Multitask Linear Genetic Programming with Shared Individuals and its Application to Dynamic Job Shop Scheduling

Zhixing Huang , *Graduate Student Member, IEEE,* Yi Mei , *Senior Member, IEEE,* Fangfang Zhang , *Member, IEEE,* Mengjie Zhang , *Fellow, IEEE*

***Abstract*—Multitask genetic programming methods have been applied to various domains, such as classification, regression, and combinatorial optimization problems. Most existing multitask genetic programming methods are designed based on tree-based structures, which are not good at reusing building blocks since each sub-tree passes its outputs to only one parent. It may limit the design and performance of knowledge sharing in multitask optimization. Different from tree-based genetic programming, building blocks in linear genetic programming can be easily reused by more than one parent. Besides, existing multitask genetic programming methods always allocate each individual to a specific task and have to duplicate genetic materials from task to task in knowledge transfer, which is inefficient and often produces redundancy. Contrarily, it is natural for a linear genetic programming individual to produce multiple distinct outputs, which enables each linear genetic programming individual to solve multiple tasks simultaneously. With this in mind, we propose a new multitask linear genetic programming method that transfers knowledge via multi-output individuals (i.e., shared individuals among tasks). By integrating different solutions into one multi-output individual, the proposed method efficiently reuses common knowledge among tasks and maintains distinct behaviors for each task. The empirical results show that the proposed method has a significantly better test performance than state-of-the-art multitask genetic programming methods. Further analyses verify that the new knowledge transfer mechanism can adjust the transfer rate automatically and thus improves its effectiveness.**

***Index Terms*—Multitask optimization, Linear genetic programming, Directed acyclic graph, Dynamic job shop scheduling.**

I. Introduction

Evolutionary multitask optimization is an emerging research area in the last decade [1], [2]. In the light of the outstanding association ability of human brains, evolutionary multitask optimization techniques aim to design evolutionary computation methods that fully exploit the latent synergies among tasks. In contrast to solving every single task independently from scratch, evolutionary multitask optimization techniques solve similar tasks simultaneously and exchange useful information in the course of evolution. Genetic materials such as elite individuals and building blocks are shared among tasks, to enhance convergence speed and search effectiveness. Nowadays, evolutionary multitask techniques have shown remarkable performance on both continuous and discrete optimization problems [3].

Genetic programming (GP) [4] is a popular evolutionary computation method. Its individuals can represent mathematical formulas or computer programs based on a given primitive set. Part of the existing literature has applied multitask optimization to enhance tree-based GP in solving machine learning problems and combinatorial optimization and has shown very impressive results [5]–[7]. However, conventional tree-based GP is not good at reusing building blocks, since each node in the tree has at most one parent node. For reusing a sub-tree (building block), tree-based GP has to duplicate that sub-tree or design multiple outputs by complicated tricks [8], which is inefficient in both space and computation and reduces the diversity of genetic materials in the population.

On the other hand, linear genetic programming (LGP) [9]–[11], which is a kind of graph-based GP [12], often has a more compact and flexible representation than tree-based structures. By decoding LGP individuals into directed acyclic graphs, each node (i.e., primitive in LGP individuals) can have multiple parent nodes, and each graph can have multiple outputs. These graph characteristics allow the building blocks of LGP (i.e., sub-graphs) to pass their outputs to multiple graph nodes in the calculation and enable LGP to represent multiple solutions within a single individual naturally.

However, extending LGP to existing multitask GP methods cannot fully utilize the graph characteristic. The existing multitask GP methods simply see each individual as a solution/heuristic for a specific task and transfer knowledge by duplicating genetic materials (e.g., instruction segments in LGP individuals). A primary investigation on multitask LGP verified that directly applying existing multitask GP methods to LGP has similar performance with existing multitask tree-based GP methods [13]. Multitask LGP methods have not yet

Manuscript received XXX; revised XXX; accepted XXX. This work is supported in part by the Marsden Fund of New Zealand Government under Contract MFP-VUW1913, and by the MBIE SSIF Fund under Contract VUW RTVU1914. The work of Zhixing Huang was supported by the China Scholarship Council (CSC)/Victoria University Scholarship. (Corresponding author: Fangfang Zhang.)

The authors are with the Evolutionary Computation and Machine Learning Research Group at the School of Engineering and Computer Science, Victoria University of Wellington, New Zealand (E-mail: zhixing.huang@ecs.vuw.ac.nz; yi.mei@ecs.vuw.ac.nz; fangfang.zhang@ecs.vuw.ac.nz; mengjie.zhang@ecs.vuw.ac.nz).

This article has supplementary downloadable material available at XXX, provided by the authors.

Colour versions of one or more of the figures in this article are available online at XXX.

Digital Object Identifier XXX

been well investigated.

In this paper, we aim to propose a new multitask framework based on the graph characteristic of LGP, named **M**ultitask **L**GP with **S**hared **I**ndividuals (MLSI). MLSI evolves a sub-population of multi-output individuals (i.e., shared individuals). Each shared individual simultaneously encodes more than one solution, each for a specific task and with a specific output, within one directed acyclic graph. These solutions share common building blocks to perform knowledge transfer. The shared individuals then participate in the evolution of all the tasks by shifting their graph outputs, in which way they behave like task-specific individuals, but intrinsically carry common building blocks from the other tasks.

There are four main contributions in this paper:

1) A new knowledge transfer strategy is proposed based on LGP. The proposed strategy fully utilizes the topological structures in LGP by encoding the solutions from different tasks into a directed acyclic graph. Due to the high reusability and flexibility of graph-based structures, the proposed strategy is expected to evolve compact solutions for multiple tasks, and further improve the efficiency of knowledge transfer. It reveals the great potential of graph-based structures in multitask GP methods. To the best of our knowledge, this is the first multitask GP method that designs knowledge transfer mechanisms based on graph-based structures.
2) New genetic operators are designed based on the new knowledge transfer strategy. Specifically, a riffle shuffle operator is proposed to integrate elite individuals into shared individuals. A better-parent reservation strategy is developed to further enhance the effectiveness of the crossover in multitask LGP. Given that shared individuals are evaluated on multiple tasks simultaneously and have more than one fitness, tournament selection is updated accordingly to select elite individuals for specific tasks.
3) Comprehensive experiments based on dynamic scheduling are conducted. The experiments cover six multitask scenarios, in which tasks have different levels of similarity. The results show that the proposed method significantly outperforms three state-of-the-art multitask GP methods in terms of both training efficiency and test effectiveness.
4) Further analyses verify that the superior performance of the newly proposed method stems from the intrinsical adaptation ability on knowledge transfer rate. It implies that adjusting the knowledge transfer rate based on the evolution process and the similarity among tasks is a useful strategy in improving search effectiveness. The results also verify the necessity of the key components in the proposed method and show a different pattern of sharing common building blocks in GP individuals, which promotes future study on graph-based GP.

The rest of this paper is organized as follows. Section II introduces multitask optimization and the existing literature on linear genetic programming. Section III presents the details of the proposed method. Experiment designs including problem formulation and comparison design are illustrated in Section IV. The results and further analyses are shown in Section V and VI respectively. Finally, Section VII draws the conclusions.

II. Literature review

A. Evolutionary Multitask Optimization

Existing evolutionary multitask methods can be categorized into two paradigms, implicit and explicit genetic transfer [14], [15]. Methods with implicit genetic transfer exchange knowledge among tasks by applying a suite of genetic operators to perform implicit genetic mating (e.g., applying crossover operators on two parents from different tasks). One of the most typical methods with the implicit genetic transfer is the multifactorial evolutionary algorithm (MFEA) [3]. MFEA designed four new concepts: factorial cost, factorial rank, skill factor, and scalar fitness. The first two terms are vectors for multiple tasks, denoting the fitness and rank in corresponding tasks. The scalar fitness is the minimum factorial rank among tasks. The task with the minimum factorial rank is recorded by skill factor. An assortative mating and a vertical cultural transmission are also proposed in [3] to facilitate knowledge transfer. MFEA integrates multiple solutions into one individual based on the skill factor. However, most MFEA methods are designed based on numerical representation. The idea of integrating multiple numerical solutions in MFEA cannot be easily extended to GP methods whose search space is symbolic. Based on MFEA, many studies developed new techniques to enhance its performance. For example, Bali et al. [16] proposed a linear transformation strategy to transform the decision space among tasks. Ding et al. [17] developed a decision variable translation strategy and decision variable shuffling strategy for MFEA to improve the effectiveness of knowledge sharing. Zheng et al. [18] proposed a self-regulated method to perform knowledge sharing based on the relatedness among tasks. Besides, MFEA has been applied to many applications, such as capacitated vehicle routing problems [19], robot path planning problem [20], and bi-level optimization [21]. Gupta et al. [22] also extended MFEA to multi-objective optimization.

However, the implicit genetic transfer has a key limitation. It unnecessarily limits the information exchange within genetic mating [14]. In practice, genetic mating might not be effective enough to transfer knowledge among different tasks. To address this issue, the explicit genetic transfer methods are proposed to explicitly consider different representations and search mechanisms among tasks in knowledge transfer. For example, Feng et al. [14], [23] proposed to use an artificial neural network (e.g., denoising autoencoder) to perform knowledge transfer among different tasks. The artificial neural network is trained beforehand on uniformly sampled data. When performing knowledge transfer, solutions from one task are mapped to another space by the artificial neural network. This method demonstrates the superior performance of explicit genetic transfer over implicit genetic transfer in both single- and multi-objective optimization. To capture the essential features of different tasks, Tang et al. [24] transformed the distribution of sub-populations into task-specific

low-dimension spaces, and made pairwise mapping among tasks by well-trained alignment matrices. When the mapping discrepancies among tasks are minimized, individuals from different tasks can be transferred to another task based on low-dimension spaces and alignment matrices. Chen et al. [25] treated decision spaces of different tasks as manifolds and projected the decision spaces of tasks to a joint manifold to represent the task relationship. The knowledge transfer among tasks is performed based on the latent task relationship. Kullback-Leibler divergence [26] and Naive Bayes classifier [27] are also extended as methods of selective knowledge transfer. Nowadays, multitask optimization techniques with explicit genetic transfer have been applied to some real-world applications, such as time series prediction [28] and capacitated vehicle routing problem [29].

B. Linear Genetic Programming

LGP is a representative graph-based GP. It was firstly designed to evolve programs written in machine codes in the 1990s [9], [10]. Each LGP individual is a sequence of register-based instructions. All instructions manipulate the same set of registers to pass the intermediate results, and each instruction contains a function to represent a specific operation. The instruction sequence can be converted into a directed acyclic graph (DAG).

Fully utilizing the graph-based structure of LGP provides a different perspective from tree-based structures in designing GP methods. For example, Fogelberg [30] and Downey [31], [32] respectively applied LGP to multi-class classification and showed very promising results. Since each LGP individual has multiple outputs easily, and these outputs share common building blocks, it is quite efficient for LGP to simultaneously evolve multiple sub-classifiers. Based on the graph-based structure of LGP, Kantschik et al. [33] proposed a linear-tree GP for solving classification and symbolic regression problems. Sotto et al. [34], [35] took LGP as an example and further compared the effectiveness among different graph-based GP methods. They found that the graph-based structure of LGP which has limited graph width defined by the registers is suitable for solving even parity circuits. Since the directed acyclic graph neglects the non-effective instructions in LGP, an investigation on the relationship between the bloat effect and the impact of non-effective instructions in LGP was conducted by Sotto et al [36], [37] to better understand the genotype-phenotype mapping.

Nowadays, LGP has been applied to different applications and has undergone various developments. For example, Brameier and Banzhaf [38] applied LGP to the classification of medical data and proposed several effective genetic operators to enhance LGP performance [39], [40]. In [41], Sotto et al. enhanced LGP by a stochastic context-free grammar, and achieved significantly better results than conventional LGP in some symbolic regression problems. LGP can also be applied as a hyper-heuristic method to solve combinatorial optimization problems and showed very promising results [13], [42]. Besides, because of the concise chromosome representation, LGP is seen as a representative example to analyze the evolvability and robustness of GP methods [43], [44].

C. Multitask GP

Multitask techniques are also applied to GP methods to enhance their performance in different domains. For example, Zhong et al. [5] and Wei et al. [45] applied multitask techniques to gene expression programming for solving symbolic regression and multi-class classification. Bi et al. [6] proposed to use multitask techniques to encourage GP to construct effective features for image classification. These GP-constructed features from different tasks are concatenated together based on an ensemble method. Bi et al. [46] proposed to use a common tree structure to construct shared features in similar classification tasks. Multitask GP is applied to job shop scheduling problems. In [7], [47], Zhang et al. proposed a multi-population based multitask GP method and verified the effectiveness of the explicit genetic transfer. To further improve the training convergency, surrogate models are also introduced in multitask framework to selectively share knowledge [48].

Based on the review, we found that existing multitask GP methods are mostly designed based on tree-based GP. Knowledge is transferred mainly by duplicating elite individuals or sub-trees from one task to another, which is inefficient. Further, since there is usually one output in tree-based structures (i.e., the root), it is uneasy for the existing tree-based multitask GP methods to switch among the representation of solutions for different tasks, which is inflexible in utilizing common building blocks. An investigation is conducted on LGP with the existing multitask frameworks [13]. The results showed that the existing multitask frameworks fail to utilize the graph-based characteristic of LGP to evolve compact solutions since they treat LGP solutions separately. To further improve the effectiveness of knowledge sharing and encourage compact representations, an improved LGP-based multitask method is needed.

III. Proposed Method

This section demonstrates the proposed MLSI in detail. The chromosome representation and evolutionary framework of MLSI are firstly described, followed by the selection method. The key new genetic operators are finally introduced.

A. Program Representation

An LGP individual $\mathbf{f}$ is a sequence of instructions ($\mathbf{f} = [f_0, f_1, ..., f_{l-1}]$), where l is the total number of instructions. To limit the search space of LGP, l has to be in a predefined range $[l_{min}, l_{max}]$. Fig. 1 shows an example of an LGP individual in solving multitask optimization. Every instruction f consists of three parts: a destination register $R_{f,d}$ (on the left of the equal mark), a function $fun_f(\cdot)$ (the operation on the right of the equal mark), and two source registers $R_{f,s}$ (the registers on the right of the equal mark). The instruction takes the values from the source registers, calculates the output value by the function, and passes the output value to the destination register. Both destination and source registers are selected from the same register set $\mathbb{R} = \{R_0, ..., R_{|\mathbb{R}|-1}\}$ where $|\cdot|$ denotes the cardinality of a set or a list, and functions are selected from a predefined function set. In addition, constants such

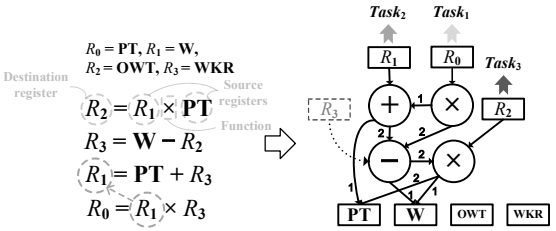


Fig. 1. An example of LGP individual with three outputs (i.e., R_0, R_1, and R_2). PT: processing time of each operation, W: weight of job, OWT: waiting time of an operation, WKR: remaining processing time of a job.

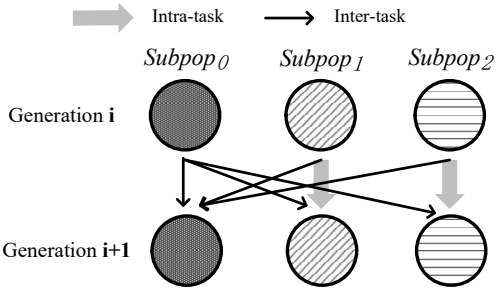


Fig. 2. The intra- and inter-flow of genetic materials among tasks.

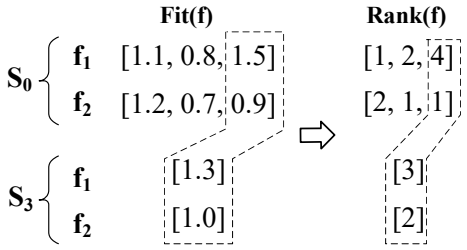


Fig. 3. An example of determining the rank of the third task (i.e., $Rank_3(\mathbf{f})$) for $\mathbb{S}_0$ and $\mathbb{S}_3$, each sub-population with two individuals.

as input features are treated as a special type of read-only registers. They can act as source registers, but cannot be used as destination registers.

The instructions in the LGP program are executed one by one from the top of the program to the bottom. Before execution, the registers are initialized by the designated problem features. In Fig. 1, R_0 to R_3 are respectively initialized by the input features (i.e., processing time of a job (PT), weight of a job (W), waiting time of an operation (OWT), and the remaining processing time of a job (WKR)). To produce the results for different tasks, multiple output registers are defined for LGP. For the sake of simplicity, we designate the first k registers as the output registers for the k tasks (e.g., R_0 to R_2 in Fig. 1). We can see that some of the registers are shared among different tasks (i.e., different output registers), such as R_3 in the second instruction.

Fig. 1 also gives an example of converting LGP instructions into a DAG. First, each function or constant in the instructions is instantiated as a graph node. Duplicated constants (e.g., **PT**) are merged into one graph node. Then, we connect the functions based on the registers (e.g., connecting function $\times$ to $+$ since $\times$ in the last instruction reads the result in the register R_1 in the third instruction written by $+$) and connect the functions to specific constants (e.g., connecting $+$ to **PT**). Following the existing style [11], the direction of arrows denotes accepting inputs from another graph node and the number beside the arrow indexes the first or second argument of the function. The output of the DAG is stored in output registers R_0, R_1, and R_2. When an LGP individual is shared by multiple tasks, it is evaluated on these tasks respectively and obtains a list of fitness values, each for a specific task.

B. Algorithm Framework

Different from existing multi-population multitask evolutionary frameworks in which each sub-population solves a specific task, the first sub-population $\mathbb{S}_0$ of MLSI is a generalist sub-population that aims to solve all the tasks, while the remaining sub-populations $\mathbb{S}_i(i = 1, ..., k)$ are specialist sub-populations that aim to solve a single task respectively. Fig. 2 shows the flow of genetic materials (e.g., building blocks) among tasks. The knowledge of different tasks is shared via $\mathbb{S}_0$. The specialist sub-populations give genetic material to the generalist one and accept knowledge from the generalist sub-population for every generation. Since the individuals in the generalist sub-population $\mathbb{S}_0$ are shared on all tasks, the evolution of the generalist sub-population is seen as another kind of sharing knowledge. By this means, the common building blocks across tasks can be carried in a compact representation in $\mathbb{S}_0$ and flexibly switch representations based on different output registers (i.e., cooperate with task-specific sub-graphs in the individuals).

Each individual $\mathbf{f} \in \mathbb{S}_0$ is evaluated on all the k tasks, and thus has a vector of fitnesses, denoted as $\mathbf{Fit}(\mathbf{f}) = [Fit_1(\mathbf{f}), ..., Fit_k(\mathbf{f})]$. Contrarily, the individuals in $\mathbb{S}_i(i > 0)$ only have a single fitness value for its corresponding task (i.e., $(\mathbf{Fit}(\mathbf{f}) = [Fit_i(\mathbf{f})], i \in \{1, ..., k\})$. Based on $\mathbf{Fit}(\mathbf{f})$, the rank of an individual $\mathbf{Rank}(\mathbf{f})$ is designed accordingly. For each generalist individual $\mathbf{f} \in \mathbb{S}_0$, the rank of $\mathbf{f}$ is denoted as $\mathbf{Rank}(\mathbf{f}) = [Rank_1(\mathbf{f}), ..., Rank_k(\mathbf{f})]$. For each specialist individual $\mathbf{f} \in \mathbb{S}_i(i > 0)$, $\mathbf{Rank}(\mathbf{f}) = [Rank_i(\mathbf{f})], i \in \{1, ..., k\}$. To determine the rank of $\mathbf{f}$, i.e., $\mathbf{Rank}(\mathbf{f})$ for solving task t, all the individuals $\mathbf{f} \in \mathbb{S}_0 \bigcup \mathbb{S}_i(t > 0, t = i)$ are combined and sorted together. The example of determining $\mathbf{Rank}(\mathbf{f})$ is shown in Fig. 3. There are two individuals in each sub-population. When we are identifying the rank of the third task (i.e., $Rank_3(\mathbf{f})$), those individuals which are evaluated on the third task (i.e., $\mathbf{f} \in \mathbb{S}_0 \bigcup \mathbb{S}_3$) are sorted together based on the corresponding fitness, i.e., $Fit_3(\mathbf{f})$. Smaller fitness has a better rank.

The pseudo-code of MLSI is shown in Alg. 1 [1] where the underlines "$\sim$" highlight the major differences from existing multitask GP methods. To solve k similar tasks simultaneously, MLSI first initializes $k + 1$ sub-populations. At each generation, the individuals in $\mathbf{S}_0$ are evaluated on all the k tasks, and the individuals in $\mathbf{S}_i(i > 0)$ are evaluated on task i respectively. Then, $\mathbf{Fit}(\mathbf{f})$ and $\mathbf{Rank}(\mathbf{f})$ of the all individuals

[1] `rand`(a, b) and `randint`(a, b) return a random floating point number or a random integer between $[a, b)$ respectively. The notations are used in the rest of the paper.

are updated based on the new fitness. Elitism selection is applied to retain competent individuals.

New offspring are produced by five genetic operators, including macro mutation, micro mutation, crossover, reproduction, and *riffle shuffle* based on their rates. The riffle shuffle is the newly proposed genetic operator in this paper to vary the individuals with multiple outputs, while the other four genetic operators are off-the-shelf operators [11], [42]. Specifically, *effective macro mutation*, *effective micro mutation*, and *linear crossover* in [11] act as mutation and crossover respectively. Linear crossover is enhanced by a newly proposed better-parent reservation strategy (i.e., `BetterParentRes`$(\cdot)$). Given that each individual in $\mathbf{S}_0$ has a list of fitnesses, a new tournament selection `TournamentSpecific`$(\cdot)$ is developed to select parents based on a specific task. More specifically, $\mathbf{S}_0$ applies reproduction, linear crossover, and the riffle shuffle (i.e., `RiffleShuffle`$(\cdot)$) to produce offspring, while $\mathbf{S}_i(i > 0)$ apply reproduction, linear crossover, and mutation to produce offspring. To share knowledge among tasks, the parents of the linear crossover and riffle shuffle are selected from a merged sub-population, formed by $\mathbb{S}_0$ and specialist sub-populations. As suggested by [11], all macro operators (i.e., crossover, riffle shuffle, and macro mutation) are followed by a micro mutation. All the produced offspring replace the parents without further comparison and form the population of the next generation. This evolutionary process is iterated for every generation until the stopping criteria are met. The best individuals of the tasks in all the sub-populations are outputted as the final results.

C. Selection

The parent selection is implemented by the `TournamentSpecific`$(\cdot)$ method. This method is extended from the standard tournament selection by enabling cross-sub-population selection for a specific task. The pseudo-code of `TournamentSpecific`$(\cdot)$ is shown in Alg. 2, in which $\mathbf{Fit}(\mathbf{f}, t)$ returns the t^{th} element of $\mathbf{Fit}(\mathbf{f})$ if $|\mathbf{Fit}(\mathbf{f})| > 1$, and the only element in $\mathbf{Fit}(\mathbf{f})$ otherwise[2].

Note that in the proposed multitask framework, the parents of linear crossover and riffle shuffle for a certain task t are selected from the generalist sub-population (i.e., $\mathbb{S}_0$) and specialist sub-population (i.e., $\mathbb{S}_t$) simultaneously. Specifically, $\mathbb{S}_0$ and $\mathbb{S}_t$ are merged into one sub-population, and the two parents for one mating are selected from the merged sub-population based on the paradigm of tournament selection. Therefore, the parents for linear crossover and riffle shuffle can both come from $\mathbb{S}_0$, or both come from $\mathbb{S}_t$, or come from either $\mathbb{S}_0$ or $\mathbb{S}_t$. The knowledge sharing among different tasks for the linear crossover and riffle shuffle is achieved by selecting the parents from $\mathbb{S}_0$ whose individuals are shared by all the tasks. The parents for task t are selected based on their fitness on that task. When the individuals in $\mathbb{S}_0$ have better fitness on task t than those in $\mathbb{S}_t$, they have a higher probability to be selected by `TournamentSpecific`$(\cdot)$, and vice versa. Such design helps MLSI determine the suitable timing and frequency of knowledge transfer across tasks (i.e., transfer knowledge when the individuals with common building blocks have better performance).

[2]same as $\mathbf{Rank}(\mathbf{f}, t)$ in Alg. 4.

Algorithm 1: Framework of MLSI

Input: k tasks, macro mutation rate θ_{ma}, micro mutation rate θ_{mi}, crossover rate θ_c, reproduction rate θ_r, step size of `RiffleShuffle` η, tournament selection size s.
Output: k best heuristics $h_t(t = 1, ..., k)$, each for a specific task.

Initialize $k+1$ sub-populations $\mathbb{S}_0$ to $\mathbb{S}_k$.
while *stopping criteria are not satisfied* **do**
  `// Evaluation`
  Evaluate $\mathbf{f} \in \mathbb{S}_0$ on all the tasks. Evaluate $\mathbf{f} \in \mathbb{S}_i(i = 1, .., k)$ on task i respectively;
  Update $\mathbf{Fit}(\mathbf{f})$ and $\mathbf{Rank}(\mathbf{f})$ for $\mathbf{f} \in \{\mathbb{S}_0, ..., \mathbb{S}_k\}$;
  `// Breeding`
  foreach $\mathbb{S}_i(i = 0, ..., k)$ **do**
    $\mathbb{S}'_i \leftarrow \emptyset$;
    Clone elite individuals of $\mathbb{S}_i$ into $\mathbb{S}'_i$;
    while $|\mathbb{S}'_i| < |\mathbb{S}_i|$ **do**
      $rnd \leftarrow$ `rand`$(0, 1)$;
      if $rnd < \theta_r$ **then**
        $\mathbf{c} \leftarrow$ `TournamentSpecific`$(\mathbb{S}_i, i, s)$;
      else if $rnd < \theta_r + \theta_c$ **then**
        $\mathbf{c} \leftarrow$ LGP crossover with `BetterParentRes`$(\{\mathbb{S}_0, ..., \mathbb{S}_k\}, i, s)$;
      else
        `// generalist sub-population`
        if $i = 0$ **then**
          $\mathbf{c} \leftarrow$ `RiffleShuffle`$(\{\mathbb{S}_0, ..., \mathbb{S}_k\}, \eta, s)$;
        `// specialist sub-population`
        else
          $\mathbf{p} \leftarrow$ `TournamentSpecific`$(\mathbb{S}_i, i, s)$;
          if $rnd - (\theta_c + \theta_r) < \theta_{ma}$ **then**
            Apply LGP macro mutation on $\mathbf{p}$ to produce offspring $\mathbf{c}$;
          else
            Apply LGP micro mutation on $\mathbf{p}$ to produce offspring $\mathbf{c}$;
      if $\mathbf{c}$ *is produced by macro operators* **then**
        Apply LGP micro mutation on $\mathbf{c}$ to update it;
      $\mathbb{S}'_i \leftarrow \mathbb{S}'_i \bigcup \{\mathbf{c}\}$;
    $\mathbb{S}_i \leftarrow \mathbb{S}'_i$;
  Update the best heuristics $h_t(t = 1, .., k)$ for the k tasks;
Return $h_t(t = 1, .., k)$.

Algorithm 2: `TournamentSpecific`

Input: A merged sub-population $\mathbb{S}_{meg}$, task index t, tournament size s
Output: An LGP individual $\mathbf{f}$

Randomly select an LGP individual $\mathbf{f}'$ from $\mathbb{S}_{meg}$;
$\mathbf{f} \leftarrow \mathbf{f}'$;
$Fit \leftarrow \mathbf{Fit}(\mathbf{f}, t)$;
for $j \leftarrow 1$ *to* $s - 1$ **do**
  Randomly select an LGP individual $\mathbf{f}'$ from $\mathbb{S}_{meg}$;
  $Fit' \leftarrow \mathbf{Fit}(\mathbf{f}', t)$;
  if Fit' *is better than* Fit **then**
    $\mathbf{f} \leftarrow \mathbf{f}'$, $Fit = Fit'$;
Return $\mathbf{f}$;

D. Breeding Offspring

To improve the effectiveness of sharing knowledge, the better-parent reservation is proposed to enhance the crossover operator. The riffle shuffle is proposed to effectively integrate

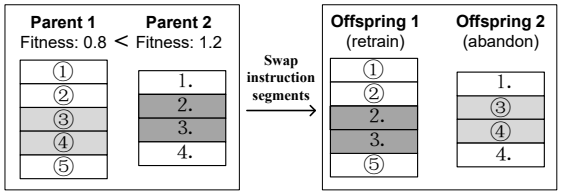


Fig. 4. An example of linear crossover with better-parent reservation strategy. Smaller fitness is better.

Algorithm 3: Linear crossover with `BetterParentRes`$(\cdot)$

Input: Set of sub-populations $\{\mathbb{S}_0, ..., \mathbb{S}_k\}$, index of current sub-population i, tournament selection size s
Output: A new offspring $\mathbf{c}$.

1 $t = i$;
2 **if** $i = 0$ **then**
3 $\quad t \leftarrow$ `randint`$(1, k)$;
4 $\mathbb{S}_{meg} \leftarrow \mathbb{S}_0 \bigcup \mathbb{S}_t$
5 Apply `TournamentSpecific`$(\mathbb{S}_{meg}, t, s)$ to select $\mathbf{p_a}$ and $\mathbf{p_b}$ respectively;
6 Apply linear crossover on $\mathbf{p_a}$ and $\mathbf{p_b}$ to produce two offspring $\mathbf{c_a}$ and $\mathbf{c_b}$;
`// Better-parent reservation`
7 $\mathbf{c} \leftarrow \mathbf{c_a}$;
8 **if** $\mathbf{p_b}$ *is better than* $\mathbf{p_a}$ **then**
9 $\quad \mathbf{c} \leftarrow \mathbf{c_b}$;
10 **Return** $\mathbf{c}$;

knowledge from different tasks.

1) Linear crossover with better-parent reservation: Multi-task optimization framework promotes effectiveness by sharing knowledge among similar tasks. However, different tasks may have different specialist knowledge. The crossover operator should not only share useful common knowledge among different sub-populations but also retain useful specialist building blocks for each specific task. It is reasonable to assume that the parent with better fitness on a certain task is more likely to have more useful building blocks for that task. Based on this assumption, the crossover operator in this work only retains the child whose corresponding parent (i.e., the parent accepting a new genome to form the offspring) has better fitness. An example of the better-parent reservation is shown in Fig. 4 in which the to-be-swapped instructions are highlighted as dark. Since *Parent 1* has better (i.e., smaller) fitness than *Parent 2*, only *offspring 1* which is generated from *Parent 1* by accepting new instructions is retained. The pseudo-code of the linear crossover with better-parent reservation is shown as Alg. 3, in which the linear crossover is followed by the better-parent reservation based on the fitness of the parents.

2) Riffle shuffle: Riffle shuffle is a technical term for playing cards. When there are two decks of cards, the riffle shuffle integrates them into a single deck by alternatively interleaving the two decks of cards and maintaining the relative order of cards within every deck. This concept is extended to our work to integrate individuals from different tasks into a new offspring. By this means, heuristics for different tasks cannot only maintain the relative order of their instructions, but also they can use the building blocks from other tasks.

The pseudo-code of the proposed riffle shuffle operator is

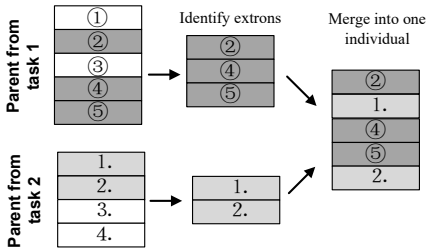


Fig. 5. An example of the riffle shuffle operator for LGP individuals

shown in Alg. 4. First, the riffle shuffle operator samples several LGP parents from different tasks. Then, the effective instructions (i.e., extrons) are extracted based on their output registers (lines 5 to 13) (The extraction method of effective instructions can be referred to [11]). The extracted effective instructions for different tasks are stored in lists respectively. The task index whose sampled parent has the best rank is recorded for later use. For lines 14 to 19, the riffle shuffle operator alternatively interleaves instructions from the lists to form a new instruction sequence. To prevent LGP programs from increasing size too rapidly, a maximum step size η is defined to limit the largest variation of program size. Specifically, the new program size L of the offspring is defined by aggregating the average program size of parents and a random variation size based on η (lines 20 to 21). L is also limited by the maximum and minimum program size of LGP individuals (L_{max} and L_{min}). For lines 22 to 27, when the actual program size after merging exceeds L, instructions are randomly removed until the actual program size is consistent with L. To protect the useful building blocks, the instructions from the parent with a better rank have a higher priority to be kept in the offspring. Specifically, the instructions from the parent with a better rank are extracted based on the output register R_{t^*-1} (line 24). When the program size of the parent with a better rank is smaller than L and the offspring still contains the instructions from other tasks (i.e., $|\mathbf{G}| < |\mathbf{c}|$), the instructions of better parent are protected from being removed (lines 25 and 26).

Fig. 5 shows an example of the riffle shuffle. The two LGP parents come from two different tasks and have three and two effective instructions respectively. They first extract the effective instructions, which are the second, fourth, and fifth instructions of the first parent, and the first and second instructions of the second parent. Then, the two sequences of effective instructions are integrated into one sequence alternatively to form an offspring.

Both better-parent reservation and riffle shuffle can be applied to other methods. Specifically, better-parent reservation strategy can be applied together with other crossover operators that accept more than one parent. Riffle shuffle is an LGP-based genetic operator for transferring genetic materials. Further, riffle shuffle is designed for merging building blocks from different tasks. Applying riffle shuffle to evolve a single task might produce many redundant instructions since the building blocks from a single task are often too similar. The effectiveness of the two operators and their collaboration with

Algorithm 4: `RiffleShuffle`

Input: Set of sub-populations $\{\mathbb{S}_0, ..., \mathbb{S}_k\}$, step size η, tournament selection size s
Output: A new offspring $\mathbf{c}$.

$n \leftarrow$ `randint`$(2, k)$;
$\mathbb{T}, \mathbb{P}, \mathbb{G}, \mathbf{c} \leftarrow \emptyset$;
Randomly select n unique task indices to initialize $\mathbb{T}$;
$t^*, Rank^* \leftarrow +\infty$
foreach t *in* $\mathbb{T}$ **do**
  $\mathbf{p}' \leftarrow$ `TournamentSpecific`$(\mathbb{S}_0 \bigcup \mathbb{S}_t, t, s)$;
  $\mathbb{P} \leftarrow \mathbb{P} \bigcup \mathbf{p}'$;
  $\mathbf{G} \leftarrow$ `IdentifyExtrons`$(\mathbf{p}', \{R_{t-1}\})$;
  if $\mathbf{G} \neq \emptyset$ **then**
    $\mathbb{G} \leftarrow \mathbb{G} \bigcup \mathbf{G}$;
  if $\mathbf{Rank}(\mathbf{p}', t) < Rank^*$ **then**
    $Rank^* \leftarrow \mathbf{Rank}(\mathbf{p}', t)$;
    $t^* \leftarrow t$;
// merge into one individual
while $\mathbb{G} \neq \emptyset$ **do**
  $\mathbf{G} \leftarrow$ Randomly select an element from $\mathbb{G}$;
  $f \leftarrow$ get and remove the first element from $\mathbf{G}$;
  Append f to $\mathbf{c}$;
  if $\mathbf{G} = \emptyset$ **then**
    remove $\mathbf{G}$ from $\mathbb{G}$;
// randomly remove instructions
$L \leftarrow \frac{\Sigma_{\mathbf{p} \in \mathbb{P}} |\mathbf{p}|}{n} +$ `randint`$(-\eta, \eta)$;
$L \leftarrow l_{min}$ if $L < l_{min}$ (or $L \leftarrow l_{max}$ if $L > l_{max}$);
while $|\mathbf{c}| > L$ **do**
  $f_{rmv} \leftarrow$ randomly select an instruction from $\mathbf{c}$;
  $\mathbf{G} \leftarrow$ `IdentifyExtrons`$(\mathbf{c}, R_{t^*-1})$;
  while $|\mathbf{G}| < L$ *and* $|\mathbf{G}| < |\mathbf{c}|$ *and* $f_{rmv} \in \mathbf{G}$ **do**
    $f_{rmv} \leftarrow$ randomly select an instruction from $\mathbf{c}$;
  Remove f_{rmv} from $\mathbf{c}$;
Return $\mathbf{c}$;

other methods are investigated in Section VI-A.

IV. A Case Study On Dynamic Job Shop Scheduling

A. Problem Description

In this paper, we consider dynamic job shop scheduling (DJSS) as a case study to verify the performance of the newly proposed method. DJSS is a ubiquitous combinatorial optimization problem in modern production lines [49]–[51], and often has many similar optimization problems. For example, the peak and off seasons of the same production line might have similar scheduling methods, and the customers might share similar requirements with the manager on the production line. Fully utilizing the common knowledge among these tasks (i.e., multitask optimization) is more efficient than searching from a scratch for every single task. Further, multitask tree-based GP methods have shown to be effective in solving multitask DJSS problems [7], [48], [52]. It is convincing to verify the proposed MLSI by comparing with the existing multitask GP methods.

Technically, the job shop processes totally n jobs $\mathcal{J} = \{\mathcal{J}_1, ..., \mathcal{J}_n\}$. Each $\mathcal{J}_j$ consists of a sequence of operations $\mathcal{O}_\mathbf{j} = \{\mathcal{O}_{j1}, ..., \mathcal{O}_{jm_j}\}$ where m_j is the number of operations for job $\mathcal{J}_j$. Every operation $\mathcal{O}_{ji}$ can only be processed by a predefined machine $\pi(\mathcal{O}_{ji})$ with a processing time $p(\mathcal{O}_{ji})$. Besides, $\mathcal{J}_j$ has an arrival time $\alpha(\mathcal{J}_j)$ and a weight $\omega(\mathcal{J}_j)$. The job shop has a set of machines $\mathcal{M} = \{\mathcal{M}_1, ..., \mathcal{M}_{|\mathcal{M}|}\}$. All of these machines have a buffer to store ready operations for their own. The execution of operations on machines is assumed to be atomic, which means machines do not interrupt an operation once it starts processing. Machines can process at most one operation at any time. Dynamic events may occur during the execution of DJSS [53], [54]. Specifically, we focus on DJSS with new job arrival, in which unknown new jobs arrive to the job shop over time [55], [56].

The flowtime and tardiness of jobs which are popular metrics in existing literature [7], [13], are considered as performance measures in this paper. Flowtime is the total time cost of jobs from arrival to being completed. Tardiness denotes the delay of completion from a given due date. The due date of each job is defined as the total processing time of the job multiplied by a due-date factor. The due-date factor for determining due date of jobs is defined as 1.5 in the simulation. The flowtime and tardiness of a job $\mathcal{J}_j$ are defined as follows.

- $\text{flowtime}(\mathcal{J}_j) = C_j - \alpha(\mathcal{J}_j)$
- $\text{tardiness}(\mathcal{J}_j) = \max(0, C_j - d_j)$

where C_j and d_j are the completion time and the due date of $\mathcal{J}_j$ respectively. The objectives of our DJSS problems are minimizing the mean, max, and weighted mean flowtime (denoted as Fmean, Fmax, WFmean) and tardiness (denoted as Tmean, Tmax, WTmean), which are defined as follows.

- $\text{Fmean} = \frac{\sum_{\mathcal{J}_j \in \mathcal{J}} \text{flowtime}(\mathcal{J}_j)}{n}$
- $\text{Fmax} = \max_{\mathcal{J}_j \in \mathcal{J}} \text{flowtime}(\mathcal{J}_j)$
- $\text{WFmean} = \frac{\sum_{\mathcal{J}_j \in \mathcal{J}} \text{flowtime}(\mathcal{J}_j) \times \omega(\mathcal{J}_j)}{n}$
- $\text{Tmean} = \frac{\sum_{\mathcal{J}_j \in \mathcal{J}} \text{tardiness}(\mathcal{J}_j)}{n}$
- $\text{Tmax} = \max_{\mathcal{J}_j \in \mathcal{J}} \text{tardiness}(\mathcal{J}_j)$
- $\text{WTmean} = \frac{\sum_{\mathcal{J}_j \in \mathcal{J}} \text{tardiness}(\mathcal{J}_j) \times \omega(\mathcal{J}_j)}{n}$

B. Applying GP to DJSS

New coming jobs often disturb the schedule of the job shop. To improve the effectiveness of the overall job shop, instant reaction (e.g., re-scheduling) is necessary. Dispatching rules are designed to schedule new coming jobs as soon as they arrive [57]–[60]. But those manually designed dispatching rules are not effective enough to handle complex situations. To improve the effectiveness of dispatching rules, GP is applied to help humans design promising dispatching rules automatically [52], [61]–[63], which is known as the GP-based hyper-heuristic method.

In contrast to searching the specific schedules for job shops, GP-based hyper-heuristic generates a heuristic (i.e., dispatching rule) based on the given primitives [64]. The obtained dispatching rule is then applied to DJSS problem instances to construct schedules in a greedy manner. Specifically, the dispatching rules are used to prioritize operations in machines [57], [59], [65]. In this paper, LGP is applied to DJSS as a hyper-heuristic method to design dispatching rules.

C. Simulation Configuration

Based on the problem formulation of DJSS, this paper sets up a DJSS simulation to evaluate the fitness of GP individuals. Every GP individual is decoded into a dispatching rule. Once

TABLE I
THE SETTING OF MULTITASK SCENARIOS REPRESENTED BY OPTIMIZED OBJECTIVES AND UTILIZATION LEVELS

Scenarios	#tasks (k)	Specific settings of tasks
A	3	$<$ Fmean, 0.95 $>$,$<$ Fmean, 0.85 $>$,$<$ Fmean, 0.75 $>$
B	3	$<$ Tmean, 0.95 $>$,$<$ Tmean, 0.85 $>$,$<$ Tmean, 0.75 $>$
C	3	$<$ WTmean, 0.95 $>$,$<$ WTmean, 0.85 $>$,$<$ WTmean, 0.75 $>$
D	2	$<$ Fmax, 0.95 $>$,$<$ Tmax, 0.95 $>$
E	2	$<$ WFmean, 0.95 $>$,$<$ WTmean, 0.95 $>$
F	3	$<$ Fmean, 0.95 $>$,$<$ Tmean, 0.85 $>$,$<$ WTmean, 0.75 $>$

a machine is idle and its buffer is not empty, the machine will pick up one available operation with the best priority given by the dispatching rule. The job shop totally has 10 machines (i.e., $|\mathcal{M}| = 10$). The number of operations m for every job ranges from 2 to 10. Every operation has a floating-point processing time ranging from 1 to 99 time units. Jobs have different weights, i.e., 20%, 60%, and 20% of jobs have weights of 1, 2, and 4 respectively. To evaluate GP methods in a steady state, our simulation has 1000 warm-up jobs and takes the performance of the following 5000 jobs into account.

Jobs arrive to the job shop based on a Poisson distribution, as shown in Eq. 1. t is the time interval before the next coming job, and λ is the mean processing time of a job in the job shop, defined by Eq. 2. ν is the average number of operations in all jobs and μ is the average processing time of all operations. We control the arrival rate of jobs by the utilization level of machines ρ. When ρ is relatively large, the mean actual processing time of jobs in the job shop is short, and thus new jobs arrive to the job shop more frequently.

$$P(t = \text{next job arrival time}) \sim \exp(-\frac{t}{\lambda}) \tag{1}$$

$$\lambda = \frac{\nu \cdot \mu}{\rho \cdot |\mathcal{M}|} \tag{2}$$

D. Multitask Scenarios

In multitask optimization of DJSS, six multitask scenarios are designed for verifying the performance of the proposed method according to the above objectives [7]. In these multitask scenarios, each specific optimization problem with a specific optimized objective and a specific utilization level of the simulation is defined as a task, denoted as "$<$ objective, $\rho >$". The settings of the six multitask scenarios are shown in Table I. To have a comprehensive investigation, Scenarios A to F cover a wide range of tasks, with respect to the optimized objective and utilization level. It should be noted that the six scenarios cover different job sets which are specified by the utilization level. For example, Sce. A, B, C, and F have the same settings of utilization level for the three tasks (i.e., 0.95, 0.85, and 0.75) and thus have the same job sets with each other. Sce. D and E have different job sets from Sce. A, B, C, and F. Further, the three tasks in Sce. A are configured with different job sets since each task is set with a different utilization level. Other parameters in the simulation are the same for all tasks, as introduced in Section IV-C.

E. Design of Comparison

We select four compared methods to verify the performance of MLSI. First, multi-population LGP (MPLGP) is included as the baseline method. It solves multiple tasks simultaneously, each by a sub-population. But these sub-populations do not exchange any information and solve tasks independently. Second, one of the state-of-the-art multitask tree-based GP methods, M^{2}GP [7], is compared in the experiment. M^{2}GP is a recently proposed method for solving multitask dynamic scheduling. In addition, to comprehensively verify the effectiveness of the proposed mechanism (i.e., transfer knowledge via shared individuals), we compare with two recent multitask LGP methods proposed in [13], denoted as M^{2}LGP and MFLGP respectively. Specifically, M^{2}LGP replaces the tree-based GP individuals in M^{2}GP [7] with LGP individuals, and MFLGP replaces the individual representation and genetic operators in MFEA [3] (a popular framework for multitask optimization) with that of LGP.

The parameters of the compared methods are designed based on their original paper or existing literature [7], [13]. Specifically, the population size and generations are set separately based on the recommended settings of tree-based GP and LGP. More specifically, tree-based GP is recommended to have a large population and a small number of generations while LGP is recommended to have a small population and a large number of generations [42]. All the compared methods have the same total number of simulations to ensure fairness. All methods keep elite individuals directly to the next generation and apply tournament selection methods to select parents. For MLSI, `TournamentSpecific`$(\cdot)$ replaces the conventional tournament selection. M^{2}GP, which is designed based on tree-based GP, mainly breeds offspring by crossover, mutation, and reproduction. The rates of these three operators are 80%, 15%, and 5% respectively [7]. On the other hand, the LGP-based methods, including MPLGP, M^{2}LGP, MFLGP, and MLSI, employ macro mutation, micro mutation, linear crossover, and reproduction with rates of $\theta_{ma} : \theta_{mi} : \theta_c : \theta_r = 30\% : 30\% : 35\% : 5\%$ to produce offspring. MLSI enhances the linear crossover by the better-parent reservation, and the generalist sub-population in MLSI replaces mutation operators by riffle shuffle. All the compared methods use the same set of terminal and function sets to generate dispatching rules [13], [53]. The terminal set contains sixteen DJSS features, as shown in Table II. The function set in this experiment contains $\{+, -, \times, \div, \max, \min\}$ where the division returns one if divided by zero. LGP-based methods also utilize a register set of 8 registers. The registers are initialized by the first eight features of Table II before every execution.

The transfer rate of M^{2}GP is set as 0.3 as suggested in [7]. Given that their knowledge transfer is mainly implemented based on the crossover, actually $80\% \times 0.3 = 24\%$ of crossover mate parents from different sub-populations. To keep the rate of knowledge transfer the same, 68.6% of linear crossover ($35\% \times 68.6\% = 24\%$) in M^{2}LGP and MFLGP produce offspring based on parents from different sub-populations. Since MLSI selects parents from a merged population, the parameter of transfer rate is not needed for MLSI. Contrarily, MLSI introduces two new parameters, the population size of the generalist sub-population and the step size of riffle shuffle η. Without loss of generality, the size of the generalist sub-population is defined as 30 to approximate the transfer rate of

TABLE II
THE TERMINAL SET

Notation	Description
PT	Processing time of an operation in a job
NPT	Processing time of the next operation in a job
WINQ	Total processing time of operations in the buffer of a machine which is the corresponding machine of the next operation in a job
WKR	Total remaining processing time of a job
rFDD	Difference between the expected due date of an operation and the system time
OWT	Waiting time of an operation
NOR	Number of remaining operations of a job
NINQ	Number of operations in the buffer of a machine which is the corresponding machine of the next operation in a job
W	Weight of a job
rDD	Difference between the expected due date of a job and the system time
NWT	Waiting time of the next to-be-ready machine
TIS	Difference between system time and the arrival time of a job
SL	Slack: difference between the expected due date and the sum of the system time and WKR
NIQ	Number of operations in the buffer of a machine
WIQ	Total processing time of operations in the buffer of a machine
MWT	Waiting time of a machine

TABLE III
PARAMETERS OF ALL COMPARED METHODS

Parameters	M^2GP	MPLGP	M^2LGP	MFLGP	MLSI
number of sub-population	k	k	k	1	k+1
(sub) population size	400	100	100	k*100	generalist:30, specialist:70
generations	50	200	200	200	200
elitism selection size for each sub-population	10	3	3	6	3
tournament selection size	5	7	7	7	7
crossover parameters	inner node 90%, leaf node 10%	segment length$\leq$30, segment length difference$\leq$5, crossover point distance$\leq$30			
mutation parameters	inner node 90%, leaf node 10%	macro(insertion 67%, deletion 33%), micro (function 50%, destination register 25%, source register 12.5%, constant 12.5%)			
initial program size	min depth=2, max depth=6	min instruction=1, max instruction=10			
maximum program size	max depth=8	max instruction=50			

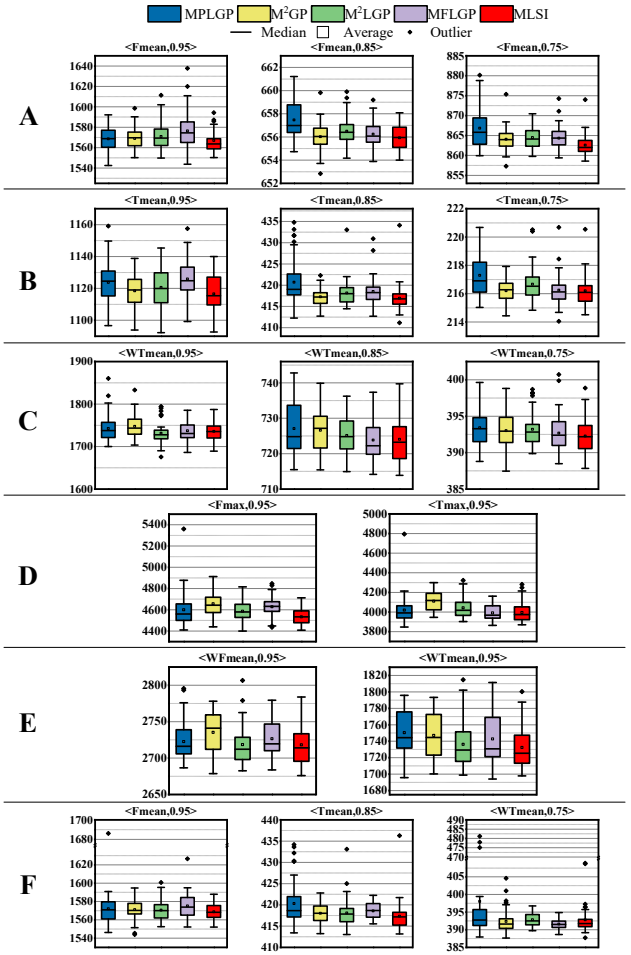


Fig. 6. The box plots of test performance on all tasks in different multitask scenarios.

TABLE IV
RESULTS OF FRIEDMAN TEST

Metrics	MPLGP	M^2GP	M^2LGP	MFLGP	MLSI
mean rank	3.78	3.56	3.13	3.16	**1.38**
p-value	$1.14E^{-4}$				

0.3. η is defined as 15 by default. The other parameters are set as Table III.

In the training phase, all the compared methods are trained by k DJSS instances for every generation, each DJSS instance for a specific task. The random seeds of instances are rotated every generation. All the compared methods are tested on 50 unseen DJSS instances after training, and the mean performance on these unseen instances is seen as the test performance.

V. RESULTS

In this section, we compare the training and test performance of all the compared methods. We analyze the objective values on test instances and training convergence curves. Friedman test and Wilcoxon test with a significance level of 0.05 are also applied to make a comprehensive analysis.

A. Test performance

Fig. 6 shows the test performance of all the compared methods on different multitask scenarios. We can see that the results of MLSI have relatively small objective values in most tasks. For example, in the complex tasks of Scenarios A, B, and D (e.g., A-<Fmean,0.95>, B-<Tmean,0.95>, and D-<Fmax,0.95>), MLSI has better medians and averages of test performance than the other compared methods over 50 independent runs. In other tasks such as C-<WTmean,0.95> and F-<Fmean,0.95>, although we cannot see significant differences between MLSI and the other compared methods, we can confirm that the medians and averages of MLSI test performance are similar to the best medians and best averages in the tested problems.

To have a more comprehensive comparison, we analyze the test performance by the Friedman test, as shown in Table IV. The p-value of the Friedman is $1.14E^{-4}$, which is much smaller than 0.05. It implies that there is a significant difference among these compared methods. Besides, the mean rank from the Friedman test shows that MLSI has the best average ranking on these tasks.

Based on the Friedman test, post-hoc analyses by the Wilcoxon rank sum test with Bonferroni correction and a significance level of 0.05 are conducted to further analyze the

TABLE V
THE POST-HOC ANALYSES BY WILCOXON TEST WITH BONFERRONI TEST (VS. MLSI) AND SIGNIFICANCE LEVEL OF 0.05.

Scenarios	Tasks	MPLGP	M^2GP	M^2LGP	MFLGP
A	<Fmean,0.95>	≈	≈	−	−
	<Fmean,0.85>	−	−	−	−
	<Fmean,0.75>	−	≈	−	≈
B	<Tmean,0.95>	−	≈	≈	−
	<Tmean,0.85>	−	≈	−	−
	<Tmean,0.75>	−	≈	−	≈
C	<WTmean,0.95>	≈	−	≈	≈
	<WTmean,0.85>	−	−	≈	≈
	<WTmean,0.75>	−	≈	≈	≈
D	<Fmax,0.95>	−	−	−	−
	<Tmax,0.95>	≈	−	−	≈
E	<WFmean,0.95>	≈	−	≈	≈
	<WTmean,0.95>	−	−	≈	≈
F	<Fmean,0.95>	≈	≈	≈	−
	<Tmean,0.85>	−	−	≈	−
	<WTmean,0.75>	≈	≈	≈	≈
win-draw-lose		0-6-10	0-8-8	0-9-7	0-9-7
p-value with Bonferroni correction		$1.53E^{-4}$	$8.24E^{-4}$	0.015	0.012

performance on different tasks. In Table V, "+" denotes that a method is significantly better than MLSI, "−" denotes that a method is significantly worse than MLSI on a certain task, and "≈" denotes competitive performance with MLSI [3]. It can be seen that the four compared methods are significantly inferior to MLSI on most tasks. Specifically, MLSI is significantly better than the three state-of-the-art methods (i.e., M^2GP, M^2LGP, MFLGP) in nearly half of the tasks. The p-values with Bonferroni correction also validate that MLSI has a significantly better overall performance than the others since all of them are much smaller than 0.05. These results verify that the newly proposed multitask mechanism is very effective in improving the performance of multitask LGP.

B. Training Efficiency

To validate the training efficiency of MLSI, we compare the training convergence. Fig. 7 shows the test objectives of all compared methods over evaluation times. We select one of the most complex tasks (i.e., those with a utilization level of 0.95) in each scenario for investigation. It can be seen that MLSI converges faster and lower than the others overall. Specifically, MLSI achieves better performance than the other compared methods over the whole training processing in A-<Fmean,0.95>, B-<Tmean,0.95>, D-<Fmax,0.95> and F-<Fmean,0.95>. Although MLSI performs similarly to other compared methods at the final stage of training in C-<WTmean,0.95> and E-<WFmean,0.95>, MLSI has a faster training efficiency at the early stage of training (e.g., before 10000 simulations) in E-<WFmean,0.95>. The results imply that MLSI has better efficiency in designing effective dispatching rules.

VI. FURTHER ANALYSIS

In this section, we make further analyses to answer the following research questions:

[3]The similar notations are used in the rest of the paper

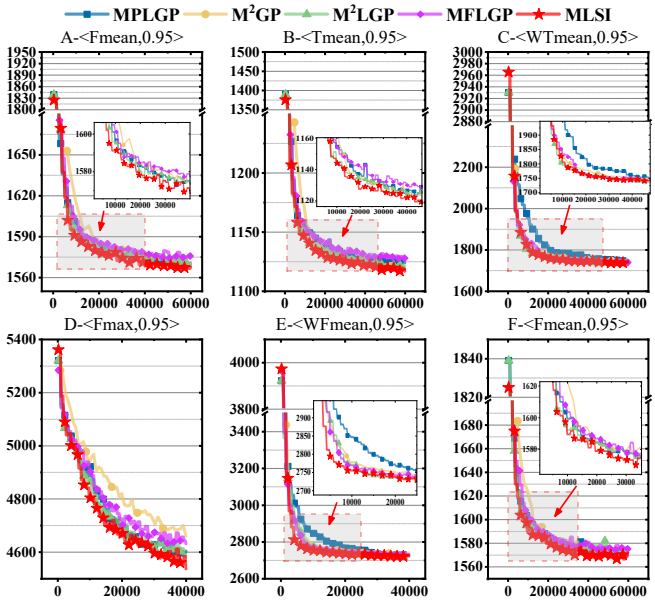


Fig. 7. Test objectives during evolution. X-axis:evaluation times, Y-axis:objective values.

- What is the effectiveness of the newly proposed riffle shuffle and the better-parent reservation strategy?
- How does performance change with the key parameters of MLSI (i.e., the population size of the generalist population and the step size of riffle shuffle η)?
- How does MLSI control the rate of knowledge transfer?
- What do MLSI individuals look like?

These questions are answered respectively as follows.

A. Component Analysis

To investigate the effectiveness of each key component of MLSI, this section conducts a component analysis. First, to verify the effectiveness of riffle shuffle, an MLSI variant without riffle shuffle, denoted as "MLSI/RS" is developed. Second, to verify the effectiveness of the better-parent reservation, we replace the crossover with the better-parent reservation with conventional crossover, which retains both of the two offspring after mating. This variant of MLSI is denoted as "MLSI/BPR". Further, to verify the performance gain from genetic transfer rather than genetic operator bias, we apply riffle shuffle and better-parent reservation to MPLGP, but has no knowledge transfer among tasks, denoted as "MPLGP+". Other settings of these two compared methods are the same as MLSI by default. The comparative results are shown in Table VI.

First, Table VI shows that simply applying the riffle shuffle and better-parent reservation to MPLGP is harmful to MPLGP performance. MPLGP+ performs much worse than MLSI on all the tasks. Since riffle shuffle is specialized for merging various building blocks from different tasks, the similar building blocks in the parents from a single task might be unnecessarily duplicated by riffle shuffle, which produces a large number of redundant instructions in offspring. The results further verify that the performance gain of MLSI is due to genetic transfer. Second, removing riffle shuffle from MLSI reduces

TABLE VI
MEAN (STD.) TEST PERFORMANCE OF MLSI WITH DIFFERENT COMPONENTS AND MPLGP+. THE BEST MEAN VALUES ARE HIGHLIGHTED BY **BOLD** FONT

Tasks	MPLGP+	MLSI/RS	MLSI/BPR	MLSI
A-<Fmean,0.95>	1584.6 (21.4) −	1570 (11.27) ≈	1570.9 (10.85)−	**1567.1** (16.51)
A-<Fmean,0.85>	1118.4 (192.7)−	863.4 (2.67) ≈	863.4 (3.62) ≈	**862.6** (2.53)
A-<Fmean,0.75>	761.3 (100.3) −	656 (1.01) ≈	656.2 (1.25) ≈	**655.9** (1.02)
B-<Tmean,0.95>	1133.8 (23.3) −	1123.3 (18.82) −	1123.9 (13.42)−	**1116.6** (11.25)
B-<Tmean,0.85>	691.2 (164.8) −	417.2 (2.53) ≈	417.2 (2.28) ≈	**417** (3.14)
B-<Tmean,0.75>	332.4 (141.9) −	**216** (1.12) ≈	216.2 (1.12) ≈	216.2 (1.06)
C-<WTmean,0.95>	1925.8 (302.3)−	1743.3 (22.13) ≈	**1732.3** (24.65)≈	1735.8 (23.08)
C-<WTmean,0.85>	1670.2 (691.2)−	726.4 (5.34) −	**724** (5.51) ≈	**724** (6.05)
C-<WTmean,0.75>	682.5 (192.5) −	393.2 (2.8) ≈	**392.3** (1.95) ≈	**392.3** (2.41)
D-<Fmax,0.95>	4628.9 (117.6)−	4624.5 (208.65)−	4559.2 (96.6) ≈	**4533.5** (75.13)
D-<Tmax,0.95>	1.6E4 (23142) −	4024.3 (102.35)≈	**3968** (73.01) ≈	3994.7 (92.4)
E-<WFmean,0.95>	2856.4 (233.2)−	2722.8 (25.9) ≈	**2713.5** (21.78)≈	2718.2 (26.33)
E-<WTmean,0.95>	4799.1 (1517) −	1734.2 (24.55) ≈	**1727.1** (26.63)≈	1732.4 (25.78)
F-<Fmean,0.95>	1587.3 (16.5) −	1569.7 (11.88) ≈	**1568** (11.26) ≈	1568.8 (9.34)
F-<Tmean,0.85>	671.5 (180.1) −	418.1 (2.45) −	418 (2.62) −	**417.3** (3.3)
F-<WTmean,0.75>	786.6 (211.5) −	394.1 (2.65) −	393 (2.66) ≈	**392.8** (3.86)
win-draw-lose	0-0-16	0-11-5	0-13-3	
mean rank	3.94	2.75	1.81	**1.50**
p-value	5.0E-07	0.03	0.90	

the effectiveness since MLSI/RS is significantly worse than MLSI on five tasks and has a significantly worse (higher) mean rank than MLSI based on the Friedman rank sum test with the Bonferroni correction. The results imply that the riffle shuffle operator is prominent for MLSI. Third, MLSI without the better-parent reservation performs similarly to MLSI on average, implying that selectively retaining offspring based on superior parents is not so crucial for MLSI.

B. Parameter Sensitivity Analysis

There are two new parameters in MLSI, which are the population size of the generalist sub-population and step size η of the riffle shuffle. To investigate the parameter sensitivity, we compare the performance of MLSI with different parameter settings. Four different MLSI versions are developed. Specifically, "MLSI-pop15", "MLSI-pop20", and "MLSI-pop45" set the population size of the shared population as 15, 20, and 45 respectively. "MLSI-η5", "MLSI-η10" and "MLSI-η20" set η as 5, 10 and 20 respectively. When the population size of the shared population is changed, to be fair, the population size of other specialized sub-populations is updated accordingly (e.g., MLSI-pop15 has specialized sub-populations with 85 individuals). Other parameters of the four methods are set the same as MLSI.

The test performance of different settings is shown in Table VII. The results show that the performance of all different settings is quite similar to the default ones in most cases based on the Wilcoxon rank sum test. Though significant differences can be seen on a few tasks, the mean rank by the Friedman test and the p-values with Bonferroni correction verify that different parameters have no significant difference in terms of overall performance on these tasks. The results verify the performance of MLSI is robust to the two newly introduced parameters.

C. Rate of Knowledge Transfer

When selecting parents from a merged set of individuals that consists of generalist and specialist sub-populations, MLSI always selects parents that are good at a certain task, no matter where they come from. It helps MLSI flexibly adjust the rate of knowledge transfer (i.e., selecting parents from the generalist sub-population) in the course of evolution. To validate the necessity of selecting a suitable rate of knowledge transfer, we compare MLSI with different fixed transfer rates and analyze the ratio of knowledge transfer over generations in this section.

First, we explicitly distinguish generalist and specialist sub-populations in the crossover. We select parents from generalist and specialist sub-population based on a fixed rate. The fixed rate of selecting parents from the generalist sub-population is set as 0.1, 0.2, and 0.35 respectively. When the rate is small, the sub-populations will mainly mate within their individuals for crossover, and thus the knowledge transfer among tasks is weak. These compared methods are denoted as "MLSI-fix0.1", "MLSI-fix0.2", and "MLSI-fix0.35" respectively. As shown in Table VIII, when fixing the knowledge transfer rate, the performance of all fixed-rate MLSI variants is decreased. They have a larger mean rank than MLSI and perform significantly worse on two tasks. Furthermore, MLSI-fix0.1 has a significantly worse overall performance than MLSI based on the p-value with Bonferroni correction. Besides, we can see that these fixed-rate variants have different performances on different tasks. For example, while MLSI-fix0.35 has worse mean performance on tasks of Scenario B, MLSI-fix0.2 is not so effective on tasks of Scenario D. It implies that to further improve the performance of specific tasks, a suitable transfer rate should be deliberated.

Second, we investigate the adaptation ability of knowledge transfer rate in MLSI, which is defined as the mean rate of mating with generalist sub-population parents. The curves of mating rate over generations are drawn. If the mating rate is high, MLSI prefers to produce offspring based on the parents from the generalist sub-population. Thus, the knowledge transfer among tasks is frequent. Contrarily, the individuals from the generalist sub-population are hardly selected by MLSI. The specialist sub-populations mainly evolve independently.

As shown in Fig. 8, the decline in mating rate can be seen in all scenarios. At the beginning of evolution, sharing knowledge among tasks is very useful. Individuals in generalist sub-populations often have superior performance, which is more likely to produce effective offspring. However, the mating rate decreases with generations. When heuristics become more sophisticated, evolving them specifically is more effective than sharing knowledge (e.g., building blocks) with other similar tasks. Besides, if we look at the level of the mating rate, we can also find that problems with less similar tasks have lower mating rates. For example, Scenario F whose both optimization objectives and utilization levels are different has a mating rate of 0.1 at the final stage of evolution. Contrarily, other problems in which only the objective or utilization level is different, have a mating rate of 0.15 at the end. Based on these results, we believe MLSI not only adjusts the transfer rate in different stages of evolution but also decides the rate based on similarity among tasks.

TABLE VII
MEAN (STD.) TEST PERFORMANCE OF MLSI WITH DIFFERENT PARAMETER SETTINGS.

Tasks	MLSI-pop15	MLSI-pop20	MLSI-pop45	MLSI-η5	MLSI-η10	MLSI-η20	MLSI
A-<Fmean,0.95>	1565.8 (11.6) ≈	1569.7 (12.25) −	1569.1 (11.61) ≈	**1564.4** (8.61) ≈	1570.7 (11.3) −	1570.3 (17.2) ≈	1567.1 (16.51)
A-<Fmean,0.85>	**862.6** (2.27) ≈	863.5 (2.48) −	863.3 (2.41) −	863 (2.71) ≈	863.5 (1.84) −	862.9 (2.42) ≈	**862.6** (2.53)
A-<Fmean,0.75>	656.2 (1) ≈	656 (0.96) ≈	655.9 (1.12) ≈	655.8 (1.08) ≈	656.1 (0.97) ≈	**655.7** (1.12) ≈	655.9 (1.02)
B-<Tmean,0.95>	**1115.9** (12.63) ≈	1119.5 (19.76) ≈	1121.3 (11.19) −	1117.9 (10.44) ≈	1120.7 (13.14) ≈	1120.9 (11.47) ≈	1116.6 (11.25)
B-<Tmean,0.85>	**416.5** (2.29) ≈	416.7 (2.72) ≈	417.1 (2.46) ≈	417.4 (2.64) ≈	417 (2.62) ≈	417.3 (2.63) ≈	417 (3.14)
B-<Tmean,0.75>	**216.1** (0.83) ≈	216.2 (1.05) ≈	216.3 (1.01) ≈	216.2 (1.02) ≈	216.3 (1.25) ≈	216.3 (1.12) ≈	216.2 (1.06)
C-<WTmean,0.95>	1727.5 (21.91) +	**1726.9** (22.4) +	1731.2 (24.44) ≈	1729.4 (23.85) ≈	1730.1 (23.79) ≈	1728 (22.87) ≈	1735.8 (23.08)
C-<WTmean,0.85>	**722.4** (6.35) ≈	723.1 (4.63) ≈	724.4 (7.19) ≈	723.6 (6.27) ≈	723.8 (6.14) ≈	723.1 (5.64) ≈	724 (6.05)
C-<WTmean,0.75>	392.2 (2.07) ≈	392.1 (2.3) ≈	392.1 (2.1) ≈	392.1 (2.54) ≈	392.2 (2.34) ≈	**391.9** (2.09) ≈	392.3 (2.41)
D-<Fmax,0.95>	4542.1 (74.6) ≈	4541.8 (83.84) ≈	4557.7 (85.67) ≈	4559 (89.01) ≈	4538.8 (93.97) ≈	4549.3 (89.26) ≈	**4533.5** (75.13)
D-<Tmax,0.95>	3984.6 (94.79) ≈	3970.2 (81.27) ≈	**3956** (85.99) +	3961.9 (93.57) ≈	3984.2 (95.44) ≈	3963.6 (83.82) ≈	3994.7 (92.4)
E-<WFmean,0.95>	2714.3 (25.23) ≈	2713.9 (25.96) ≈	2717.7 (25.08) ≈	2714.4 (23.89) ≈	**2712.5** (28.03) ≈	2713.5 (20.68) ≈	2718.2 (26.33)
E-<WTmean,0.95>	1728.9 (23.36) ≈	1728.9 (24.42) ≈	1728.6 (21.82) ≈	1728.7 (22.46) ≈	1726.8 (27.66) ≈	**1725.3** (23.76) ≈	1732.4 (25.78)
F-<Fmean,0.95>	1569.6 (18.02) ≈	1566.2 (11.44) ≈	1568.4 (15.68) ≈	1569.9 (11.14) ≈	**1564.8** (11.46) ≈	1568.5 (11.32) ≈	1568.8 (9.34)
F-<Tmean,0.85>	417.4 (3.23) ≈	**416.5** (2.32) ≈	417.2 (2.5) ≈	417.7 (2.58) ≈	416.9 (2.37) ≈	417.5 (3.1) ≈	417.3 (3.3)
F-<WTmean,0.75>	392.6 (1.98) ≈	**391.8** (1.96) ≈	392.2 (2.53) ≈	392.3 (1.87) ≈	392.6 (3.72) ≈	394.1 (12.07) ≈	392.8 (3.86)
win-draw-lose	1-15-0	1-13-2	1-13-2	0-16-0	0-14-2	0-16-0	
mean rank	4.09	**3.13**	4.75	3.94	3.97	3.91	4.22
p-value	1.000	0.782	0.993	1.000	1.000	1.000	

TABLE VIII
MEAN (STD.) TEST PERFORMANCE OF MLSI WITH FIXED TRANSFER RATES

Tasks	MLSI-fix0.1	MLSI-fix0.2	MLSI-fix0.35	MLSI
A-<Fmean,0.95>	1569.5 (11.7) ≈	1568.6 (11.0) ≈	1570.9 (12.3) −	**1567.1** (16.5)
A-<Fmean,0.85>	864 (2.1) −	863.3 (2.4) −	863.6 (2.7) −	**862.6** (2.5)
A-<Fmean,0.75>	656.4 (1.1) ≈	656.1 (1.1) ≈	656.2 (1.1) ≈	**655.9** (1.0)
B-<Tmean,0.95>	1120.4 (12.8) ≈	1118.8 (12.1) ≈	1121.7 (12.1) ≈	**1116.6** (11.3)
B-<Tmean,0.85>	417.7 (3.3) ≈	417.3 (3.3) ≈	417.4 (2.3) ≈	**417** (3.1)
B-<Tmean,0.75>	216.3 (0.9) ≈	216.3 (1.3) ≈	216.4 (1.1) ≈	**216.2** (1.1)
C-<WTmean,0.95>	1730.8 (22.5) ≈	1739.1 (30.7) ≈	**1723.4** (24.8) +	1735.8 (23.1)
C-<WTmean,0.85>	724.6 (5.4) ≈	725.1 (6.6) ≈	**724** (6.9) ≈	**724** (6.1)
C-<WTmean,0.75>	**391.7** (2.0) ≈	392.7 (2.4) ≈	392.4 (2.1) ≈	392.3 (2.4)
D-<Fmax,0.95>	4558.5 (91.1) ≈	4586.9 (106.8) −	**4532.4** (83.8) ≈	4533.5 (75.1)
D-<Tmax,0.95>	3985.6 (89.1) ≈	4004.9 (93.4) ≈	**3970.6** (86.2) ≈	3994.7 (92.4)
E-<WFmean,0.95>	2722.2 (27.9) ≈	**2714.8** (25.3) ≈	2719 (28.1) ≈	2718.2 (26.3)
E-<WTmean,0.95>	1734.4 (25.9) ≈	1733.6 (28.1) ≈	**1731.7** (27.5) ≈	1732.4 (25.8)
F-<Fmean,0.95>	1573.4 (12.9) ≈	1567.9 (12.0) ≈	**1567.6** (11.2) ≈	1568.8 (9.3)
F-<Tmean,0.85>	418.4 (2.6) −	417.4 (2.5) ≈	**417.2** (2.3) ≈	417.3 (3.3)
F-<WTmean,0.75>	393.9 (12.4) ≈	393.8 (12.2) ≈	**391.8** (1.9) ≈	392.8 (3.9)
win-draw-lose	0-14-2	0-14-2	1-13-2	
mean rank	3.31	2.63	2.22	**1.84**
p-value	0.01	0.31	0.84	

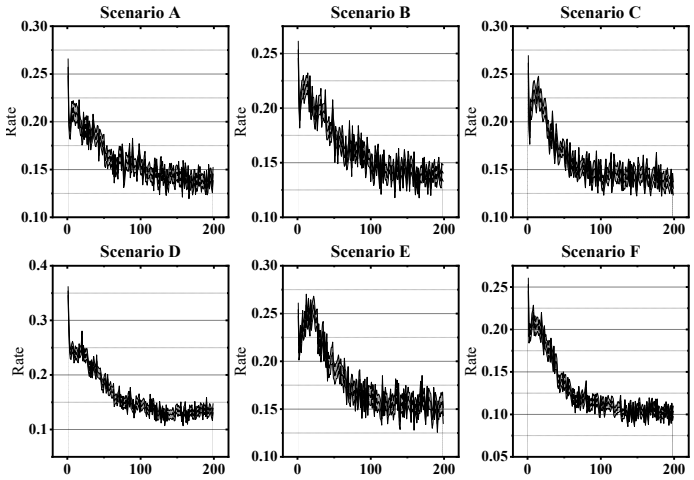


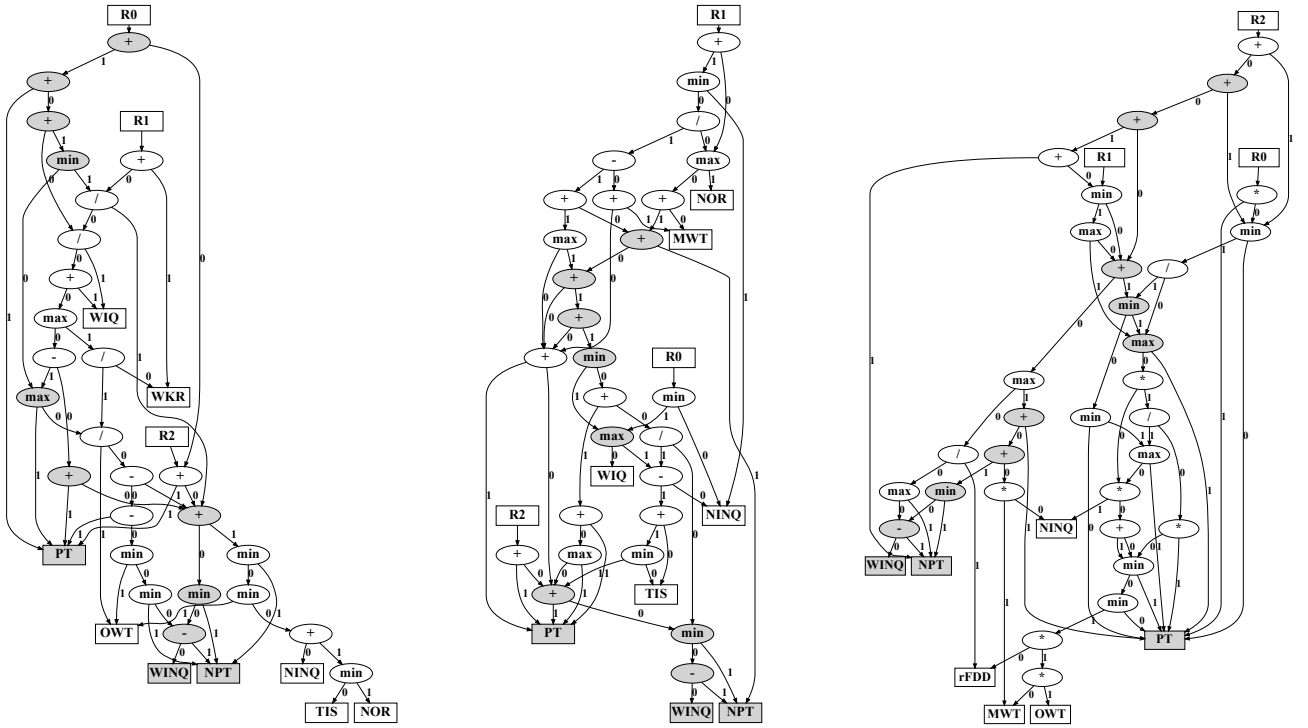
Fig. 8. The mean rate of knowledge transfer over generations of MLSI. X-axis: generations, Y-axis: the mean rate of mating with the generalist sub-population.

D. Example Program Analysis

To further analyze the behavior of MLSI, we investigate the obtained programs by MLSI. Specifically, we pick the final obtained programs for different tasks (i.e., individuals with the best training fitness for each task) from an independent run in Scenario A as examples, which are shown in Fig. 9. In this figure, ovals denote functions while rectangles denote terminals. The directed edges specify the inputs of functions. The numbers (i.e., 0 and 1) beside the edges respectively indicate the first or second argument for the function.

It can be seen that these outputted programs for the three similar tasks share some similarities. First, some common building blocks can be found in these outputted programs. For example, these three programs share a building block of "$\min(\mathrm{WINQ}-\mathrm{NPT},\mathrm{NPT})+\mathrm{PT}$". It implies that minimizing the processing time (PT) and the smaller value between the processing time of the next operation (i.e., NPT) and the remaining total processing time in the next corresponding machine buffer (i.e., WINQ-NPT), is a useful strategy to minimize mean flowtime. Second, they have a similar distribution of terminals. All of them utilize the processing time of operations and the next operation (i.e., PT and NPT), and the total processing time and the number of operations in the next corresponding machine buffer (i.e., WINQ and NINQ). Additionally, at least two of the three programs include a number of remaining operations (NOR), waiting time of an operation (OWT), and other machine-related terminals (e.g., WIQ and MWT) as primitives.

Despite the similarity, we can find that the outputted programs are specialized for different tasks respectively. For example, outputted programs for <Fmean,0.95> and <Fmean,0.85> mainly apply "PT" together with addition. Contrarily, the one for <Fmean,0.75> directly minimizes "PT" (i.e., $\min(\mathrm{PT},*)$, "*" denotes any input) in most cases. It shows that the three programs have different building blocks when using a terminal. Besides, the output register for specific tasks (e.g., R0 for <Fmean,0.95>) aggregates the results from most the graph nodes, standing for the most sophisticated program in this LGP individual. On the contrary, the other output registers only store the intermediate results from part of the computer program. It implies that these programs are



(a) <Fmean,0.95> (b) <Fmean,0.85> (c) <Fmean,0.75>

Fig. 9. The final outputted programs for different tasks from an independent run in Scenario A. The common building blocks are highlighted in dark nodes

specifically designed for a certain task. Simply seeing the whole individual from one task as the parents of another is likely to be ineffective.

By comparing the learnt heuristics from multitask tree-based GP and transfer learning methods for tree-based GP [63], [66], we find that the common building blocks in graph-based structures can be flexibly distributed in different parts of the graphs and can be reused multiple times without duplication, which is different from tree-based GP whose common building blocks appear in particular parts of a tree, and each can be typically used by a single part of tree only. To summarise, graph-based structures show a flexible and concise way of reusing shared knowledge among tasks.

VII. Conclusions

The main goal of this paper was to improve the knowledge transfer efficiency in multitask LGP methods. It has been successfully achieved by designing a new knowledge transfer mechanism that evolves shared individuals with multiple outputs, each for one task, based on the graph-based structure.

The experiment results show that the proposed method (i.e., MLSI) is significantly better than the baseline method and three state-of-the-art multitask GP methods. Further analyses verify that the adaptation ability of transfer rate based on the evolution process and the similarity among tasks is the essential reason for the superior performance. These results fully imply the great potential of graph-based structures in multitask optimization.

The proposed knowledge transfer mechanism enriches the methodologies in transferring knowledge, but also provides an effective example of designing graph-based knowledge transfer. The improvement can be easily extended to other domains such as classification and symbolic regression to which LGP has been applied. This paper is expected to evoke the research interest in LGP, which is a representative graph-based GP method but not well investigated compared to tree-based GP.

In future work, some characteristics of MLSI will be further investigated. For example, though MLSI can adjust the transfer rate based on the similarity of tasks in the course of evolution, MLSI only has a slightly better performance than fixed-rate variants. Fully utilizing this kind of ability of MLSI should be an important direction to further improve MLSI. Besides, the riffle shuffle merges LGP individuals randomly, which might be not effective enough. Developing some selective methods to identify valuable building blocks in the riffle shuffle is another potential direction.

References

[1] Y. S. Ong, "Towards Evolutionary Multitasking: A New Paradigm in Evolutionary Computation," in *Computational Intelligence, Cyber Security and Computational Models*, 2015, pp. 25–26.

[2] Q. Xu, N. Wang, L. Wang, W. Li, and Q. Sun, "Multi-task optimization and multi-task evolutionary computation in the past five years: A brief review," *Mathematics*, vol. 9, no. 8, pp. 1–44, 2021.

[3] A. Gupta, Y. S. Ong, and L. Feng, "Multifactorial Evolution: Toward Evolutionary Multitasking," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 3, pp. 343–357, 2016.

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

[5] J. Zhong, L. Feng, W. Cai, and Y. S. Ong, "Multifactorial Genetic Programming for Symbolic Regression Problems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, no. 11, pp. 4492–4505, 2020.

[6] Y. Bi, B. Xue, and M. Zhang, "Multitask Feature Learning as Multiobjective Optimization: A New Genetic Programming Approach to Image Classification," *IEEE Transactions on Cybernetics*, pp. 1–14, 2022, doi:10.1109/TCYB.2022.3174519.

[7] F. Zhang, Y. Mei, S. Nguyen, K. C. Tan, and M. Zhang, "Multitask Genetic Programming-Based Generative Hyperheuristics: A Case Study in Dynamic Scheduling," *IEEE Transactions on Cybernetics*, vol. 52, no. 10, pp. 10 515–10 528, 2021.

[8] M. Zhang and M. Johnston, "A Variant Program Structure in Tree-Based Genetic Programming for Multiclass Object Classification," in *Book chapter of Evolutionary Image Analysis and Signal Processing.Studies in Computational Intelligence.Vol.213*. Springer, 2009, ch. 3, pp. 55–72.

[9] P. Nordin, "A compiling genetic programming system that directly manipulates the machine code," *Advances in genetic programming*, vol. 1, pp. 311–331, 1994.

[10] P. Nordin, "Evolutionary program induction of binary machine code and its applications," Ph.D. dissertation, University of Dortmund, 1997.

[11] M. Brameier and W. Banzhaf, *Linear genetic programming*. Springer, Boston, MA, 2007.

[12] G. Wilson and W. Banzhaf, "A comparison of cartesian genetic programming and linear genetic programming," in *Proceedings of European Conference on Genetic Programming*, 2008, pp. 182–193.

[13] Z. Huang, F. Zhang, Y. Mei, and M. Zhang, "An Investigation of Multitask Linear Genetic Programming for Dynamic Job Shop Scheduling," in *Proceedings of European Conference on Genetic Programming*, 2022, pp. 162–178.

[14] L. Feng, L. Zhou, J. Zhong, A. Gupta, Y. S. Ong, K. C. Tan, and A. K. Qin, "Evolutionary Multitasking via Explicit Autoencoding," *IEEE Transactions on Cybernetics*, vol. 49, no. 9, pp. 3457–3470, 2019.

[15] T. Wei, S. Wang, J. Zhong, D. Liu, and J. Zhang, "A Review on Evolutionary Multi-Task Optimization: Trends and Challenges," *IEEE Transactions on Evolutionary Computation*, vol. 26, no. 5, pp. 941–960, 2021.

[16] K. K. Bali, A. Gupta, L. Feng, Y. S. Ong, and T. P. Siew, "Linearized domain adaptation in evolutionary multitasking," in *IEEE Congress on Evolutionary Computation*, 2017, pp. 1295–1302.

[17] J. Ding, C. Yang, Y. Jin, and T. Chai, "Generalized Multitasking for Evolutionary Optimization of Expensive Problems," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 1, pp. 44–58, 2019.

[18] X. Zheng, A. K. Qin, M. Gong, and D. Zhou, "Self-Regulated Evolutionary Multitask Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 1, pp. 16–28, 2020.

[19] L. Zhou, L. Feng, J. Zhong, Y. S. Ong, Z. Zhu, and E. Sha, "Evolutionary multitasking in combinatorial search spaces: A case study in capacitated vehicle routing problem," in *IEEE Symposium Series on Computational Intelligence*, 2016, pp. 1–8.

[20] J. Yi, J. Bai, H. He, W. Zhou, and L. Yao, "A Multifactorial Evolutionary Algorithm for Multitasking under Interval Uncertainties," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 5, pp. 908–922, 2020.

[21] A. Gupta, J. Mańdziuk, and Y.-S. Ong, "Evolutionary multitasking in bi-level optimization," *Complex & Intelligent Systems*, vol. 1, no. 1-4, pp. 83–95, 2015.

[22] A. Gupta, Y. S. Ong, L. Feng, and K. C. Tan, "Multiobjective Multifactorial Optimization in Evolutionary Multitasking," *IEEE Transactions on Cybernetics*, vol. 47, no. 7, pp. 1652–1665, 2017.

[23] X. Chen, Y. Huang, W. Zhou, and L. Feng, "Evolutionary Multitasking via Artificial Neural Networks," in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, 2021, pp. 1545–1552.

[24] Z. Tang, M. Gong, Y. Wu, W. Liu, and Y. Xie, "Regularized Evolutionary Multitask Optimization: Learning to Intertask Transfer in Aligned Subspace," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 2, pp. 262–276, 2021.

[25] Z. Chen, Y. Zhou, X. He, and J. Zhang, "Learning Task Relationships in Evolutionary Multitasking for Multiobjective Continuous Optimization," *IEEE Transactions on Cybernetics*, vol. 52, no. 6, pp. 5278–5289, 2020.

[26] Y. Chen, J. Zhong, L. Feng, and J. Zhang, "An Adaptive Archive-Based Evolutionary Framework for Many-Task Optimization," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 4, no. 3, pp. 369–384, 2020.

[27] J. Lin, H. L. Liu, B. Xue, M. Zhang, and F. Gu, "Multiobjective Multitasking Optimization Based on Incremental Learning," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 5, pp. 824–838, 2020.

[28] R. Chandra, Y. S. Ong, and C. K. Goh, "Co-evolutionary multi-task learning for dynamic time series prediction," *Applied Soft Computing Journal*, vol. 70, pp. 576–589, 2018.

[29] L. Feng, Y. Huang, L. Zhou, J. Zhong, A. Gupta, K. Tang, and K. C. Tan, "Explicit Evolutionary Multitasking for Combinatorial Optimization: A Case Study on Capacitated Vehicle Routing Problem," *IEEE Transactions on Cybernetics*, vol. 51, no. 6, pp. 3143–3156, 2021.

[30] C. Fogelberg, "Linear Genetic Programming for Multi-class Classification Problems," Ph.D. dissertation, Victoria University of Wellington, 2005.

[31] C. Downey, M. Zhang, and W. N. Browne, "New crossover operators in linear genetic programming for multiclass object classification," in *Proceedings of the Annual Genetic and Evolutionary Computation Conference*, 2010, pp. 885–892.

[32] C. Downey, M. Zhang, and J. Liu, "Parallel Linear Genetic Programming for multi-class classification," *Genetic Programming and Evolvable Machines*, vol. 13, no. 3, pp. 275–304, 2012.

[33] W. Kantschik and W. Banzhaf, "Linear-tree gp and its comparison with other gp structures," 2001, pp. 302–312.

[34] L. F. D. P. Sotto, P. Kaufmann, T. Atkinson, R. Kalkreuth, and M. P. Basgalupp, "A study on graph representations for genetic programming," 2020, pp. 931–939.

[35] L. F. D. P. Sotto, P. Kaufmann, T. Atkinson, R. Kalkreuth, and M. P. Basgalupp, "Graph representations in genetic programming," *Genetic Programming and Evolvable Machines*, vol. 22, pp. 607–636, 2021.

[36] L. F. D. P. Sotto and V. V. de Melo, "Studying bloat control and maintenance of effective code in linear genetic programming for symbolic regression," *Neurocomputing*, vol. 180, pp. 79–93, 2016.

[37] L. F. D. P. Sotto, F. Rothlauf, V. V. de Melo, and M. P. Basgalupp, "An Analysis of the Influence of Noneffective Instructions in Linear Genetic Programming," *Evolutionary Computation*, vol. 30, no. 1, pp. 51–74, 2022.

[38] M. Brameier and W. Banzhaf, "A comparison of linear genetic programming and neural networks in medical data mining," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 1, pp. 17–26, 2001.

[39] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming: An Introduction On the Automatic Evolution of Computer Programs and Its Applications*, San Francisco, California, 1998.

[40] W. Banzhaf, M. Brameier, M. Stautner, and K. Weinert, "Genetic Programming and Its Application in Machining Technology," *Advances in Computational Intelligence – Theory and Practice*, pp. 194–242, 2003.

[41] L. F. D. P. Sotto and V. V. de Melo, "A probabilistic linear genetic programming with stochastic context-free grammar for solving symbolic regression problems," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2017, pp. 1017–1024.

[42] Z. Huang, Y. Mei, and M. Zhang, "Investigation of Linear Genetic Programming for Dynamic Job Shop Scheduling," in *Proceedings of IEEE Symposium Series on Computational Intelligence*, 2021, pp. 1–8.

[43] T. Hu, J. L. Payne, W. Banzhaf, and J. H. Moore, "Robustness, evolvability, and accessibility in linear genetic programming," in *Proceedings of European Conference on Genetic Programming*, 2011, pp. 13–24.

[44] T. Hu, W. Banzhaf, and J. H. Moore, "Robustness and evolvability of recombination in linear genetic programming," in *Proceedings of European Conference on Genetic Programming*, 2013, pp. 97–108.

[45] T. Wei and J. Zhong, "A Preliminary Study of Knowledge Transfer in Multi-Classification Using Gene Expression Programming," *Frontiers in Neuroscience*, vol. 13, no. January, pp. 1–14, 2020.

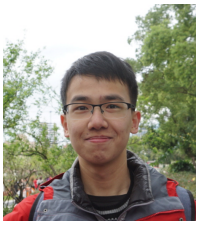
[46] Y. Bi, B. Xue, and M. Zhang, "Learning and Sharing: A Multitask Genetic Programming Approach to Image Feature Learning," *IEEE Transactions on Evolutionary Computation*, vol. 26, no. 2, pp. 218–232, 2022.

[47] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "A preliminary approach to evolutionary multitasking for dynamic flexible job shop scheduling via genetic programming," 2020, pp. 107–108.

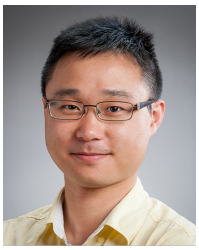
[48] F. Zhang, Y. Mei, S. Nguyen, M. Zhang, and K. C. Tan, "Surrogate-Assisted Evolutionary Multitasking Genetic Programming for Dynamic Flexible Job Shop Scheduling," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 4, pp. 651–665, 2021.

[49] T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J. M. Pierson, and A. V. Vasilakos, "Cloud computing: Survey on energy efficiency," *ACM Computing Surveys*, vol. 47, no. 2, pp. 1–36, 2015.

[50] F. Corman and E. Quaglietta, "Closing the loop in real-time railway control: Framework design and impacts on operations," *Transportation Research Part C: Emerging Technologies*, vol. 54, pp. 15–39, 2015.
[51] L. Lamorgese and C. Mannino, "A noncompact formulation for job-shop scheduling problems in traffic management," *Operations Research*, vol. 67, no. 6, pp. 1586–1609, 2019.
[52] F. Zhang, Y. Mei, S. Nguyen, K. C. Tan, and M. Zhang, "Task Relatedness Based Multitask Genetic Programming for Dynamic Flexible Job Shop Scheduling," *IEEE Transactions on Evolutionary Computation*, pp. 1–15, 2022, doi:10.1109/TEVC.2022.3199783.
[53] J. Branke, S. Nguyen, C. W. Pickardt, and M. Zhang, "Automated Design of Production Scheduling Heuristics: A Review," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 1, pp. 110–124, 2016.
[54] J. Park, Y. Mei, S. Nguyen, G. Chen, and M. Zhang, "Investigating a machine breakdown genetic programming approach for dynamic job shop scheduling," in *Proceedings of European Conference on Genetic Programming*, 2018, pp. 253–270.
[55] A. D'Ariano, D. Pacciarelli, M. Pistelli, and M. Pranzo, "Real-time scheduling of aircraft arrivals and departures in a terminal maneuvering area," *Networks*, vol. 65, no. 3, pp. 212–227, 2015.
[56] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Correlation Coefficient-Based Recombinative Guidance for Genetic Programming Hyperheuristics in Dynamic Flexible Job Shop Scheduling," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 3, pp. 552–566, 2021.
[57] M. Pfund, J. W. Fowler, A. Gadkari, and Y. Chen, "Scheduling jobs on parallel machines with setup times and ready times," *Computers and Industrial Engineering*, vol. 54, no. 4, pp. 764–782, 2008.
[58] W. E. Smith, "Various optimizers for single-stage production," *Naval Research Logistics Quarterly*, vol. 3, pp. 59–66, 1956.
[59] A. P. Vepsalainen and T. E. Morton, "Priority Rules for Job Shops With Weighted Tardiness Costs." *Management Science*, vol. 33, no. 8, pp. 1035–1047, 1987.
[60] J. R. Jackson, "Scheduling a production line to minimize maximum tardiness," *Management Science Research Project*, 1955.
[61] S. Nguyen, Y. Mei, and M. Zhang, "Genetic programming for production scheduling: a survey with a unified framework," *Complex & Intelligent Systems*, vol. 3, no. 1, pp. 41–66, 2017.
[62] F. Zhang, S. Nguyen, Y. Mei, and M. Zhang, "Genetic Programming for Production Scheduling: An Evolutionary Learning Approach," in *Machine Learning: Foundations, Methodologies, and Applications*, ser. Machine Learning: Foundations, Methodologies, and Applications. Singapore: Springer, 2021, pp. XXXIII+338.
[63] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Multitask Multi-objective Genetic Programming for Automated Scheduling Heuristic Learning in Dynamic Flexible Job Shop Scheduling," *IEEE Transactions on Cybernetics*, 2022, doi:10.1109/TCYB.2022.3196887.
[64] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A Classification of Hyper-Heuristic Approaches," in *Handbook of Metaheuristics*, ser. International Series in Operations Research & Management Science, 2010, pp. 449–468.
[65] M. Liu, W. U. Cheng, and Y. J. Yang, "Genetic Algorithm Method Based on Combinatorial Rules in Identaical Parallel Machine Scheduling Problem," *ACTA ELECTRONICA SINICA*, vol. 28, no. 5, pp. 52–54, 2000.
[66] D. O'Neill, H. Al-Sahaf, B. Xue, and M. Zhang, "Common subtrees in related problems: A novel transfer learning approach for genetic programming," in *Proceedings of IEEE Congress on Evolutionary Computation*, 2017, pp. 1287–1294.



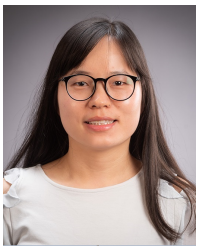
Zhixing Huang received the B.S. and M.S. degrees from the School of Computer Science and Engineering, South China University of Technology, China, in 2018 and 2020 respectively. He is currently pursuing the Ph.D. degree with the School of Engineering and Computer Science, Victoria University of Wellington, New Zealand. His research interests include evolutionary computation (e.g., genetic programming and its applications), combinatorial optimization, and program synthesis.



Yi Mei received the BSc and PhD degrees from the University of Science and Technology of China, Hefei, China, in 2005 and 2010, respectively.

He is an Associate Professor at the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. His research interests include evolutionary computation and machine learning for combinatorial optimisation, genetic programming, hyper-heuristics, and explainable AI. He has over 180 fully referred publications, including the top journals in EC and Operations Research such as IEEE TEVC, IEEE TCYB, Evolutionary Computation Journal, European Journal of Operational Research, ACM Transactions on Mathematical Software.

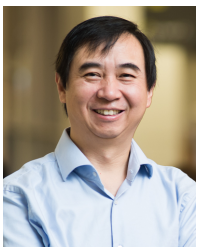
He is an Associate Editor of IEEE Transactions on Evolutionary Computation, and a guest editor of the Genetic Programming Evolvable Machine journal. He is the Founding Chair of IEEE Taskforce on Evolutionary Scheduling and Combinatorial Optimisation. He serves as a Vice-Chair of the IEEE CIS Emergent Technologies Technical Committee, and a member of Intelligent Systems Applications Technical Committee. He is a reviewer of over 50 international journals. He is a Fellow of Engineering New Zealand and an IEEE Senior Member.



Fangfang Zhang received the B.Sc. and M.Sc. degrees from Shenzhen University, China, and the PhD degree in Computer Science from Victoria University of Wellington, New Zealand, in 2014, 2017, and 2021, respectively.

She is currently a research fellow in computer science with the School of Engineering and Computer Science, Victoria University of Wellington, New Zealand. She has more than 45 publications in refereed international journals and conferences. Her research interests include evolutionary computation, hyper-heuristic learning/optimisation, job shop scheduling, surrogate, and multitask learning.

Dr Fangfang is an Associate Editor of Expert Systems With Applications. She is a Vice-Chair of the Task Force on Evolutionary Scheduling and Combinatorial Optimisation. She is a member of the IEEE Computational Intelligence Society and Association for Computing Machinery, and has been severing as reviewers for top international journals such as the IEEE Transactions on Evolutionary Computation and the IEEE Transactions on Cybernetics, and conferences including the Genetic and Evolutionary Computation Conference and the IEEE Congress on Evolutionary Computation. She is the Secretary of IEEE New Zealand Central Section, and was the Chair of the IEEE Student Branch at Victoria University of New Zealand and the chair of Professional Activities Coordinator.



Mengjie Zhang received the B.E. and M.E. degrees from Artificial Intelligence Research Center, Agricultural University of Hebei, Baoding, China, and the Ph.D. degree in computer science from RMIT University, Melbourne, VIC, Australia, in 1989, 1992, and 2000, respectively.

He is currently a Professor of Computer Science, the Head of the Evolutionary Computation and Machine Learning Research Group, and the Associate Dean (Research and Innovation) with the Faculty of Engineering, Victoria University of Wellington, New Zealand. His current research interests include genetic programming, image analysis, feature selection and reduction, job-shop scheduling, and evolutionary deep learning and transfer learning. He has published over 800 research papers in refereed international journals and conferences. He is a Fellow of the Royal Society of New Zealand, a Fellow of Engineering New Zealand, a Fellow of IEEE, and an IEEE Distinguished Lecturer.