Multifactorial Genetic Programming for Symbolic Regression Problems

Jinghui Zhong, Liang Feng, Wentong Cai, and Yew-Soon Ong

***Abstract*—Genetic programming (GP) is a powerful evolutionary algorithm that has been widely used for solving many real-world optimization problems. However, traditional GP can only solve a single task in one independent run, which is inefficient in cases where multiple tasks need to be solved at the same time. Recently, multifactorial optimization (MFO) has been proposed as a new evolutionary paradigm toward evolutionary multitasking. It intends to conduct evolutionary search on multiple tasks in one independent run. To enable multitasking GP, in this paper, we propose a novel multifactorial GP (MFGP) algorithm. To the best of our knowledge, this is the first attempt in the literature to conduct multitasking GP using a single population. The proposed MFGP consists of a novel scalable chromosome encoding scheme which is capable of representing multiple solutions simultaneously, and new evolutionary mechanisms for MFO based on self-learning gene expression programming. Further, comprehensive experimental studies are conducted on multitask scenarios consisting of commonly used GP benchmark problems and real world applications. The obtained empirical results confirmed the efficacy of the proposed MFGP.**

***Index Terms*—Genetic programming (GP), multifactorial evolutionary algorithm (MFEA), multifactorial optimization (MFO), multitask learning (MTL), symbolic regression problem (SRP).**

I. Introduction

GENETIC programming (GP), which was first proposed by Cramer [1], is a powerful population-based evolutionary algorithm for solving user-defined tasks by automatic generation of computer programs [2]. In the past few years, GP has undergone a rapid development and many improved GP variants have been proposed, such as grammatical evolution [3], gene expression programming (GEP) [4], Cartesian GP [5], linear GP [6], and semantic GP [7]–[9]. GPs have also been successfully applied to solve a wide range of scientific and engineering applications such as classification problem, time series prediction problem, and rule identification problem [10]–[17].

In the literature, existing research works of GP could be generally divided into two groups. The first group is for single objective optimization (SOO), which aims to find a single optimal or near-optimal solution for the encountered problem. The second group focuses on multiobjective optimization (MOO), which tries to obtain a set of equally good solutions (i.e., the Pareto front) that holds unique tradeoff among multiple conflicting objectives. However, it is worth noting that, as GP contains an iterative evolution process, it could be extremely slow for solving complex optimization problems, in which a single solution evaluation may take minutes or even hours (e.g., those evolved simulations [18], [19]).

Today, it is well established that problems seldom exist in isolation, and problems often contain useful information that if properly harnessed, can lead to enhanced problem-solving process when another related problem is encountered [20]–[23]. For example, if two problems happen to have a common global optimum, solving one problem will get the other solved. Inspired by this, multitask learning (MTL) has been proposed in machine learning to learn multiple related tasks simultaneously for improving the generalization performance of each task. Over the years, various advanced MTL algorithms have been developed using the learners like artificial neural networks [24]–[26], and support vector machines [27]–[29]. These MTL algorithms have been further successfully applied to many real world applications, such as image processing [30] and data mining [31].

In spite of the accomplishments made in machine learning, application of multitasking for efficient search in evolutionary optimization, GP in particular, has to date received far less attention. In the literature, Krawiec and Wieloch [32], presented an evolutionary framework that uses a set of instructions provided by a GP problem to automatically build a repertoire of related problems that are used to bias the evolutionary search process. However, as this framework requires that all the problems share a common representation, it can only solve problems within the same domain and would fail to apply GP across different problem domains.

Manuscript received January 19, 2018; accepted June 28, 2018. This work was supported in part by the National Natural Science Foundation of China under Grant 61602181 and Grant 61603064, in part by the Program for Guangdong Introducing Innovative and Enterpreneurial Teams under Grant 2017ZT07X183, in part by the Fundamental Research Funds for the Central Universities under Grant 2017ZD053, in part by the Frontier Interdisciplinary Research Fund for the Central Universities under Grant 106112017CDJQJ188828, and in part by the Data Science and Artificial Intelligence Center at the Nanyang Technological University. This paper was recommended by Associate Editor J.-H. Chou. *(Corresponding author: Liang Feng.)*

J. Zhong is with the Guangdong Provincial Key Laboratory of Computational Intelligence and Cyberspace Information, School of Computer Science and Engineering, South China University of Technology, Guangzhou 510640, China (e-mail: jinghuizhong@gmail.com).

L. Feng is with the College of Computer Science, Chongqing University, Chongqing 400044, China (e-mail: liangf@cqu.edu.cn).

W. Cai and Y.-S. Ong are with the School of Computer Science and Engineering, Nanyang Technological University, Singapore (e-mail: aswtcai@ntu.edu.sg; asysong@ntu.edu.sg).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMC.2018.2853719

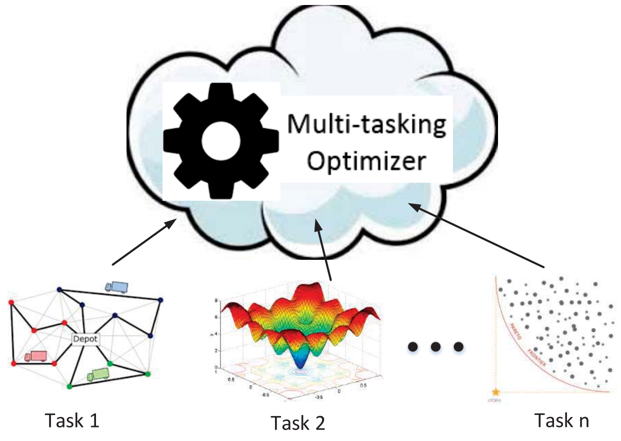


Fig. 1. MFO paradigm for evolutionary multitasking.

Recently, the concept of multifactorial optimization (MFO) has been introduced in [33] as a new optimization paradigm toward evolutionary multitasking with a single population of individuals. In contrast to traditional evolutionary search paradigm, as illustrated in Fig. 1, MFO intends to conduct evolutionary search on multiple concurrently search spaces corresponding to different tasks or optimization problems, each possessing a unique function landscape. As MFO keeps only one population for multiple tasks, useful traits of different tasks could be transferred among individuals via the sexual reproduction process, leading to enhanced search performance. The efficacy of MFO has been confirmed by a particular design of multifactorial evolutionary algorithm (MFEA) on a set of continuous and combinatorial optimization problems in [33]. However, despite the efficacy of MFEA, the backbones in MFEA such as common solution representation and task decoding, are problem dependent. These components have to be particularly designed in GP for solving GP problems such as symbolic regression problems (SRPs). To the best of our knowledge, there is no existing study on GP for evolutionary multitasking with a single population. Thus, this paper presents an attempt to fill this gap.

In particular, in this paper, we propose a multifactorial GP (MFGP) paradigm toward evolutionary multitasking GP. The proposed MFGP involves a novel scalable chromosome representation which is capable of providing a flexible solution encoding for problems across different domains. Further, novel evolutionary multitask mechanisms based on a recently published GP variant named SL-GEP [34], are designed to take both knowledge transfer and solution quality into consideration during the evolution process. Lastly, to evaluate the efficacy of the proposed MFGP for evolutionary multitasking, comprehensive empirical studies are conducted under multitask scenarios which are generated by two sets of problems.

The rest of this paper is organized as follows. Section II begins with an introduction of necessary background and related works for facilitating the understanding of our proposed MFGP. The details of the proposed MFGP toward evolutionary multitasking is then presented in Section III. Next, Sections IV and V provide the experiment studies on the two sets of problems. Last but not least, the concluding remarks of this paper are drawn in Section VI.

II. Preliminaries

In this section, we first give the concept of MFO for evolutionary multitasking. Next, a particular evolutionary multitask algorithm [33], which inspired the proposed MFGP in this paper, is then introduced and discussed. Lastly, the review of related works on evolutionary multitasking in the literature is presented.

A. Multifactorial Optimization

The MFO has been defined in [33] as an evolutionary multitask paradigm that builds on the implicit parallelism of population-based search with the aim of finding the optimal solutions for multiple tasks simultaneously. To solve multiple tasks, the traditional SOO (or MOO) requires different solvers with unique problem specific representation for each task, while MFO employs a unified problem representation and solves multiple tasks with one single solver simultaneously. In this manner, the implicit transfer of useful genetic material from one task to another for efficient evolutionary optimization may occur in MFO while search progresses.

In particular, suppose there are K unconstraint minimization problems needed to be solved concurrently. The ith problem is denoted as task T_i and the corresponding objective function of T_i is given by $f_i : X_i \to \mathbb{R}$, where X_i is the search space of T_i. The objective of MFO is to find a set of solutions $\{\mathbf{x}_1^*, \mathbf{x}_2^*, \ldots, \mathbf{x}_K^*\}$ in one single run, where $\mathbf{x}_i^*$ minimizes T_i, i.e.,

$$\mathbf{x}_i^* = \underset{\mathbf{x}}{\operatorname{argmin}} f_i(\mathbf{x}), i = 1, 2, \ldots, K. \tag{1}$$

To evaluate the population in multitasking environment, the following properties are defined in [33] for each individual. Note that individuals are all encoded in a unified search space encompassing the search space of all the tasks, which can be decoded into task-specific solution representation with respect to each of the K tasks.

1) *Factorial Cost:* The factorial cost f_i of an individual p denotes the objective value on a particular task T_i. For K tasks, there will be a vector with length K, in which each dimension gives the objective value of p on the corresponding task.
2) *Factorial Rank:* The factorial rank r_p^j simply denotes the index of individual p in the list of population members sorted in ascending order with respect to their factorial costs on the jth task.
3) *Skill Factor:* The skill factor τ_p of individual p denotes the task, on which p is most effective among all tasks in MFO, i.e., $\tau_p = \operatorname{argmin}\{r_p^j\}$, where $j \in \{1, \ldots, K\}$.
4) *Scalar Fitness:* The scalar fitness φ_p of an individual p is defined based on its best rank over all tasks, which is given by $\varphi_p = (1/[\min_{j\in\{1,\ldots,K\}} r_p^j])$.

With the concepts given above, the individuals are evaluated based on three phases, which is illustrated in Fig. 2. In particular, in the first phase, the *factorial costs* of each individual on all the tasks are calculated. In the second phase, for each task, the whole population is ranked in ascending order

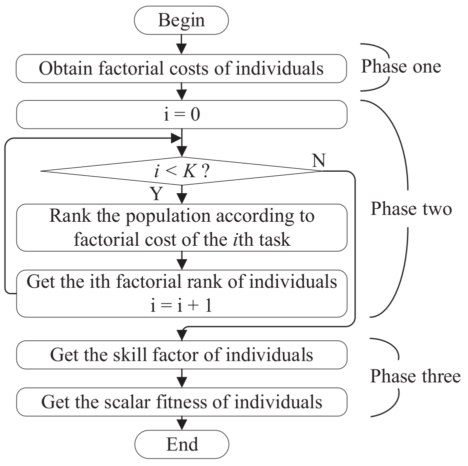

Begin
Obtain factorial costs of individuals
Phase one
i = 0
i < K ?
N
Y
Rank the population according to factorial cost of the ith task
Get the ith factorial rank of individuals i = i + 1
Phase two
Get the skill factor of individuals
Get the scalar fitness of individuals
Phase three
End

Fig. 2. Fitness evaluation procedure in MFO.

with respect to the *factorial cost* of each individual on the task. Next, based on the ranking results, the *factorial ranks* of individuals on each task are obtained. In this way, each individual possesses K *factorial ranks*, where K is the number of tasks to be solved. In the third phase, the *Skill Factor* of each individual is obtained based on its K *factorial ranks*, and the *scalar fitness* of each individual is then set as $\varphi_p = 1/r_p^j$, where j is the skill factor and r_p^j is the factorial rank of the individual on task j.

In MFO, the performance comparison between solutions are carried out based on the scalar fitness values of individuals. In particular, individual p_a is considered to dominate p_b in multifactorial sense simply if $\varphi_{p_a} > \varphi_{p_b}$. Therefore, suppose all the tasks are minimization problems, the definition of multifactorial optimality is given as [33]:

Multifactorial Optimality: An individual p^*, with a list of objective values $\{f_1^*, f_2^*, \ldots, f_K^*\}$ on K tasks, is considered optimum in multifactorial sense if and only if $\exists j \in [1, \ldots, K]$, such that $f_j^* \le f_j(\mathbf{x}_j)$, where $\mathbf{x}_j$ denotes all the feasible solutions in the search space of task T_j.

B. Multifactorial Evolutionary Algorithm

The MFEA was proposed by Gupta *et al.* [33] for MFO. The main procedures of the MFEA consist of five main steps. For the initialization of population (denote the initial population and the population size as P and N, respectively), each individual in MFEA is represented by a real vector with dimension $D_{\text{multitask}}$, where $D_{\text{multitask}} = \max\{D_j | j = 1, \ldots, K\}$, and each dimension of the individual is given by a randomly generated value lying in the range of [0, 1]. For task T_i with dimension D_i, the dimensions from 1 to D_i are considered for the individual evaluation on this task. The evaluation of P is conducted on all the K tasks to obtain the corresponding scalar fitness and skill factor for each individual.

Next, the *Population reproduction* kicks in to generate a set of offspring (denoted as Q) by using an assortative mating scheme. Specifically, for generating two offspring q_i and q_{i+1}, two parents p_a and p_b are randomly selected from P to undergo crossover with probability *rmp* which is a user defined parameter. Otherwise, q_i and q_{i+1} are obtained by performing mutation on p_a and p_b, respectively.

Furthermore, each individual in Q is evaluated by a selective evaluation that uses a vertical cultural transmission mode [21]. In the selective evaluation, the skill factor of each new individual is inherited from its parent. If an offspring has two parents, its skill factor is then configured to be one of the parent's skill factors with equal probability. In other words, each individual will only be evaluated on one task according to its skill factor, and the objective values of the unevaluated tasks are set to be $+\infty$. Subsequently, the current population and the newly generated offspring are concatenated to form a pool of population (denoted as temppop), i.e., temppop $= P \cup Q$. The skill factors and scalar fitness values of individuals in temppop are further updated by ranking the objective values of each individual on the K tasks.

Lastly, the fitter N individuals in temppop that have higher scalar fitness values are selected to form the new population for the next generation. If the stop criteria are not satisfied, MFEA will return back to the *Population reproduction*.

The efficacy of MFEA has been confirmed on a set of continuous (e.g., Sphere and Ackley) and discrete (e.g., knapsack problem and quadric assignment problem) problems. However, based on the descriptions above, it is straightforward to see that particular designs such as unified representation and task specific decoding, are required for conducting evolutionary multitasking in MFO on different problems, and the MFEA in [33] cannot be directly applied to GP.

C. Review on Existing Evolutionary Multitask Algorithms

Due to the efficacy of conducting multitask optimization, increasing research efforts on evolutionary multitasking has emerged in the literature. In particular, Gupta *et al.* [35] extended the MFEA framework for solving multiple MOO problems, while Zhou *et al.* [36] proposed a permutation-based unified representation and a split-based decoding operator for conducting MFO on the NP-hard capacitated vehicle routing problems. Toward positive knowledge sharing across tasks, Bali *et al.* [37] proposed a linearized domain adaptation strategy that transforms the search space of a simple task to the search space similar to its constitutive complex task. Further, Liaw and Ting [38] explored the resource allocation mechanism for evolutionary multitasking, and then proposed an evolutionary algorithm of biocoenosis through symbiosis for solving many-tasking optimization problem in [39]. To explore the generality of evolutionary multitasking with different search mechanisms, Feng *et al.* [40] presented two MFO approaches with particle swarm optimizer and differential evolution, respectively. Tang *et al.* [41] introduced an MFO algorithm for training multiple extreme learning machine with different number of hidden neurons for classification problems.

In contrast to the existing works, in this paper, we focus on designing multitasking GP for SRP. To the best of our knowledge, this is the first work in the literature on GP for multitask optimization. The detailed designs of the proposed multifactorial GP for SRPs will be presented in the next section.

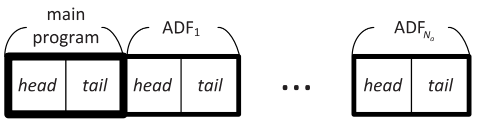


Fig. 3. Structure of the C-ADF proposed in [34].

III. Multifactorial Genetic Programming

In this section, the proposed MFGP is detailed. The proposed MFGP contains a novel scalable gene expression representation for encoding solutions across different domains, and new reproduction operators for conducting multifactorial evolution.

A. Scalable Gene Expression Representation

The solutions offered by GP (e.g., mathematical formulas and logical rules) are typically represented by expression tree (ET), which consists of interior function nodes and leave nodes. The function nodes denote functions such as "+," "sin," etc., while leave nodes represent terminals like variables and constants. The children of a function node are the inputs of this function. The output of a function could be either the final output or an input of another function node.

Gene expression representation with automatically defined functions [34] (labeled as C-ADF hereafter) is a recently proposed effective encoding scheme for GP, using a fixed length of strings. As illustrated in Fig. 3, C-ADF contains one main function and multiple automatically defined functions (ADFs). The main function gives the final output, while the ADFs represent subfunctions of the main function. Further, as depicted in Fig. 3, both the main function and ADFs are represented by the Karva expression (or K-expression) proposed in [42] which describes a solution via a fixed-length string that consists of functions and terminals. The K-expression can be converted to ET by the breadth first traversal method. Each K-expression contains two parts: *head* and *tail*. The *head* contains functions or terminals, while the *tail* consists of terminals only. To ensure that each chromosome can be converted to a valid ET, the lengths of *head* (h) and *tail* (l) are imposed with the following constraint:

$$l = h \cdot (u - 1) + 1 \tag{2}$$

where u is the number of children of the function with the maximum arity. A particular example of a chromosome including one main function and one ADF is given in Fig. 4. The decoded main function and ADF are also provided in Fig. 4. As can be observed, the final solution can be decoded as

$$2 \cdot \left(2 \cdot a^2 \cdot c + a\right)^2 \cdot b \cdot c. \tag{3}$$

The proposed scalable gene expression representation for multitasking is an extension of the C-ADF encoding scheme. We label the new representation as SC-ADF to facilitate description. As only a single population is required in MFO for solving multiple optimization tasks, the key challenge here is how to encode multiple solutions using a common representation. Note that different tasks across domains may have unique functions and terminals. To address this issue, in our proposed scalable gene expression representation, integers are employed to represent both functions and terminals. A single integer could represent various symbols for different tasks.

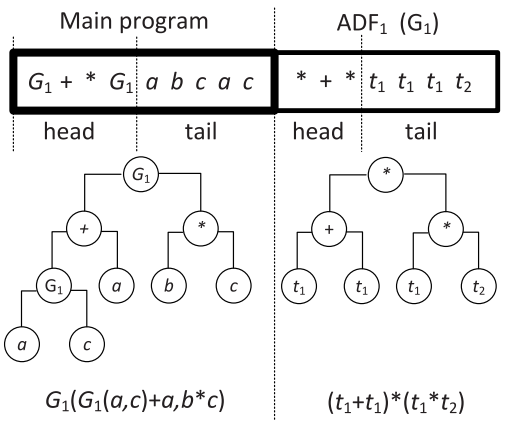


Fig. 4. Illustrating chromosome of GER/ADF.

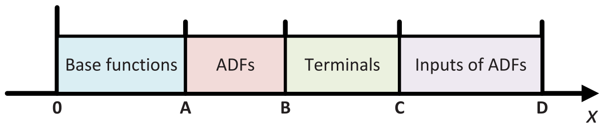


Fig. 5. Integer ranges of element types.

In particular, suppose there are K tasks need to be solved. The function set and terminal set of the ith task are $\mathbf{F}_i$ and $\mathbf{T}_i$, respectively. To ensure that any chromosome can be properly converted to a valid ET, the lengths of *head* and *tail* are imposed with the following constraint:

$$l = h \cdot \left(\max_{a \in \mathbf{F}_1 \cup \mathbf{F}_2 \ldots \cup \mathbf{F}_K} \xi(a) - 1 \right) + 1 \tag{4}$$

where $\xi(a)$ returns the number of arguments of function a. After determining the chromosome length, the second issue is to represent elements of each chromosome by using integers. Denote the number of ADFs in each chromosome and the number of input arguments in each ADF as N_a and N_g, respectively. Four integer ranges are defined to represent the elements (i.e., functions, terminals, ADFs, and input arguments), which are $[0, A - 1]$, $[A, B - 1]$, $[B, C - 1]$, and $[C, D - 1]$, as illustrated in Fig. 5. The values of A, B, C, and D are calculated by

$$A = \max_{i \in \{1, \ldots K\}} |\mathbf{F}_i| \tag{5}$$

where $|\mathbf{F}_i|$ returns the number of elements in set $\mathbf{F}_i$

$$B = A + N_a \tag{6}$$
$$C = B + \max_{i \in \{1, \ldots K\}} |\mathbf{T}_i| \tag{7}$$
$$D = C + N_g. \tag{8}$$

With this common chromosome representation, the decoding process that transforms a common chromosome to a task specific chromosome for evaluation can be conducted in the following two phases.

1) The first phase will figure out its corresponding type represented by each dimension of the chromosome.

This can be achieved by checking the range the integer belongs to, as illustrated in Fig. 5.

2) The second phase is to scale the integer according to the maximum number of elements for the type defined for the task, which is given by

$$x_i' = \lfloor (x_i - L_x)/(U_x - L_x) * N_x \rfloor \tag{9}$$

where $[L_x, U_x - 1]$ is the range of the element type found in the first phase and N_x is the maximum number of elements for this identified type. In this way, x_i' is mapped to an integer between 0 and $N_x - 1$, and we can subsequently use the scaled value as an index to map x_i' to a meaningful symbol for a specific task.

For example, suppose we have two tasks to solve. The first task is an SRP. The function set, ADFs, terminal set, and input arguments are defined as $\{+, -, *, /\}$, $\{G_1\}$, $\{a, b, c\}$, and $\{t_1, t_2\}$, respectively. The second one is an even-parity problem. The function set, terminal set, ADFs, and input arguments are defined as $\{\&, |\}$, $\{G_1\}$, $\{x_1, x_2\}$, and $\{t_1, t_2\}$, respectively. So, the maximum numbers of functions, terminals, ADFs, and input arguments of these two problems are then given by 4, 1, 3, and 2, respectively. Based on the proposed scalable gene expression representation, we have $A = 4, B = 5, C = 8, D = 10$. Let [4, 0, 2, 4, 5, 6, 7, 5, 7, 2, 0, 2, 8, 8, 8, 9] be an example of the SC-ADF encoded chromosome with one ADF. In this example, the head and tail lengths, i.e., (h, t), of the main program and ADF are set as (4, 5) and (3, 4), respectively. With this setting, the main function can be obtained based on the first nine dimensions (i.e., [4, 0, 2, 4, 5, 6, 7, 5, 7]), while the ADF is then achieved based on the rest dimensions (i.e., [2, 0, 2, 8, 8, 8, 9]). Further, to translate the chromosome to a solution of SRP, each dimension (x_i) of the chromosome has be to mapped to a meaningful symbol based on the number of functions, terminals, and ADFs defined for the SRP. In particular, this process consists of two phases. The first phase is to identify the type of x_i, while the second phase is to assign the correct symbol to x_i based on the corresponding type obtained in the first phase. For example, as the value of the first dimension is 4, which belongs to $[A, B - 1]$, the element type of the first dimension is an ADF (note that $A = 4$, $B = 5$ in this example). Then, in the second phase, we use (9) to assign the correct ADF to x_i. Since $\lfloor (4 - A)/(B - A) * 1 \rfloor = 0$, we set x_i as the first ADF (i.e., G_1, whose index is 0). In this way, we can translate the chromosome to the SRP solution $[G_1, +, *, G_1, a, b, c, a, c, *, +, *, t_1, t_1, t_1, t_2]$, which can be further decoded to a final expression as illustrated in (3).

Meanwhile, this chromosome can also be transformed to the even parity problem solution $[G_1, \&, |, G_1, x_1, x_1, x_2, x_1, x_2, |, \&, |, t_1, t_1, t_1, t_2]$ through the two-phase decoding. The corresponding final expression for the even parity problem is then given by

$$((((x_1 \& x_1)|(x_1|x_2))\&x_1)\&(((x_1\&x_1)|(x_1|x_2))\&x_1)) \\ |((((x_1\&x_1)|(x_1|x_2))\&x_1)|(x_1|x_2)). \tag{10}$$

B. Algorithmic Framework

Based on the SC-ADF representation, the details of our proposed MFGP are presented in this section. The outline of the MFGP is given in Algorithm 1. Some notations in Algorithm 1 are defined as follows: rand(a, b) returns a random value uniformly distributed within $[a, b]$; ϵ is a mutation probability calculated by (14).

In particular, the proposed MFGP mainly contains four steps, i.e., *Initialization*, *Population Reproduction*, *Concatenation*, and *Selection*, which are detailed as follows.

1) Step 1—Initialization: This step generates a random initial population (denoted the population and population size as *pop* and N, respectively.). Each individual is represented by a vector of integers using the SC-ADF representation, i.e.,

$$X_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,n}] \tag{11}$$

where $x_{i,j}$ is an integer within the feasible value range. In particular, a feasible type among {function, ADF, terminal, input argument} is randomly selected at the beginning. Next, $x_{i,j}$ is then set to a random integer within the value range of the selected type. For example, suppose $x_{i,j}$ belongs to the *head* of the main function, the set of feasible types for $x_{i,j}$ is then given by {function, ADF, terminal}. If "terminal" is the selected type and the value range of terminal is $[a, b]$, then $x_{i,j}$ is set to a random integer within $[a, b]$. Further, the number of integers in each individual (i.e., the length of chromosome) is obtained by

$$n = h + l + N_a * (h' + l') \tag{12}$$

where N_a is the number of ADFs in each individual, h (l) and h' (l') denote the *head* (*tail*) length of the main function and the ADFs, respectively.

Once all the individuals are initialized, their fitness values are evaluated by using the three phases as described in Section II-A.

2) Step 2—Population Reproduction: This step is to generate an offspring population (denoted as newpop) based on the current population pop. In contrast to the population reproduction conducted in traditional single task evolutionary search, two aspects should be considered in this process for concurrent evolution of multiple tasks. First of all, this reproduction should encourage the discovery and implicit transfer of useful genetic material across tasks, so that the useful traits found on one task could be transferred to improve the search on other tasks. Further, this reproduction should be able to provide good search capability for the problem encountered. Keeping these in mind, the proposed population reproduction in MFGP is designed based on assortative mating in MFEA [33] and the SL-GEP [34]. In particular, the ith individual in newpop is generated by either of the following two processes.

The first process is performed with a probability of *rmp*. In this process, the one-point crossover is employed to generate an offspring U_i by crossing X_i with a randomly selected parent X_{r1}. As different individuals may have unique skill factors, this crossover operation enhances the transfer of useful genetic materials found for different tasks. After that, a uniform mutation operation is performed on U_i to bring

Algorithm 1: Pseudo-Code of MFGP

```
   /* Step 1: Initialization */
1  Randomly generate an initial population;
2  Calculate the factorial costs of all tasks for each individual;
3  Evaluate the scalar fitness values of all individuals;
4  while stopping conditions are not satisfied do
       /* Step 2: Population Reproduction */
5      for $i = 1$ to $N$ do
6          if $rand(0, 1) < rmp$ then
               /* Perform one-point crossover and uniform mutation to generate offspring */
7              $r1 \leftarrow$ select a random individual index;
8              $Y_i \leftarrow$ perform crossover on $X_i$ and $X_{r1}$;
9              $U_i \leftarrow$ perform uniform mutation on $Y_i$;
               /* $U_i$ 's skill factor is inherited from $X_i$ or $X_{r1}$ */
10             $\tau$ of $U_i \leftarrow \tau$ of $X_i$ or $X_{r1}$
11         else
               /* Perform the DE mutation and crossover in SL-GEP to generate offspring */
12             $F \leftarrow rand(0, 1)$; $CR \leftarrow rand(0, 1)$;
13             $k \leftarrow$ a random integer between 1 and $n$;
14             for $j = 1$ to $n$ do
15                 Calculate mutation rate $\epsilon$ by Eq.(14)
16                 if ($rand(0, 1) < CR$ or $k = j$) and $rand(0, 1) < \epsilon$ then
17                     Set $u_{i,j}$ by the frequency-based assignment scheme in SL-GEP
18                 else
19                     $u_{i,j} \leftarrow x_{i,j}$
               /* $U_i$'s skill factor is inherited from $X_i$ */
20             $\tau$ of $U_i \leftarrow \tau$ of $X_i$
21         Evaluate one factorial cost of $U_i$ based on its $\tau$;
       /* Step 3: Concatenation */
22     $temppop \leftarrow \{U_1, \ldots, U_N\} \bigcup \{X_1, \ldots, X_N\}$;
23     Rank $temppop$ and update the fitness and skill factor of each individual in $temppop$;
       /* Step 4: Selection */
24     for $i = 1$ to $N$ do
25         $a \leftarrow \varphi(U_i)$; $b \leftarrow \varphi(X_i)$;
26         if $a == 1$ and $b == 1$ then
               /* Keep the best individual of each task in the new population */
27             $r1 \leftarrow$ a random individual index;
28             $X_{r1} \leftarrow U_i$;
29         else
               /* Remove the worse or redundant individuals */
30             if $a > b$ or $X_i$ is redundant then
31                 $X_i \leftarrow U_i$;
```

more population diversity and to avoid local stagnation. The skill factor of the generated offspring U_i is inherited from one of its parents with equal probability. Then, the factorial cost of U_i on the τth task (τ is the skill factor inherited from its parent) is evaluated. Fig. 6 shows a typical example of this reproduction process. In this example, the crossover point is seven. Thus, the offspring created by the crossover operator is comprised of the first seven dimensions of X_i and the last eight dimensions of X_{r1}. The 5th and the 12th dimensions of the offspring are further changed to new values after the mutation operator. The skill factor of the offspring is inherited from X_{r1} in this example.

On the other hand, if the assortative mating operation is not performed, the SL-GEP-based reproduction operation will be performed on X_i to generate an offspring U_i. Each dimension

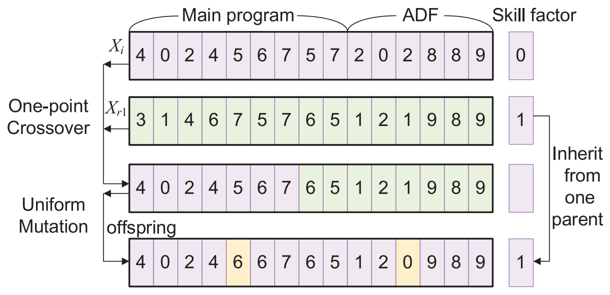


Fig. 6. Illustration of generating offspring based on the one-point crossover and uniform random mutation.

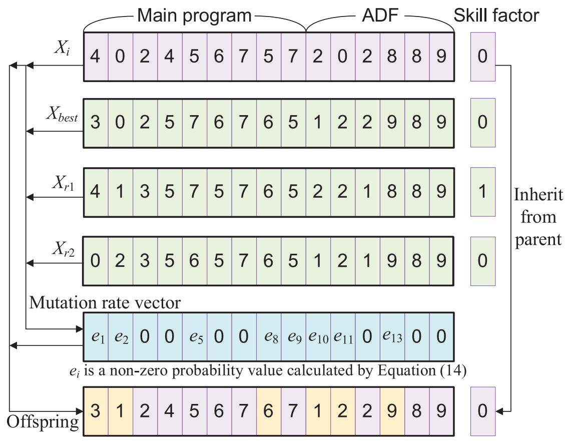


Fig. 7. Illustration of generating offspring based on the DE operators.

of U_i is set by

$$u_{i,j} = \begin{cases} t_{i,j}, & \text{if condition1 is satisfied} \\ x_{i,j}, & \text{otherwise} \end{cases} \tag{13}$$

where $t_{i,j}$ is a randomly generated value and condition1 is the mutation condition. We will describe how to generate $t_{i,j}$ and calculate condition1 in the following parts. Specifically, condition1 is set to be [rand(0, 1) $< CR$ or $j = k$] and [rand(0, 1) $< \epsilon$], where CR is a random value within 0 and 1, k is a random integer between 1 and n, $u_{i,j}$ and $x_{i,j}$ are the jth variables of U_i and X_i, respectively. The value of ϵ is calculated by

$$\epsilon = 1 - \left(1 - F \cdot \left[x_{\text{best},j} \neq x_{i,j}\right]\right) * \left(1 - F \cdot \left[x_{r2,j} \neq x_{r3,j}\right]\right) \tag{14}$$

where F is the scaling factor defined by user, $r2$ and $r3$ are two distinct random indices, $[a]$ is the Iverson bracket which returns 1 if a is true and 0 otherwise. Condition1 determines the probability of $u_{i,j}$ being assigned with a new value rather than $x_{i,j}$. When a mutation occurs, if $u_{i,j} \in$ ADF, $u_{i,j}$ is then assigned with a random feasible value following the process introduced in the initialization step. Else, $u_{i,j}$ is assigned based on the frequencies of functions and terminals in the current population. Element that appears more often in the population is more likely to be selected and assigned to $u_{i,j}$. Fig. 7 shows a typical example of this reproduction process. In this example, as the mutation rates of the 3rd, 4th, 6th, 7th, 12th, 14th, and 15th dimensions are zero according to (14), these dimensions are kept the same as X_i, while other dimensions may mutate to new values. The skill factor of the offspring is directly inherited from X_i. Once all the elements of U_i have been determined, the skill factor (τ) of U_i is set the same as X_i, and the factorial cost of U_i on the τth task is evaluated accordingly. The other factorial costs of U_i are set to be $+\infty$.

Based on the factorial costs of the individuals, the scalar fitness values of individuals can be calculated by using the three phases described in Section II-A. It is worth noting that each individual can represent different solutions of multiple tasks. However, the fitness evaluation process only calculates the factorial cost of one task rather than all the tasks. This could help save computational cost if the tasks to be solved are similar.

3) Step 3—Concatenation: In this step, the newly generated offspring population newpop is concatenated with the parent population pop to form a temporary population temppop, i.e.,

$$\text{temppop} = [U_1, U_2, \ldots, U_N, X_1, X_2, \ldots, X_N]. \tag{15}$$

Subsequently, the $2 * N$ individuals in temppop are ranked on each task independently according to their corresponding factorial costs. By doing this, each individual will have K ranks, where K is the total number of tasks. After that, the *Skill Factor* and *Scalar Fitness* of each individual are updated.

4) Step 4—Selection: To select individuals from temppop for survival in the next generation, the one-to-one selection strategy used in differential evolution algorithm is considered. In particular, in temppop, each offspring U_i is compared with its parent X_i. If both U_i and X_i rank first on their skill factors, X_i survives and U_i will be compared against another random individual in the population for survival. Otherwise, the fitter between U_i and X_i will survive for the next generation, i.e.,

$$X_i = \begin{cases} U_i, & \text{if } \varphi(U_i) > \varphi(X_i) \text{ or } X_i \text{ is redundant} \\ X_i, & \text{otherwise} \end{cases} \tag{16}$$

where $\varphi(X)$ returns the *Scalar Fitness* of X. X_i is deemed redundant when there exists another individual $X_j, j < i$ which is exactly the same as X_i. We replace a redundant individual with a new offspring so as to bring more population diversity.

IV. Experiment Studies on Benchmark Problems

In this section, empirical studies on benchmark SRPs are conducted to evaluate the effectiveness of the proposed MFGP. First, the details of the experiment settings such as the test problems, the algorithms's configurations, the performance metrics, and a distance measure for problem similarity analysis, are presented. Next, the experiment results are provided and discussed.

A. Experiment Settings

SRPs are the most common benchmark problems considered in the field of GP, which have a wide range of real-world applications [43]. In an SRP, a set of measurement data which consists of input variables and the corresponding output responses are given. The objective is to find a correct mathematical formula Γ to describe the relationship between the input variables and the outputs. Once the correct Γ is obtained,

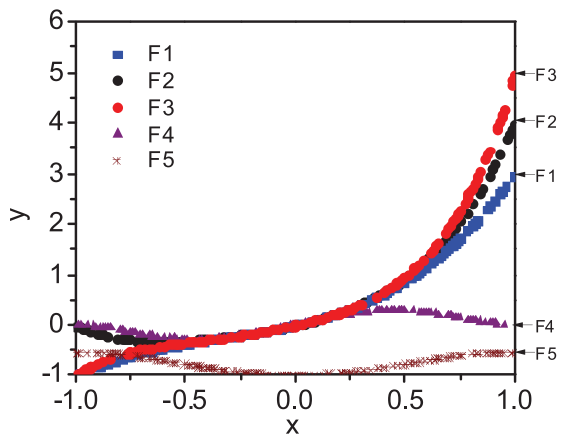


Fig. 8. Data sets of the five benchmark SRPs.

TABLE I
FIVE SRPS USED IN THE FIST CASE STUDY

Problem	Objective Function	Data set
F_1	$x^3 + x^2 + x$	$U[-1, 1, 200]$
F_2	$x^4 + x^3 + x^2 + x$	$U[-1, 1, 200]$
F_3	$x^5 + x^4 + x^3 + x^2 + x$	$U[-1, 1, 200]$
F_4	$x^5 - 2x^3 + x$	$U[-1, 1, 200]$
F_5	$sin(x^2)cos(x) - 1$	$U[-1, 1, 200]$

$U[a, b, c]$ represents c uniform random samples from a to b;

it can then be used to analyze the physical system that generates the data, and to predict the outputs of new input variables. Specifically, denote the ith sample data as

$$[t_{i,1}, t_{i,2}, \ldots, t_{i,m}, o_i] \quad (17)$$

where m is the number of input variables (i.e., the dimension of an SRP benchmark), $t_{i,j}$ denotes the jth variable value of the ith sampled data, and o_i gives the corresponding output. The formula Γ consists of function (e.g., +, "×," sin) and terminal (e.g., variables and constants). The functions and terminals are defined in advance for a given problem, and the goal of an SRP is to construct the optimal formula Γ^* by combining the functions and terminals so as to minimize the fitting error

$$\Gamma^* = \arg \min_{\Gamma} g(\Gamma) \quad (18)$$

where $g(\Gamma)$ returns the fitting error of Γ. Commonly, $g(\Gamma)$ is achieved by the root-mean-square-error (RMSE):

$$g(\Gamma) = \sqrt{\frac{\sum_{i=1}^{M} \left(\Gamma\left(t_{i,1}, \ldots, t_{i,m}\right) - o_i\right)^2}{M}} \quad (19)$$

where $\Gamma(t_{i,1}, \ldots, t_{i,m})$ is the output of Γ for the ith input data, o_i is the true output of the ith input data, and M is the number of samples in the training data set. The objective formulas of the five SRPs in this paper are listed in Table I and the data sets of the five problems are plotted in Fig. 8. In the literature, these benchmarks are commonly used to test the performance of GP variants. Here, we apply the proposed MFGP to solve two different SRPs simultaneously. Our objective is to investigate whether the proposed MFGP can solve two different SRPs more efficiently and effectively when compared to the solver that tackles a single SRP independently. We also investigate that how does the distance between two SRPs being solved affect the performance of MFGP. Further, to solve these problems, as suggested in [43], the terminal set is set to $\{x\}$ and the function set is set to $\{+, -, \times, \div, \sin, \cos, e^x, \ln(|x|)\}$.[1]

The proposed MFGP is designed based on a recently published GP variant named SL-GEP [34], which has been confirmed to be effective in solving single-task SRPs. Thus, we compare MFGP with SL-GEP to investigate the effectiveness of the proposed multitask evolutionary mechanism. According to [34], the parameters of SL-GEP are configured as: $N = 50$; $h = 10$; $h' = 3$; the number of ADFs is set as 2 [l and l' are set to h+1 and h'+1, respectively, based on (2)]. Meanwhile, for fair comparison, the common parameters of MFGP are kept the same as SL-GEP for all the problems. The distinct parameters in MFGP are configured as $rmp = 0.2$ and $pm = 0.002$ for all the problems. The maximum number of fitness evaluations are set to 1 000 000 in both MFGP and SL-GEP.

Further, the SL-GEP terminates when a perfect hit is achieved or the maximum number of fitness evaluations is reached, while the MFGP terminates when the perfect hits of both problems are achieved or the maximum number of fitness evaluations is reached. Lastly, 100 independent runs with different random seeds are conducted for both MFGP and SL-GEP on all the test problems.

In the experiment studies, we use the tenfold cross validation method for training and testing. First, the regression data of each problem is evenly divided into ten folds. Then, nine folds are used for training and the remaining one fold is used to test the best solution found by the algorithm. There are ten different training cases, and we perform each algorithm on each training case for ten times. In this way, each algorithm is performed for 100 times on each problem. We consider that an algorithm has reached a perfect hit when the algorithm converges to a solution Γ with $g(\Gamma) < 10^{-4}$. Further, to evaluate the algorithm performance, the first performance metric considered here is the average best fitting errors over 100 independent runs. In addition, the success rate of achieving perfect hits (denoted as SUC) is adopted as the second metric. The SUC is computed by

$$\text{SUC} = \frac{C_s}{C} \cdot 100\% \quad (20)$$

where C is the number of independent runs and C_s is the number of successful runs which achieved a perfect hit. In addition, when an algorithm achieves perfect hit on all the 100 independent runs, the averaged number of fitness evaluations that are required to reach a perfect hit (denoted as FES), is also investigated. The FES gives an indication of the convergence speed of an algorithm. In the case that the algorithm fails to achieve a perfect hit, the corresponding FES is ignored. It is worth noting that the SL-GEP is performed for each task independently. The FES of SL-GEP is thus independently counted on each task accordingly. Meanwhile, as MFGP solves two tasks

[1]In the simulation study, operators $\div$ and e^x are protected: 1) to avoid invalid calculation, $x \div 0 = 0$ and 2) to avoid float overflow, if $|x| > 20$, then $e^x = e^{20}$.

TABLE II
COMPARISON RESULTS OF MFGP AND SL-GEP

		F_1			F_2			F_3			F_4			F_5		
	Paired Problem	SUC	FES	$RMSE$	SUC	FES	$RMSE$	SUC	FES	$RMSE$	SUC	FES	$RMSE$	SUC	FES	$RMSE$
MFGP	F_1	N/A	N/A	N/A	100	6,748	0∼	100	12,053	0∼	59	N/A	7.92E-4∼	58	N/A	0.00167‡
	F_2	100	3,405	0∼	N/A	N/A	N/A	100	**11,887**	0∼	73	N/A	9.24E-4∼	63	N/A	0.00201‡
	F_3	100	**2,814**	0∼	100	**6,726**	0∼	N/A	N/A	N/A	68	N/A	**5.50E-4**∼	61	N/A	0.00211‡
	F_4	100	5,780	0∼	100	1,277	0∼	100	26,949	0∼	N/A	N/A	N/A	73	N/A	9.24E-4‡
	F_5	100	4,678	0∼	100	17,573	0∼	98	N/A	5.12E-4∼	63	N/A	9.37E-4∼	N/A	N/A	N/A
SL-GEP		100	6,272	0	100	15,300	0	100	25,203	0	72	N/A	8.93E-4	87	N/A	**4.45E-4**

∼ (or ‡) means the $RMSE$ results of MFGP are similar to (or significantly worse than) those of SL-GEP, according to the Wilcoxon test at 95% confidence level.

simultaneously, the corresponding FES on one task is the number of function evaluations that are used only to evaluate this task.

To conduct evolutionary multitasking, the relatedness between tasks has great impact on the performance of the MFO search paradigm. Similar tasks usually possess common information that can be transferred across tasks to enhance the problem-solving process if properly harnessed. Therefore, to facilitate investigating the impact of similarity between tasks on the performance of the proposed algorithm, we define a distance measure for the SRPs considered in this paper. For the SRPs considered in this paper, their input–output pairs can be treated as time series. Thus, the distance measure for time series can be used to calculate the distance between SRPs. Taking this cue, here we consider the perceptually important point (PIP) approach [44], which is a simple and effective distance measure for two time series segments, for calculating distances between SRPs.

Particularly, the PIP-based distance measure between SRPs is based on three phases. In phase one, the overlap interval $[x_{\min}, x_{\max}]$ of the datasets of two SRPs is first calculated. If two problems have no overlap, then these two problems are deemed as totally different. Next, each point (x, y) in the two data segments that falls into the overlap interval are normalized by

$$\begin{cases} x' = (x - x_{\min})/(x_{\max} - x_{\min}) \\ y' = (y - y_{\min})/(y_{\max} - y_{\min}) \end{cases} \tag{21}$$

where (x', y') is the normalized point, $x_{\min}$ and $x_{\max}$ are the minimum and maximum inputs of the two data segments, respectively, $y_{\min}$ and $y_{\max}$ are the minimum and maximum outputs of the two data segments, respectively. Further, in phase two, κ PIPs are identified to capture the general shape of the normalized segments. The set of PIP points are obtained by the method proposed in [44]. Lastly, the third phase is to calculate the distance between the two SRPs based on the obtained PIPs, which is given by

$$D(\mathbf{U}, \mathbf{V}) = \frac{1}{2 \cdot \kappa}\left(\sum_{i=1}^{\kappa} \operatorname{dis}(p_i, \mathbf{V}') + \sum_{i=1}^{\kappa} \operatorname{dis}\big(q_i, \mathbf{U}'\big)\right) \tag{22}$$

where $\mathbf{U}$ and $\mathbf{V}$ are the two SRPs. $\mathbf{U}'$ and $\mathbf{V}'$ denote the PIPs of $\mathbf{U}$ and $\mathbf{V}$, respectively. p_i and q_i give the ith points in $\mathbf{U}'$ and $\mathbf{V}'$, respectively. κ represents the number of PIPs and dis$(p, \mathbf{V})$ is the minimum Euclid distance between p and the points in $\mathbf{V}$.

TABLE III
DISTANCES BETWEEN THE FIVE BENCHMARK PROBLEMS

Problem	F_1	F_2	F_3	F_4	F_5
F_1	**0**	**0.118**	**0.084**	0.357	0.312
F_2	**0.118**	**0**	**0.051**	0.398	0.351
F_3	**0.084**	**0.051**	**0**	0.381	0.331
F_4	0.357	0.398	0.381	**0**	0.443
F_5	0.312	0.351	0.331	0.443	**0**

B. Results and Analysis

In this case study, the MFGP is applied to solve a pair of problems simultaneously in a single run. Five benchmark SRPs are used for testing, thus there are totally $C_5^2 = 10$ pairs of problems and the MFGP is applied to solve all the ten pairs of problems. Table II summarizes the performances of SL-GEP and MFGP on the five SRPs. In Table II, the first row represents the tasks which are solved together with the paired problems. The results of MFGP on each task are listed and compared against those of SL-GEP. For example, the MFGP achieved an FES of 3405 on F_1 when it is applied to solve the problem pair $\{F_1, F_2\}$, while the SL-GEP achieved an FES of 6272 on F_1. It can be observed that MFGP can achieve better performance than SL-GEP in terms of FES, when the pair of problems are from the same problem set $\{F_1, F_2, F_3\}$. However, the performance of MFGP may degrade if the first task is from $\{F_1, F_2, F_3\}$, while the paired problem is from $\{F_4, F_5\}$.

To provide deeper insights of the results obtained by the MFGP above, we first recall the curves plotted in Fig. 8. As can be observed, among all the five SRPs, F_1, F_2, and F_3 share similar shapes, which indicates that great similarity buried among these three SRPs. Therefore, useful traits found and transferred across these problems could enhance the corresponding optimization process accordingly. To quantify this, in Table III, we calculate the distances between the five problems based on the distance measure defined by (22). As can be observed, the distance between F_1, F_2, and F_3 are quite small, while F_4 and F_5 are relatively dissimilar to F_1, F_2, and F_3.

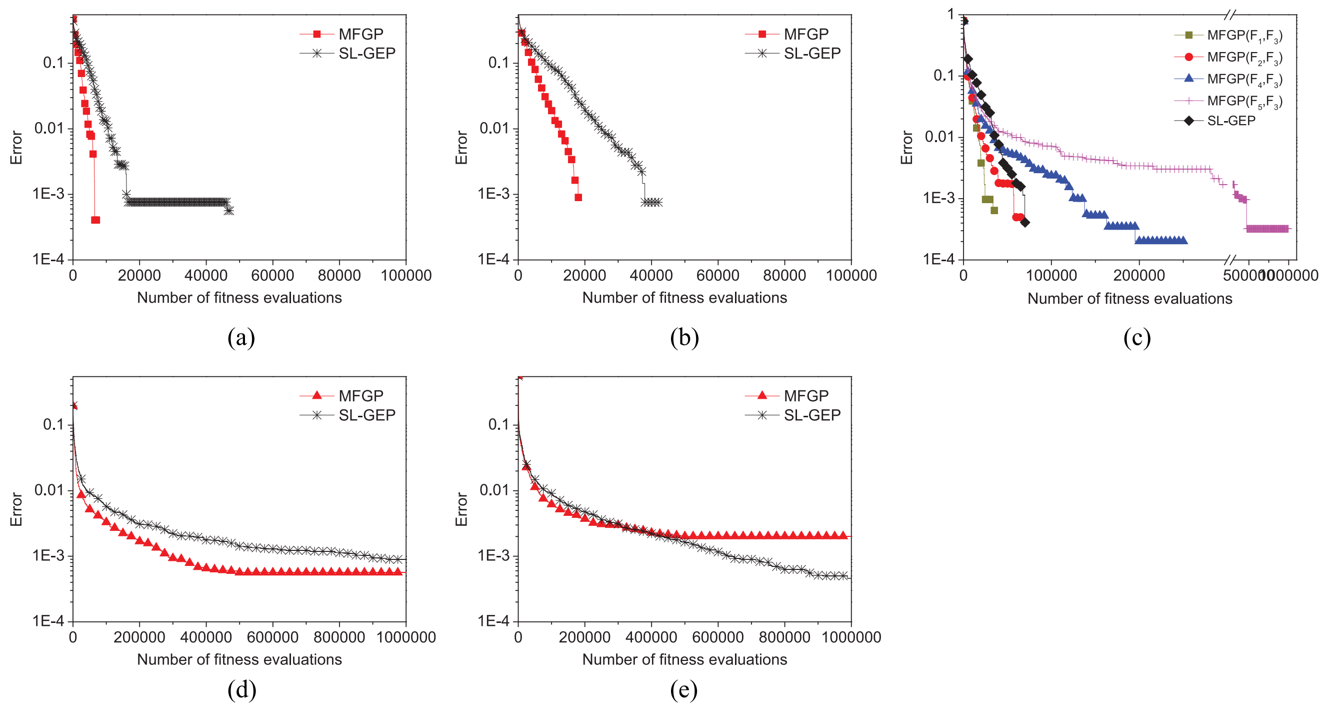


Fig. 9. Evolution curves of the average best fitting errors found by MFGP and SL-GEP on the first test set. (a) F_1, (b) F_2, (c) F_3, (d) F_4, and (e) F_5.

Further, Fig. 9 provides the evolution curves of the averaged best fitting errors found by SL-GEP and MFGP on four pairs of problems, i.e., $\{F_1, F_3\}$, $\{F_2, F_3\}$, $\{F_4, F_3\}$, $\{F_5, F_3\}$. It can be observed that the evolution curves are consistent with the results as discussed above. In particular, when solving $\{F_1, F_3\}$ and $\{F_2, F_3\}$, MFGP converges much faster over SL-GEP on F_1 and F_2, respectively. Meanwhile, for solving $\{F_5, F_3\}$, MFGP evolves slower than SL-GEP on F_5 since these two problems are dissimilar (see Table III). However, as natural selection in the evolutionary search can automatically neglect the detrimental solutions, it could reduce the effect of negative transfer when solving dissimilar problem pairs. Thus, enhanced search performance of MFGP has also been observed on F_4 over SL-GEP when solving the problem pair F_4, F_3.

Further, we study whether common building blocks exist in the final solutions obtained by MFGP on the studied problem pairs. Table IV lists three example solutions found by the MFGP when solving the problem pair of $\{F_1, F_2\}$. Each row of Table IV contains the best solutions of the two problems found by MFGP in a single run. As F_1 and F_2 share great similarity (see Table III), we can see that ADFs of F_2 can also be the ADFs of F_1. For example, for the first pair of solutions, the second ADF of the solution for F_2 [i.e., $G_1(t_1, t_2) = (t_1 + (t_1 * t_2))$] is exactly the same as the second ADF of the solution for F_1. Similar results can also be observed from the second and third pairs of solutions. These results demonstrate that the building blocks learned from one task is indeed useful for constructing solutions for other similar tasks. By using this feature, significantly improved search performance has been achieved by the proposed MFGP, as presented above.

TABLE IV
EXAMPLES OF THE BEST SOLUTIONS FOUND BY MFGP WHEN SOLVING $\{F_1, F_2\}$

Run		Solution
1st	F_1	$\Gamma = (x + G_1((x * G_0(x, x)), G_1(x, G_0(x, x))))$
		$G_0(t_1, t_2) = t_1$
		$G_1(t_1, t_2) = (t_1 + (t_1 * t_2))$
	F_2	$\Gamma = G_1(x, G_0(x, G_1(x, x)))$
		$G_0(t_1, t_2) = t_2$
		$G_1(t_1, t_2) = (t_1 + (t_1 * t_2))$
2nd	F_1	$\Gamma = ((x * G_1(x, G_1(x, x))) + x)$
		$G_0(t_1, t_2) = t_1$
		$G_1(t_1, t_2) = (t_1 + (t_2 * t_1))$
	F_2	$\Gamma = (G_0(x, (G_0(G_1(x, x), x) * G_1(x, x))) + x)$
		$G_0(t_1, t_2) = t_2$
		$G_1(t_1, t_2) = (t_1 + (t_2 * t_1))$
3rd	F_1	$\Gamma = G_1(G_1(G_0(G_0(G_1(x, x), x), x), G_0(x, G_1(x, x))), x)$
		$G_0(t_1, t_2) = t_1$
		$G_1(t_1, t_2) = (t_2 + (t_1 * t_2))$
	F_2	$\Gamma = G_1(G_0((G_0(x, x) + x), G_0(x, G_1(x, x))), x)$
		$G_0(t_1, t_2) = t_2$
		$G_1(t_1, t_2) = (t_2 + (t_1 * t_2))$

V. Experimental Studies on Real-World Problems

A. Experiment Settings

In this section, we further validate the performance of the proposed MFGP on real-world applications. In particular, we

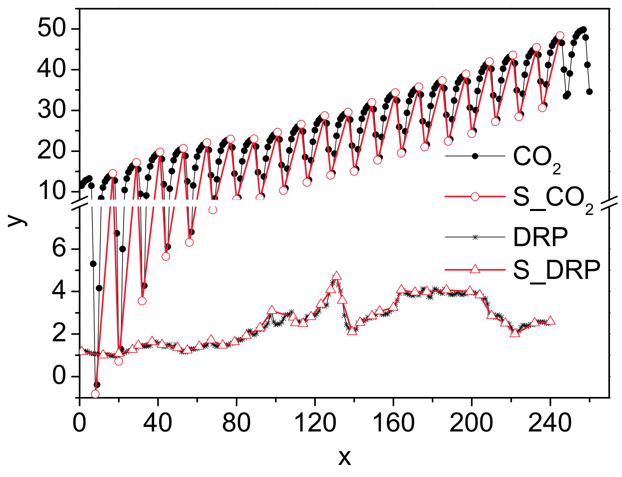


Fig. 10. Data sets of the four time series prediction problem.

apply the proposed method for solving two real-world time series prediction problems. The first problem contains 260 sample data points, which are the monthly average atmospheric CO_2 concentrations, derived from flask air samples, which are collected at Alert, Northwest Territiries, Canada from January 1986 to August 2007 [45].[2] To make the problem tractable, in this paper, the output is shifted by an offset of -340 for all data points to form the target y values. The x value represents the number of months starting from January 1986 (e.g., 1 represents January 1986, and 2 represents February 1986). The objective is to find a formula to model the relationships between x and y. The second problem contains 240 data points, which are the monthly U.S. No 2 Diesel Retail Prices (Cents per Gallon) from September 1997 to August 2017.[3] In this problem, the y values are the monthly Diesel Retail Prices, while the x value represents the number of months starting from September 1997. Further, we create two simplified time series for the two problems by selecting 40 important points from the original time series as training data. These selected points are important as they can capture the general shape of the curve of the original problem. Our objective is to investigate whether solving these simplified problems can help solve the original problems. To simplify the description, we mark the first two time series prediction problems as CO_2 and DRP, respectively. The corresponding simplified problems are labeled as S_CO_2 and S_DRP, respectively. The sample data of the four problems are plotted in Fig. 10.

Further, as the problems in this case study are more complex than the SRPs studies above, the head length and the maximum number of fitness evaluations are set as 20 and 2 000 000, respectively, for both SL-GEP and MFGP. Other parameter settings of SL-GEP and MFGP are kept the same as that in the first case study. The tenfold cross validation method and the performance metrics used in the first case study are also adopted in this paper. To investigate the effectiveness of the proposed method, the MFGP is applied to solve three pairs of problems, i.e., {CO_2, S_CO_2}, {CO_2, DRP}, and {DRP, S_DRP}.

[2]The data is available from http://cdiac.ornl.gov/ftp/trends/co2/altsio.co2.

[3]The data is available from https://www.eia.gov/dnav/pet/hist/LeafHandler.ashx?n=PET&s=EMD_EPD2D_PTE_NUS_DPG&f=W.

TABLE V
EXPERIMENT RESULTS OF MFGP AND SL-GEP ON CO_2 AND DRP

		CO_2	DRP
MFGP	paired problem	$RMSE$	$RMSE$
	CO_2	N/A	0.494-
	S_CO_2	**4.828**	N/A
	DRP	5.495-	N/A
	S_DRP	N/A	**0.478**
SL-GEP		5.504-	0.534-

$-$ means the result is significantly worse than the best result (marked in bold font) according to the Wilcoxon test at 95% confidence level.

TABLE VI
DISTANCES BETWEEN THE TIME SERIES PREDICTION PROBLEMS

Problem	CO_2	S_CO_2	DRP	S_DRP
CO_2	0	0.002	0.207	0.208
DRP	0.207	0.223	0	0.044

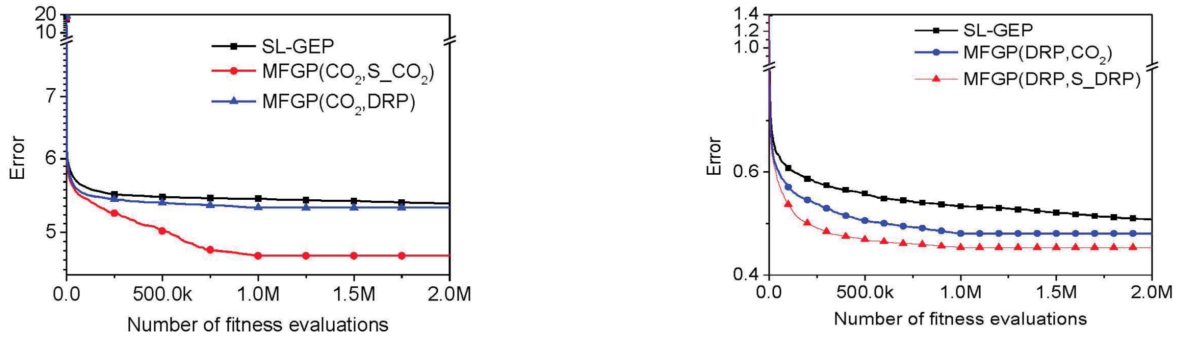
B. Results and Analysis

Table V lists the results obtained by SL-GEP and MFGP on the test problems. It can be observed that the performances of MFGP on both CO_2 and DRP are significantly improved when the corresponding simplified problems are paired. When a different problem is paired, due to the natural selection and diversity increased by knowledge transfer, the proposed MFGP also performed better than SL-GEP on both CO_2 and DRP in terms of RMSE. As MFGP and SL-GEP only differs in the proposed knowledge transfer across tasks, the observed superior performance again confirmed the efficacy of the proposed method.

Further, the evolution curves of the averaged best fitting errors are illustrated in Fig. 11. It can be observed that when the simplified problems are paired with the original problems, the best fitting errors of MFGP on the original problems converge much faster than those of SL-GEP. On the other problem pairs, better fitting errors have also been obtained by the proposed MFGP.

Moreover, we calculate the distance between the problems by the distance measure defined in (22), and the results are listed in Table VI. It can be observed that {CO_2, S_CO_2} and {DRP, S_DRP} are the problem pairs that have the closest distances. For these similar problem pairs, positive knowledge transferring is more likely to happen during the evolution process. Thus, the MFGP can achieve much superior performance on these problem pairs.

Lastly, Table VII lists four example solutions found by MFGP in two independent runs. The first run is to solve CO_2 and S_CO_2, while the second run is to solve DRP and S_DRP. It can be observed that the two solutions found by



(a) (b)

Fig. 11. Evolution curves of the average best fitting errors found by MFGP and SL-GEP on the third test case.

TABLE VII
EXAMPLES OF THE BEST SOLUTIONS FOUND BY MFGP WHEN SOLVING {OP, S-OP}

Run	Problem	Solution
1st	S_CO$_2$	$\Gamma = (x/ln(\lvert G_0((x - G_1(x, ln(\lvert x\rvert))), (x/ln(\lvert G_0($ $G_1((x-x),(x/x)), G_0((x/x),(x/x)))\rvert))))\rvert))$ $G_0(t_1,t_2) = (t_1/cos(t_2))$ $G_1(t_1,t_2) = (sin(t_2) + (t_2 * t_2))$
	CO$_2$	$\Gamma = (x/ln(\lvert G_0((G_1(x, G_0(ln(\lvert ln(\lvert x\rvert)\rvert), (x/x))) - x),$ $((x/G_1(G_0(x,x),(x/x))) - ln(\lvert x\rvert))))\rvert))$ $G_0(t_1,t_2) = (t_1/cos(t_2))$ $G_1(t_1,t_2) = (exp(t_2) + t_2)$
2nd	S_DRP	$\Gamma = (G_0((G_1((x+x),(x+x)) * (ln(\lvert x\rvert) + (x/x))),$ $ln(\lvert G_0(ln(\lvert x\rvert),(x*x))\rvert)) + G_1(G_1(G_1(x,x),x),$ $(G_1(x,x) * (x+x))))$ $G_0(t_1,t_2) = cos(sin((t_2 - t_1)))$ $G_1(t_1,t_2) = exp(cos(ln(\lvert t_2\rvert)))$
	DRP	$\Gamma = (G_0((G_1(G_1(x,x),(x+x)) * (ln(\lvert x\rvert) + (x/x))),$ $(G_0(ln(\lvert x\rvert), ln(\lvert x\rvert))/ln(\lvert x\rvert))) + G_1(ln(\lvert ln(\lvert x\rvert)\rvert),$ $(G_1(x,x) * (x+x))))$ $G_0(t_1,t_2) = cos(sin((t_2 + t_1)))$ $G_1(t_1,t_2) = exp(cos(ln(\lvert t_2\rvert)))$

MFGP in one single run are similar to each other. For example, in the first run, the first ADFs of the two solutions are exactly the same. The results in Table VII demonstrate that the building blocks learned from S_CO$_2$ (or S_DRP) are useful for constructing solutions of CO$_2$ (or DRP). Similar results can also be observed on other solution pairs found by MFGP. The above results confirmed that when multiple similar problems are solved concurrently, the proposed MFGP is capable of accelerating the search efficiency by utilizing knowledge learned across tasks.

VI. CONCLUSION

In this paper, we have proposed an MFGP toward evolutionary multitasking GP. In particular, we have presented a novel scalable gene expression chromosome representation, which allows to encode multiple solutions across domains in a unified representation. Based on this representation, new evolutionary mechanisms that consider both the implicit transfer of useful traits across tasks and the effective evolutionary search capability, have also been presented. To investigate the effectiveness of the proposed MFGP, comprehensive empirical studies conducted in multitask scenarios have been provided and analyzed. The obtained results confirmed the efficacy of the proposed MFGP.

Although the results of this paper have been encouraging, we would like to note that this paper is only a first step in the research direction of conducting multitasking GP. More investigation of both practical and theoretical aspects of the proposed method is needed in the further. For example, one future work would be to apply the proposed method to real world applications such as image classification problems and rule identification problems in agent-based simulation. Besides, as shown in the experimental studies, the distance between problems has a significant impact on the performance of the algorithm. Hence, developing a generic distance measure for problems in different domains is a very promising research direction. Once such a generic distance measure is available, adaptive controlling strategy could be proposed and integrated into the proposed framework to adaptively decide which problem set should be solved concurrently.

REFERENCES

[1] N. L. Cramer, "A representation for the adaptive generation of simple sequential programs," in *Proc. 1st Int. Conf. Genet. Algorithms*, 1985, pp. 183–187.

[2] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, vol. 1. Cambridge, MA, USA: MIT Press, 1992.

[3] M. O'Neill and C. Ryan, "Grammatical evolution," *IEEE Trans. Evol. Comput.*, vol. 5, no. 4, pp. 349–358, Aug. 2001.

[4] C. Ferreira, *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence* (Studies in Computational Intelligence). New York, NY, USA: Springer, 2006.

[5] J. F. Miller and P. Thomson, "Cartesian genetic programming," in *Proc. 3rd Eur. Conf. Genet. Program.*, vol. 1802, Apr. 2000, pp. 121–132.

[6] M. F. Brameier and W. Banzhaf, *Linear Genetic Programming*. New York, NY, USA: Springer, 2007.

[7] A. Moraglio, K. Krawiec, and C. G. Johnson, "Geometric semantic genetic programming," in *Proc. Int. Conf. Parallel Problem Solving Nat.*, Taormina, Italy, 2012, pp. 21–31.

[8] R. Ffrancon and M. Schoenauer, "Memetic semantic genetic programming," in *Proc. ACM Annu. Conf. Genet. Evol. Comput.*, Madrid, Spain, 2015, pp. 1023–1030.

[9] M. Castelli, L. Vanneschi, and S. Silva, "Semantic search-based genetic programming and the effect of intron deletion," *IEEE Trans. Cybern.*, vol. 44, no. 1, pp. 103–113, Jan. 2014.

[10] C. Zhou, W. Xiao, T. M. Tirpak, and P. C. Nelson, "Evolving accurate and compact classification rules with gene expression programming," *IEEE Trans. Evol. Comput.*, vol. 7, no. 6, pp. 519–531, Dec. 2003.

[11] M. Schmidt and H. Lipson, "Distilling free-form natural laws from experimental data," *Science*, vol. 324, no. 5923, pp. 81–85, 2009.

[12] P. G. Espejo, S. Ventura, and F. Herrera, "A survey on the application of genetic programming to classification," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 40, no. 2, pp. 121–144, Mar. 2010.

[13] N. R. Sabar, M. Ayob, G. Kendall, and R. Qu, "Automatic design of a hyper-heuristic framework with gene expression programming for combinatorial optimization problems," *IEEE Trans. Evol. Comput.*, vol. 19, no. 3, pp. 309–325, Jun. 2015.

[14] T. Weise and K. Tang, "Evolving distributed algorithms with genetic programming," *IEEE Trans. Evol. Comput.*, vol. 16, no. 2, pp. 242–265, Apr. 2012.

[15] N. R. Sabar, M. Ayob, G. Kendall, and R. Qu, "A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems," *IEEE Trans. Cybern.*, vol. 45, no. 2, pp. 217–228, Feb. 2015.

[16] K. Y. Chan, H. K. Lam, C. K. F. Yiu, and T. S. Dillon, "A flexible fuzzy regression method for addressing nonlinear uncertainty on aesthetic quality assessments," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 47, no. 8, pp. 2363–2377, Aug. 2017.

[17] A. Bailey, M. Ventresca, and B. Ombuki-Berman, "Genetic programming for the automatic inference of graph models for complex networks," *IEEE Trans. Evol. Comput.*, vol. 18, no. 3, pp. 405–419, Jun. 2014.

[18] J. Zhong, L. Luo, W. Cai, and M. Lees, "Automatic rule identification for agent-based crowd models through gene expression programming," in *Proc. Int. Conf. Auton. Agents Multi Agent Syst.*, Paris, France, 2014, pp. 1125–1132.

[19] J. Zhong, L. Feng, and Y.-S. Ong, "Gene expression programming: A survey [review article]," *IEEE Comput. Intell. Mag.*, vol. 12, no. 3, pp. 54–72, Aug. 2017.

[20] R. Meuth, M.-H. Lim, Y.-S. Ong, and D. C. Wunsch, II, "A proposition on memes and meta-memes in computing for higher-order learning," *Memetic Comput.*, vol. 1, no. 2, pp. 85–100, 2009.

[21] X. Chen, Y.-S. Ong, M.-H. Lim, and K. C. Tan, "A multi-facet survey on memetic computation," *IEEE Trans. Evol. Comput.*, vol. 15, no. 5, pp. 591–607, Oct. 2011.

[22] M. Iqbal, W. N. Browne, and M. Zhang, "Reusing building blocks of extracted knowledge to solve complex, large-scale Boolean problems," *IEEE Trans. Evol. Comput.*, vol. 18, no. 4, pp. 465–480, Aug. 2014.

[23] M. Iqbal, "Improving the scalability of XCS-based learning classifier systems," Ph.D. dissertation, Dept. Comput. Sci., Victoria Univ. Wellington, Wellington, New Zealand, 2014.

[24] R. Caruana, "Multitask learning," *Mach. Learn.*, vol. 28, no. 1, pp. 41–75, 1997.

[25] L. Wen, L. Gao, and X. Li, "A new deep transfer learning based on sparse auto-encoder for fault diagnosis," *IEEE Trans. Syst., Man, Cybern., Syst.*, to be published, doi: 10.1109/TSMC.2017.2754287.

[26] S. Li, S. Song, G. Huang, and C. Wu, "Cross-domain extreme learning machines for domain adaptation," *IEEE Trans. Syst., Man, Cybern., Syst.*, to be published, doi: 10.1109/TSMC.2017.2735997.

[27] T. Evgeniou and M. Pontil, "Regularized multi–task learning," in *Proc. 10th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min.*, Seattle, WA, USA, 2004, pp. 109–117.

[28] O. Chapelle *et al.*, "Multi-task learning for boosting with application to Web search ranking," in *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min.*, Washington, DC, USA, 2010, pp. 1189–1198.

[29] G. Wang, G. Zhang, K.-S. Choi, and J. Lu, "Deep additive least squares support vector machines for classification with model transfer," *IEEE Trans. Syst., Man, Cybern., Syst.*, to be published, doi: 10.1109/TSMC.2017.2759090.

[30] Y. Luo, D. Tao, B. Geng, C. Xu, and S. J. Maybank, "Manifold regularized multitask learning for semi-supervised multilabel image classification," *IEEE Trans. Image Process.*, vol. 22, no. 2, pp. 523–536, Feb. 2013.

[31] A. Evgeniou and M. Pontil, "Multi-task feature learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 19, 2007, pp. 41–48.

[32] K. Krawiec and B. Wieloch, "Automatic generation and exploitation of related problems in genetic programming," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Barcelona, Spain, 2010, pp. 1–8.

[33] A. Gupta, Y.-S. Ong, and L. Feng, "Multifactorial evolution: Toward evolutionary multitasking," *IEEE Trans. Evol. Comput.*, vol. 20, no. 3, pp. 343–357, Jun. 2016.

[34] J. Zhong, Y.-S. Ong, and W. Cai, "Self-learning gene expression programming," *IEEE Trans. Evol. Comput.*, vol. 20, no. 1, pp. 65–80, Feb. 2016.

[35] A. Gupta, Y.-S. Ong, L. Feng, and K. C. Tan, "Multiobjective multifactorial optimization in evolutionary multitasking," *IEEE Trans. Cybern.*, vol. 47, no. 7, pp. 1652–1665, Jul. 2017.

[36] L. Zhou *et al.*, "Evolutionary multitasking in combinatorial search spaces: A case study in capacitated vehicle routing problem," in *Proc. IEEE Symp. Series Comput. Intell. (SSCI)*, Athens, Greece, 2016, pp. 1–8.

[37] K. K. Bali, A. Gupta, L. Feng, Y. S. Ong, and T. P. Siew, "Linearized domain adaptation in evolutionary multitasking," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2017, pp. 1295–1302.

[38] R.-T. Liaw and C.-K. Ting, "Evolutionary many-tasking based on biocoenosis through symbiosis: A framework and benchmark problems," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2017, pp. 2266–2273.

[39] Y.-W. Wen and C.-K. Ting, "Parting ways and reallocating resources in evolutionary multitasking," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2017, pp. 2404–2411.

[40] L. Feng *et al.*, "An empirical study of multifactorial PSO and multifactorial DE," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2017, pp. 921–928.

[41] Z. Tang, M. Gong, and M. Zhang, "Evolutionary multi-task learning for modular extremal learning machine," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2017, pp. 474–479.

[42] C. Ferreira, "Gene expression programming: A new adaptive algorithm for solving problems," *Complex Syst.*, vol. 13, no. 2, pp. 87–129, 2001.

[43] J. McDermott *et al.*, "Genetic programming needs better benchmarks," in *Proc. 14th Annu. Conf. Genet. Evol. Comput.*, Philadelphia, PA, USA, 2012, pp. 791–798.

[44] F.-L. Chung, T.-C. Fu, V. Ng, and R. W. P. Luk, "An evolutionary approach to pattern-based time series segmentation," *IEEE Trans. Evol. Comput.*, vol. 8, no. 5, pp. 471–489, Oct. 2004.

[45] C. K. Williams and C. E. Rasmussen, *Gaussian Processes for Machine Learning*. Cambridge, MA, USA: MIT Press, vol. 2, 2006, p. 4.

Jinghui Zhong received the Ph.D. degree in computer applied technology from the School of Information Science and Technology, Sun Yat-sen University, Guangzhou, China, in 2012.

He is currently an Associate Professor with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China. From 2013 to 2016, he was a Post-Doctoral Research Fellow with the School of Computer Engineering, Nanyang Technological University, Singapore. His current research interests include evolutionary computation such as genetic programming, differential evolution, and the applications of evolutionary computation.

Liang Feng received the Ph.D. degree in computational intelligence from the School of Computer Engineering, Nanyang Technological University, Singapore, in 2014.

He was a Post-Doctoral Research Fellow with the Computational Intelligence Graduate Laboratory, Nanyang Technological University, Singapore. He is currently an Assistant Professor with the College of Computer Science, Chongqing University, Chongqing, China. His current research interests include computational and artificial intelligence, memetic computing, big data optimization and learning, and transfer learning.

Wentong Cai received the Ph.D. degree in computer science from the University of Exeter, Exeter, U.K., in 1991.

He is a Professor with the School of Computer Engineering, Nanyang Technological University, Singapore. He is also the Director of the Parallel and Distributed Computing Centre. His expertise is mainly in the areas of modeling and simulation (particularly, modeling and simulation of large-scale complex systems, and system support for distributed simulation and virtual environments) and parallel and distributed computing (particularly, cloud, grid, and cluster computing).

Dr. Cai is an Associate Editor of the *ACM Transactions on Modeling and Computer Simulation* and an Editor of the *Future Generation Computer Systems*.

Yew-Soon Ong received the Ph.D. degree on artificial intelligence in complex design from the Computational Engineering and Design Center, University of Southampton, Southampton, U.K., in 2003.

He is a Professor and the Chair of the School of Computer Science and Engineering, Nanyang Technological University, Singapore, where he is the Director of the Data Science and Artificial Intelligence Research Center, the Director of the A*STAR SIMTECH-NTU Joint Laboratory on Complex Systems, and a Principal Investigator of the Data Analytics and Complex System Programme in the Rolls-Royce@NTU Corporate Lab. His current research interests include computational intelligence span across memetic computation, complex design optimization, intelligent agents, and big data analytics.

Dr. Ong was a recipient of the 2015 *IEEE Computational Intelligence Magazine* Outstanding Paper Award and the 2012 IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION Outstanding Paper Award for his work pertaining to Memetic Computation. He is the Founding Editor-in-Chief of the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTATIONAL INTELLIGENCE, and an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, IEEE TRANSACTIONS ON NEURAL NETWORK AND LEARNING SYSTEMS, and IEEE TRANSACTIONS ON CYBERNETICS.