A Further Investigation to Improve Linear Genetic Programming in Dynamic Job Shop Scheduling

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

***Abstract*—Dynamic Job Shop Scheduling (DJSS) is an important problem with many real-world applications. Genetic programming is a promising technique to solve DJSS, which automatically evolves dispatching rules to make real-time scheduling decisions in dynamic environments. Linear Genetic Programming (LGP) is a notable variant of genetic programming methods. Compared with Tree-based Genetic Programming (TGP), LGP has high flexibility of reusing building blocks and easy control of bloat effect. Due to these advantages, LGP has been successfully applied to various domains such as classification and symbolic regression. However, for solving DJSS, the most commonly used GP method is TGP. It is interesting to see whether LGP can perform well, or even outperform TGP in the DJSS domain. Applying LGP as a hyper-heuristic method to solve DJSS problems is still in its infancy. An existing study has investigated some basic design issues (e.g., parameter sensitivity and training and test performance) of LGP. However, that study lacks a comprehensive investigation on the number of generations and different genetic operator rates, and misses the investigation on register initialization strategy of LGP. To have a more comprehensive investigation, this paper investigates different generations, genetic operator rates, and register initialization strategies of LGP for solving DJSS. A further comparison with TGP is also conducted. The results show that sufficient evolution generations and initializing registers by diverse features are important for LGP to have a superior performance.**

***Index Terms*—Linear Genetic Programming, Dynamic Job Shop Scheduling, Hyper Heuristics.**

I. Introduction

Dynamic Job Shop Scheduling (DJSS) is a vital problem for real-world manufacturing. Different from static job shop scheduling, the information of the job shop and the to-be-solved jobs in DJSS are not exactly known before scheduling, since dynamic events can only be known until they happen incidentally during the execution. It requires problem solvers to not only optimize the schedule based on the currently available information, but more importantly, make instant reactions on dynamic events to repair or reschedule the existing solutions. In manufacturing, new job arrival [1] is one of the most common dynamic events. Nowadays, DJSS has shown a prosperous commercial value in many applications such as cloud computing [2] and railway management [3].

Because of the dynamic characteristics of DJSS, designing dispatching rules to make instant reactions for DJSS based on available information is a practical method. Dispatching rules are used to prioritize available jobs or machines for further decisions for every decision situation. Since they are applied to DJSS in a greedy manner, they have a much better tolerance to dynamic events than complete solutions of DJSS which may be greatly affected by dynamic events. There have been many manually designed dispatching rules for various DJSS applications, such as earliest due date [4] and weighted apparent tardiness cost [5]. However, these manually designed rules are not effective enough especially when facing complex DJSS problems. In recent years, many studies have shown that Genetic Programming (GP) is a promising method to automatically evolve dispatching rules [6].

Linear Genetic Programming (LGP) is a well-known GP variant. The similar idea of LGP has appeared as early as 1990s, proposed by Nordin [7], [8]. And it was subsequently developed into the existing LGP framework by Wolfgang [9]. Basically, there are two main characteristics of LGP, linear arrangement and sequential execution of instructions. The two characteristics define the key meaning of the term "linear". LGP has been successfully applied to various applications such as classification [10], symbolic regression [11], and automatical algorithm design [12].

LGP is applied to DJSS as a hyper-heuristic method. In DJSS, LGP-based hyper-heuristic is mainly used to evolve priority dispatching rules. A prior investigation of LGP in solving DJSS [13] attempted to investigate how different LGP parameter settings affect its effectiveness and rule size for evolving DJSS rules. The experimental studies found a proper setting for some parameters, but showed that LGP still performs slightly inferior to Tree-based Genetic Programming (TGP) in some scenarios with high utilization levels. Besides, the program sizes of LGP-based dispatching rules are slightly smaller than the ones of TGP.

Though the prior investigation recommended the settings of part of parameters, it has some limitations. Firstly, the number of generations and genetic operator rates are not comprehensively investigated in [13]. The existing investigation only investigated a small range of generations, from 50 to 200. The small number of generations in the prior investigation may not be sufficient enough to fully utilize the variation step size of LGP, especially when the variation step size of LGP genetic operators are limited to a small value [14]. Besides, the existing study investigated the number of generations and different rates of genetic operators separately. However

2022 IEEE Symposium Series on Computational Intelligence (SSCI) | 978-1-6654-8768-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/SSCI51031.2022.10022208

these two genetic operators may be dependent, since they keep the balance of exploration and exploitation cooperatively (i.e., large generations and mutation-dominated evolution can highlight the exploitation and vise versa). A comprehensive grid search on these settings is absent in the existing study.

Secondly, the initialization strategy of registers for DJSS has not been investigated in existing studies. In classification and symbolic regression problems, Brameier et al. [14] have shown that initializing registers by valuable input features significantly improves LGP performance. However, the number of features in DJSS is much larger than the recommended number of registers used in LGP. It is still an open question how to set the initial value of the registers which are much fewer than the number of features.

Thirdly, the comparison between TGP and LGP in the primary investigation does not compare the training time between LGP and TGP. Due to the variable-length representation, the number of fitness evaluations might not lead to fair comparison, since individuals with different sizes can have very different computational complexities, leading to very different training time.

To gain a better understanding on the behaviour of LGP, we aim to extend our prior investigation [13] and make a more comprehensive one on LGP to verify its performance by answering the following key research questions.

1) What are the proper settings of the number of generations and genetic operator rates for LGP in solving DJSS?
2) What is the suitable strategy of register initialization of LGP in solving DJSS?
3) What are the differences between TGP and the LGP with the fine-tuned parameter settings (particularly the number of generations and genetic operator rates) and register initialization strategy, for solving DJSS under the same amount of training time?

II. Background

A. Dynamic Job Shop Scheduling

This paper focuses on solving DJSS problems with new job arrival. The main characteristic of this type of DJSS problems is that there will be new jobs arriving to the job shop in real time. The information of these new arrival jobs is not exactly known until they arrive. All of these jobs form a job set $\mathbf{J}$ and are completed by a job shop. The job shop has a finite set of machines $\mathbf{M}$. All arrival jobs are stored in the buffers of corresponding machines respectively. Every job $j \in \mathbf{J}$ consists of a sequence of operations $\mathbf{O}$, an arrival time α_j, a due date d_j, and a weight ω_j which specifies the importance of j. There are l_j operations in the operation sequence $\mathbf{O} = (o_{j1}, ..., o_{jl_j})$. Every $o_{ji}(1 \leq i \leq l_j)$ is designated to a certain machine $\pi(o_{ji}) \in \mathbf{M}$ and it will be completed with a positive processing time $\sigma(o_{ji})$. The execution of o_{ji} must follow the execution order specified in $\mathbf{O}$, which means $o_{ji}(i > 1)$ can only be processed until $o_{j,i-1}$ is completed. The execution of o_{ji} is also assumed to be uninterruptable, and every machine can process at most one job for any time.

To verify the performance, this paper focuses on three performance metrics of DJSS. They are maximum tardiness for the whole simulation (T_{max}), mean tardiness (T_{mean}), and weighted mean tardiness (WT_{mean}). Denote the completion time of j as c_j, these three optimization objectives are formulated as follows.

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

B. Related Work

Genetic Programming-based Hyper-Heuristic (GPHH) is an emerging topic in dynamic scheduling. A lot of literature of GPHH are proposed to enhance the performance in solving DJSS in recent years [15], [16]. For example, Planinić et al. [17] developed an ϵ-Lexicase selection for GPHH which successfully improved the solution diversity and training convergence speed. Besides, to relief the time consumption of DJSS simulation, surrogate models are developed for GPHH [18], [19]. Zhang et al. [20], [21] also further integrated surrogate models with multitask learning and knowledge transfer to improve the training efficiency of GPHH. The generalization ability to unseen DJSS instances is also an important concern of GPHH. To improve the generalization ability, Durasević et al. [22] and Park et al. [23] respectively investigated the performance of GPHH enhanced by different ensemble methods. They found that summing up the decision from different rules is one of the most effective schemes to enhance GPHH performance. To improve the interpretability of dispatching rules evolved by GPHH, grammar and other kinds of domain knowledge are introduced to GPHH. For example, Planinić et al. [24] proposed a simplification method which consists of algebraic reduction and permutation-based pruning to simplify the outputted dispatching rules Mei et al. [25] also developed a dimension gap to measure the distance between the physical meaning of DJSS attributes to improve the interpretability of dispatching rules.

Although there are many different GP variants which have been successfully applied in other applications, the existing GPHH methods are mainly designed based on TGP, and only few of other GP variants are applied to dynamic scheduling. For example, gene expression programming [26], which encodes tree structures into a linear representation, is applied to dynamic scheduling by Nie et al. [27] and Xiao et al. [28]. Their results show gene expression programming has a similar effectiveness with TGP and has better interpretability.

LGP is one of notable GP variants. It has some advantages over TGP. For example, because of its linear representation, the search space of LGP is easily to be formulated. Based on the characteristics of LGP, Hu et al. [29], [30] took LGP as an example to make a comprehensive investigation on the robustness and evolvability of GP methods. LGP also shows superior performance to TGP in classification and symbolic

```
R[0] = R[1] − x0
R[2] = R[0] * R[0]
R[1] = min(R[2], x1)
R[0] = R[1] * R[0]
```

Fig. 1. An example program of LGP

regression. For example, because LGP individuals naturally have multiple outputs by defining multiple output registers, LGP is more ready to address multi-class classification than TGP whose individuals usually only have one output. The existing literatures validated the superior performance of LGP in multi-class classification [31]–[33]. Besides, LGP is successfully applied to symbolic regression [11], [34], [35] and automatical algorithm design [12].

Despite these encouraging achievements of LGP, applying LGP to solve dynamic scheduling is at an early stage. A prior investigation of LGP for DJSS is conducted by Huang et al. [13]. That investigation found some recommended parameter settings for LGP and that LGP can evolve more compact programs than tree-based GP. But it also shows that DJSS is still a very challenging problem for LGP, and LGP is less competitive than TGP in some scenarios in terms of test performance. This paper will make a further and deeper investigation in this direction.

III. Applying LGP to DJSS

This paper applies LGP as a hyper-heuristic method to search dispatching rules, which can minimize these three metrics mentioned in II-A respectively. It is detailed by the following sections.

A. Representation

Every LGP individual is an instruction sequence. The length of the instruction sequence ranges between specified minimum and maximum program length. Every instruction in the sequence is a register-based instruction which consists of a destination register, a function, and some source registers. All of the destination and source registers come from a predefined finite set of registers. Functions in instructions accept the values from source registers, and pass the calculation result to subsequent instructions by assigning results to destination registers. In execution, the registers are initialized at first. Then, all instructions are executed sequentially. The final output of LGP programs are outputted from pre-specified output registers. Fig. 1 is an example program of LGP, representing a heuristic $\min((1-x_0)^2, x_1) * (1-x_0)$ where x_0 and x_1 are the inputs and $R[i](i \in 0, 1, 2)$ are registers. All registers in the example are initialized as 1 (i.e., the first instruction in Fig. 1 can be seen as "$R[0] = 1 - x_0$"). The final output of the individual is outputted from $R[0]$.

B. Evolutionary Framework of LGP-based Hyper-Heuristic

The pseudo code of LGP for solving DJSS are shown in Alg. 1. First, an LGP population $\mathbf{P}$ is initialized. The fitness of LGP individuals is evaluated by a DJSS simulation. Then based on the fitness, an elitism selection is used to retain elite individuals to the next generation. In breeding, LGP parent individuals are selected by a tournament selection. These parents are varied to produce offspring by micro mutation, macro mutation, and crossover respectively based on the given probability. The offspring form the new population and the best-of-run individual is updated. To improve the generalization ability of dispatching rules, DJSS instances are also rotated for every generation. After generations of evolution, the best individual of the population is outputted.

Algorithm 1: The pseudo code of LGP for DJSS

Input: The settings of the DJSS simulation.
Output: The best-of-run LGP individual F^*.

Initialize the population $\mathbf{P}$;
Generate a DJSS instance I based on the settings of the simulation;
while *The stopping criteria are not satisfied* **do**
  Evaluate every individual F on I by a DJSS simulation;
  Parent individuals $F_1 = F_2 = null$, offspring $F' = F'' = null$;
  $\mathbf{P}' =$ individuals from an elitism selection on $\mathbf{P}$;
  while *size of* $\mathbf{P}' <$ *size of* $\mathbf{P}$ **do**
    Select F_1 and F_2 by a tournament selection on $\mathbf{P}$;
    $p \sim U(0, 1)$;
    if $p \leq$ *micro mutation rate* **then**
      Produce F' by micro mutation on F_1;
    else if $p \leq$ *micro mutation rate* $+$ *macro mutation rate* **then**
      Produce F' by macro mutation on F_1;
    else if $p \leq$ *micro mutation rate* $+$ *macro mutation rate* $+$ *crossover rate* **then**
      Produce F' and F'' by crossover operator on F_1 and F_2;
    else
      $F' = F_1$;
    $\mathbf{P}' = \mathbf{P}' \bigcup F'$;
    if *size of* $\mathbf{P}' <$ *size of* $\mathbf{P}$ **then**
      $\mathbf{P}' = \mathbf{P}' \bigcup F''$;
  $\mathbf{P} = \mathbf{P}'$;
  Update $F^* \in \mathbf{P}$;
  Rotate the random seed of I.
Return F^*.

C. Fitness Evaluation

An LGP individual is evaluated by a DJSS simulation. In the simulation, the LGP individual is regarded as a dispatching rule. The dispatching rule is used to prioritize available jobs or machines for making decisions. Specifically, for LGP-based dispatching rules, the registers in LGP individuals are re-initialized before every execution. Then, the instructions in LGP individuals are executed sequentially, and the final output is seen as the priority of a certain candidate decision. The decision with the best priority is then executed. Finally, the performance of the generated schedule is measured by a predefined objective function (i.e., T_{max} or T_{mean} or WT_{mean} in section II-A). The value of the objective function is regarded as the fitness of the LGP individual.

D. Genetic Operators

LGP has three basic kinds of genetic operators: micro mutation, macro mutation, and crossover. Micro mutation

aims to produce offspring by changing the primitives in LGP individuals. The term "micro" means this type of genetic operators does not change the total number of instructions, but just makes modifications inside an instruction. Macro mutation updates LGP individuals by inserting a randomly generated instruction into a parent individual or removing a random instruction from the parent. It always changes the total number of instructions. The main idea of crossover in LGP is very similar with the crossover in genetic algorithm, which produces offspring by swapping instruction segments from two different parent individuals. Specifically, this investigation adopts an effective micro mutation and an effective macro mutation [36] as the mutation operators, and uses a basic implement of LGP crossover, linear crossover [9], as the crossover operator. The effective micro and macro mutation ensure that the modification must have an effect on the final output, to encourage LGP offspring to have different behaviours from their parents. Noted that the variation step size of crossover is usually larger than the one of micro and macro mutation, since it is equivalent to removing a sequence of instructions and re-inserting another sequence.

IV. Experiment Design

A. Simulation Configuration

The configurations of the DJSS simulation are set according to the common settings of existing studies [18], [25]. Specifically, our job shop contains totally 10 machines. Operations in new arrival jobs are designated to these machines randomly and are completed by the designated machine with a continuous processing time ranging from 1 to 99. Every job has at least 2 operations and at most 10 operations. The new jobs arrive to the job shop based on a Poisson process whose utilization levels are set as 0.85 and 0.95 respectively in our experiment. The new jobs also have different weights based on a pre-defined probability. Specifically, 20% have a weight of 1, 60% have a weight of 2, while the rest have a weight of 4. Our investigation mainly focuses on the steady-state performance of DJSS simulation, in which there are 1000 warm-up jobs in the experiment and only the subsequent 5000 jobs are counted in optimization objectives. Based on the three optimization objectives and the two utilization levels, there are totally six scenarios in the experiments, denoted as "$\langle A, B\rangle$" where "A" is the objective and "B" is the utilization level such as $\langle T_{max}, 0.95\rangle$. Every DJSS scenario has 30 independent runs. The outputted dispatching rules from different runs are tested on 50 unseen instances.

B. Parameter Settings

Based on the investigation in [13], we set the default values of LGP parameters in following experiments as follows. The number of registers is 6. Every LGP individual initially has 1 to 10 instructions, and can have at most 64 instructions and at least 1 instructions during evolution. For micro mutation, the mutation rates of different primitive types (i.e., function, constant, destination and source registers) are set as $\mathrm{fun} : \mathrm{destin} : \mathrm{cons} : \mathrm{source} = 50\% : 25\% : 12.5\% : 12.5\%$. For macro mutation, the insertion and deletion rate are set as 67% and 33% respectively. For the linear crossover, the maximal length of two crossover segments is 30, the maximal length difference of the two segments is 5, and the maximal distance of the two crossover points is 30. Besides the three genetic operators, a reproduction whose rate is 10% is also adopted in the evolution. The tournament size in tournament selection is set as 7 and the elitism rate is 1%. There are totally 51200 simulations by default. Besides, LGP in this investigation uses a function set including $\{+, -, \times, \div, \max, \min\}$ where $\div$ returns 1.0 if denominator equals to 0.0, and uses sixteen DJSS attributes as the terminal set, as shown in Table I.

TABLE I
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 operation
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	relative flow due date from now
rDD	relative due date from now
W	the weight of the job
TIS	time in the system since the arrival
SL	slack

V. Results and Analyses

A. Investigation on Generations and Evolution Settings

To properly set the number of generations and genetic operator rates, we make a grid search on the two parameters. Intuitively, when the number of generations is large and mutation dominates the evolution, LGP population can have a more thorough search in the neighbor region of existing solutions. Elite solutions can be further elaborated by small changes. On the other hand, when there are a small number of generations and crossover dominates the evolution, LGP population can search more different regions in solution space easily, and thus has a high degree of exploration. Here, we select two settings of generations, 50 and 400. To retain the same number of total simulations, the population size of LGP is adjusted accordingly (total number of simulations divided by the number of generations). Besides, we compare two crossover-dominated and mutation-dominated settings. Denoting micro mutation rate as θ_{micro}, macro mutation rate as θ_{macro}, and crossover rate as θ_{cross}, we set $\theta_{micro} : \theta_{macro} : \theta_{cross} = 10\% : 10\% : 70\%$ for crossover-dominated settings, and set $\theta_{micro} : \theta_{macro} : \theta_{cross} = 30\% : 30\% : 30\%$ for mutation-dominated settings.

The results of the grid search are shown in Table II. A Wilcoxon rank-sum test with a significance level of

TABLE II
MEAN (STANDARD DEVIATION) TEST PERFORMANCE OF LGP WITH DIFFERENT GENERATIONS AND GENETIC OPERATOR RATES

Scenarios		generations=50 popsize=1024	generations=400 popsize=128
$\langle T_{max}, 0.85\rangle$	crossover-dominated	2026.2(86.61)	1928.61(41.68)+
$\langle T_{max}, 0.95\rangle$		4179.05(231.24)	4003.52(119.78)+
$\langle T_{mean}, 0.85\rangle$		419.14(3.22)	419.91(6.11)≈
$\langle T_{mean}, 0.95\rangle$		1120.39(9.18)	1115.7(8.62)≈
$\langle WT_{mean}, 0.85\rangle$		728.94(7.26)	727.55(7.1)≈
$\langle WT_{mean}, 0.95\rangle$		1746.12(24.47)	1735.12(32.14)≈
$\langle T_{max}, 0.85\rangle$	mutation-dominated	1987.46(60.36)+	1953.52(59.54)+
$\langle T_{max}, 0.95\rangle$		4109.88(150.49)≈	3999.86(158.62)+
$\langle T_{mean}, 0.85\rangle$		418.15(2.43)≈	417.16(2.95)+
$\langle T_{mean}, 0.95\rangle$		1117.89(6.24)≈	1116.51(10.76)≈
$\langle WT_{mean}, 0.85\rangle$		730.6(7.03)≈	726.32(8.02)≈
$\langle WT_{mean}, 0.95\rangle$		1753.39(28.34)≈	1725.95(26.13)+

TABLE III
MEAN (STANDARD DEVIATION) TEST PERFORMANCE OF DIFFERENT GENERATION SETTINGS WITH MUTATION-DOMINATED LGP

Scenarios	generations=50 popsize=1024	generations=200 popsize=256	generations=400 popsize=128	generations=800 popsize=64
$\langle T_{max}, 0.85\rangle$	2026.2(86.61)	1918.58(39.69)+	1953.52(59.54)+	1943.39(54.17)+
$\langle T_{max}, 0.95\rangle$	4179.05(231.24)	3995.55(173.69)+	3999.86(158.62)+	3977.28(120.35)+
$\langle T_{mean}, 0.85\rangle$	419.14(3.22)	418.14(4.58)+	417.16(2.95)+	419.89(5.16)≈
$\langle T_{mean}, 0.95\rangle$	1120.39(9.18)	1116.5(11.7)≈	1116.51(10.76)≈	1120.11(16.63)≈
$\langle WT_{mean}, 0.85\rangle$	728.94(7.26)	727.13(6.78)≈	726.32(8.02)≈	728.02(11.15)≈
$\langle WT_{mean}, 0.95\rangle$	1746.12(24.47)	1733.26(26.61)+	1725.95(26.13)+	1735.83(31.43)≈

0.05 is also applied to these results. The notation "+", "−", and "≈" respectively denote the compared algorithm is significantly better than, significantly worse than, or similar with the baseline (i.e., generation=50 and crossover-dominated in this experiment). It can be observed that LGP has a significantly improvement when exploitation is highlighted. When generations equal to 400 and mutation-dominated evolution is adopted, LGP is significantly better than "(crossover-dominated, generations=50)" in four of the six scenarios. Besides, if looking at the "crossover-dominated" row and "generations=50" column respectively, we can see that increasing exploitation, no matter by increasing generations to allocate more training resources for convergent phase or letting mutation dominate evolution, is helpful to LGP. They significantly improve two or one scenarios respectively.

To identify a proper setting of generations for LGP, we make a further investigation on the number of generations. Two generation settings, 200 and 800, are conducted based on the mutation-dominated setting. Table III shows that when the number of generations equals to 200 or 400, LGP is significantly better than the baseline setting (i.e., "generations=50") in four of the six scenarios. On the other hand, extremely increasing the number of generations of LGP has a negative impact on the performance. LGP with 800 generations only has a superior performance to the baseline in the two T_{max} scenarios. Thus, the number of generations of LGP is suggested to set as 200 or 400 to strike a good exploration-and-exploitation balance.

However, increasing generations increases the training time. As shown in Table IV, the training time grows up consistently from "generation=50" to "generation=800". The main reason of the increasing training time is the increase of program size.

TABLE IV
MEAN (STANDARD DEVIATION) TRAINING TIME OF DIFFERENT GENERATIONS WITH MUTATION-DOMINATED LGP (SECONDS)

Scenarios	generations=50 popsize=1024	generations=200 popsize=256	generations=400 popsize=128	generations=800 popsize=64
$\langle T_{max}, 0.85\rangle$	2483.7(65)	4129.9(115.8)	4360.1(67.8)	4449.9(55.6)
$\langle T_{max}, 0.95\rangle$	5641.9(209.9)	10089.4(228.1)	11311.3(246.3)	11710.7(234.2)
$\langle T_{mean}, 0.85\rangle$	2262.8(41.1)	3870.1(117.3)	4050.2(75.7)	4081.6(73.8)
$\langle T_{mean}, 0.95\rangle$	4782.7(121.2)	9391.6(341)	9792.1(366.2)	10580.6(443.6)
$\langle WT_{mean}, 0.85\rangle$	2289.9(58.8)	4341.6(138.7)	4258.4(74.3)	4478.2(113)
$\langle WT_{mean}, 0.95\rangle$	4928.4(174.4)	9087.3(310.6)	9550.9(238.8)	9758.1(297)

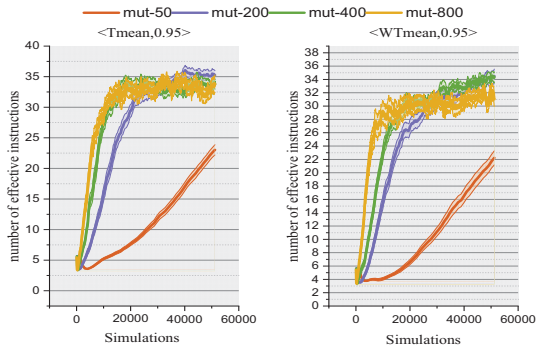

Fig. 2. Average program sizes of different settings over generations

When LGP evolves a large population for a small number of generations, there is few long individuals in LGP population because of the limited number of generations and limited variation step size. Contrarily, LGP individuals grow long enough after generations of evolution. Long dispatching rules inevitably increase decision time and make DJSS simulation more time consuming.

To verify the program size increase, we compare the average program size of LGP population over generations with different generation settings. Specifically, we denote the number of effective instructions in an individual as the program size of LGP and use "mut-X", "mut" denoting mutation and X for generation settings, to denote evolution settings. As shown in Fig. 2, the curves of small generations climb up much slower than the ones of large generations. Based on the results, it is recommended to set the number of generations as 200 and use mutation-dominated settings to balance the training efficiency and test performance.

B. Investigation on Register Initialization Strategy

This section investigates the effectiveness of different register initialization strategies. In previous experiment, we initialize the six registers by the first six DJSS attributes in Table I in default. However, the first three attributes are machine related, which is helpless in prioritizing operations in a same machine queue. To find an effective register initialization strategy, we compare three strategies in this investigation. First, the default initialization strategy acts as the baseline. For the second strategy, we design two principles. The initialization should 1) be as diverse as possible and 2) use job-related attributes. This strategy encourages

TABLE V
MEAN (STANDARD DEVIATION) TEST PERFORMANCE OF DIFFERENT REGISTER INITIALIZATION STRATEGIES

Scenarios	Default	DivJob	All-ones
$\langle T_{max}, 0.85\rangle$	1953.52(59.54)	1914.35(34.99)+	1952.21(45.28)≈
$\langle T_{max}, 0.95\rangle$	3999.86(158.62)	3973(135.48)≈	3963.31(81.94)≈
$\langle T_{mean}, 0.85\rangle$	417.16(2.95)	418.04(3.88)≈	418.01(4.03)≈
$\langle T_{mean}, 0.95\rangle$	1116.51(10.76)	1112.16(8.65)≈	1117.18(13.95)≈
$\langle WT_{mean}, 0.85\rangle$	726.32(8.02)	727.72(6.04)≈	728.45(8.47)≈
$\langle WT_{mean}, 0.95\rangle$	1725.95(26.13)	1737.68(39.06)≈	1725.63(27.92)≈

dispatching rules to consider different information of available jobs and ensures that the initial values are at least helpful in distinguishing operations. Based on the domain knowledge, the attribute pairs such as WKR and NOR, rFDD and rDD, WINQ and NINQ, are respectively regarded as similar attributes. So, we simply put one of the attributes in these pairs at the font of the list and leave the other to the end part. The attribute list in Table. I is thus rearranged into

$$\{\text{PT}, \text{NPT}, \text{WINQ}, \text{WKR}, \text{rFDD}, \text{OWT}, \text{NOR}, \text{NINQ}, \text{W}, \text{rDD}, \text{NWT}, \text{TIS}, \text{SL}, \text{NIQ}, \text{WIQ}, \text{MWT}\}.$$

The first k attributes are used to initialize k registers respectively. This diverse job-related attribute initialization is denoted as "DivJob". Third, we initialize all registers as "1", which is a simplest way for initialization. Other experiment settings follow Section IV-B and mutation-dominated settings, evolving 400 generations.

The test performance of the three initialization strategies are shown in Table V. Overall, the test performance of the three strategies are very similar, which means LGP is relatively robust to register initialization in terms of test performance. Nevertheless, DivJob still significantly outperforms the other two initialization strategies on one scenario. Besides, DivJob also has a better mean performance than the default initialization on two of the three scenarios with high utilization levels.

To further investigate the effectiveness of initialization strategies, the training performance is also compared. Fig. 3 shows the test performance of the best dispatching rule of every generation. As shown in Fig. 3, DivJob (i.e., the red curves) averagely drops down faster and deeper than the other two initialization strategies in the two T_{max} scenarios. Besides, for the two T_{mean} scenarios, DivJob has a relatively fast convergence speed at the beginning and has a more stable performance (i.e., narrower bias range) at the final stage of evolution. Though in WT_{mean} scenarios, DivJob has a slower convergence speed than the others at the beginning of evolution, DivJob can still converge to a similar level with other initialization strategies before half of the evolution. Based on these results, initializing registers as various job-related attributes is helpful for LGP evolution.

Given that the number of registers is often smaller than the number of DJSS attributes, we also conduct an investigation on the suitable number registers based on the newly recommended initialization strategy. The comparison of average test performance is shown in Table VI. The default setting (i.e., 6 registers) is regarded as the baseline. Basically, the test performance of different number of registers is similar in most cases. However, when the number of registers is set as 8, LGP outperforms the default setting in $\langle WT_{mean}, 0.85\rangle$. Besides, LGP with 8 registers can also have relatively small mean test performance in most high-utilization-level scenarios. Based on the results, 8 registers is a recommended setting.

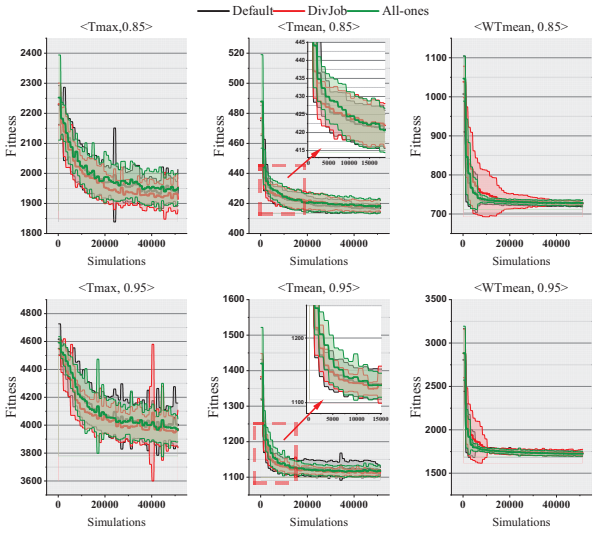

Fig. 3. Test performance over generations

TABLE VI
MEAN (STANDARD DEVIATION) TEST PERFORMANCE OF DIFFERENT NUMBER OF REGISTERS

Scenarios \ #registers	4	8	10	12
$\langle T_{max}, 0.85\rangle$	1918.65(29.99)≈	1927.03(50.67)≈	1944.89(53.67)−	1943.65(63.67)≈
$\langle T_{max}, 0.95\rangle$	3974.87(154.94)≈	3938.6(117.75)≈	3975.27(102.66)≈	3993.53(168.22)≈
$\langle T_{mean}, 0.85\rangle$	418.63(3.98)≈	418.06(4.27)≈	416.43(3.18)≈	417.84(2.26)≈
$\langle T_{mean}, 0.95\rangle$	1115.59(9.97)≈	1115.77(9.37)≈	1115.55(11.13)≈	1116.11(9.92)≈
$\langle WT_{mean}, 0.85\rangle$	724.42(7.1)≈	721.59(6.02)+	726.69(6.51)≈	726.75(6.95)≈
$\langle WT_{mean}, 0.95\rangle$	1725.07(22.07)≈	1729.49(22.2)≈	1731.1(20.1)≈	1742.57(30.17)≈

*The test performance of 6 registers with new register initialization strategy is referred to DivJob in Table V.

C. Comparison with TGP

Based on the results of prior sections, we set the parameters of LGP as Table VII and make a comparison between TGP and LGP. To improve the training efficiency of LGP, we reduce the maximum program length to 50 and use smaller variation step size in linear crossover. The parameters of TGP are set based on the recommended parameters of [25]. Specifically, TGP in the experiment has a population of 1024 individuals and evolves 50 generations. The results are shown in Table VIII. It can be observed that LGP is significantly better than TGP in two scenarios, in terms of test performance. Besides, LGP has better mean performance than TGP in most of other scenarios. However, the training time of LGP is much longer than the ones of TGP.

A further comparison is made to see what is the performance difference between TGP and LGP within a similar training

TABLE VII
KEY PARAMETER SETTINGS OF LGP IN THE COMPARISON WITH TGP

Parameters	Values
generation	200
population size	256
number of registers	8
initiali program size	[1,10]
program size	[1,50]
$\theta_{micro} : \theta_{macro} : \theta_{cross}$	30%:30%:30%
register initialization strategy	DivJob

TABLE VIII
MEAN (STANDARD DEVIATION) TEST PERFORMANCE AND TRAINING TIME OF TGP AND LGP

Scenarios	TGP		LGP	
	Fitness	Training time	Fitness	Training time
$\langle T_{max}, 0.85\rangle$	1931.04(44.39)	3036.1(81.9)	1931.14(37.22)≈	3483.5(52.4)
$\langle T_{max}, 0.95\rangle$	4105.46(193.07)	6386.1(209.4)	3974.2(112)+	8759.4(205.4)
$\langle T_{mean}, 0.85\rangle$	417.34(2.95)	2398.6(67)	417.3(2.15)≈	3480.7(104.9)
$\langle T_{mean}, 0.95\rangle$	1115.64(10.99)	4543.8(131.1)	1115.53(9.45)≈	7727.3(252)
$\langle WT_{mean}, 0.85\rangle$	727.93(9.14)	2544.7(80.9)	723.96(6.35)≈	3273.6(72.4)
$\langle WT_{mean}, 0.95\rangle$	1745.24(25.5)	4687.7(116)	1728.6(24.1)+	8132.6(279.9)

time. To answer this extended question, we respectively evolve TGP with 70 and 100 generations. The comparison in terms of test performance and training time are shown in Table. IX. Specifically, TGP with 70 and 100 generations are denoted as "TGP70" and "TGP100" respectively. Table IX shows that increasing the number of generations of TGP can improve the performance of TGP. However, the performance of TGP70 and TGP100 are still similar with the one of LGP. Besides, increasing the number of generations also increases the training time of TGP. The two versions of TGP both have longer training time than LGP in most scenarios.

To verify the training performance of LGP, the test performance over generations of TGP, TGP70, TGP100, and LGP are also compared. As shown in Fig. 4, LGP (i.e., red curves) has a very competitive performance with the other three compared methods. Besides, LGP also drops down much faster than its adversaries in $\langle T_{max}, 0.95\rangle$.

The program sizes of outputted dispatching rules are compared in Fig. 5. The number of tree nodes after simplification is denoted as the program size of TGP rules, and the number of effective instructions multiplied by a factor of 2.0 is denoted as the size of LGP rules. We see that LGP has a lower (or at least competitive) distribution of program size than (with) TGP, which means LGP is more likely to evolve more compact programs than TGP methods.

The results verify that within the same number of simulations, LGP has a significantly better learning ability than TGP (i.e, better effectiveness and more compact programs). The training efficiency and test performance of LGP are also very competitive with TGP if given the same training resources.

VI. CONCLUSIONS

This paper aims to extend the prior investigation by answering three critical research questions for solving DJSS using LGP. These questions have been successfully answered by the empirical studies and analyses. Firstly, we make a grid search on different generations and genetic operator rates, and compare their training and test performance. The results show that LGP is suitable to large generations and mutation-

TABLE IX
MEAN (STANDARD DEVIATION) TEST PERFORMANCE AND TRAINING TIME OF TGP WITH 70 AND 100 GENERATIONS AND LGP

Test performance			
Scenarios	TGP70	TGP100	LGP
$\langle T_{max}, 0.85\rangle$	1918.87(41.5)≈	1920.31(50.86)≈	1931.14(37.22)
$\langle T_{max}, 0.95\rangle$	4012.22(88.63)≈	3985.49(85.69)≈	3974.2(112)
$\langle T_{mean}, 0.85\rangle$	416.8(3.18)≈	416.03(3.04)≈	417.3(2.15)
$\langle T_{mean}, 0.95\rangle$	1112.65(11.19)≈	1114.02(15.64)≈	1115.53(9.45)
$\langle WT_{mean}, 0.85\rangle$	726.09(5.52)≈	725.21(6.58)≈	723.96(6.35)
$\langle WT_{mean}, 0.95\rangle$	1730.11(27.46)≈	1726.83(27.84)≈	1728.6(24.1)
Training time (seconds)			
$\langle T_{max}, 0.85\rangle$	4868.7(185)	7714.2(260.7)	3483.5(52.4)
$\langle T_{max}, 0.95\rangle$	10356.5(313.4)	15809.1(514.5)	8759.4(205.4)
$\langle T_{mean}, 0.85\rangle$	4014.8(80.9)	5698.9(156.5)	3480.7(104.9)
$\langle T_{mean}, 0.95\rangle$	7695.1(280.9)	11653.8(391)	7727.3(252)
$\langle WT_{mean}, 0.85\rangle$	3948.5(90.5)	5507.1(124.6)	3273.6(72.4)
$\langle WT_{mean}, 0.95\rangle$	7692.7(294.5)	11637.7(386.3)	8132.6(279.9)

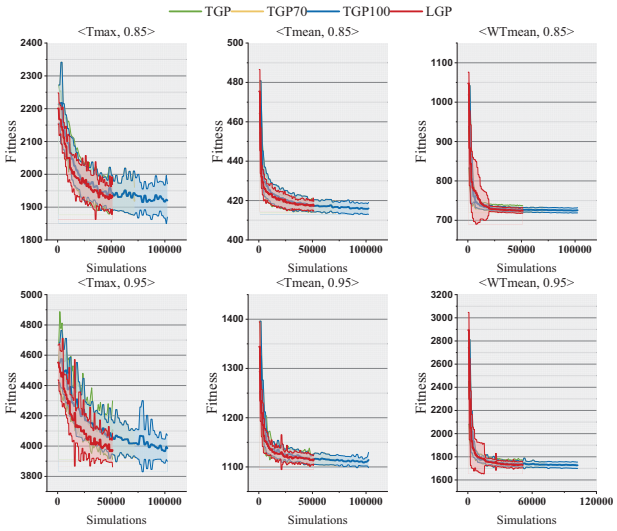

Fig. 4. Test performance over generations of TGP and LGP

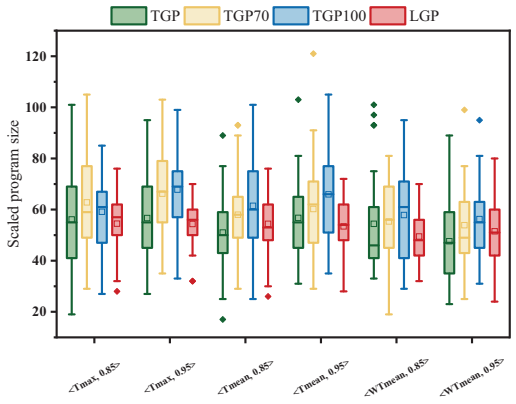

Fig. 5. Scaled program sizes of TGP with different generations and LGP

dominated settings. Based on the results, a recommended setting is also concluded to balance training efficiency and test performance. For the second research question, we propose three initialization strategies and compare their performance. The results verify that initializing registers as diverse job-related attributes is helpful for LGP evolution. It not only significantly improves the training and test performance in specific cases, but also performs competitively with other initialization strategies in other scenarios. Thirdly, a comparison between TGP and LGP is conducted to answer the third research question. The comparison validates the superior learning ability of LGP for solving DJSS problems. We expect this investigation provides a clearer benefit to scheduling and combinatorial optimization community and represents a starting point for LGP to be applied in this area. More work should be done in this direction in the future, to promote the application of LGP in combinatorial optimization.

REFERENCES

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

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

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

[10] 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.

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

[12] L. F. d. P. Sotto, V. V. d. 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.

[13] Z. Huang, Y. Mei, and M. Zhang, "Investigation of linear genetic programming for dynamic job shop scheduling," in *Proceedings of the IEEE Symposium Series on Computational Intelligence*, December 2021.

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

[15] M. Sanchez, J. M. Cruz-Duarte, J. C. Ortiz-Bayliss, H. Ceballos, H. Terashima-Marin, and I. Amaya, "A Systematic Review of Hyper-Heuristics on Combinatorial Optimization Problems," *IEEE Access*, vol. 8, pp. 128 068–128 095, 2020.

[16] F. Zhang, S. Nguyen, Y. Mei, and M. Zhang, *Genetic Programming for Production Scheduling: An Evolutionary Learning Approach*. Singapore: Springer, 2021. DOI:10.1007/978-981-16-4859-5.

[17] L. Planinić, M. Durasević, and D. Jakobović, "On the Application of ϵ-Lexicase Selection in the Generation of Dispatching Rules," in *IEEE Congress on Evolutionary Computation*, 2021, pp. 2125–2132.

[18] T. Hildebrandt and J. Branke, "On using surrogates with genetic programming," *Evolutionary Computation*, vol. 23, no. 3, pp. 343–367, 2015.

[19] S. Nguyen, M. Zhang, and K. C. Tan, "Surrogate-Assisted Genetic Programming with Simplified Models for Automated Design of Dispatching Rules," *IEEE Transactions on Cybernetics*, vol. 47, no. 9, pp. 2951–2965, 2017.

[20] 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.

[21] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Collaborative Multifidelity-Based Surrogate Models for Genetic Programming in Dynamic Flexible Job Shop Scheduling," *IEEE Transactions on Cybernetics*, pp. 1–15, 2021.

[22] M. Durasević and D. Jakobović, "Comparison of ensemble learning methods for creating ensembles of dispatching rules for the unrelated machines environment," *Genetic Programming and Evolvable Machines*, vol. 19, no. 1-2, pp. 53–92, 2018.

[23] 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.

[24] L. Planinić, M. Durasević, and D. Jakobović, "Towards Interpretable Dispatching Rules: Application of Expression Simplification Methods," in *IEEE Symposium Series on Computational Intelligence*, December 2021.

[25] Y. Mei, S. Nguyen, and M. Zhang, "Constrained dimensionally aware genetic programming for evolving interpretable dispatching rules in dynamic job shop scheduling," in *Asia-Pacific Conference on Simulated Evolution and Learning*, vol. 10593, 2017, pp. 435–447.

[26] C. Ferreira, "Gene Expression Programming: a New Adaptive Algorithm for Solving Problems," *Complex Systems*, vol. 13, no. 2, pp. 87–129, 2001.

[27] 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.

[28] Q. Xiao, J. Zhong, L. Feng, L. Luo, and J. Lv, "A Cooperative Coevolution Hyper-Heuristic Framework for Workflow Scheduling Problem," *IEEE Transactions on Services Computing*, vol. 15, no. 1, pp. 150–163, 2019.

[29] 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.

[30] T. Hu, W. Banzhaf, and J. H. Moore, "Robustness and evolvability of recombination in linear genetic programming," in *European Conference on Genetic Programming*, vol. 7831, 2013, pp. 97–108.

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

[32] 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.

[33] 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.

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

[35] 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.

[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.