**
      $t \leftarrow$ Randomly select a leaf node from $tree$;
      Extend t by a randomly selected operator;
    else if $r < \frac{2}{3}$ **then**
      $t \leftarrow$ Randomly select a subtree from $tree$;
      $t' \leftarrow$ Randomly select a subtree from t;
      Replace t with t' in $tree$;
    else
      $t \leftarrow$ Randomly select a node from $tree$;
      $t' \leftarrow$ Randomly generate a node with the same category to t;
      Replace t with t' in $tree$;
return O;

f and g is measured according to the objective values:

$$Sim(f,g) = \sqrt{\sum_{\mathbf{x} \in X} (f(\mathbf{x}) - g(\mathbf{x}))^2}. \quad (4)$$

Based on the similarity between f and and 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, the 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 remained. 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 $\mathbf{x} \in X$ and $f(\mathbf{x})$ can be remained.

At each generation, N parents are first selected from P by binary tournament selection according to the fitness, and they are used to generate N offsprings by crossover and mutation operators. After optimizing the constants and calculating the objective values of the offsprings, the population is combined with the offsprings and N individuals having better fitness values survive to the next generation. Algorithm 6 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 t from the tree and replace the subtree with one of its subtree; c) randomly select a node from the tree and randomly modify its notation. The offspring population consists of all the trees generated by the crossover operator and tuned by the mutation operator.

TABLE VI
TEST ACCURACY AND TRUE RANKING OBTAINED BY BASELINE0, BASELINE1, BASELINE2, AND AR-NLP+SR ON THE CREATED BENCHMARK PROBLEMS. THE BEST RESULTS ARE HIGHLIGHTED.

Method	Time series forecast	
	Test accuracy	True ranking
Baseline0	41.27%	2.347
Baseline1	63.80%	2.314
Baseline2	66.22%	2.119
AR-NLP+SR	74.82%	1.980

B. Comparison Between AR-NLP+SR and Existing Methods

To study the performance of the proposed AR-NLP+SR, it is compared to Baseline0, Baseline1, Baseline2 on the 100,000 created benchmark problems with five-fold cross-validation. The k-NN with k=3 is used in Baseline1 and Baseline2, the total number of sampled solutions on each sample for Baseline1, Baseline2, and AR-NLP+SR is set to 500, the population size N, the number of generations G, and the mutation probability in the symbolic regressor of AR-NLP+SR are set to 20, 10, and 0.1, respectively. According to the test accuracy and true ranking obtained by the compared methods shown in Table VI, the proposed AR-NLP+SR is worse than AR-NLP but better than the baselines, having achieved a test accuracy of 74.82%. Therefore, the effectiveness of the proposed AR-NLP for the algorithm recommendation on black-box problems can be verified.

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. 10 shows the test accuracy of Baseline1, Baseline2, and AR-NLP+SR 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 systematic method is necessary for training an effective recommendation

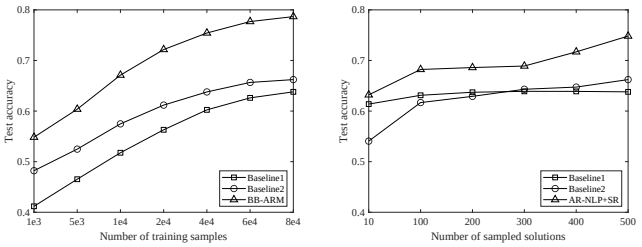


Fig. 10. Test accuracy of Baseline1, Baseline2, and AR-NLP+SR with different numbers of training samples (left) and sampled solutions (right).

model. On the other hand, the test accuracy reduces slightly with the decreased number of sampled solutions, hence it is possible to sample fewer solutions on the problems with computationally expensive functions.

C. Transferability of AR-NLP+SR

The transferability of AR-NLP+SR is studied here on some real-world problems. The time series forecast problem [73] and the portfolio optimization problem [71] 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 [74]. The detailed definitions of these two problems can be found in Supplementary Materials III.

Table VII 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 problems, and the deep recurrent neural network in AR-NLP+SR is also trained with the 100,000 created benchmark problems. According to the experimental results in Table VII, AR-NLP+SR 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-NLP+SR has better overall performance than the compared methods on real-world problems, and the high transferability of AR-NLP+SR can be verified.

TABLE VII
TEST ACCURACY AND TRUE RANKING OBTAINED BY BASELINE0, BASELINE1, BASELINE2, AND AR-NLP+SR 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-NLP+SR	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-NLP+SR	68.86%	2.290

V. CONCLUSIONS

To tackle the algorithm recommendation task on continuous optimization problems, this paper has proposed a systematic method for creating random problems as data samples. The proposed systematic method defines a number of nodes representing different operands and operators, and creates functions with diverse difficulties by constructing a binary tree with randomly selected nodes. Moreover, an algorithm recommendation method has been proposed, which converts the problem to a reverse Polish expression and feeds it to a deep recurrent neural network trained with a large number problems created by the proposed systematic method. Meanwhile, the proposed recommendation method is extended to black-box problems, where the reverse Polish expression of the problem is obtained by estimating the operands and operators via a novel symbolic regressor.

According to the experimental results of ten algorithms on 100,000 problems created by the proposed systematic method, some observations can be made to illustrate the merits and demerits of each algorithm on continuous optimization problems. It is thus interesting to extend the systematic method to other types of problems such as combinatorial optimization problems [18] and multi-objective optimization problems [30] as a tool for performance analysis. On the other hand, the experimental results have demonstrated that the proposed recommendation methods are more effective than existing algorithm recommendation methods. Hence, it is also desirable to enhance the proposed recommendation methods with more informative features and effective classifiers for other types of 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. Liefooghe, 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] 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.

[23] T. Rodemann, "Industrial portfolio management for many-objective optimization algorithms," in *Proceedings of the 2018 IEEE Congress on Evolutionary Computation*, 2018.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] M. Stein, "Large sample properties of simulations using Latin hypercube sampling," *Technometrics*, vol. 29, no. 2, pp. 143–151, 1987.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] R. Pasupathy and S. G. Henderson, "A testbed of simulation-optimization problems," in *Proceedings of the 2006 Winter Simulation Conference*, 2006, pp. 255–263.

[41] 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.

[42] 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.

[43] 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.

[44] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," Erciyes University, Tech. Rep. tr06, 2005.

[45] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[46] 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.

[47] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, pp. 82–102, 1999.

[48] K. Deb and R. B. Agrawal, "Simulated binary crossover for continuous search space," *Complex Systems*, vol. 9, no. 4, pp. 115–148, 1995.

[49] 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.

[50] D. Bertsimas, J. Tsitsiklis *et al.*, "Simulated annealing," *Statistical Science*, vol. 8, no. 1, pp. 10–15, 1993.

[51] 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.
[52] 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.
[53] 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.
[54] M. Hellwig and H.-G. Beyer, "Evolution under strong noise: A self-adaptive evolution strategy can reach the lower performance bound - the pcCMSA-ES," in *Proceedings of the 2016 Parallel Problem Solving from Nature*, 2016, pp. 26–36.
[55] 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.
[56] 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.
[57] 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.
[58] C. Zhang, M. Ren, and R. Urtasun, "Graph hypernetworks for neural architecture search," *arXiv preprint arXiv:1810.05749*, 2018.
[59] 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.
[60] P. V. Krtolica and P. S. Stanimirović, "On some properties of reverse Polish notation," *Filomat*, pp. 157–172, 1999.
[61] T. Mikolov, 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.
[62] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *ICLR Workshop*, 2013.
[63] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[64] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[65] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. CRC Press, 1984.
[66] Y. Wu, K. Ianakiev, and V. Govindaraju, "Improved k-nearest neighbor classification," *Pattern Recognition*, vol. 35, no. 10, pp. 2311–2318, 2002.
[67] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer, 2008.
[68] I. D. Longstaff and J. F. Cross, "A pattern recognition approach to understanding the multi-layer perception," *Pattern Recognition Letters*, vol. 5, no. 5, pp. 315–319, 1987.
[69] N. Christianini and J. C. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, 2000.
[70] 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.
[71] 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.
[72] R. Poli, W. B. Langdon, and N. F. McPhee, *A field guide to genetic programming*. http://lulu.com, 2008.
[73] 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.
[74] 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.