Investigation of Linear Genetic Programming for Dynamic Job Shop Scheduling

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

2021 IEEE Symposium Series on Computational Intelligence (SSCI) | 978-1-7281-9048-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/SSCI50451.2021.9660091

***Abstract*—Using genetic programming-based hyper-heuristic methods to automatically design dispatching rules has become one of the most effective methods to solve dynamic job shop scheduling. However, most genetic programming-based hyper-heuristic methods are developed based on tree-like structures. On the other hand, linear genetic programming variants, whose individuals are designed in a linear fashion, also have been successfully applied to some classification and symbolic regression problems and achieved promising results. But the studies of linear genetic programming as a hyper-heuristic for evolving dispatching rules for job shop scheduling are still in the infancy. To apply linear genetic programming to dynamic job shop scheduling (DJSS), this paper makes a comprehensive investigation on the design issues of linear genetic programming (e.g., the number of registers and the variation step size) and validates that linear genetic programming has a competitive performance with standard genetic programming and can produce compact dispatching rules.**

***Index Terms*—genetic programming, hyper-heuristic, linear genetic programming, dynamic job shop scheduling**

I. Introduction

Dynamic job shop scheduling (DJSS) is a special type of job shop scheduling problem. Different from static job shop scheduling where all information of the problem are known beforehand, there are dynamic events in DJSS problems whose information is not known until the dynamic event happens. These dynamic events will occur incidentally during the execution process of the schedule and greatly affect the effectiveness of original schedule. The common dynamic events in the industry are the new job arrival [1] and machine breakdown [2]. Since DJSS caters for many applications such as cloud computing [3] and railway management [4], it has attracted a lot of research interest from both academia and industry.

Hyper-heuristic is a state-of-the-art method for solving DJSS [5], [6]. Specifically, hyper-heuristic is a big umbrella of methods which tries to select or generate different types of heuristics to solve a certain task [7]. Thus, hyper-heuristic methods search in the heuristic space rather than the solution space. Among different hyper-heuristic methods, using genetic programming (GP) [8] as a hyper-heuristic (GPHH) to automatically design dispatching rules is one of the important branches and has undergone a good development in the last decade in solving DJSS [9]. To evolve an effective dispatching rule, GPHH encodes dispatching rules as individuals and applies genetic operators to reproduce new rules. The performance of these generated rules are evaluated on the training instances of DJSS. GPHH has successfully found better rules than the existing man-made dispatching rules for many benchmark DJSS scenarios [10], [11].

Most existing GPHH methods for DJSS are of tree-based genetic programming (TGP). Though a few of other GP variants like gene expression programming [12], [13] have been applied to DJSS, the research on other GP variants in solving DJSS is still limited. Linear genetic programming (LGP) [14] is a popular GP variant, which has been widely applied to classification and symbolic regression [15]–[20], and gained a promising performance in many datasets. The empirical results show that LGP can evolve more compact heuristics than TGP [14], and has an easy control on the variation step size [21]. To the best of our knowledge, LGP has never been applied to DJSS.

This paper aims to make a comprehensive investigation on the effectiveness of the off-the-shelf LGP in solving DJSS. It should be noted that DJSS problems have an essential difference with classification and symbolic regression. Due to the differences, some recommended settings for LGP in classification and symbolic regression may not be suitable in DJSS. We will also investigate the effectiveness and efficiency of the basic LGP instance by comparing with standard GP [8] (SGP), as a representative example of TGP. Finally, the compact program size and the interpretability of LGP rules will be analysed. Specifically, this paper attempts to answer the following three research questions.

- What is the proper parameter settings of LGP in evolving dispatching rules for DJSS?
- What are the differences between off-the-shelf LGP and SGP in terms of effectiveness and efficiency when evolving dispatching rules for DJSS?
- Are the dispatching rules evolved by LGP more compact and more explainable than the ones from SGP?

The rest of the paper is organised as follows. In Section II, we introduce the DJSS problem and the GPHH framework, and review the related work of using GPHH to solve DJSS. Then, the rationale of LGP is introduced in Section III. After that, we will investigate the proper parameter settings of LGP, and validate its effectiveness and advantages in sections IV-VI. Finally, the conclusion will be drawn in Section VII.

978-1-7281-9048-8/21/$31.00 ©2021 IEEE

II. Background

A. Problem Definition

In this paper, we focus on DJSS with new job arrivals. The DJSS studied in this paper has a finite set of machines M and a set of jobs J. Jobs will come into J continuously over time. Every job j has an arrival time α_j, a due date d_j, and a weight ω_j which specifies the importance of the job. Besides, for every job $j \in J$, we have a sequence of operations $(o_1^j, ..., o_h^j, ..., o_{l_j}^j)$ which specifies the execution order of operations for the job j. In this sequence, every operation is completed by a specific machine $m(o_h^j) \in M$ and consumes a positive processing time $p(o_h^j)$. Specifically, the first operation o_1^j can only be scheduled after job j arrived to the job shop, and the operation $o_h^j (h = 2, ..., l_j)$ can only be scheduled after o_{h-1}^j is completed. Every operation also has its own due date and the due date of a job in our experiment is extended from the due date of the last operation by a predefined factor. Each machine can process at most one operation at any time and the processing is assumed to be uninterruptable for any jobs.

Three optimization objectives are considered in this work. They are the maximum tardiness T_{max}, mean tardiness $\overline{T}$, and mean weighted tardiness $\overline{T_w}$. Denoting the finishing time of job j as c_j, these objectives are formulated as follows.

- $T_{max} = \max_{j \in J}(\max(c_j - d_j, 0))$
- $\overline{T} = \frac{\sum_{j \in J}(\max(c_j - d_j, 0))}{|J|}$
- $\overline{T_w} = \frac{\sum_{j \in J}(\max(c_j - d_j, 0) \cdot \omega_j)}{|J|}$

where $c_j = x(o_{l_j}^j) + p(o_{l_j}^j)$ and $x(o_{l_j}^j)$ is the starting time of the final operation of job j.

B. GPHH Framework

The GPHH methods for DJSS contain four main modules: GP module, simulation module, training data, and test data. The GP module encodes dispatching rules into GP individuals and contains the process of evolving dispatching rules, each as a GP individual. The fitness of a GP individual is calculated by running the simulation based on the GP individual and the training data. Specifically, a training instance includes the information of the jobs and machines (e.g., arrival times, processing times and due dates). During the simulation, when a machine becomes idle, the GP individual (i.e., a dispatching rule) selects the job from the machine's queue to process next. Given a GP individual and a training instance, the simulation module generates a schedule. Then, the fitness of the GP individual can be defined as the average objective value of the schedules obtained for all the training instances. When the GP module reaches its stopping criteria, the best dispatching rule from the training process is validated on the test data.

C. Related Work

Using GPHH to evolve dispatching rules has undergone a rapid development in the past decade. Many advanced techniques have been proposed to improve the effectiveness of GPHH. For example, a feature selection technique is designed for GPHH to improve effectiveness and interpretability [22], and surrogate functions and multitask framework are also introduced to GPHH to enhance the training efficiency [23]. Designing effective dispatching rule representations is one of the common methods to enhance the effectiveness of GPHH. A representative work is named "GP-3" proposed by Jakobović in 2006 [24]. There are three rules for different purposes in GP-3 method. One rule is designed to distinguish bottleneck machines, and the other two rules are the dispatching rules for bottleneck and non-bottleneck machines respectively. It is reported that GP-3 can outperform other human-designed dispatching rules in DJSS. A similar idea of composite rules is also adopted by Nguyen et al. [25]. In [25], various types of composite rules are investigated. For example, the tree structure is divided into a decision tree and man-made dispatching rules or divided into a decision tree and GP-evolved rules. Their computational analysis found that a GP tree composing a decision tree and GP-evolved rules has the best performance. To fully utilize the machines with different properties, Pickardt et al. [26] designed a multi-tree paradigm to evolve a group of dispatching rules and assign them to different machines. Besides, Park et al. [27]–[30] improved the robustness of GPHH by ensemble techniques. They investigated the effectiveness of different ensemble schemes and proposed an ensemble paradigm based on multi-level GP [31]. Their results validate the effectiveness of ensemble techniques in DJSS. Some studies also attempt to apply other representations to encode dispatching rules. For example, Nie et al. [12] applied the gene expression programming for DJSS. Because gene expression programming adopts a kind of linear representation in its chromosome, the program size can be well controlled without the loss of effectiveness. An investigation by Durasević et al. [13] also validates the similar effectiveness and a higher interpretability of gene expression programming compared with conventional TGP.

The methods mentioned above succeeded in improving the effectiveness of GPHH in DJSS. But these GPHH methods are mainly designed based on SGP. In some cases, it is non-trivial for SGP to have a further improvement. For example, because SGP is designed in a tree-like fashion and the tree-like structures only have one output (i.e., the root), when multiple decisions are required for every decision situation (e.g., ensemble methods), it is common for GPHH methods to maintain a multi-tree paradigm, which may enlarge the search space. Besides, most existing methods share the common patterns among different trees by a stochastic sub-tree swapping. These designs enlarge the search space to some extent. SGP also limits the reuse of intermediate results. The results from the sub-trees can only be accessed by their own parents. When there are repetitive patterns in the calculation, take summing up a for 16 times as an example (i.e., $\sum_{i=1}^{16} a$), an SGP solution may need 16 additions while the calculation can be done at least by 4 additions if intermediate results can be reused flexibly.

To remove these limitations, this paper intends to apply LGP to evolve dispatching rules for DJSS. LGP is another popular

r[0] = r[1] + x_0
r[2] = r[0] * r[0]
r[1] = max(r[2], x_1)
r[0] = r[1] * r[2]

Fig. 1. An example program of LGP

GP variant. Every individual of LGP can have multiple outputs for different tasks, and these outputs have a natural sharing of some common patterns. It means that LGP has a more compact and efficient representation than TGP by sharing common patterns between sub-tasks. The effectiveness of the multiple outputs of LGP has been validated in multi-class classification problems [19], [32]. LGP can also evolve compact individuals by reusing the intermediate results of registers [14]. Besides, because of its instruction-based representation, LGP rules cater to the human reading habit better than tree-like structures. Some work have been done to analyze the robustness of LGP [21], [33], [34] and improve its efficiency in other domains, such as symbolic regression problems [18], but few studies apply LGP to scheduling problems, especially DJSS. Thus, in this paper, we intend to apply LGP to DJSS and investigate its effectiveness.

Algorithm 1: The pseudo code of LGP

Input: the population size S_{pop}, the step size of macro mutation λ, the maximum segment length of exchanging segments L_{cross}, the maximum difference of exchanging segments $\Delta_{L_{cross}}$, the maximum distance of crossover points $D_{crosspoint}$, the rate of different genetic operators, and the maximum length of the program L_{prog}.
Output: The best-of-run dispatching rule R^*.

```
Initialize the population pop based on S_pop and L_prog;
while The stopping criteria are not satisfied do
    Evaluate every individual R by the simulation on DJSS instances;
    Parent individuals R_1 = R_2 = null, offspring R' = null;
    newpop = individuals from an elitism selection on pop;
    while the size of newpop < S_pop do
        Select R_1 and R_2 by a tournament selection on pop;
        p = uniform(0, 1);
        if p ≤ macro mutation rate then
            R' = MacroMutation(R_1, λ);
        else if p ≤ macro mutation rate+crossover rate then
            R' = Crossover(R_1, R_2, L_cross, Δ_{L_cross}, D_crosspoint);
        else if p ≤ macro mutation rate+crossover rate+micro mutation rate then
            R' = MicroMutation(R_1);
        else
            R' = R_1;
        newpop = newpop ⋃ R';
    pop = newpop;
    Update R* ∈ pop;
Return R*.
```

III. Linear Genetic Programming

A. Representation

One of the key differences of LGP from TGP is the representation. The representation of LGP is a sequence of register-based instructions [14]. Every register-based instruction consists of three parts: destination register, operator (also termed as function indicator), and source registers. In execution, the operation in the instruction accepts the values from source registers as inputs, and assigns the calculation results to the destination register. These instructions will be executed in order. The values in the pre-specified output registers will be regarded as the output of the whole program. It should be noted that the way to arrange these instructions and the sequential execution of all these instructions form the core meaning of the term "linear". Fig. 1 is a simple example of LGP to represent a heuristic $\max(x_0^2, x_1) * x_0^2$ where x_0 and x_1 are the inputs. All registers (i.e., $r[i], i \in 0, 1, 2$) are initialized as 0 for simplicity. In the example program, x_0, x_0^2, and $\max(x_0^2, x_1)$ are stored to the registers accordingly and these intermediate calculation results are finally aggregated to the predefined output register (i.e., $r[0]$).

B. Algorithm Design

The evolutionary framework of LGP follows a typical evolutionary algorithm framework. A brief pseudo code of LGP is shown in Alg. 1. Firstly, a population of individuals is initialized and evaluated. It then applies tournament selection to select parent individuals from the current population in every generation. These parent individuals produce offspring by different genetic operators. These offspring form the new population and are evaluated on the DJSS instance. LGP will keep evolving until it meets the given stopping criteria. The best individual is returned as the final solution.

Specifically, there are two types of variations in LGP [14]. They are *macro variations* which insert or delete instructions in an individual, and *micro variations* which do not change the total number of instructions but modify the primitive within instructions. Though there have been many advanced genetic operators in both variation types, in this paper, we only adopt two typical macro variations and one micro variation for a fair comparison with canonical TGP. The two macro variations are the linear crossover (denoted as "*cross*") [35] and the effective macro mutation (denoted as "*effmut*") [36]. The main idea of *cross* is exchanging the instruction segments of parent individuals to generate offspring. There are three features of the selected segments, which are the segment length, the length difference of the segments, and the distance between crossover points. They can be used to control the variation step size.

For *effmut*, it firstly chooses insertion or deletion randomly, and then inserts at least one effective instruction into random places of the parent individual or deletes a number of random instructions from the individual. The term "effective" implies the inserted instruction must be able to affect the final output of the offspring for at least one input. Besides, we adopt a micro mutation which selects a random instruction from the parent individual and mutates one of the three parts of the instruction based on a distribution. More details of these three genetic operators can be found in the chapter 5 and 6 of [14].

TABLE I
PARAMETERS OF LGP

to-be-tuned parameters	S_{pop}	1024,**512**,256
	N_{reg}	2,**4**,6,16
	λ_{effmut}	**1**,3,5
	L_{cross}	**5**,10,15
	$\Delta_{L_{cross}}$	0,**1**,3,5
	$D_{crosspoint}$	25,50,**100**
	macro variations%(effmut:cross)	**70:0 or 0:70**, 35:35, 50:20, 20:50,
	L_{prog}	40,60,80,**100**,120
default parameters	micro mutation%	20
	reproduction%	10
	insertion%:deletion%	80:20
	tournament size	7
	elitism %	2
	total evaluations	52224
	function set	$+, -, \times, \div_{pro}, \max, \min$
	terminal set	see table II

TABLE II
THE TERMINAL SET

Name	Description
NIQ	the number of operations in the queue
WIQ	the total processing time of operations in the queue
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

As recommended by [14], an additional micro variation will be appended to the macro variation in our work, to better solve complex problems like DJSS.

IV. EFFECT OF LGP PARAMETERS

A. Experiment Design

Although the existing studies have investigated the LGP parameter settings in other problems such as symbolic regression, it is still unknown whether the common LGP parameter settings will be suitable for evolving dispatching rules for DJSS, due to the different characteristics between DJSS and other problems. Therefore, in this study, we firstly conduct a comprehensive investigation on the parameter sensitivity for LGP in the context of DJSS. Specifically, we investigate the following LGP parameters which may affect LGP performance. They are: (1) the population size S_{pop}, (2) the number of calculation registers N_{reg}, (3) the step size of *effmut* λ_{effmut}, (4) the maximum segment length of exchanging segments L_{cross}, (5) the maximum difference of exchanging segments $\Delta_{L_{cross}}$, (6) the maximum distance of crossover points $D_{crosspoint}$, (7) the rate of $effmut$ and $cross$, and (8) the maximum length of the program L_{prog}.

All investigated values of the to-be-tuned parameters are shown in Table I, where the bold values are the default ones. When tuning a certain parameter, the rest of parameters are set as their default values. The default values are defined based on the recommended settings from existing studies and some preliminary experiments. It is noted that, in [14], it recommends to use only one type of macro variation (i.e., *effmut* or $cross$) in LGP. Thus, in our experiment, we follow their settings and respectively use mutation-only ($effmut : cross = 70 : 0$) or crossover-only ($0 : 70$) paradigms as the default settings in investigating the proper step size of *effmut* and $cross$. To ensure that the macro variation plays the major role in the evolution, the total rate of macro variation is set to 70% while micro mutation and that reproduction take 20% and 10% respectively. Besides these to-be-tuned parameters, there are some other parameters in LGP which are not so influential to the algorithm performance. These parameters are set as the default values recommended by some existing work [5], [14], [19]. The values of these parameters are shown in Table I, where $\div_{pro}$ is the protected version of division and returns 1.0 if the denominator is zero. The insertion and deletion rate (%) in the table is the distribution of the *effmut* to select inserting (or deleting) a random instruction into (from) the individual. Before every simulation, the terminals in Table II are used to initialize the registers sequentially.

B. Simulation Configuration

There are totally 10 machines in our job shop. The processing time of all operations is a continuous value ranging from 1 to 99, and the number of operations in a job ranges from 2 to 10. 20% of jobs have a weight of 1, 60% of jobs have a weight of 2, while the rest of jobs have a weight of 4. Jobs arrive to the job shop following a Poisson process. Two utilization levels (i.e., 0.85 and 0.95) are adopted in the simulation. To evaluate the steady-state performance, the first 1000 jobs will not be counted (i.e., 1000 warm-up jobs) and the subsequent 5000 jobs are used to evaluate the dispatching rule. To normalize the objective values, the earliest due date, the apparent tardiness cost, and the weighted apparent tardiness cost dispatching rules are regarded as the benchmark rules respectively for T_{max}, $\overline{T}$, and $\overline{T_w}$. The tardiness of an LGP-evolved dispatching rule over the one of corresponding benchmark rule is defined as the normalized objective value. All experiment settings have 30 independent runs with different random seeds. In every independent run, a different DJSS instance is used for training for every generation, and 50 unseen DJSS instances are used as test data.

TABLE III
TEST PERFORMANCE WITH DIFFERENT S_{pop} AND N_{reg}

S_{pop}	N_{reg}	T_{max}-0.85(std.)		$\overline{T}$-0.95(std.)	
1024	2	0.520 (0.032)	≈	**0.731 (0.037)**	–
	4	**0.502 (0.018)**	≈	0.739 (0.050)	–
	6	0.522 (0.031)	≈	0.732 (0.029)	–
	16	0.514 (0.022)	≈	0.745 (0.040)	–
512	2	0.509 (0.027)	≈	0.705 (0.022)	≈
	4	**0.504 (0.019)**	def	0.706 (0.031)	def
	6	0.506 (0.024)	≈	**0.694 (0.023)**	≈
	16	0.509 (0.017)	≈	0.699 (0.033)	≈
256	2	0.520 (0.032)	≈	0.713 (0.043)	≈
	4	0.517 (0.029)	≈	0.687 (0.023)	≈
	6	0.518 (0.029)	≈	**0.685 (0.019)**	≈
	16	**0.511 (0.024)**	≈	0.689 (0.029)	≈

C. Results and Discussion

In the investigation on the effect of different parameters, we select the maximum tardiness with 0.85 utilization level (denoted as T_{max}-0.85) and the mean tardiness with 0.95 utilization level (denoted as $\overline{T}$-0.95) as two example scenarios. Wilcoxon rank sum test is conducted to analyze these results. For the wilcoxon rank sum test, notation "def" denotes the default settings, notation "+" means the corresponding setting significantly outperforms the default setting, notation "≈" means these settings are statistically similar, and "-" means the corresponding setting is inferior to the default setting. The bold results are the lowest average test performance.

1) S_{pop} and N_{reg}: To identify a robust setting for S_{pop} and N_{reg}, we begin with investigating the effectiveness of different population size and number of registers together. In the experiments here, only *effmut* plays the role of macro variation. The test performance with 512 individuals and 4 registers are defined as the baseline results. As shown in Table III, for the problem T_{max}-0.85, all settings of S_{pop} have a statistically similar test performance, while for the problem $\overline{T}$-0.95, LGP with population size of 512 and 256 tends to be superior to the one with population size of 1024. The results with different N_{reg} also showed a statistically similar performance. But when the number of registers changes from 4 to 2 in $\overline{T}$-0.95, the test performance of $S_{pop} = 256$ is decreased. It should be noted that the setting from [14] of 16 registers did not perform better than the other settings. It is probably because within the limited evolution generations, it is hard for LGP to fully utilize a large number of registers.

2) λ_{effmut}: The test performance with different λ_{effmut} values are shown in Table IV. The LGP with one mutation step serves as the baseline method. When λ_{effmut} is larger than 1, an integer will be sampled from $Z \in [1, \lambda_{effmut}]$ as the actual step for every *effmut*. As shown in Table IV, it is believed that λ_{effmut} is relatively robust in these two problems since there is no significant difference among different λ_{effmut} values. But the test performance tends to decrease as the step size increases. It is because when the mutation step size is too large, every mutation acts like a random jumping in the phenotypic space of LGP, which prevents LGP from convergence.

TABLE IV
TEST PERFORMANCE WITH DIFFERENT λ_{effmut}

λ_{effmut}	T_{max}-0.85(std.)		$\overline{T}$-0.95(std.)	
1	**0.504 (0.019)**	def	0.706 (0.031)	def
3	0.507 (0.027)	≈	**0.703 (0.017)**	≈
5	0.511 (0.027)	≈	0.710 (0.028)	≈

TABLE V
TEST PERFORMANCE WITH DIFFERENT L_{cross},$\Delta_{L_{cross}}$, AND $D_{crosspoint}$

Parameters		T_{max}-0.85(std.)		$\overline{T}$-0.95(std.)	
L_{cross}	5	**0.503 (0.030)**	def	0.716 (0.033)	def
	10	0.507 (0.032)	≈	0.716 (0.035)	≈
	15	**0.503 (0.026)**	≈	**0.700 (0.023)**	≈
$\Delta_{L_{cross}}$	0	**0.497 (0.016)**	≈	0.717 (0.027)	≈
	1	0.503 (0.030)	def	0.716 (0.033)	def
	3	0.501 (0.032)	≈	0.709 (0.027)	≈
	5	0.503 (0.025)	≈	**0.704 (0.022)**	≈
$D_{crosspoint}$	25	**0.501 (0.031)**	≈	0.719 (0.034)	≈
	50	0.503 (0.029)	≈	0.717 (0.032)	≈
	100	0.503 (0.030)	def	**0.716 (0.033)**	def

3) L_{cross},$\Delta_{L_{cross}}$, and $D_{crosspoint}$: To investigate the effect of these parameters on the performance, we only use *cross* as the macro variation in the experiment. The results with default settings (i.e., $L_{cross} = 5$,$\Delta_{L_{cross}} = 1$,$D_{crosspoint} = 100$) is regarded as the baseline and all the results are shown in Table V. From a first glance, these parameters do not have a significant effect on these two DJSS problems. For L_{cross}, the longest crossover segment has the best performance in both problems. However, as shown in Table VI, both of the average absolute program length (denoted by L_{abs}), the average effective program length (denoted by L_{eff}), and the average training time (abbreviated as "Time") are all increased rapidly with the increase of L_{cross}. The results imply that a large L_{cross} reduces the training efficiency of LGP by evolving more complex individuals than other settings. For $\Delta_{L_{cross}}$, the test performance of both simple and difficult problems become stable and promising with the increase of $\Delta_{L_{cross}}$. Based on Table VI, a large $\Delta_{L_{cross}}$ also increases the training time. But it only increases the training time by 10-20%. For $D_{crosspoint}$, the performance of all different settings on these two problems are similar.

4) macro variation rates: With the proper settings of different genetic operators, we would like to answer a question: will coordinating different macro variations be better than only using a single type of them, and if yes, how to arrange their rate. Specifically, N_{reg} is set as 4 and λ_{effmut} is set as 1. Table VII compares the results with different macro variation rates. The *effmut*-only setting is defined as the baseline. The table shows that the compared settings all have a similar performance. It implies that different macro variation rates do not exert a significant effect on the test performance of LGP. But a larger crossover rate seems to give slightly better performance.

5) L_{prog}: To identify the suitable setting of L_{prog}, the test performance of five different L_{prog} are investigated. The

TABLE VI
PROGRAM LENGTH AND RUNNING TIME WITH DIFFERENT L_{cross} AND $\Delta_{L_{cross}}$

Parameters		T_{max}-0.85(std.)			$\overline{T}$-0.95(std.)		
		L_{abs}	L_{eff}	Time(sec)	L_{abs}	L_{eff}	Time(sec)
L_{cross}	5	39.00(7.038)	19.05(5.409)	3165.6(293.7)	36.31(6.715)	20.76(5.159)	5886.4(859.1)
	10	59.25(10.36)	27.63(8.346)	3747.0(374.0)	64.17(12.83)	32.35(9.43)	7819.3(1559.9)
	15	74.37(11.83)	32.01(7.42)	4306.4(634.0)	87.55(8.51)	41.77(10.61)	9709.0(1945.8)
$\Delta_{L_{cross}}$	0	31.65(5.19)	16.42(4.53)	2965.0(289.8)	34.88(7.44)	19.20(5.19)	6076.0(879.8)
	1	39.00(7.038)	19.05(5.409)	3165.6(293.7)	36.31(6.715)	20.76(5.159)	5886.4(859.1)
	3	49.97(6.72)	22.27(4.39)	2964.6(448.4)	49.40(7.62)	28.11(6.65)	6186.9(1035.6)
	5	51.07(8.40)	28.11(6.65)	3578.0(318.7)	56.05(8.67)	28.70(5.92)	7049.6(1206.6)

TABLE VII
TEST PERFORMANCE WITH DIFFERENT MACRO VARIATION RATE

effmut:*cross*	T_{max}-0.85(std.)		$\overline{T}$-0.95(std.)	
70:0	0.504 (0.019)	def	0.706 (0.031)	def
50:20	0.505 (0.022)	$\approx$	0.705 (0.023)	$\approx$
35:35	0.507 (0.033)	$\approx$	0.706 (0.027)	$\approx$
20:50	**0.498 (0.020)**	$\approx$	0.705 (0.031)	$\approx$
0:70	0.503 (0.025)	$\approx$	**0.704 (0.022)**	$\approx$

TABLE VIII
TEST PERFORMANCE WITH DIFFERENT L_{prog}

L_{prog}	T_{max}-0.85(std.)		$\overline{T}$-0.95(std.)	
40	**0.490 (0.008)**	$\approx$	0.710 (0.033)	$\approx$
60	0.495 (0.017)	$\approx$	0.710 (0.028)	$\approx$
80	0.498 (0.020)	$\approx$	**0.705 (0.029)**	$\approx$
100	0.498 (0.020)	def	**0.705 (0.031)**	def
120	0.498 (0.020)	$\approx$	**0.705 (0.031)**	$\approx$

parameter settings in the prior sub-sections and the recommended settings of the two macro variations (i.e., *effmut* : *cross* $= 20 : 50$) are adopted in the experiments here. The performance with the default L_{prog} (i.e., 100) is defined as the baseline setting. As shown in Table VIII, there is a similar pattern of test performance among these settings. Besides, the experiments with L_{prog} ranging from 80 to 120 have a nearly the same test performance. It is because when L_{prog} is large enough, LGP hardly reaches the maximum program length within the limited generations. On the other hand, when L_{prog} decreases, an improvement on test performance can be seen for the problem with a lower utilization level while a deterioration of test performance is seen for the problem with a higher utilization level.

6) summary: Overall, the parameters of this LGP instance are relative robust in DJSS problems given that a wide range of parameter settings have a statistically similar test performance with the baseline settings. But we still have some useful findings. Firstly, for S_{pop} and N_{reg}, setting the population size as 512 and providing 4 or 6 additional computation registers show a slightly better overall performance than the other settings in the two example scenarios. Secondly, to enhance the training efficiency within limited generations, it is suggested to use a moderate step size (i.e., $\lambda_{effmut} = 1, 3$, $L_{cross} = 5$, $\Delta_{L_{cross}} = 5$, and $D_{crosspoint} = 100$) in variations, rather than a small step size as recommended in other problems. Thirdly, cooperating different genetic operators and giving *cross* a higher rate (i.e., *effmut* : *cross* $= 20 : 50$) is a suggested choice. For other parameters, default settings (i.e., $L_{prog} = 100$) are good and robust.

TABLE IX
TEST PERFORMANCE OF SGP AND LGP

Scenarios	SGP (std.)	LGP (std.)	
T_{max}-0.85	0.495(0.017)	0.498(0.020)	$\approx$
T_{max}-0.95	0.705(0.021)	0.698(0.019)	$\approx$
$\overline{T}$-0.85	0.526(0.054)	0.494(0.036)	$+$
$\overline{T}$-0.95	0.681(0.026)	0.705(0.031)	$-$
$\overline{T_w}$-0.85	0.588(0.099)	0.534(0.070)	$+$
$\overline{T_w}$-0.95	0.800(0.046)	0.836(0.050)	$-$

V. COMPARISON WITH SGP

To investigate the effectiveness of LGP, we compare LGP with SGP on six different scenarios in terms of test performance and training efficiency, which are T_{max}-0.85 and $\overline{T}$-0.95 introduced above, T_{max}-0.95, $\overline{T}$-0.85, $\overline{T_w}$-0.85, and $\overline{T_w}$-0.95. All scenarios have 30 independent runs with different random seeds and a wilcoxon rank sum test is conducted between SGP and LGP. The parameters of SGP are set according to [37], while the parameters of LGP are set as the recommended settings (Specifically, $N_{reg} = 4$ and $\lambda_{effmut} = 1$).

Table IX shows the mean and standard deviation of the test performance of the two methods. LGP has a significantly superior performance in $\overline{T}$ and $\overline{T_w}$ with utilization level of 0.85 and is similar with SGP in the two T_{max} problems. But when the utilization levels of $\overline{T}$ and $\overline{T_w}$ increase to 0.95, the performance of LGP is inferior to SGP. Both of LGP and SGP show a similar pattern of standard deviation.

To further validate the training efficiency of LGP, we also show the convergence curves of LGP and SGP in Fig.2. From the figure, we can see that except for $\overline{T}$-0.95 and $\overline{T_w}$-0.95 in which LGP has an inferior performance to SGP, the other convergence curves of SGP and LGP are nearly overlapped. Besides, for the problems in which LGP has a better performance (e.g., $\overline{T}$-0.85 and $\overline{T_w}$-0.85), the curves of LGP are narrower than the ones of SGP, which mean a better stability. Based on these results, LGP has a competitive effectiveness in designing dispatching rules for DJSS compared with SGP.

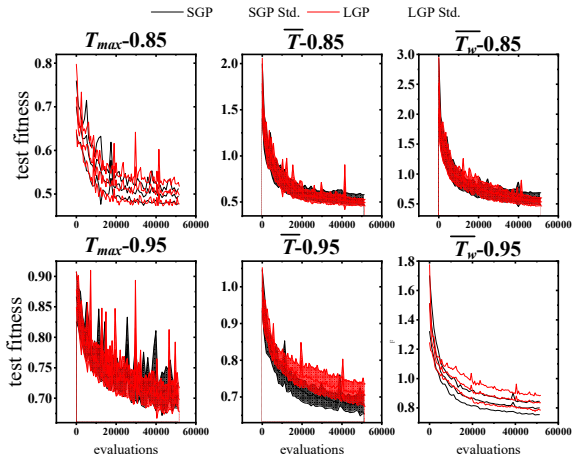

Fig. 2. Convergence curves on six scenarios.

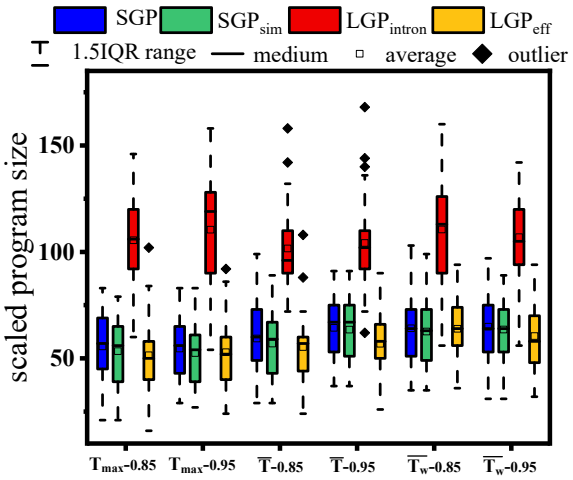

Fig. 3. Scaled program size of SGP and LGP.

A=SL−WKR+3PT+NIQ
rule= max(A−WKR, A+PT− 2)

Fig. 4. Example program of LGP.

VI. Interpretability Analysis

Evolving compact programs is one of the advantages of LGP in other applications [14], [32]. A more compact program usually facilitates human users to explain the evolved rule. To validate this advantage, in this section, we firstly compare the program size of the best LGP and SGP rules by a box plot (i.e., Fig. 3), and then give some explanations on some example programs.

To make a fair comparison of program size between LGP and SGP is non-trivial. There are more computer manipulations in an LGP instruction than in SGP tree nodes. Every instruction needs to read the values from two source registers, execute one function calculation, and rewrite one target register, while a function primitive in TGP only needs to accept two inputs and calculate the function. Thus, the number of instructions of LGP rules is scaled up by a factor of 2.0, to make a fair comparison with the number of tree nodes in SGP rules. To make a comprehensive comparison, the program size of SGP and LGP before and after simplification (respectively denoted as "SGP", "SGP_{sim}", "LGP_{intron}", and "LGP_{eff}") are compared. Specifically, "LGP_{intron}" counts both of intron and effective instructions, and "LGP_{eff}" only counts effective instructions. As shown in Fig. 3, LGP only with effective instructions (i.e., yellow boxes) has an averagely smaller program size than SGP (i.e., blue boxes) and the simplified SGP (i.e., green boxes) in six scenarios. Although the program size of LGP rules with introns (i.e., red boxes) is much larger than the other three settings, given that the introns of LGP have no effect on the final results and they will not be executed, it is concluded that LGP can achieve a competitive performance with SGP by shorter programs.

To further analyze the explainability, we try to analyze an example program of LGP. Fig. 4 shows an LGP-evolved rule for Tmax-0.85, after some manual simplification. We can see that the rule reuses a building block A, which tries to minimize "SL", "PT", and "NIQ". It implies that the operations which are going to be late or have a short processing time will be scheduled first. Besides, the building block tries to maximize the remaining processing time of the job so that the jobs with a large amount of uncompleted work will have a high priority in reducing the maximum tardiness. The building block is reused in a maximum comparison. However, these two terms in the maximum comparison still show a favour to "−WKR" and "PT", which is consistent with the building block. Given that "−WKR" is always non-positive and "PT−2" is positive in most cases, the dispatching rule can be further simplified to the latter element of the maximum comparison.

VII. Conclusions

In this paper, we mainly investigate the application of LGP in DJSS problems. Three main questions are tried to be answered. Firstly, a series of comprehensive experiments are conducted and the most suitable parameter settings for DJSS are recommended. Generally speaking, we should place more emphasis on *cross* which is more effective than *effmut* in DJSS with the limited evolution generations. A proper setting of variation step size is needed for both *cross* and *effmut*. Secondly, LGP is very competitive compared with SGP. It shows a superior performance in some scenarios, but still needs a further improvement in the others. These improvement will be our future work. Finally, the results show that LGP can evolve more compact rules than SGP, and LGP rules are able to reuse some building blocks.

References

[1] Y. Mei and M. Zhang, "A comprehensive analysis on reusability of GP-evolved job shop dispatching rules," *2016 IEEE Congress on Evolutionary Computation, CEC 2016*, pp. 3590–3597, 2016.

[2] J. Park, Y. Mei, S. Nguyen, G. Chen, and M. Zhang, "Investigating a machine breakdown genetic programming approach for dynamic job shop scheduling," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10781 LNCS, pp. 253–270, 2018.

[3] 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, 2015.

[4] 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.

[5] 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.

[6] H. Al-Sahaf, Y. Bi, Q. Chen, A. Lensen, Y. Mei, Y. Sun, B. Tran, B. Xue, and M. Zhang, "A survey on evolutionary machine learning," *Journal of the Royal Society of New Zealand*, vol. 49, no. 2, pp. 205–228, 2019.

[7] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A Classification of Hyper-Heuristic Approaches: Revisited," in *Handbook of Metaheuristics*, ser. International Series in Operations Research & Management Science. Springer International Publishing, 2018, pp. 453–477.

[8] J. R. Koza, "Genetic programming as a means for programming computers by natural selection," *Statistics and Computing*, vol. 4, no. 2, pp. 87–112, 1994.

[9] 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.

[10] K. Miyashita, "Job-shop scheduling with genetic programming," *GECCO 2000 - Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 505–512, 2000.

[11] D. Jakobović, "LNCS 3905 - Dynamic Scheduling with Genetic Programming," pp. 73–84, 2006.

[12] L. Nie, L. Gao, P. Li, and X. Li, "A GEP-based reactive scheduling policies constructing approach for dynamic flexible job shop scheduling problem with job release dates," *Journal of Intelligent Manufacturing*, vol. 24, no. 4, pp. 763–774, 2013.

[13] M. Durasević, D. Jakobović, and K. Knežević, "Adaptive scheduling on unrelated machines with genetic programming," *Applied Soft Computing Journal*, vol. 48, pp. 419–430, 2016.

[14] M. Brameier and W. Banzhaf, *Linear genetic programming*, 2007, vol. 53, no. 9.

[15] L. F. D. P. Sotto, V. V. de Melo, and M. P. Basgalupp, "λ -LGP: an improved version of linear genetic programming evaluated in the Ant Trail problem," *Knowledge and Information Systems*, vol. 52, no. 2, pp. 445–465, 2017.

[16] 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.

[17] S. Provorovs and A. Borisov, "Use of Linear Genetic Programming and Artificial Neural Network Methods to Solve Classification Task," *Scientific Journal of Riga Technical University. Computer Sciences*, vol. 45, no. 1, pp. 133–139, 2012.

[18] 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.

[19] C. Downey, M. Zhang, and W. N. Browne, "New crossover operators in linear genetic programming for multiclass object classification," *Proceedings of the 12th Annual Genetic and Evolutionary Computation Conference, GECCO '10*, pp. 885–892, 2010.

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

[21] T. Hu, J. L. Payne, W. Banzhaf, and J. H. Moore, "Robustness, evolvability, and accessibility in linear genetic programming," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6621 LNCS, pp. 13–24, 2011.

[22] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, *Genetic Programming with Adaptive Search Based on the Frequency of Features for Dynamic Flexible Job Shop Scheduling*. Springer International Publishing, 2020, vol. 12102 LNCS.

[23] 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(Early Access)*, pp. 1–15, 2021.

[24] D. Jakobović and L. Budin, "Dynamic Scheduling with Genetic Programming," *European Conference on Genetic Programming*, vol. 3905, pp. 73–84, 2006.

[25] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 5, pp. 621–639, 2013.

[26] C. W. Pickardt, T. Hildebrandt, J. Branke, J. Heger, and B. Scholz-Reiter, "Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems," *International Journal of Production Economics*, vol. 145, no. 1, pp. 67–77, 2013.

[27] J. Park, Y. Mei, S. Nguyen, G. Chen, and M. Zhang, "An investigation of ensemble combination schemes for genetic programming based hyper-heuristic approaches to dynamic job shop scheduling," *Applied Soft Computing Journal*, vol. 63, pp. 72–86, 2018.

[28] J. Park, S. Nguyen, M. Zhang, and M. Johnston, "A single population genetic programming based ensemble learning approach to job shop scheduling," *GECCO 2015 - Companion Publication of the 2015 Genetic and Evolutionary Computation Conference*, pp. 1451–1452, 2015.

[29] J. Park, Y. Mei, S. Nguyen, G. Chen, M. Johnston, and M. Zhang, "Genetic programming based hyper-heuristics for dynamic job shop scheduling: Cooperative coevolutionary approaches," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9594, pp. 115–132, 2016.

[30] J. Park, Y. Mei, G. Chen, and M. Zhang, "Niching genetic programming based hyper-heuristic approach to dynamic job shop scheduling: An investigation into distance metrics," *GECCO 2016 Companion - Proceedings of the 2016 Genetic and Evolutionary Computation Conference*, pp. 109–110, 2016.

[31] S. X. Wu and W. Banzhaf, "Rethinking multilevel selection in genetic programming," *Genetic and Evolutionary Computation Conference, GECCO'11*, pp. 1403–1410, 2011.

[32] C. Fogelberg, "Linear Genetic Programming for Multi-class Classification Problems," Ph.D. dissertation, 2005.

[33] T. Hu, J. L. Payne, W. Banzhaf, and J. H. Moore, "Evolutionary dynamics on multiple scales: A quantitative analysis of the interplay between genotype, phenotype, and fitness in linear genetic programming," *Genetic Programming and Evolvable Machines*, vol. 13, no. 3, pp. 305–337, 2012.

[34] T. Hu, W. Banzhaf, and J. H. Moore, "Robustness and evolvability of recombination in linear genetic programming," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7831 LNCS, pp. 97–108, 2013.

[35] W. Banzhaf, *Genetic programming: an introduction on the automatic evolution of computer programs and its applications*, 1998.

[36] 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.

[37] Y. Mei, S. Nguyen, and M. Zhang, "Constrained dimensionally aware genetic programming for evolving interpretable dispatching rules in dynamic job shop scheduling," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10593 LNCS, pp. 435–447, 2017.