An Investigation of Multitask Linear Genetic Programming for Dynamic Job Shop Scheduling

Zhixing Huang, Fangfang Zhang(✉), Yi Mei, and Mengjie Zhang

School of Engineering and Computer Science, Victoria University of Wellington, PO Box 600, Wellington 6140, New Zealand
{zhixing.huang,fangfang.zhang,yi.mei,mengjie.zhang}@ecs.vuw.ac.nz

Abstract. Dynamic job shop scheduling has a wide range of applications in reality such as order picking in warehouse. Using genetic programming to design scheduling heuristics for dynamic job shop scheduling problems becomes increasingly common. In recent years, multitask genetic programming-based hyper-heuristic methods have been developed to solve similar dynamic scheduling problem scenarios simultaneously. However, all of the existing studies focus on the tree-based genetic programming. In this paper, we investigate the use of linear genetic programming, which has some advantages over tree-based genetic programming in designing multitask methods, such as building block reusing. Specifically, this paper makes a preliminary investigation on several issues of multitask linear genetic programming. The experiments show that the linear genetic programming within multitask frameworks have a significantly better performance than solving tasks separately, by sharing useful building blocks.

Keywords: Multitask · Linear genetic programming · Hyper-heuristic · Dynamic job shop scheduling

1 Introduction

Job shop scheduling (JSS) is a typical combinatorial optimization problem and has a large commercial value in manufacturing systems. There are a set of machines and a set of jobs in the job shop. The job shop processes the set of jobs by the given machines so that some objectives, such as tardiness and makespan, are optimized. For dynamic job shop scheduling (DJSS), there are some dynamic events such as new job arrivals, which need to be considered when making schedules. In DJSS with new job arrivals, the information of the new jobs is not known in advance. Such characteristic requires optimization techniques to be able to make an instant reaction (e.g., re-scheduling or repairing existing schedules) to the newly arrived jobs. It also limits the application of some existing exact optimization algorithms such as branch-and-bound and dynamic programming whose computation burden may be too large for the instant reaction.

E. Medvet et al. (Eds.): EuroGP 2022, LNCS 13223, pp. 162–178, 2022.
https://doi.org/10.1007/978-3-031-02056-8_11

Hyper-heuristic methods (HH) have been successfully applied to many applications [18]. They try to search a suitable scheduling heuristic for a certain problem by selecting or recombining some existing scheduling heuristics [6]. Different from heuristic methods whose search space consists of solutions (i.e., complete schedules), HH methods search in a heuristic space given by users. Specifically, the heuristics in JSS are also known as dispatching rules. It has been shown that HH methods can obtain more sophisticated and effective priority dispatching rules than human designed ones in DJSS [1,5,28]. Genetic programming-based hyper heuristic (GPHH) is one of the most popular branches of HH methods [5,14]. Specifically, for GPHH, scheduling heuristics are encoded into genetic programming (GP) individuals. These heuristics will be modified by genetic operators and evaluated on problem instances. The performance of heuristics on the problem instances will be regarded as the fitness of those heuristics. The quality of scheduling heuristics are improved generation by generation.

Although there have been many advanced techniques to assist GPHH to find more effective heuristics in solving a certain DJSS problem [15,24–26], it is a tedious and expensive task for GPHH methods to search effective heuristics for each single scenario. In recent years, some researchers found that sharing knowledge among different scenarios is a potential research direction to enhance GPHH in solving different DJSS scenarios [16,22]. Given that many DJSS scenarios share similarities in objective functions or job shop environments, they may require the same building blocks (e.g., subtrees in tree-based GP) to form an effective scheduling heuristic.

Evolutionary multitask optimization is an emerging topic of evolutionary computation area, which aims to fully utilize the search information among different tasks [20]. The evolutionary multitask methods will accept more than one optimization tasks and solve them simultaneously by an evolutionary computation method within a unified search space. The evolutionary multitask optimization has a wide spectrum of applications nowadays, such as vehicle routing [2,29], time series prediction [7,11], and robot path planning [21]. The evolutionary multitask techniques in these applications are validated to be more effective than solving the problems separately.

To adapt multitask techniques to GPHH methods for dynamic scheduling, several GPHH-specific multitask techniques are proposed. For example, Park et al. [16] proposed a niched GP to improve the generalization ability with different machine breakdown levels (i.e., different optimization tasks). Specifically, the term "niched" means the GP method has a light-weight grouping mechanism. Every group in niched GP is assigned to a specific task. The breeding and selection of GP population were also designed based on these groups. Besides, Zhang et al. respectively made an investigation on multitask GPHH methods [23] and proposed a multitask multi-population GPHH method to design dispatching rules for dynamic flexible JSS [22]. To improve the training efficiency and precisely share the search information, Zhang et al. [27] further proposed a surrogate-assisted multitask GPHH method. Their designed

surrogate model successfully enhances the GPHH in terms of test performance and convergence speed.

However, most of existing studies of multitask GPHH are designed based on tree-based GP [13]. Since the tree-like structures usually only have one output, GP individuals have to be assigned to a specific task in the off-the-shelf evolutionary multitask optimization frameworks. On the other hand, linear genetic programming (LGP) [4], which can reuse useful building blocks easily, may be more suitable for multitask optimization. Though LGP has some advantages over tree-based GP in designing multitask frameworks, none of the existing studies apply LGP to multitask optimization frameworks.

To consolidate the foundation of applying LGP to multitask framework, this paper serves as a preliminary work to investigate the performance and behaviour of linear genetic programming with multitask optimization frameworks. Basically, this paper makes an investigation about several key issues of developing multitask linear genetic programming-based hyper heuristic (LGPHH). These issues are summarised into the following three research questions:

- Which existing multitask framework (e.g., multifactorial evolutionary framework and multitask multi-population framework) is most suitable with LGPHH?
- How effective is LGPHH compared with the existing tree-based GPHH for multitask DJSS?
- How does multitask LGPHH share information among the individuals for different tasks?

The rest of the paper is organised as follows. Section 2 gives the introduction to DJSS, LGP, and two existing multitask frameworks. Section 3 develops two LGPHH-based multitask methods for solving DJSS problems. The experiment settings and the result analysis are respectively introduced in Sect. 4 and 5. Finally, Sect. 6 draws out some conclusions.

2 Background

2.1 Dynamic Job Shop Scheduling

DJSS with new job arrival start with an empty job shop whose set of machines M are given beforehand. The jobs will come into the job shop to form the job set J over time. Their information cannot be known until their arrival. Every job j has a sequence of operations $(o_{j1}, ..., o_{ji}, ..., o_{jl_j})(1 \leq i \leq l_j)$, an arrival time α_j, a due date d_j, and a weight ω_j. The operation o_{ji} will be processed by a certain machine $\pi(o_{ji}) \in M$ with a processing time $\delta(o_{ji})$. A machine can only process one operation at any time and its process is assumed to be uninterruptable. The operation sequence of job j specifies the process order of those operations. $o_{ji+1}(i+1 \leq l_j)$ can only be processed after o_{ji} is completed.

This paper mainly considers the flowtime and the tardiness as the performance metrics of the job shop. To formulate these metrics, we denote the actual

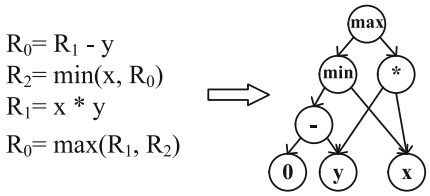


Fig. 1. An example program of LGP

starting time of an operation o_{ji} as $x(o_{ji})$ and denote the finishing time of job j as c_j, where $c_j = x(o_{jl_j}) + \delta(o_{jl_j})$. The flowtime and the tardiness are further specified as maximum flowtime (F_{max}), mean flowtime (F_{mean}), mean weighted flowtime (WF_{mean}), maximum tardiness (T_{max}), mean tardiness (T_{mean}), and mean weighted tardiness (WT_{mean}). They are formulated as below.

- $F_{max} = \max_{j \in J}(c_j - x(o_{j1}))$
- $F_{mean} = \frac{\sum_{j \in J}(c_j - x(o_{j1}))}{|J|}$
- $WF_{mean} = \frac{\sum_{j \in J}(c_j - x(o_{j1})) \cdot \omega_j}{|J|}$
- $T_{max} = \max_{j \in J}(\max(c_j - d_j, 0))$
- $T_{mean} = \frac{\sum_{j \in J}(\max(c_j - d_j, 0))}{|J|}$
- $WT_{mean} = \frac{\sum_{j \in J}(\max(c_j - d_j, 0) \cdot \omega_j)}{|J|}$

2.2 Linear Genetic Programming

LGP [4] is a GP variant which has been successfully applied to classification [3, 9, 12, 17] and symbolic regression problems [8, 19]. LGP individuals are sequences of register-based instructions. The instructions in a same sequence are executed sequentially to form a completed computer program. The linear arrangement of instructions and the sequential execution are two core meaning of the term "linear". For every single instruction in LGP, it contains three parts: source register, operation, and destination register. The values in the source registers serve as the inputs of the operation. The output from the operation is assigned to the destination register and passed to the subsequent instructions. To output the final result, at least one output register is needed for LGP. By default, the first register is regarded as the output register. Figure 1 is a simple example of LGP individual to represent a mathematical formula "$\max(xy, \min(x, -y))$". R_0, R_1 and R_2 are three registers. They serve as both source and destination registers. These registers are initialized by a certain value, such as zero in this example. The LGP individual can also be transformed into a directed acyclic graph (DAG) to be more compact.

The evolutionary framework of LGP is quite similar with the one of standard GP. But because of the different representation, LGP has two kinds of different genetic operators from standard GP [13]. The first type of genetic operators is

macro variation. The term "macro" means that this kind of genetic operators produce offspring mainly by affecting the total number of instructions. The other type of genetic operators is micro variation. Contrarily, micro variation does not change the total number of instructions, but only changes the primitives inside instructions to produce offspring.

2.3 Related Work

In the literatures, there are two popular evolutionary multitask frameworks for existing GPHH methods. One is multifactorial evolutionary algorithm (MFEA) and the other is multitask multi-population GPHH (M^2GP).

Multifactorial Evolutionary Algorithm. MFEA was firstly proposed by Gupta et al. [10]. The main idea of MFEA is to use an evolutionary algorithm with a single population of individuals to solve different optimization tasks simultaneously. All of these individuals are encoded into a unified search space and can be transformed into a problem-specific representation to solve different tasks. To evolve the individuals for different tasks simultaneously, MFEA introduces four key properties, i.e., factorial cost, factorial rank, scalar fitness, and skill factor. Based on these properties, the effectiveness of an individual in solving a certain task can be represented by the factorial cost and rank. The individuals good at solving different tasks can be identified by different skill factors. The individuals with different skill factors can also make a fair comparison together based on the scalar fitness. To enable individuals to share the information among different tasks, Gupta et al. developed an assortative mating algorithm which allows individuals with different skill factors to perform crossover with a predefined probability. A vertical cultural transmission is also developed together with the assortative mating to propagate the skill factor from parent individuals to offspring.

Multitask Multi-population GPHH. M^2GP is proposed by Zhang et al. [22], which is a GP-specific multitask optimization framework. Specifically, M^2GP splits a GP population into several sub-populations, each for a single task. Every sub-population evolves GP individuals with the conventional evolutionary framework of GP. GP individuals in M^2GP are trained on different problem instances in different generations. GP individuals in M^2GP share their knowledge by swapping sub-trees across different sub-populations. Different from MFEA in which individuals may change their skill factor (i.e., corresponding task) by imitating different parents, GP individuals in M^2GP only evolve for a certain task and will not migrate to the other sub-populations. To improve the efficiency of multitask learning, M^2GP further proposes an origin-based offspring reservation strategy, which only keeps the offspring generated based on the parent from the corresponding sub-population and discards the other offspring in crossover. The empirical results show that M^2GP has a better performance than MFEA in solving dynamic flexible JSS problems.

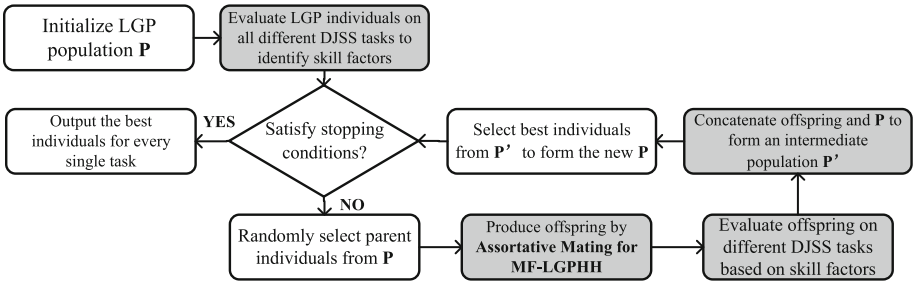


Fig. 2. Flowchart of MF-LGPHH

3 Multitask LGPHH

In this paper, we aim to make a preliminary investigation on the performance and the behaviour of LGP with off-the-shelf GPHH-specific multitask optimization techniques. Specifically, two multitask LGPHH methods are developed based on MFEA and M^{2}GP respectively.

3.1 Multi-factorial LGPHH

Multi-factorial LGPHH (MF-LGP) is developed based on MFEA. The evolutionary framework of MF-LGP is shown in Fig. 2 where the dark boxes are the key differences from basic LGPHH methods. Initially, LGP individuals are randomly generated and evaluated on all different DJSS tasks to identify their skill factors. The minimum ranking among different tasks is regarded as the fitness of LGP individuals (i.e., scalar fitness). In every generation, parent individuals are randomly selected from the population. The offspring are generated by a newly developed LGP-specific assortative mating algorithm. This algorithm generates offspring by mutating instructions or exchanging instruction segments of parent individuals based on the skill factors of parents and a predefined random mating probability (rmp). The skill factors of offspring are also updated by inheriting from one of the parents. To improve the training efficiency, every LGP individual will be evaluated on only one corresponding DJSS scenario, which is specified by the skill factor. The performance of LGP heuristic is regarded as the fitness of individuals. Then, both of parent and offspring individuals are concatenated into an intermediate population. The best individuals of the intermediate population will be selected greedily and form the new population in next generation. To ensure the fitness of LGP individuals are comparable, all LGP individuals are evaluated on a same DJSS problem instance.

To be more specific, the newly developed LGP-specific assortative mating (shown in Algorithm 1) is designed based on [10] and the three basic genetic operators of LGP (i.e., crossover, macro mutation, and micro mutation). The algorithm accepts two randomly selected parents from the population. If the two parent individuals have the same skill factor or a randomly generated number is

Algorithm 1: Assortative Mating for MF-LGP

Input: Two selected parents p_a and p_b, crossover rate r_c, and a random mating probability rmp.
Output: Two offspring c_a and c_b.

```
Generate two random numbers rand1 and rand2 between 0 and 1;
// crossover
if (p_a and p_b have same skill factor) or (rand1 < r_c) then
    Perform LGP crossover on p_a and p_b to produce two offspring c_a and c_b;
    Perform vertical cultural transmission on the skill factors of c_a and c_b;
// macro mutation
else if rand2 < 0.5 then
    Apply LGP macro mutation on p_a and p_b respectively to produce offspring
     c_a and c_b;
// micro mutation
else
    Apply LGP micro mutation on p_a and p_b respectively to produce offspring
     c_a and c_b;
Return c_a and c_b.
```

smaller than the random mating probability, the crossover is performed on the parent individuals to produce offspring. The skill factors of offspring are updated based on the vertical cultural transmission proposed in [10]. If the two parents have different skill factors and the random mating probability is not satisfied, macro and micro mutation will be performed. In this algorithm, macro and micro mutation have the same probability in producing offspring. Since macro and micro mutation only accept one parent individual each time, the skill factor of the generated offspring is the same as that of the parent in mutation.

However, MF-LGP only uses one DJSS problem instance during evolution, and the performance of MF-LGP may be limited by the insufficient training instances. To have a comprehensive investigation, a MFEA-based LGPHH with GP selection paradigm is also developed. It replaces the selection paradigm of conventional MFEA into the one of standard GP, which applies tournament selection to select parents and replaces the old population by offspring. Since there is no concatenation of parents and offspring, the problem instances can be rotated every generation. This variant of MFEA-based LGPHH is denoted as MF-LGP$_{rotate}$.

3.2 Multitask Multi-population LGPHH

In this paper, we extend M^{2}GP to LGP to develop a new algorithm called M^{2}LGP. The flowchart of M^{2}LGP is shown in Fig. 3. The dark boxes in Fig. 3 also highlight key differences from basic LGP. Basically, it firstly initializes multiple populations of LGP individuals randomly. The individuals in a sub-population are only evaluated on a certain DJSS scenario. When the stopping conditions are not satisfied, M^{2}LGP reproduces offspring by origin-based offspring reservation

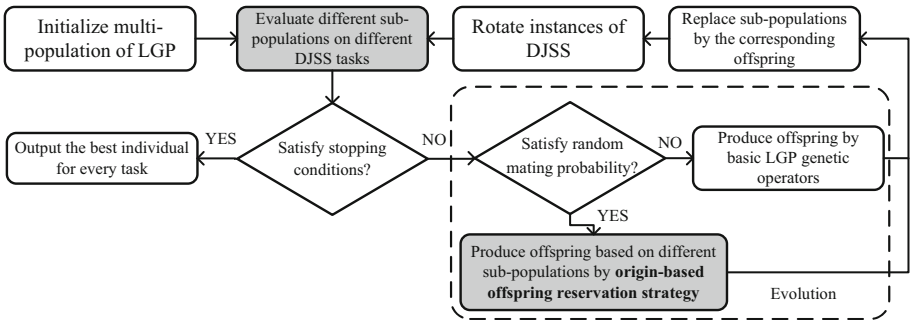


Fig. 3. Flowchart of M^2LGP

strategy or basic LGP genetic operators based on a probability specified by a random mating probability. The sub-populations are then replaced by the offspring. The DJSS problem instance is rotated every generation. Finally, the best LGP individuals in the different sub-populations are outputted to test data.

Specifically, the evolution of M^2LGP produces offspring by three basic kinds of LGP genetic operators. The pseudo code of the evolution is shown in Algorithm 2. To share the search information, the origin-based offspring reservation strategy proposed by Zhang et al. is also extended to LGP crossover. When the random number is smaller than *rmp*, LGP selects two parent individuals from different sub-populations. Then, LGP exchanges the instruction segments of the parent individuals and only retains the offspring from the parent individual of the corresponding sub-population. The old sub-population will be replaced by the newly generated offspring population.

4 Experiment Design

4.1 Multitask DJSS Scenarios

Based on the categorization in [22], there are two types of multitask settings in DJSS, i.e., heterogeneous and homogeneous multitask optimization. Specifically, heterogeneous multitask problems contain a set of DJSS problems whose optimization objectives are different but having a same utilization level. On the contrary, the DJSS problems in homogeneous multitask problems have a same optimization objective but different utilization levels. We develop three scenarios for each type of multitask settings. These six scenarios are listed in Table 1. The notation "$<x, y>$" denotes a task whose optimization objective is x and utilization level is y.

There are 10 machines for every problem instance. The new arrival jobs have a sequence of operations whose length ranges from 2 to 10. The processing time of operations is a continuous value from 1 to 99. Every job has a weight. Specifically, 20%, 20%, and 60% of the new jobs have a weight of 1, 4, and 2

Algorithm 2: Evolution of M^{2}LGP

Input: The population of the current generation *pop*, crossover rate r_c, macro mutation rate r_{macro}, micro mutation rate r_{micro}, and a random mating probability *rmp*.

Output: An updated LGP population.

for *all sub-population s in pop* **do**
  Initialize an empty sub-population s_n;
  Load elite individuals from s to s_n by an elitism selection;
  while *size of* s_n < *size of* s **do**
    Generate a random number $rand1$ between 0 and 1;
    Use tournament selection to select a parent individual p_a from s;
    `// crossover`
    if $rand1 < r_c$ **then**
      Generate a random number $rand2$ between 0 and 1;
      if $rand2 < rmp$ **then**
        Select p_b from another sub-population $s'(s' \neq s)$ by tournament selection;
        `// origin-based offspring reservation`
        Swap an instruction segment from p_b to p_a to produce offspring c_a;
        $s_n = s_n \bigcup \{c_a\}$;
      else
        Select p_b from s by tournament selection;
        Perform LGP crossover on p_a and p_b to produce two offspring c_a and c_b;
        $s_n = s_n \bigcup \{c_a, c_b\}$;
    `// macro mutation`
    else if $rand1 < r_c + r_{macro}$ **then**
      Apply LGP macro mutation on p_a to produce offspring c_a;
      $s_n = s_n \bigcup \{c_a\}$;
    `// micro mutation`
    else if $rand1 < r_c + r_{macro} + r_{micro}$ **then**
      Apply LGP micro mutation on p_a to produce offspring c_a;
      $s_n = s_n \bigcup \{c_a\}$;
    else
      reproduce p_a to s_n.
  $s = s_n$;
Return *pop*.

Table 1. Problem settings of the multitask scenarios

Scenarios	task1	task2	task3
homogeneous multitask			
homoFmean	$<F_{mean}, 0.95>$	$<F_{mean}, 0.85>$	$<F_{mean}, 0.75>$
homoTmean	$<T_{mean}, 0.95>$	$<T_{mean}, 0.85>$	$<T_{mean}, 0.75>$
homoWTmean	$<WT_{mean}, 0.95>$	$<WT_{mean}, 0.85>$	$<WT_{mean}, 0.75>$
heterogeneous multitask			
heteFTMax	$<F_{max}, 0.95>$	$<T_{max}, 0.95>$	
heteFTMean	$<F_{mean}, 0.95>$	$<T_{mean}, 0.95>$	
heteWFTMean	$<WF_{mean}, 0.95>$	$<WT_{mean}, 0.95>$	

Table 2. The terminal set

Notation	Description
NIQ	the number of operations in the queue of a machine
WIQ	the total processing time of operations in the queue of a machine
MWT	the waiting time of the machine
PT	the processing time of the operation
NPT	the processing time of the next operation
OWT	the waiting time of the operation
NWT	the waiting time of the next to-be-ready machine
WKR	the total remaining processing time of the job
NOR	the number of remaining operations of the job
WINQ	total processing time of operations in the queue of the machine which specializes in the next operation of the job
NINQ	number of operations in the queue of the machine which specializes in the next operation of the job
rFDD	the difference between the expected due date of the operation and the system time
rDD	the difference between the expected due date of the job and the system time
W	the weight of the job
TIS	the difference between system time and the arrival time of the job
SL	the difference between the expected due date and the sum of the system time and WKR

respectively. During the simulation, the first 1000 jobs will be regarded as warm-up jobs to ensure that heuristics are evaluated in a steady state of job shops. The performance of a heuristic is evaluated by the subsequent 5000 jobs. For every scenario, 30 independent runs with different random seeds are carried out, and each output heuristic is tested on 50 unseen DJSS instances.

4.2 Comparison Methods

To investigate the three research questions, two comparison methods are developed. Firstly, a baseline method of LGPHH is adopted. It simply runs the LGPHH for each single task separately. In other words, there is no knowledge transfer among the tasks. LGPHH serves as a baseline for other multitask optimization techniques. Secondly, the state-of-the-art GPHH-specific multitask method, M^{2}GP, is adopted. M^{2}GP is based on tree-based GP.

Table 3. The mean (and standard deviation) of test performance in all multitask scenarios

Scenario	Task	LGPHH	M^{2}GP	M^{2}LGP	MF-LGP	MF-LGP$_{rotate}$
homo Fmean	$<F_{mean}, 0.95>$	1584.4(21.0)	**1569.3(10.5)**	1577.0(12.8)	1609.8(26.7)	1580.6(13.2)
	$<F_{mean}, 0.85>$	870.7(6.8)	**861.5(2.7)**	863.8(3.5)	874.4(6.1)	867.2(5.6)
	$<F_{mean}, 0.75>$	658.7(1.7)	**654.8(1.3)**	**655.0(1.6)**	656.9(2.2)	655.8(1.7)
homo Tmean	$<T_{mean}, 0.95>$	1129.3(8.7)	**1125.1(15.1)**	1125.3(11.9)	1165.3(26.9)	1134.3(13.8)
	$<T_{mean}, 0.85>$	427.1(5.9)	**415.9(2.1)**	417.1(2.9)	429.4(7.0)	419.2(4.2)
	$<T_{mean}, 0.75>$	218.9(1.7)	**215.0(1.1)**	**215.2(1.0)**	217.9(2.1)	**215.7(1.3)**
homo WTmean	$<WT_{mean}, 0.95>$	1817.0(29.4)	**1771.6(27.7)**	1804.9(27.3)	1839.2(24.9)	1787.5(26.3)
	$<WT_{mean}, 0.85>$	731.1(9.1)	**724.5(4.8)**	731.2(4.6)	745.5(12.0)	728.5(6.4)
	$<WT_{mean}, 0.75>$	394.3(2.5)	**391.5(1.6)**	393.8(2.0)	400.2(4.1)	**392.6(2.2)**
heteFT Max	$<F_{max}, 0.95>$	**4470.5(108.0)**	4551.7(173.5)	**4450.1(114.2)**	4794.1(203.6)	**4461.0(96.0)**
	$<T_{max}, 0.95>$	3945.2(117.3)	3991.2(117.1)	3878.4(90.5)	4272.7(170.8)	**3849.6(82.3)**
heteFT Mean	$<F_{mean}, 0.95>$	1579.8(10.6)	**1570.2(8.4)**	**1570.1(11.7)**	1620.1(31.0)	**1572.2(15.0)**
	$<T_{mean}, 0.95>$	1132.9(23.0)	**1119.1(9.7)**	**1121.8(13.5)**	1167.0(28.1)	**1122.1(14.0)**
heteW FTMean	$<WF_{mean}, 0.95>$	2793.7(31.7)	**2763.7(23.0)**	2783.0(34.6)	2813.9(26.0)	2781.5(35.6)
	$<WT_{mean}, 0.95>$	1803.4(30.2)	**1771.4(25.6)**	1799.4(35.4)	1823.1(26.5)	1792.9(33.8)
Average rank		3.40	2.09	2.51	4.42	2.57

bold font: a method is significantly better than most of other methods on a certain task.
M^{2}LGP: Multitask Multi-population LGPHH; MF-LGP: Multi-factorial LGPHH;
MF-LGP$_{rotate}$: Multi-factorial LGPHH with training instance rotation.

The parameters of these methods are designed based on [22]. Basically, the population size and the number of sub-populations vary with the number of tasks. When there are k tasks, there will be k sub-populations for the methods based on M^{2}GP, and there will be totally $k \times 200$ individuals for all LGP methods ($k \times 400$ individuals for tree-based GP methods). The total number of generations is 102 for LGP methods and 51 for tree-based GP methods. Every LGP individual has a maximum of 50 instructions. Each manipulates four available registers. The first register is regarded as the output register. To facilitate the knowledge transfer among tasks, LGP adopts a kind of effective macro mutation and a linear crossover to produce offspring. Specifically, the effective macro mutation inserts (or removes) an effective instruction into (from) heuristics, and will remove all ineffective instructions after mutation. The maximum crossover length and the maximum length difference of the linear crossover are both 30. Specifically, the crossover, macro and micro mutation rate of M^{2}LGP are 60%, 10%, and 25% respectively. The hyper parameters of tree-based GP are set the same as the ones of [22]. A terminal set including sixteen terminals is designed for the GP methods, which is shown in Table 2.

5 Results and Discussion

5.1 Test Performance

To analyze the effectiveness of multitask LGPHH, the test performance of the five methods are compared, as shown in Table 3. A Friedman test with a significance level of 0.05 is also applied to the test performance analysis. The average rank below the table shows the overall ranking of all algorithms based on the Friedman test. The p-value of the Friedman test on the test performance is 1.3e−08, which means there is a significant difference among all of these algorithms. Therefore, a pairwise Wilcoxon test with false discovery rate correction (by the Benjamini and Hochberg method) and a significance level of 0.05 is further applied to every pair of these methods. The bold results in the table highlight the methods which are significantly better than most of other methods.

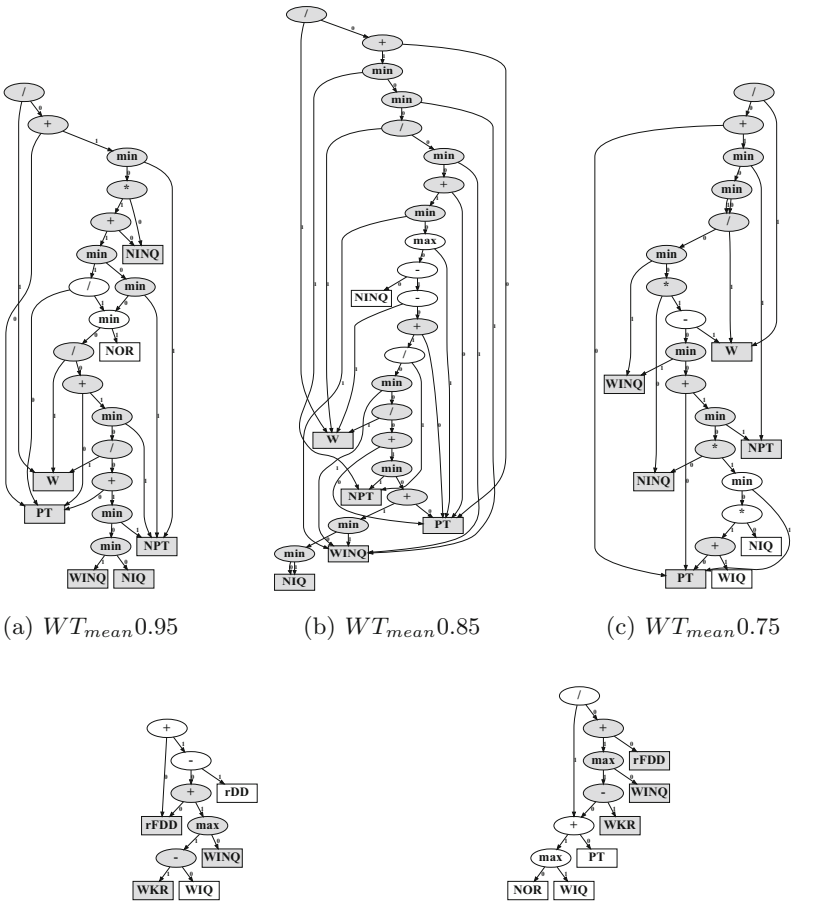
To answer the first research questions, i.e., which existing multitask framework is suitable to LGPHH, the baseline LGPHH, M^2LGP and the two MFEA-based LGPHH are compared. Generally speaking, M^2LGP and MF-LGP$_{rotate}$ have a quite competitive performance with each other. They show a similar average ranking about 2.5. They also have a significantly better performance than the baseline method in most of the scenarios based on the Wilcoxon test. However, the simple combination of LGPHH and MFEA does not work very well in all these scenarios. Its average rank is 4.42 among the five algorithms. Given that the optimization problems in the paper are minimization problems, MF-LGP has the worst performance in most cases. The results of the three multitask LGPHH methods imply that simply replacing LGPHH into the existing multitask frameworks is not always a good way. It is likely for multitask LGPHH to work poorly, especially when there are some unsuitable designs for LGPHH in multitask frameworks. Based on the results, it is advisable for LGPHH to be adopted in M^2LGP and MF-LGP$_{rotate}$ which enable LGPHH to have sufficient training instances by GP selection methods.

To investigate the effectiveness of multitask LGPHH compared with tree-based GPHH, M^2GP, M^2LGP, and MF-LGP$_{rotate}$ are further compared. Basically, M^2LGP and MF-LGP$_{rotate}$ are less effective than the state-of-the-art method, i.e., M^2GP. The results of the average rank show that M^2LGP and MF-LGP$_{rotate}$ have a bigger value than M^2GP. It implies that M^2GP has a better overall performance than the two LGPHH-based multitask algorithms. Although the performance of M^2LGP and MF-LGP$_{rotate}$ are quite competitive with M^2GP in the three heterogeneous scenarios, they are inferior to M^2GP in the three homogeneous multitask scenarios based on the Wilcoxon test. The test effectiveness of multitask LGPHH methods should be further improved.

5.2 Example Program Analysis

To analyze how LGPHH shares useful knowledge among different tasks, we sample some example heuristics from all independent runs. Here, three heuristics

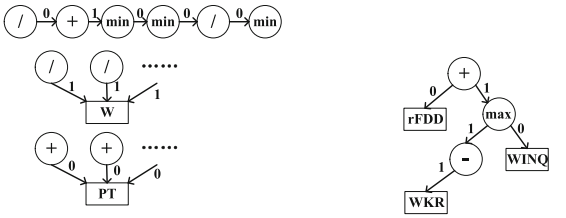
from an independent run of M^{2}LGP in solving homoWTmean and two heuristics from an independent run of MF-LGP in solving heteFTMax are selected. These LGP-based heuristics are transformed into DAGs and are shown in Fig. 4. The nodes of the DAGs represent different operations or terminals (oval for operations and rectangle for terminals). Each node has at most two output edges. These output edges accept the result from the node that it points to. The "0" and "1" beside edges respectively denote the first and second argument of operations. The final heuristic value is outputted from the top node.



(a) $WT_{mean}0.95$ (b) $WT_{mean}0.85$ (c) $WT_{mean}0.75$

(d) $F_{max}0.95$ (e) $T_{max}0.95$

Fig. 4. Three example programs from homoWTmean and two example programs from heteFTMax



(a) homoWTmean (b) heteFTMax

Fig. 5. Examples of common patterns in heuristics

If we have a closer look on these DAGs, some common patterns with at least three nodes can be found among the scheduling heuristics from a same scenario. The common patterns of these scheduling heuristics are highlighted in grey. Figure 5 shows some example common patterns in these heuristics. For the tasks of WT_{mean}, the three heuristics share common building blocks of dividing job weight and adding the processing time of next operation multiple times. These building blocks are also reused multiple times in these heuristics. It implies that it is advisable to select the operations with large weight and short processing time in minimizing the weighted mean tardiness. Besides, LGP-based heuristics can also share some "high-level" structures of operation combinations. The "$/, +, \min$" structure is adopted by two of the heuristics at the top of the DAGs. For the tasks of F_{max} and T_{max}, the two heuristics share a building block of "$rFDD + \max(WINQ, -WKR)$". It implies that, to minimize the maximum flowtime and tardiness, prioritizing the operations which are close to the due date (i.e., small $rFDD$) and have a lot of remaining work (i.e., large WKR) is a useful strategy. Besides, the operations which do not suffer from bottle neck machines (i.e., small $WINQ$) should also be processed as soon as possible. Averagely, more than half of the nodes are covered by at least one repeated patterns in these five heuristics. The results validate that LGP-based heuristics can share useful building blocks and operation combinations in the two existing multitask frameworks.

6 Conclusion

This paper makes an investigation of LGPHH methods with different evolutionary multitask frameworks. To extend LGPHH to two existing multitask frameworks, we have developed two LGPHH-based multitask methods, which are MFLGP and M^{2}LGP, based on the characteristics of LGPHH. These methods are examined on six different multitask scenarios, including three homogeneous and three heterogeneous scenarios. Some conclusions are drawn out based on the comparison. Firstly, the results show that M^{2}GP and MFEA with training instance rotation help LGPHH to have an effective test performance in multitask learning. It implies that LGPHH is suitable to those frameworks with more GPHH characteristics. Secondly, the multitask LGPHH methods based on off-the-shelf

frameworks are worse than tree-based GPHH methods in terms of test effectiveness. Designing a multitask framework to further enhance LGPHH performance will be our future work. Thirdly, the heuristics in multitask LGPHH can share some common patterns. These common patterns can be both of reusable building blocks and operation combinations. By sharing these common patterns among different tasks, the multitask LGPHH methods can be more effective than solving these tasks separately. In our future work, more LGP characteristics will be considered in the design of multitask frameworks, to further enhance the performance of multitask LGPHH. For example, LGP individuals can have multiple output registers to solve different tasks, and some selective crossovers can be developed based on different output registers.

References

1. Al-Sahaf, H., et al.: A survey on evolutionary machine learning. J. R. Soc. N. Z. **49**(2), 205–228 (2019)
2. Ardeh, M.A., Mei, Y., Zhang, M.: A novel multi-task genetic programming approach to uncertain capacitated Arc routing problem. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 759–767 (2021)
3. Brameier, M., Banzhaf, W.: A comparison of linear genetic programming and neural networks in medical data mining. IEEE Trans. Evol. Comput. **5**(1), 17–26 (2001)
4. Brameier, M., Banzhaf, W.: Linear Genetic Programming, vol. 53. Springer, Boston (2007). https://doi.org/10.1007/978-0-387-31030-5
5. Branke, J., Nguyen, S., Pickardt, C.W., Zhang, M.: Automated design of production scheduling heuristics: a review. IEEE Trans. Evol. Comput. **20**(1), 110–124 (2016)
6. Burke, E.K., Hyde, M.R., Kendall, G., Ochoa, G., Özcan, E., Woodward, J.R.: A classification of hyper-heuristic approaches: revisited. In: Gendreau, M., Potvin, J.-Y. (eds.) Handbook of Metaheuristics. ISORMS, vol. 272, pp. 453–477. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-91086-4_14
7. Chandra, R., Ong, Y.S., Goh, C.K.: Co-evolutionary multi-task learning for dynamic time series prediction. Appl. Soft Comput. J. **70**, 576–589 (2018)
8. Dal Piccol Sotto, L.F., De Melo, V.V.: A probabilistic linear genetic programming with stochastic context-free grammar for solving symbolic regression problems. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1017–1024 (2017)
9. Downey, C., Zhang, M., Liu, J.: Parallel linear genetic programming for multi-class classification. Genet. Program Evolvable Mach. **13**(3), 275–304 (2012). https://doi.org/10.1007/s10710-012-9162-9
10. Gupta, A., Ong, Y.S., Feng, L.: Multifactorial evolution: toward evolutionary multitasking. IEEE Trans. Evol. Comput. **20**(3), 343–357 (2016)
11. Huang, S., Zhong, J., Yu, W.J.: Surrogate-assisted evolutionary framework with adaptive knowledge transfer for multi-task optimization. IEEE Trans. Emerg. Top. Comput. **9**(4), 1930–1944 (2019)
12. Kantschik, W., Banzhaf, W.: Linear-tree GP and its comparison with other GP structures. In: Miller, J., Tomassini, M., Lanzi, P.L., Ryan, C., Tettamanzi, A.G.B., Langdon, W.B. (eds.) EuroGP 2001. LNCS, vol. 2038, pp. 302–312. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45355-5_24

13. Koza, J.R.: Genetic programming as a means for programming computers by natural selection. Stat. Comput. **4**(2), 87–112 (1994). https://doi.org/10.1007/BF00175355
14. Nguyen, S., Mei, Y., Zhang, M.: Genetic programming for production scheduling: a survey with a unified framework. Complex Intell. Syst. **3**(1), 41–66 (2017). https://doi.org/10.1007/s40747-017-0036-x
15. Nguyen, S., Zhang, M., Tan, K.C.: Surrogate-assisted genetic programming with simplified models for automated design of dispatching rules. IEEE Trans. Cybern. **47**(9), 2951–2965 (2017)
16. Park, J., Mei, Y., Nguyen, S., Chen, G., Zhang, M.: Evolutionary multitask optimisation for dynamic job shop scheduling using niched genetic programming. In: Mitrovic, T., Xue, B., Li, X. (eds.) AI 2018. LNCS (LNAI), vol. 11320, pp. 739–751. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03991-2_66
17. Provorovs, S., Borisov, A.: Use of linear genetic programming and artificial neural network methods to solve classification task. Sci. J. Riga Tech. Univ. Comput. Sci. **45**(1), 133–139 (2012)
18. Sanchez, M., Cruz-Duarte, J.M., Ortiz-Bayliss, J.C., Ceballos, H., Terashima-Marin, H., Amaya, I.: A systematic review of hyper-heuristics on combinatorial optimization problems. IEEE Access **8**, 128068–128095 (2020)
19. Wilson, G., Banzhaf, W.: A comparison of cartesian genetic programming and linear genetic programming. In: O'Neill, M., et al. (eds.) EuroGP 2008. LNCS, vol. 4971, pp. 182–193. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78671-9_16
20. Xu, Q., Wang, N., Wang, L., Li, W., Sun, Q.: Multi-task optimization and multi-task evolutionary computation in the past five years: a brief review. Mathematics **9**(8), 1–44 (2021)
21. Yi, J., Bai, J., He, H., Zhou, W., Yao, L.: A multifactorial evolutionary algorithm for multitasking under interval uncertainties. IEEE Trans. Evol. Comput. **24**(5), 908–922 (2020)
22. Zhang, F., Mei, Y., Nguyen, S., Tan, K.C., Zhang, M.: Multitask genetic programming-based generative hyperheuristics: a case study in dynamic scheduling. IEEE Trans. Cybern. 1–14 (2021). https://doi.org/10.1109/TCYB.2021.3065340
23. Zhang, F., Mei, Y., Nguyen, S., Zhang, M.: A preliminary approach to evolutionary multitasking for dynamic flexible job shop scheduling via genetic programming. In: Proceedings of Genetic and Evolutionary Computation Conference Companion, pp. 107–108 (2020)
24. Zhang, F., Mei, Y., Nguyen, S., Zhang, M.: Collaborative multifidelity-based surrogate models for genetic programming in dynamic flexible job shop scheduling. IEEE Trans. Cybern. 1–15 (2021). https://doi.org/10.1109/TCYB.2021.3050141
25. Zhang, F., Mei, Y., Nguyen, S., Zhang, M.: Correlation coefficient-based recombinative guidance for genetic programming hyperheuristics in dynamic flexible job shop scheduling. IEEE Trans. Evol. Comput. **25**(3), 552–566 (2021)
26. Zhang, F., Mei, Y., Nguyen, S., Zhang, M.: Evolving scheduling heuristics via genetic programming with feature selection in dynamic flexible job-shop scheduling. IEEE Trans. Cybern. **51**(4), 1797–1811 (2021)
27. Zhang, F., Mei, Y., Nguyen, S., Zhang, M., Tan, K.C.: Surrogate-assisted evolutionary multitasking genetic programming for dynamic flexible job shop scheduling. IEEE Trans. Evol. Comput. **25**(4), 651–665 (2021)

28. Zhang, F., Nguyen, S., Mei, Y., Zhang, M.: Genetic Programming for Production Scheduling - An Evolutionary Learning Approach. Springer, Singapore (2021). https://doi.org/10.1007/978-981-16-4859-5
29. Zhou, L., Feng, L., Zhong, J., Ong, Y.S., Zhu, Z., Sha, E.: Evolutionary multitasking in combinatorial search spaces: a case study in capacitated vehicle routing problem. In: IEEE Symposium Series on Computational Intelligence (2016)