





# Toward Evolving Dispatching Rules with Flow Control Operations by Grammar-guided Linear Genetic Programming

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

**Abstract**—Linear genetic programming (LGP) has been successfully applied to dynamic job shop scheduling (DJSS) to automatically evolve dispatching rules. Flow control operations are crucial in concisely describing complex knowledge of dispatching rules, such as different dispatching rules in different conditions. However, existing LGP methods for DJSS have not fully considered the use of flow control operations. They simply included flow control operations in their primitive set, which inevitably leads to a huge number of redundant and obscure solutions in LGP search spaces. To move one step toward evolving effective and interpretable dispatching rules, this paper explicitly considers the characteristics of flow control operations via grammar-guided linear genetic programming and focuses on IF operations as a starting point. Specifically, this paper designs a new set of normalized terminals to improve the interpretability of IF operations and proposes three restrictions by grammar rules on the usage of IF operations: specifying the available inputs, the maximum number, and the possible locations of IF operations. The experiment results verify that the proposed method can achieve significantly better test performance than state-of-the-art LGP methods and improves interpretability by IF-included dispatching rules. Further investigation confirms that the explicit introduction of IF operations helps effectively evolve different dispatching rules according to their decision situations.

**Index Terms**—Grammar-guided Linear Genetic Programming, Flow Control Operations, Dynamic Job Shop Scheduling, Hyper Heuristics.

## I. INTRODUCTION

Automatically designing effective dispatching rules for dynamic job shop scheduling (DJSS) has great value on both commercial and academic sides. Many real-world problems, which have to face dynamic events during optimization, such as traffic management [1] and airport management [2], are intrinsically DJSS problems. On the other hand, the dynamic events during optimization and the symbolic search space both make the dispatching rule design very complicated and challenging.

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

The authors are with the Centre for Data Science and Artificial Intelligence & School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand (E-mail: zhixing.huang@ecs.vuw.ac.nz; yi.mei@ecs.vuw.ac.nz; fangfang.zhang@ecs.vuw.ac.nz; mengjie.zhang@ecs.vuw.ac.nz).

Flow control operations are important in designing dispatching rules [3]–[5]. For example, DJSS problems need IF operations to prioritize energy-efficient jobs if the power cost rate is high and prioritize other jobs otherwise. Moreover, many human programs and expertise knowledge need flow control operations to describe their complex procedure. Finding an effective way to evolve flow control operations in dispatching rules facilitates humans to make use of domain knowledge and improve the flexibility of dispatching rules.

Genetic programming (GP) is an evolutionary computation method that searches symbolic solutions for an optimization problem [6]. It has been widely applied to a wide spectrum of applications such as classification [7], [8], regression [9], [10], and program synthesis [11]. In the last decade, there have been extensive studies applying genetic programming-based hyper heuristic (GPHH) techniques to help humans design dispatching rules for scheduling problems [12], [13].

Linear genetic programming (LGP) is a special variant of GP whose solutions are represented by a list of register-based instructions [14]. LGP can easily reuse common building blocks and adapt real-world programming skills into its search space because of its linear representation. Existing studies have shown the superior performance of linear genetic programming-based hyper heuristic (LGPHH) in designing dispatching rules for DJSS problems over tree-based GP methods [15], [16].

However, existing studies of LGPHH for DJSS did not effectively evolve flow control operations, mainly due to the following three challenges.

- 1) *Dimension inconsistency*: flow control operations likely use input features with different physical dimensions (e.g., if “3 meters” is larger than “2 seconds”). The inconsistent dimension makes dispatching rules difficult to be understood.
- 2) *Inactive sub-rules*: flow control operations easily lead to inactive sub-rules (i.e., introns [14]). For example, the contradictory conditions (i.e., conditions that are always false) of IF operations easily skip a large number of instructions, which makes a dispatching rule quite naive. The variation on flow control operations also likely makes a huge difference in the behaviors of dispatching rules since the variation often substantially changes the data flow in dispatching rules.
- 3) *Ineffective sub-rules*: the effectiveness of flow control

operations is highly dependent on their sub-rules. A flow control operation is useful only when their sub-rules are effective. To ensure the effectiveness of flow control operations, we have to take the effectiveness of sub-rules into consideration.

Existing studies of LGP did not fully consider these three challenges when evolving flow control operations. They simply introduce flow control operations into their primitive set and neglect the characteristics of flow control operations. This inevitably leads to many redundant and obscure solutions in the LGP search space, which impairs the search effectiveness and efficiency of LGP.

This paper focuses on IF operations as a step towards evolving flow control operations in dispatching rules. IF operations are the basis of many other flow control operations. Maximum functions and WHILE loops both inherently need IF operations to perform logical decisions. Investigating IF operations inspires the evolution of many other flow control operations. Specifically, this paper proposes to enhance LGPHH with IF operations by grammar-guided linear genetic programming [17]. The proposed method restricts the usage of IF operations in dispatching rules by grammar, to get rid of unreasonable input features and fragile and less-effective IF branches. Specifically, this paper has four main contributions:

- 1) To address the dimension inconsistency, we design a set of normalized terminals for DJSS and restrict that IF operations can only compare the proposed normalized terminals with constants. By this means, information in all different dimensions is used in a normalized form, which is easier for humans to understand.
- 2) To address the inactive sub-rules of IF operations, we design a set of grammar rules to constrain the number and possible positions of IF operations. We restrict dispatching rules to only use IF operations at the beginning of rules with a limited number. This limits the negative impact caused by inactive sub-rules of IF operations.
- 3) To address the ineffective sub-rules, we coordinate the grammar rules for different parts of a dispatching rule, including IF-included parts and the rest of it. By designing grammar rules for different parts in a coordinating manner, we improve the effectiveness of flow control operations.
- 4) We made an empirical investigation on IF-included dispatching rules for solving DJSS problems. The investigation verifies that IF operations are necessary and effective for complex DJSS scenarios.

## II. BACKGROUND

### A. Dynamic Job Shop Scheduling

A job shop has a set of machines  $\mathbb{M}$ , each with an available operation queue  $q(m)$ . The job shop accepts a set of jobs  $\mathbb{J}$  and processes them by the machines. A job  $j$  consists of a sequence of operations  $\mathcal{O}_j$ .  $\mathcal{O}_j = \{o_{j1}, \dots, o_{ji}, \dots, o_{jk_j}\}$  where  $k_j$  is the number of operations in job  $j$ . The operation sequence specifies the execution order of operations in each job (e.g.,  $o_{j1}$  must be executed prior to  $o_{j2}$ ). When an operation  $o_{ji}$  is available, it enters the corresponding operation queue and is

processed based on its priority by the machine.  $o_{ji}$  is removed from  $q(m)$  when it is processed, and its next operation  $o_{j,i+1}$ , if it exists, will enter the corresponding machine queue after  $o_{ji}$  is finished. The main task in job shop scheduling is to sequence the execution order of operations on each machine so that the job shop performance can be optimized.

One of the most distinctive features of DJSS problems is that there are dynamic events during optimization which greatly affect the performance of existing schedules. To ensure the performance of job shops, we have to make an instant reaction to make a new schedule or adjust the existing one. Specifically, this paper focuses on DJSS with new job arrival. Each job arrives at the job shop at time  $\alpha_j$ . The job shop does not know the information about new jobs until they arrive. Each operation  $o_{ji}(1 \leq i \leq k_j)$  is processed by a certain machine  $m$  with a given processing time  $p(o_{ji})$ , and each machine processes at most one operation at any time. Each job has a weight of  $\omega_j$  and a due date  $d_j$ . When a job is completed, we record the job completion time  $c_j$  to evaluate the performance of DJSS problems. Specifically, we take tardiness and flowtime of DJSS problems as the performance metrics. Tardiness denotes the delay of the job completion time  $c_j$  from the given due date  $d_j$ . Flowtime denotes the total time consumption of a job from completion time  $c_j$  to arrival time  $\alpha_j$ .

Dispatching rules are commonly used to schedule the new coming jobs in DJSS, which enables job shops to make instant decisions [3]. Specifically, a dispatching rule estimates the priority of all the available operations. When machines turn idle, they process the operations with the smallest (i.e., most preferred) priority value. However, designing effective dispatching rules for different DJSS scenarios is non-trivial as it requires a lot of expertise. Therefore, we use genetic programming-based hyper heuristics to automatically design dispatching rules for us [12], [13]. More specifically, this paper uses linear genetic programming [14], [18], an important GP variant, to evolve dispatching rules, which has shown very promising performance in existing studies [15], [16].

### B. Related Work

1) *Flow Control Operations in GPHH for DJSS*: Flow control operations decide which parts of a dispatching rule can be executed based on the input or program context. There are many different implementations of flow control operations in existing GPHH methods for DJSS. For example, GP-3 [4] fixed IF at the output of dispatching rules and designed an IF-included template to explicitly divide the dispatching rule into three sub-rules, one for scheduling bottleneck machines, one for scheduling non-bottleneck machines, and one for detecting bottleneck machines. If the dispatching rule detects a machine as a bottleneck in the job shop, the job shop uses the bottleneck-machine dispatching rule to make decisions and uses the non-bottleneck-machine dispatching rule otherwise. Đurasević et. al. [19] used a unary flow control operation which returns the operand if the operand is larger than 0 and returns 0 otherwise to control program execution flow. Hildebrandt et.al. [20] and Christopher et al.

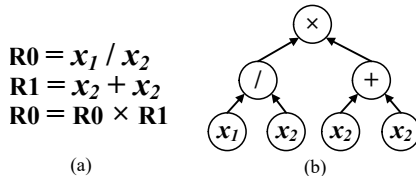


Fig. 1. (a) An LGP program and (b) a basic tree-based GP program.

[21] simultaneously include binary flow control operations (e.g., maximum and minimum) and a ternary IF operation in evolving dispatching rules. The ternary IF operation accepts three arguments, returns the second input argument if the first argument is larger than 0, and returns the third input argument otherwise.

When DJSS scenarios become more complex, flow control operations become more important in designing dispatching rules. For example, Masood et. al. [22]–[24] used maximum, minimum, and a ternary IF operation in solving multi-objective job shop scheduling. Karunakaran et. al. [25], [26] included these operations in solving DJSS problems under uncertainty. Park et. al. [27], [28] also included flow control operations in GP primitive set when solving DJSS with machine breakdown.

In addition to the flow control operations mentioned above, Miyashita [29] develop a four-argument IF operation, which returns the third argument if the first argument is less than or equal to the second argument and returns the fourth argument otherwise. Nguyen et al. [30] showed that flow control operations are effective in designing due-date estimation models.

However, the mentioned studies have not fully investigated flow control operations. They simply included flow control operations into the function set and treated them equivalently with other arithmetic operations. The existing studies did not consider the three challenges mentioned in Section I, which inevitably leads to a huge search space and a large number of redundant solutions. Moreover, designing flow control operations for tree-based programs in existing studies is not straightforward since tree-based programs are greatly different from human line-by-line programs. This precluded us from introducing our programming skills in existing studies.

2) *IF operations in Linear Genetic Programming*: Linear genetic programming (LGP) is a GP variant whose individuals are sequences of register-based instructions [14]. Each instruction normally consists of four parts, a destination register, a function, and two source registers. The function accepts the values in the source registers and writes the calculation result into the destination register. All the destination and source registers come from the same register set. Fig. 1 shows an LGP program and a basic tree-based GP program, representing a formula  $x_1/x_2 \times (x_2 + x_2)$ . R0 and R1 are registers, and  $x_1$  and  $x_2$  are inputs (or constant registers). The LGP program in Fig. 1 represents the formula by storing the intermediate results of  $x_1/x_2$  and  $x_2 + x_2$  into R0 and R1 respectively, multiplying R0 and R1, and storing the final output into R0.

LGP programs have a different design of IF operations from tree-based programs. Because of the linear representation (i.e., line-by-line instructions), IF-included sub-rules in LGP programs have a similar representation to human programs.

For example, LGP programs have to indicate the closure of IF branches (like “{...}” in C language) [14], [31]. To fulfill the closure of IF branches, existing LGP studies either define the number of instructions in an IF branch or design additional labels or pointers (e.g., `endif`) to explicitly specify the end of a branch.

This paper defines the closure of IF operations by specifying the number of instructions in IF branches. For example, the instruction “IF> #3 a b” denotes that this instruction returns true if  $a$  is larger than  $b$ , and returns false otherwise (denoted by “IF>”). If the instruction returns false, the following three instructions are skipped (denoted by “#3”). Otherwise, the following three instructions are executed.

3) *Grammar-based Genetic Programming*: Grammar is a popular tool to enforce restrictions on GP search space. GP programs can search for effective solutions faster and get rid of redundant programs by reducing to a smaller yet effective search space. There have been many studies about grammar-based genetic programming [32], [33].

Context-free-grammar-based GP is a typical grammar-guided genetic programming method [32]. A context-free grammar is a set of production rules that define the derivation from high-level concepts to low-level concepts regardless of the program context. To construct a program, grammar-guided GP recursively derives the concepts based on the production rules and forms a tree-based program. A context-free grammar is regularly defined by Backus Naur Form. Grammar-guided GP is widely applied to program synthesis problems [34], [35], regression problems [36], and automatic algorithm design [37]. Under the umbrella of grammar-guided GP, grammatical evolution is representative of linear grammar-guided GP methods [38], [39]. Unlike tree-based ones, grammatical evolution searches on bit strings (or integer strings) and maps the bit strings into computer programs based on a set of grammar rules by a MOD operator. Grammatical evolution has undergone a lot of improvement. For example, Lourenço et. al. [40], [41] developed a structured grammatical evolution that improves the locality.

LOGENPRO [42], [43] is a tree-based grammar-guided genetic programming that uses a context-sensitive grammar, PROLOG Definite Clause Grammars, to define constraints for GP search space. Due to the context-sensitive grammar, LOGENPRO is more expressive than context-free-grammar-based GP. Following LOGENPRO, Ross [44] proposed a logic-based GP system with definite clause translation grammar.

Strongly typed GP is an alternative GP method that imposes data type constraints on GP search spaces [45]. Strongly typed GP specifies all the possible data types of arguments and returns for all the non-terminals. The non-terminals can only have children with specified data types. To make the data type constraints more flexible, strongly typed GP additionally introduces generic functions and generic data types which specify a set of possible data types by algebraic quantities. Due to the flexibility in handling different data types, existing studies applied strongly typed GP to different problems, such as classification [7], [46], finance [47]–[49], and software testing [50].

Despite the differences among the mentioned grammar-guided GP methods [51], [52], most of existing grammar-guided GP encode programs into tree-based structures due to recursive grammar derivation. But tree-based programs cannot effectively reuse the common building blocks in the different sub-trees, which limits GP to produce more compact programs. Linear representation programs (e.g., a list of instructions) can naturally reuse common building blocks. A few studies tried to apply grammar to enhance GP with linear representation programs (e.g., LGP) [53]. But they did not consider flow control operations.

4) *Grammar-based Genetic Programming in Combinatorial Optimization*: Introducing domain bias by grammar is not a new idea to enhance GPHH for solving job shop scheduling problems. Nguyen et. al. [5] used grammar to define three representation templates for GP individuals, including 1) selecting simple dispatching rules based on machine attributes, 2) an arithmetic representation, and 3) selecting sub-arithmetic representations based on machine attributes. However, [5] did not show the superior performance of their grammar-like method in solving job shop scheduling problems. Hunt et. al. [54] used grammar to categorize input features into different types and defined the available input and output types for each function. However, [54] improved the interpretability of dispatching rules but sacrificed effectiveness although only slightly.

Grammar-guided GP methods have also been applied to many other combinatorial optimization problems. For example, Pawlak and O'Neill [55] used grammatical evolution to synthesize constraints for a diet plan optimization problem. Fenton et. al. [56] and Saber et. al. [57] applied grammar-guided GP for network scheduling. Correa et. al. [58] developed a grammar-guided GPHH method for solving corridor allocation problems. Pereira et al. [59], [60] developed a quantum-inspired grammar-based linear GP to schedule crude oil refinery.

Although these existing studies have applied grammar to enhance GP in solving combinatorial optimization problems, the grammar-guided GP methods for combinatorial optimization are not well investigated. Grammar improves GP interpretability or training efficiency while (not necessarily) sacrificing test effectiveness [36], [61]. Furthermore, existing studies mainly include arithmetic and domain-specific operators in their grammar rules but ignore flow control operations, which are expected to be important primitives for solving many combinatorial optimization problems.

### C. Summary

To summarize, existing GP studies have not effectively evolved IF-included solutions as IF operations inevitably introduce many redundant solutions into search spaces. Grammar-based techniques are effective in removing redundant GP solutions from search spaces. However, existing grammar-based GP methods mainly defined the basic format of flow control operations (e.g., IF operations must be followed by a boolean operation) but did not address the dimension inconsistency and inactive and ineffective sub-rules of flow control operations.

Moreover, existing grammar-based GP methods are designed based on tree-based representations and missed linear representations which can naturally accept human programming skills. To evolve effective and interpretable dispatching rules with advanced flow control operations, this paper applies grammar-guided linear genetic programming (G2LGP) [17] to evolve IF-included dispatching rules.

## III. EVOLVING IF-INCLUDED DISPATCHING RULES BY G2LGP

To evolve effective and interpretable IF-included dispatching rules, this section first designs a set of normalized terminals for IF operations to improve interpretability. Second, we use grammar rules to restrict the input, the number, and the locations of IF operations, which encourages LGP to produce more effective dispatching rules. Finally, we propose to use G2LGP to evolve IF-included dispatching rules based on the proposed grammar rules.

### A. Normalized Terminals

The newly proposed normalized terminal set transforms the existing terminals into a normalized form. Based on the common terminal sets of GPHH for solving DJSS problems [15], we design twenty normalized terminals (four of them are introduced in Sections IV-A and IV-B), as shown below. We mainly normalized terminals by their maximum values at corresponding decision situations. For example, we normalize the processing time of the operations on a machine by dividing the processing time of operations over the maximum processing time on the current machine.  $\|\cdot\|_0$  denotes the cardinality of a set or a list (e.g., the number of available operations in the queue of machine  $m$ ).  $\|\cdot\|_1$  denotes 1-norm regularization of a set or a list (e.g.,  $\|q(m)\|_1$  denotes the workload of a machine  $m$ , equivalent to  $\sum_{o \in q(m)} p(o)$ ).  $\delta(o)$  denotes the waiting time of an operation  $o$ , equivalent to the difference between the system time and the ready time of the operation  $o$ .  $q_{next}(o)$  denotes the corresponding machine queue that processes the next operation of  $o$ .  $\tau(o_{ji})$  denotes the list of remaining operations in job  $j$  after finishing  $o_{ji}$ .

To facilitate understanding, we categorize these proposed terminals into three groups: job-related, machine-related, and job shop-related terminals, as shown in Table I. These twenty normalized terminals depict various information in making decisions, which is supposed to be comprehensive enough to let IF-included dispatching rules understand decision situations.

### B. Proposed Grammar Rules

Based on the normalized terminals, we design a set of grammar rules to restrict the use of these normalized terminals in IF operations. To improve the effectiveness and interpretability of IF-included dispatching rules, the proposed grammar rules fulfill three main restrictions of IF operations:

- 1) Input restriction: IF operations should use normalized terminals and constants as inputs to improve the interpretability of IF conditions.
- 2) Location restriction: IF branches should locate in the beginning of programs and disjoint with each other (i.e.,

TABLE I  
PROPOSED NORMALIZED TERMINALS

Name	Formula	Description
Job-related normalized terminals		
Processing time ratio	$PTR(o_{ji}, m) = \frac{p(o_{ji})}{\max_{o' \in q(m)} p(o')}$	The processing time of an available operation in machine $m$ over the maximum processing time in the current queue.
The number of remaining operations ratio	$NORR(o_{ji}, m) = \frac{  \tau(o_{ji})  _0}{\max_{o' \in q(m)}   \tau(o')  _0}$	The number of remaining operations of job $j$ after processing $o_{ji}$ , divided by the maximum number of remaining operations among all available operations in $q(m)$ .
The remaining workload ratio	$WKRR(o_{ji}, m) = \frac{  \tau(o_{ji})  _1}{\max_{o' \in q(m)}   \tau(o')  _1}$	The remaining workload of job $j$ after processing $o_{ji}$ , divided by the maximum remaining workload among all available operations in $q(m)$ .
The ratio of the number of operations in the next machine	$NNQR(o_{ji}, m) = \frac{  q_{next}(o_{ji})  _0}{\max_{o' \in q(m)}   q_{next}(o')  _0}$	The number of operations in $q_{next}(o_{ji})$ , divided by the maximum number of operations in $q_{next}(o)$ , $\forall o \in q(m)$ .
The ratio of the workload of the next machine	$WNQR(o_{ji}, m) = \frac{  q_{next}(o_{ji})  _1}{\max_{o' \in q(m)}   q_{next}(o')  _1}$	The total workload of $q_{next}(o_{ji})$ , divided by the maximum workload in $q_{next}(o)$ , $\forall o \in q(m)$ .
The operation waiting time ratio	$OWTR(o_{ji}, m) = \frac{\delta(o_{ji})}{\max_{o' \in q(m)} \delta(o')}$	The waiting time of $o_{ji}$ , divided by the maximum waiting time among all operations in the current queue $q(m)$ .
The weight ratio	$WR(o_{ji}, m) = \frac{\omega(o_{ji})}{\max_{o' \in q(m)} \omega(o')}$	The weight value of $o_{ji}$ divided by the maximum weight value among all operations in $q(m)$ .
The relative flow due date ratio	$rFDR(o_{ji}, m) = \frac{\alpha_j + \sum_{i=0}^t p(o_{ji}) - t}{\max_{o'_{ji} \in q(m)} \alpha_j + \sum_{i=0}^t p(o'_{ji}) - t}$	The relative flow due date of $o_{ji}$ , divided by the maximum relative flow due date among all operations in $q(m)$ , where $t$ is the system time [62].
The ratio of energy cost rate of a job	$JERO(j) = \frac{r_e(j)}{\max r_e}$	$r_e(j)$ denotes the energy cost rate of a job $j$ . $\max r_e = 3$ in our simulation. Refer to Sections IV-A and IV-B.
Machine-related normalized terminals		
The number of operations in the machine queue ratio	$NIQR(m) = \frac{  q(m)  _0}{\sum_{m' \in M}   q(m')  _0}$	It indicates the burden of machine $m$ by comparing the number of operations in the machine queue $q(m)$ with the overall number of available operations in the job shop.
The workload in the machine queue ratio	$WIQR(m) = \frac{  q(m)  _1}{\sum_{m' \in M}   q(m')  _1}$	It indicates the burden of machine $m$ by comparing the total workload in the machine queue $q(m)$ with the overall workload in the job shop.
Deviation of processing time	$DPT(m) = \frac{\min_{o \in q(m)} p(o)}{\max_{o' \in q(m)} p(o')}$	A simple index to show the processing time discrepancy of the available operations in machine $m$ by comparing minimum processing time with maximum processing time among available operations in $q(m)$ .
Deviation of operation waiting time	$DOWT(m) = \frac{\min_{o \in q(m)} \delta(o)}{\max_{o' \in q(m)} \delta(o')}$	A simple index to show the waiting time discrepancy of the available operations in machine $m$ by comparing minimum waiting time with maximum wait time among available operations in $q(m)$ .
Deviation of the processing time of the next operation	$DNPT(m) = \frac{\min_{o_{ji} \in q(m)} p(o_{j,i+1})}{\max_{o'_{ji} \in q(m)} p(o'_{j,i+1})}$	A simple index to show the discrepancy of one-step-further decision situations of available operations in $q(m)$ by comparing the minimum processing time of the next operation and the maximum processing time of the next operation.
Deviation of the number of operations in the next machine	$DNNQ(m) = \frac{\min_{o \in q(m)}   q_{next}(o)  _0}{\max_{o' \in q(m)}   q_{next}(o')  _0}$	A simple index to show the discrepancy of one-step-further decision situations of available operations in $q(m)$ by comparing the minimum number of available operations in the next machine with the maximum number of available operations in the next machine.
Deviation of the workload of the next machine	$DWNQ(m) = \frac{\min_{o \in q(m)}   q_{next}(o)  _1}{\max_{o' \in q(m)}   q_{next}(o')  _1}$	A simple index to show the discrepancy of one-step-further decision situations of available operations in $q(m)$ by comparing the minimum workload in the next machine with the maximum workload in the next machine.
The idle energy consumption rate	$MER(m) = \frac{r_m}{\max r_m}$	The normalized energy consumption rate of an idle machine over time. $\max r_m = 7500$ in our simulation. Refer to Sections IV-A and IV-B.
Job shop-related normalized terminals		
Bottleneck workload ratio [5]	$BWR = \max_{m \in M} WIQR(m)$	An index of bottleneck by comparing the workload of bottleneck machines with overall workload. Bottleneck machines are the machines with the largest workload at a particular time [5].
Energy price rate	$EPR = \frac{p_e}{\max p_e}$	The normalized job shop-wide energy price. $\max p_e = 0.015$ in our simulation. Refer to Sections IV-A and IV-B.
The response cost rate ratio	$SFR = \frac{\varphi}{\max \varphi}$	The normalized job shop-wide cost rate for job response time. $\max \varphi = 2.3$ in our simulation. Refer to Sections IV-A and IV-B.

In our experiment, the data ranges are:  $PTR \in [0.01, 1]$ ,  $WR \in [0.25, 1]$ ,  $rFDR \in (-\infty, \infty)$ ,  $JERO \in [0.4, 1]$ ,  $NIQR \in (0, 1]$ ,  $WIQR \in (0, 1]$ ,  $DPT \in [0.01, 1]$ ,  $DNPT \in [0.01, 1]$ ,  $MER \in [0.236, 1]$ ,  $BWR \in (0, 1]$ ,  $EPR \in [0.33, 1]$ , and  $SFR \in [0.087, 1]$ . The data ranges of the other normalized terminals (i.e.,  $NORR$ ,  $WKRR$ ,  $NNQR$ ,  $WNQR$ ,  $OWTR$ ,  $DOWT$ ,  $DNNQ$ , and  $DWNQ$ ) are  $[0, 1]$ .

no nested IF branches) to avoid meaningless program output caused by IF conditions.

- 3) Number restriction: Dispatching rules should only use a limited number of IF operations to reduce redundant IF branches.

Fig. 2 is an example to illustrate the three restrictions. The raw and restricted LGP-based dispatching rules both manipulate three registers, R0, R1, and R2. The final outputs of dispatching rules are stored in the first register R0. Each dispatching rule in Fig. 2 contains three IF conditions. “IF > #N a b” denotes that if  $a$  is larger than  $b$ , the program executes  $N$  subsequent instructions. Otherwise, the program skips the next  $N$  instructions.

	Raw rule	Restricted rule	Comments
Line 0	<b>R0=R1=R2=0</b>	<b>R0=R1=R2=0</b>	//Initialize registers.
Line 1	<b>R2= PT + NPT</b>	<b>R2= PT + NPT</b>	
Line 2	<b>IF&gt;#1 WIQ WINQ</b>	<b>IF&gt;#1 WIQR 0.5</b>	//If the machine is busy
Line 3	<b>R0= R2 - NPT</b>	<b>R1= R2 - NPT</b>	//Shortest processing time
Line 4	<b>IF&lt;#1 rFDD R1</b>	<b>IF&lt;#1 rFDR 0.1</b>	//Lines 4-7: If an operation delays from a due date,
Line 5	<b>IF&gt;#2 W 2</b>	<b>R1= R2 + rFDD</b>	prioritize it by “+rFDD”.
Line 6	<b>R1= R2 + rFDD</b>	<b>IF&gt;#1 WR 0.6</b>	If a job is important,
Line 7	<b>R0= R1 / W</b>	<b>R1= R2 / W</b>	prioritize it by “/W”.
Line 8		<b>R0= R2 + R1</b>	

Fig. 2. Examples of non-restricted and restricted dispatching rules in the LGP representation. The meanings of PT, WIQ, WINQ, rFDD, and W refer to Table. II.

The two dispatching rules show a similar scheduling pattern.

```

defset FUNS {add,sub,mul,div,max,min};
defset FLOWCTRL {IfLarge1,IfLessEq1};
defset RAWINPUT
{PT,NPT,WINQ,NINQ,rFDD,rDD,SL,W,OWT,NWT,TIS,WKR,NOR};
defset conditionINPUT {NIQR,WIQR,DPT,DOWT,DNNQ,DWNQ,DNPT,BWR,
PTR,NORR,WKRR,NNQR,WKQR,OWTR,WR,rFDR};
defset INPUT {conditionINPUT,RAWINPUT};
defset REG {R0,R1,R2,R3,R4,R5,R6,R7};
defset constant {0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9};

begin modulec_0
uncondition(I\O\R) ::= <O\{FUNS}\R+I\R+I>;

branch ::= <{R0}\{FLOWCTRL}\{conditionINPUT}\{constant}>;

condition(I\O\R) ::= <O\{FUNS}\R+I\R+I> :: branch :: <O\{FUNS}\
R+I\R+I>;

PROGRAM ::= condition(I~{INPUT}\O~{REG}\
R~{REG})*5::uncondition(I~{RAWINPUT}\O~{REG}\R~{REG})*;
end modulec_0

```

Fig. 3. The proposed grammar rules for evolving IF-included dispatching rules for DJSS

When the workload in a machine queue is heavy (Line 2 for both rules), the two dispatching rules encourage the machine to finish its operations as soon as possible to improve the pipeline level of the job shop, that is, prioritizing operations mainly based on their processing time (i.e., shortest-processing-time-first rule). If an operation is already delayed from its given due date, we prioritize it by adding rFDD (rFDD<0 if an operation is delayed). If a job (and its operations) is important (i.e., with a high weight value), we prioritize the operations by dividing them by W. In summary, when the machine is busy, the operation is delayed, and the job is important, the corresponding decision will be most preferred (i.e., smallest dispatching rule value).

However, the two dispatching rules have different interpretability and effectiveness. With the input restriction, the restricted rule uses a normalized terminal WIQR to indicate the workload burden of a machine and uses a constant 0.5 to define that the workload is heavy if the workload of this machine accounts for more than half of the overall workload. Contrarily, the raw rule has to compare the features with different physical meanings (i.e., WIQ and WINQ) to approximate the heavy workload (i.e., large enough WIQ), which is not comprehensive enough. With the location restriction, the restricted rule adds one unconditional instruction (i.e., line 8) to assemble register values. But the raw rule overwrites the output register R0 in IF branches, which might lead to meaningless R0 (i.e., returning the initial value of R0) if all the branches are skipped (i.e., all the IF conditions are not satisfied). The nested IF conditions in the raw rule (i.e., lines 4 and 5) also increase the probability that the raw rule skips lines 6 and 7. Although the two rules in Fig. 2 use the same number of IF branches, we advocate that unlimited IF conditions easily lead to redundant or contradictory building blocks.

Based on the three restrictions and the domain knowledge of DJSS problems, we design the following grammar rules based on module context-free grammar (MCFG) [17], as shown in Fig. 3. Specifically, MCFG treats different sub-parts of an LGP program as modules, each with different available primitives. MCFG defines the sequential relationship of modules by “::” and defines the maximum repetition of modules by “\*”. We first categorize the input features into different concepts (i.e., primitive sets), including arithmetic functions (FUNS), IF operations (FLOWCTRL), raw job shop features

(rawINPUT), normalized terminals (conditionINPUT), registers (REG), and constants (constant). We denote the IF operations (“IF> #1” and “IF<= #1”) by IfLarge1 and IfLessEq1 respectively. The settings of the primitive set follow the results obtained from the existing LGPHH studies for DJSS problems [15], [63].

Then, we define the derivation rules to divide LGP programs into sub-programs. We divide an LGP program into conditional (i.e., condition(I\O\R)) and non-conditional sub programs (i.e., uncondition(I\O\R)), defined by PROGRAM. The conditional sub-program accepts normalized terminals and raw job shop features as inputs (i.e., I~{INPUT}) and outputs the results to any of the eight registers (i.e., O~{REG}). The conditional subprogram includes three instructions, two for arithmetic instructions (i.e., <O\{FUNS}\R+I\R+I>) and one for logical instruction (i.e., branch). To implement the IF-ELSE structure, the conditional sub-program simply first executes an arithmetic instruction unconditionally and then executes the other arithmetic instruction based on the IF condition.

To fulfill the three proposed restrictions, we have three designs in Fig. 3. 1) The IF condition (branch) only uses normalized terminals (conditionINPUT) and constants as inputs based on the input restriction. Note that the predefined output register R0 in the IF condition is useless since IF operations do not overwrite registers. 2) Based on the number restriction, the conditional sub program repeats at most five times (i.e., “\*5” after condition(I~{INPUT}\O~{REG}\R~{REG})) in PROGRAM) to limit the number of logical operations. 3) The unconditional sub-program is executed after the conditional sub-program to fulfill the location restriction. For the sake of simplicity, the unconditional sub-program only accepts raw job shop features as inputs and outputs the results to any of the eight registers. Note that the design details in Fig. 3 are based on our preliminary investigation and existing studies [14]. For example, the compared method without limiting the number of conditional sub programs averagely has five conditional sub programs. Thus, we limit the conditional sub programs to at most repeating five times in Fig. 3 to further reduce the search space.

### C. Individual Derivation from Grammar

Based on the grammar rules, we generate LGP individuals by first constructing a derivation tree and second stochastically generating a list of instructions based on the leaf nodes of the derivation tree. Fig. 4 shows an example of the initialization of an LGP individual.

We construct the derivation tree in a top-down way. PROGRAM is the starting symbol in the derivation. Each tree node is a module, remembering the input arguments specified in the grammar rules and specifying the repeating time of sub-modules. The input arguments are actually feasible primitives in different positions of the program. We derive the tree nodes to sub-trees based on the grammar rules in a recursive manner and finally end up with instruction modules (abbreviated as “instr<...>” in Fig. 4). All the leaf nodes of derivation trees

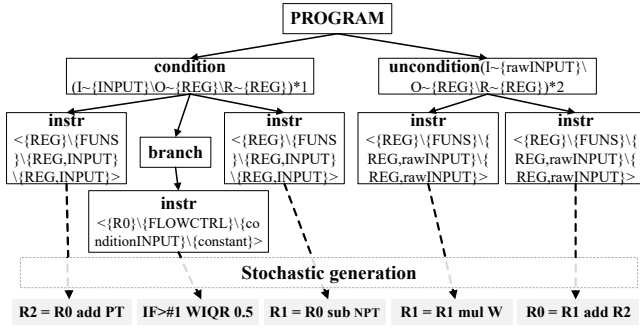


Fig. 4. An example of initializing an LGP individual based on the proposed grammar rules. The upper part is the derivation tree, and the lower part is the LGP individual.

must be instruction modules. Each instruction module specifies the feasible primitives for its four components, including a destination register, a function, the first source register, and the second source register. For example, the leftmost leaf node specifies that 1) the destination register is one of the eight registers (i.e.,  $\{REG\}$ ), 2) the function is one of the six arithmetic functions (i.e.,  $\{FUNS\}$ ), and 3) the two source registers are registers, normalized terminals, or raw job shop features (i.e.,  $\{REG, INPUT\}$ ). We generate one instruction for each leaf node by randomly selecting one of the primitives given by each component to form an LGP instruction. Specifically, there are five leaf nodes in Fig. 4, which leads to an LGP individual with five instructions. The LGP individual is obtained by sequentially arranging generated instructions from the left to the right.

#### D. Evolutionary Framework

To evolve the individuals, we apply the evolutionary framework of G2LGP [17]. We apply grammar-guided micro mutation, grammar-guided macro mutation, and grammar-guided crossover to produce new dispatching rules. The main idea of these genetic operators is to produce new rules by first varying the derivation tree of LGP parents and then updating the rule based on the new derivation tree. Specifically, grammar-guided micro mutation directly changes an LGP instruction by randomly selecting a new primitive from the feasible primitive set given by the leaf node. Grammar-guided macro mutation varies a derivation tree by increasing or reducing the repeating times of sub-modules and re-deriving sub derivation trees and instruction sequences. Grammar-guided crossover accepts two LGP parents and swaps the sub-derivation trees (and their corresponding instruction sub-sequences) whose roots are the same modules. By this means, we ensure that all LGP offspring adhere to the predefined grammar rules.

### IV. EXPERIMENT DESIGN

To verify the effectiveness of IF-included dispatching rules and G2LGP, we design three scenario sets with different complexities. Specifically, the first scenario set is the basic one, only optimizing the tardiness or flowtime. The second scenario set increases the complexity of the first scenario set

by additionally considering energy cost in the optimization. The third scenario set further increases the complexity by additionally considering energy cost and the response time of operations.

#### A. Simulation Design

We built a job shop simulator based on the common settings from the existing studies [13], [64], [65]. The job shop in our simulation contains 10 machines. Jobs arrive at the job shop based on a Poisson process:

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

$$\lambda = \frac{v \cdot u}{\rho \cdot |M|}$$

where  $v$  and  $u$  are the average number of operations and average processing time of the operations of jobs, and  $|M|$  is the number of machines in the job shop (i.e., 10). Every new arrival job consists of 2 to 10 operations, and each operation has a processing time ranging from 1 to 99 time units. The number of operations and their processing time are determined by two uniform distributions, in which  $v$  and  $u$  have a value of 6 and 50 respectively. We increase the utilization level of machines  $\rho$  based on the Poisson process to simulate a busy job shop. Specifically, we set two utilization levels in the scenario sets,  $\rho = 0.85$  or  $0.95$ . We define the due date of a job as its arrival time adding with 1.5 times the total processing time of a job. Although we only use one due date factor in our experiments, jobs have different emergencies by having different weights. All the jobs have a weight of 1, 2, or 4. The jobs with different weight values account for 20%, 60%, and 20% respectively.

1) *Basic scenario set*: Based on the general settings, we first develop a basic scenario set. The basic scenario set mainly optimize tardiness or flowtime. Specifically, we define six optimization objectives, including maximum tardiness ( $T_{max}$ ), mean tardiness ( $T_{mean}$ ), weighted mean tardiness ( $WT_{mean}$ ), maximum flowtime ( $F_{max}$ ), mean flowtime ( $F_{mean}$ ), and weighted mean flowtime ( $WF_{mean}$ ).

- 1)  $T_{max} = \max_{j \in J} (\max(c(j) - d(j), 0))$
- 2)  $T_{mean} = \frac{\sum_{j \in J} \max(c(j) - d(j), 0)}{|J|}$
- 3)  $WT_{mean} = \frac{\sum_{j \in J} \max(c(j) - d(j), 0) \times \omega(j)}{|J|}$
- 4)  $F_{max} = \max_{j \in J} (c(j) - \alpha(j))$
- 5)  $F_{mean} = \frac{\sum_{j \in J} c(j) - \alpha(j)}{|J|}$
- 6)  $WF_{mean} = \frac{\sum_{j \in J} (c(j) - \alpha(j)) \times \omega(j)}{|J|}$

Together with the two utilization level settings, the basic scenario set totally contains twelve DJSS scenarios, which are  $\langle T_{max}, 0.85 \rangle$ ,  $\langle T_{max}, 0.95 \rangle$ ,  $\langle T_{mean}, 0.85 \rangle$ ,  $\langle T_{mean}, 0.95 \rangle$ ,  $\langle WT_{mean}, 0.85 \rangle$ ,  $\langle WT_{mean}, 0.95 \rangle$ ,  $\langle F_{max}, 0.85 \rangle$ ,  $\langle F_{max}, 0.95 \rangle$ ,  $\langle F_{mean}, 0.85 \rangle$ ,  $\langle F_{mean}, 0.95 \rangle$ ,  $\langle WF_{mean}, 0.85 \rangle$ , and  $\langle WF_{mean}, 0.95 \rangle$ . Each scenario includes a set of training DJSS instances and a set of 50 test instances. The training and test DJSS instances of the same scenario use the same configurations but with different sets of random seeds. To measure the performance of the job shop in its stable status, we warmup the job shop by the first 1000



completed jobs and only take the subsequent 5000 completed jobs into account.

2) *Second scenario set*: To increase the complexity of the problems, we introduce energy cost into optimization objectives of the second scenario set. Specifically, each job in the second scenario set has an energy cost rate  $r_e \sim U(1.2, 3)$ . The machines consume energy in both idle and working time. Each machine has an idle energy consumption rate  $r_m$  and a working energy consumption rate  $r_m \times r_e$  ( $r_e$  is the energy cost rate of on-going jobs). The settings of  $r_m$  follow [66] (i.e., machines have different energy consumption rates). To simulate the floating power prices in daily life, the energy price rate  $p_e$  in our simulation changes every 10 arrival jobs. The energy price is sampled from three values 0.005, 0.01, and 0.015, based on a uniform distribution. The average energy consumption per job per machine  $\bar{E}$  is obtained as follows.

$$\bar{E} = \frac{\sum_{m \in \mathbb{M}} E(m)}{|\mathbb{M}| \times |\mathbb{J}|}$$

$$E(m) = \sum_t (\tau_{idle}(t) \times r_m \times p_e(t))$$

$$+ \sum_t \sum_{j \in \Theta(m)} \tau_{work}(t) \times r_m \times r_e(j) \times p_e(t)$$

where  $E(m)$  is the total energy consumption of machine  $m$ .  $\tau_{idle}(t)$  and  $\tau_{work}(t)$  are the idle and working running time for a machine in a time period  $t$  respectively.  $\Theta(m)$  is all the processed jobs by machine  $m$ .

Based on  $\bar{E}$ , we extend the six tardiness and flowtime optimization objectives by simply averaging  $\bar{E}$  and tardiness and flowtime objective values. For example,  $T_{max}$  is transformed to  $T_{max}^E = 0.5T_{max} + 0.5\bar{E}$ , and  $T_{mean}$  is transformed to  $T_{mean}^E = 0.5T_{mean} + 0.5\bar{E}$ , etc. Note that the two objective values in the linear combination have a similar magnitude based on our preliminary investigation. Together with the two utilization level settings, the second scenario set also has twelve scenarios.

3) *Third scenario set*: The third scenario set further increases the complexity of DJSS problems by additionally considering job response time in the second scenario. Job response time is an important performance metric in many controlling systems, such as operating systems on computers. To optimize the job response time, we define a response cost  $R$  by multiplying the job response time  $R_t$  with a response cost rate  $\varphi$ . There are three levels of  $\varphi$ , 0.2, 1, and 2.3. We reset  $\varphi$  values among the three values every 10 arrival jobs based on a probability of 40%:40%:20%. The average response cost  $\bar{R}$  is obtained by

$$\bar{R} = \frac{\sum_{j \in \mathbb{J}} R(j)}{|\mathbb{J}|}$$

$$R(j) = \sum_t \sum_{o \in \mathcal{O}_j} R_t(o) \times \varphi(t)$$

where  $R(j)$  is the response cost of job  $j$ ,  $R_t(o)$  is the response time for operations, and  $\varphi(t)$  is the response cost rate during the waiting time of the operation  $o$ . We integrate the response cost  $R$  with tardiness, flowtime, and energy

TABLE II  
THE RAW TERMINAL SET

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

cost by the same linear combination to develop six optimization objectives  $T_{max}^{ER}$ ,  $T_{mean}^{ER}$ ,  $WT_{max}^{ER}$ ,  $F_{max}^{ER}$ ,  $F_{mean}^{ER}$ , and  $WF_{max}^{ER}$ . For example,  $T_{max}^{ER} = 0.4T_{max} + 0.3\bar{E} + 0.3\bar{R}$ , and  $T_{mean}^{ER} = 0.4T_{mean} + 0.3\bar{E} + 0.3\bar{R}$ . The three objective values here also have similar magnitudes.

### B. Comparison Design

We use five compared methods to verify the effectiveness of the proposed grammar rules. 1) The first algorithm is the state-of-the-art grammar-guided LGPHH [17], which has shown promising performance for solving DJSS problems, denoted as G2LGP. 2) The second compared algorithm extends the basic LGPHH by including IF operations and the proposed normalized terminals into its primitive set directly but without grammar-guided evolutionary framework and the proposed grammar rules, denoted as LGP+. 3) and 4) The third and fourth compared methods are variants of the proposed method which extend the proposed grammar rules by removing some restrictions from Fig. 3. Specifically, the third compared method (denoted as G2LGP/input) removes the input restriction and evolves based on a set of grammar rules shown in Fig. 5. The source registers of IF operations can be any of the terminals, including all the input features, registers, and constants. The fourth compared method (denoted as G2LGP/locnum) removes the location restriction and the number restriction and evolves based on a set of grammar rules shown in Fig. 6. The “\*” after the PROGRAM derivation indicates that there can be any number of conditional sub-programs (condition). The condition subprogram can derive to either one logical instruction (branch) and one arithmetic instruction ( $\langle O \setminus \{FUNS\} \setminus R+I \setminus R+I \rangle$ ), or unconditional sub-programs ( $\langle uncondition(I \sim I \setminus O \sim O \setminus R \sim R) \rangle$ ) whose input arguments are the same as the parent of the derivation. Thus, there can be a large number of IF operations in a program, and the IF operations can also be used at any position in a program. The last compared method is the proposed algorithm, which evolves G2LGP based on the normalized terminals and the proposed grammar rules, denoted as G2LGP-IF.

The raw input features of all the compared methods are designed based on the common settings [16], as shown in



```

... /* other rules are the same as the proposed rules */
branch ::= <{R0}\{FLOWCTRL}\{conditionINPUT}\{constant}>;
branch ::= <{R0}\{FLOWCTRL}\{INPUT,REG,constant}\{INPUT,REG,constant}>;
... /* other rules are the same as the proposed rules */

```

Fig. 5. The grammar rules of G2LGP/input

```

... /* other rules are the same as the proposed rules */
condition(I\O\R) ::= <O\{FUNS}\R\I\R\I> | branch | <O\{FUNS}\R\I\R\I>;
condition(I\O\R) ::= branch | <O\{FUNS}\R\I\R\I> |
uncondition(I-I\O-O\R-R);
PROGRAM ::= condition(I-{INPUT}\O-{REG}\
R-{REG}) * 5 :: uncondition(I-{RAWINPUT}\O-{REG}\R-{REG}) *;
PROGRAM ::= condition(I-{RAWINPUT}\O-{REG}\R-{REG}) *;
end module c_0

```

Fig. 6. The grammar rules of G2LGP/locnum

TABLE III

AVERAGE TEST OBJECTIVE VALUES (STD.) IN THE BASIC SCENARIO SET.

Scenarios	G2LGP	LGP+	G2LGP/input	G2LGP/locnum	G2LGP-IF
(TmaxE,0.85)	1922.1 (42.9) ≈	1978.1 (162.1) –	1927.5 (52) ≈	1939.2 (51.4) ≈	1931 (44.6)
(TmaxE,0.95)	3943.1 (84) ≈	4040.5 (218.9) –	3946.7 (79.5) ≈	4045.6 (123.3) –	3968.8 (112.3)
(TmeanE,0.85)	417.7 (2.6) ≈	428.8 (40) ≈	416.9 (2.2) ≈	418 (2.8) –	416.8 (2.9)
(TmeanE,0.95)	1116.7 (8.7) ≈	1195.6 (167.4) ≈	1116.4 (11.2) ≈	1122.2 (12.1) ≈	1116.3 (12.1)
(WmeanE,0.85)	723.6 (7.5) ≈	770.4 (112.7) ≈	723.1 (5.4) ≈	728.4 (7.3) ≈	726.5 (7.3)
(WmeanE,0.95)	1724.4 (26.6) ≈	2103.9 (1739.3) –	1722.1 (25.5) ≈	1740.2 (27.7) ≈	1733 (33.9)
(FmaxE,0.85)	2534.6 (74.1) –	2529.2 (63.8) –	2490 (65.2) ≈	2535.7 (53.1) –	2503.1 (79.7)
(FmaxE,0.95)	4599.7 (80.6) –	4760.9 (623.5) –	4505.6 (73.4) ≈	4638.4 (120) –	4501 (74.8)
(FmeanE,0.85)	864.6 (3.2) ≈	910 (193.4) ≈	862.4 (2.6) ≈	865.1 (3.5) –	863.7 (2.6)
(FmeanE,0.95)	1565.3 (10.9) ≈	1649 (245.8) ≈	1561.7 (9.3) ≈	1571.3 (16.5) ≈	1565.9 (12.6)
(WmeanE,0.85)	1701.7 (6.1) ≈	1826.4 (661) ≈	1701.4 (6.5) ≈	1706.5 (7) ≈	1703.4 (7.5)
(WmeanE,0.95)	2722.8 (25.4) ≈	3610.6 (3943.9) –	2708 (24.7) ≈	2724.1 (24.5) –	2711.7 (21.5)
win/draw/lose	0-10-2	0-6-6	0-12-0	0-6-6	
mean rank	2.46	4.25	1.5	4.58	2.21
p-values	1.000	0.015	1.000	0.002	

TABLE IV

AVERAGE TEST OBJECTIVE VALUES (STD.) IN THE SECOND SCENARIO SET.

Scenarios	G2LGP	LGP+	G2LGP/input	G2LGP/locnum	G2LGP-IF
(TmaxE,0.85)	2093.8 (24.1) –	2106.1 (38) –	2082.9 (26.5) ≈	2096 (25.4) ≈	2078.4 (23.1)
(TmaxE,0.95)	3207.8 (86.1) –	3231.4 (302.8) –	3153.2 (63.5) ≈	3233 (128.6) –	3152.8 (65.9)
(TmeanE,0.85)	1331.1 (2.2) ≈	1342.7 (44.8) ≈	1330 (1.3) ≈	1330.6 (1.8) ≈	1330.4 (1.5)
(TmeanE,0.95)	1651 (6.4) ≈	1676.6 (67.5) ≈	1652.9 (10.6) ≈	1652.9 (7.9) ≈	1651.7 (9.6)
(WmeanE,0.85)	1487.2 (3.6) ≈	1526.4 (82.3) ≈	1485.6 (3.1) +	1486.8 (3.6) ≈	1487.6 (3.7)
(WmeanE,0.95)	1985.8 (17.3) ≈	2196 (828) –	1982 (15.3) ≈	1993.7 (17.1) ≈	1987.2 (15.2)
(FmaxE,0.85)	2385.3 (26.2) –	2513.3 (895.8) –	2378.6 (53.8) ≈	2397.7 (45.5) –	2369.7 (23.1)
(FmaxE,0.95)	3465.3 (54.9) ≈	3474.5 (77) –	3431.8 (42.4) ≈	3494.2 (58.4) –	3443.7 (56.3)
(FmeanE,0.85)	1554.1 (2.2) –	1599 (198.4) –	1553.4 (1.9) ≈	1553.9 (1.8) –	1553 (1.4)
(FmeanE,0.95)	1875.6 (5.9) ≈	1913.2 (98.7) ≈	1873.4 (5.9) ≈	1878.7 (9.1) ≈	1876.3 (5.7)
(WmeanE,0.85)	1975.6 (3.5) ≈	1999.8 (65.3) ≈	1974.4 (3.7) ≈	1977.7 (4) –	1975.2 (4.1)
(WmeanE,0.95)	2484.1 (17.4) ≈	2592.8 (436.1) –	2483.1 (17.6) ≈	2480.4 (14.8) ≈	2481.1 (14.7)
win/draw/lose	0-8-4	0-5-7	1-11-0	0-6-6	
mean rank	2.92	4.38	1.58	4.13	2.00
p-values	1.000	0.002	1.000	0.010	

Table. II. G2LGP only includes raw input features in its primitive set as suggested in [17], and the rest of the compared methods also include the proposed normalized terminals in their primitive sets. The rest of the settings for the compared methods are set as those in [17]. Specifically, all the compared methods evolve a population of 256 individuals for 200 generations. Each LGP individual manipulates eight registers and has maximally 50 instructions. The function set for G2LGP is  $\{+, -, \times, \div, \max, \min\}$ , and the function set for the other four compared methods is  $\{+, -, \times, \div, \max, \min, \text{IF} > \#1, \text{IF} \leq \#1\}$ .

## V. MAIN RESULTS

### A. Test Performance

This section compares the test performance of the five compared methods on the three scenario sets, as shown in Table III to V. The test performance indicates the actual tardiness and

TABLE V

AVERAGE TEST OBJECTIVE VALUES (STD.) IN THE THIRD SCENARIO SET.

Scenarios	G2LGP	LGP+	G2LGP/input	G2LGP/locnum	G2LGP-IF
(TmaxE,0.85)	1643.9 (26.3) –	1640.2 (36.7) –	1621.5 (18) ≈	1637.3 (25.4) –	1624.9 (20)
(TmaxE,0.95)	2807.6 (52.5) –	2838 (112.7) –	2790.4 (61.5) ≈	2833.9 (66.9) –	2794.5 (142.5)
(TmeanE,0.85)	995 (2.4) –	1007.9 (35.1) ≈	992.9 (2.6) ≈	990.1 (4.4) +	992.7 (3.4)
(TmeanE,0.95)	1457.5 (9.1) –	1542.5 (235.7) –	1444.7 (25.4) –	1405.3 (21.1) +	1424 (29.2)
(WmeanE,0.85)	1132.2 (9.6) ≈	1174.1 (81.5) –	1128.6 (3.4) ≈	1131.9 (4.6) ≈	1130.4 (4.7)
(WmeanE,0.95)	1774.5 (21.6) ≈	2003.5 (806) –	1767.9 (18.9) ≈	1754.2 (29.4) –	1761.2 (23.8)
(FmaxE,0.85)	1882.5 (28.5) –	1889.2 (31) –	1858.5 (18.9) ≈	1882.3 (61) –	1866.2 (22.8)
(FmaxE,0.95)	3053.9 (49) –	3293 (1052.1) –	3014.4 (46.1) ≈	3054.2 (59.5) –	3000.6 (47.5)
(FmeanE,0.85)	1173.2 (2.6) –	1206.7 (72.7) –	1171.4 (2.9) ≈	1167.3 (4.7) +	1170.4 (3.7)
(FmeanE,0.95)	1636.2 (9.7) –	1665.1 (95.8) –	1625.7 (20.5) –	1583.9 (22.3) +	1602.8 (29.8)
(WmeanE,0.85)	1523 (3.8) –	1566.3 (82.5) –	1520.6 (3.6) ≈	1522.8 (4.5) –	1521 (3.8)
(WmeanE,0.95)	2174 (17.9) –	2276.6 (342) –	2162.6 (14.6) ≈	2142.5 (29.6) +	2154.5 (22.3)
win/draw/lose	0-2-10	0-1-11	0-10-2	5-3-4	
mean rank	4.25	4.42	2.17	2.33	
p-values	0.002	0.000	1.000	1.000	1.83

flowtime (by time units) of the compared methods for solving unseen instances in different scenarios. For each scenario set, we first analyze the overall performance of the compared methods by the Friedman's test and then analyze the performance on each scenario based on the Wilcoxon rank-sum test with Bonferroni correction. The significance levels of the Friedman's test and the Wilcoxon test are 0.05. Specifically, the "+" in Table III to V indicates that a certain method is significantly better (i.e., having a smaller objective value) than the proposed G2LGP-IF, the "≈" indicates that a method is statistically similar to G2LGP-IF, and the "–" indicates that a method is significantly worse than G2LGP-IF. The p-values at the last row indicate the pair-wise comparison between a certain method and G2LGP-IF, with a null hypothesis that the performances of the two compared methods belong to the same distribution and an alternative hypothesis of different distributions.

For the basic scenario set, the p-value of the Friedman's test is 4.25E-07, which indicates a significant difference among the compared methods. Based on the pair-wise comparison and the mean rank, we confirm that the proposed G2LGP-IF has a significantly better overall test performance than directly evolving IF-included dispatching rules by basic LGPHH (i.e., LGP+) and G2LGP without location and number restrictions. On the other hand, we cannot see significant performance differences between state-of-the-art LGP (i.e., G2LGP) whose primitive set has been well designed and G2LGP-IF, and between G2LGP/input and G2LGP-IF. It is likely that the existing primitive set (i.e., excluding the IF operations and normalized terminals) is large enough to compose effective dispatching rules for the basic scenario set, and the grammar of G2LGP-IF and G2LGP/input effectively reduces the search space to a similar size with G2LGP. The Wilcoxon test confirms our observations on the inferior performance of LGP+ and G2LGP/locnum.

The second scenario set which concerns energy cost shows a similar pattern to the basic scenario set, with a Friedman's test p-value of 5.53E-06. In the second scenario, G2LGP-IF is also superior to LGP+ and G2LGP/locnum and performs similarly to G2LGP and G2LGP/input. It is worth mentioning that, despite the insignificant overall performance discrepancy between G2LGP and G2LGP-IF, G2LGP-IF performs significantly better than G2LGP on four scenarios and has better mean performance on eight scenarios. Together with the results in Table III, the results confirm that G2LGP-IF

has a very competitive performance with the state-of-the-art LGPHH methods and is superior to basic LGPHH when solving relatively simple scenarios.

The third scenario set which considers tardiness (or flow-time), energy cost, and job response time, shows a substantial difference. The p-value of the Friedman's test is 7.34E-06, indicating a significant difference among the compared methods. Based on the pair-wise comparison, we see that G2LGP-IF significantly outperforms G2LGP and LGP+ in terms of test performance. The Wilcoxon test further verify the superior performance of the proposed G2LGP-IF. On the other hand, the other two G2LGP variants G2LGP/input and G2LGP/locnum have a very competitive performance with G2LGP-IF, but with worse mean ranks (i.e., 2.17 for G2LGP/input and 2.33 for G2LGP/locnum are worse than 1.83 for G2LGP-IF). The results confirm that G2LGP with the proposed grammar restrictions is very effective in solving the complex scenario set which simultaneously optimizes multiple performance metrics.

Based on the results from the basic scenario set to the more complicated ones, we have the following observations:

1) When scenario sets become more and more complicated, evolving IF operations by grammar rules becomes more and more important. The evidences are twofold. First, the performance gap between non-grammar-guided LGP and grammar-guided LGP becomes larger and larger when the scenarios have to optimize more and more performance metrics (e.g., the mean rank of LGP+ increases with scenario complexity). Second, G2LGP, which has no IF primitives and necessary grammar rules, cannot handle complex scenarios effectively. It performs inferior to G2LGP-IF on more scenarios when scenario sets become more complex (i.e., from 2 significantly worse scenarios to 10 worse scenarios).

2) Directly evolving IF-included dispatching rules is too difficult for existing LGPHH methods since IF operations introduce a large number of redundant solutions into their search spaces. For example, LGP might waste a lot of time searching the contradictory and tautological IF operations that do not contribute to the final output. As shown in Table III to V, LGP+ which directly includes IF operations in its primitive set always has the worst performance among the compared methods. However, without IF operations, it is hard for existing LGPHH methods to solve complicated scenarios.

3) Tables III to V give some insights into the number and location of IF operations. For example, limiting the maximum number of IF operations to five (i.e., G2LGP-IF and G2LGP/input) is effective since G2LGP-IF and G2LGP/input show a good performance in all the three scenario sets. Setting the number of IF operations too small (i.e., G2LGP and LGP+) or too large (i.e., G2LGP/locnum) likely reduces the effectiveness in complex scenarios.

To further analyze the test performance of the compared methods, we show the average test performance of all the compared methods over generations, as shown in Fig. 7. Specifically, we select  $Tmax$ ,  $WTmean$ ,  $Fmax$ , and  $Fmean$  with a high utilization level of 0.95 in the three scenario sets as the example scenarios.

We can see that the proposed G2LGP-IF (i.e., the red

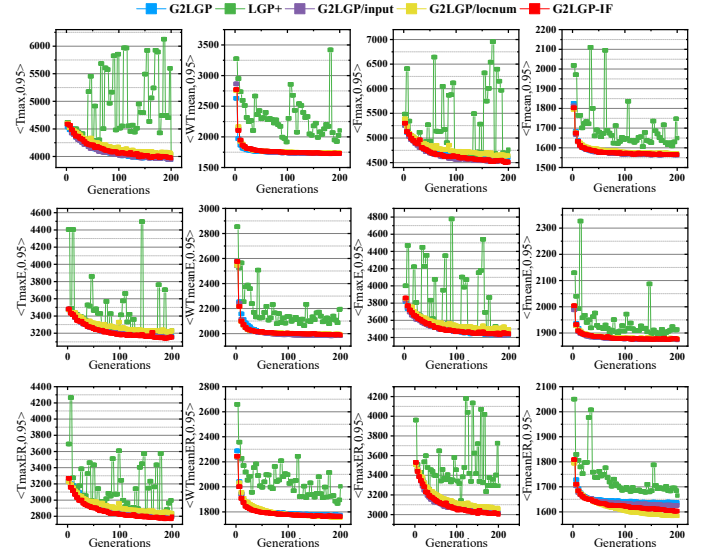


Fig. 7. The test performance over generations in example scenarios.

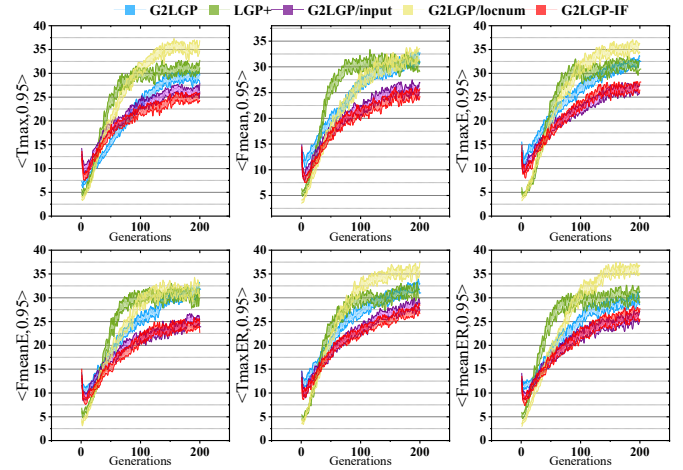


Fig. 8. The average effective program size ( $\pm$  std.) of best-of-run individuals of the compared methods over generations and 50 independent runs.

curves) shows a very competitive performance with other compared methods. On the contrary, basic LGPHH cannot find stable IF-included dispatching rules (i.e., LGP+, the green curves). In some certain generations, the test performance of LGP+ soars up to an extremely poor level, implying that basic LGPHH fails to evolve IF-included dispatching rules with a good generalization ability.

## B. Program Size

To have a brief understanding of the interpretability of output rules, Fig. 8 shows the average effective program size (i.e., the average number of effective instructions) of best-of-run individuals for each compared method for solving six example scenarios over 50 independent runs. We simply assume that a concise rule (i.e., a smaller program) has good interpretability. The curves of mean values and the shadows of standard deviation show that the proposed G2LGP-IF and G2LGP/input averagely achieves significantly smaller program

size than the others. For example, in the first example scenario  $\langle T_{max}, 0.95 \rangle$ , the red and purple curves nearly have no overlap with the others at the end of their evolution, indicating a significant program size difference of output programs. Given that both G2LGP-IF and G2LGP/input have the restrictions on the number and locations of IF operations by grammar rules, we believe the two proposed restrictions are essential reasons for producing concise programs.

### C. Dimension Consistency

This section analyzes the example dispatching rules of G2LGP-IF and G2LGP/input to demonstrate the dimension consistency achieved by the proposed normalized terminals and input restriction. Specifically, we randomly select two best rules of G2LGP-IF and G2LGP/input from 50 independent runs respectively, for solving  $\langle T_{mean}, 0.95 \rangle$  and  $\langle WF_{meanER}, 0.95 \rangle$ . Each pair of the selected rules for the same scenario has a similar test performance. All the four rules have been manually simplified by replacing registers with intermediate results and removing contradictory and tautological IF operations.

1)  $\langle T_{mean}, 0.95 \rangle$ : The example rule from G2LGP-IF for  $\langle T_{mean}, 0.95 \rangle$  is shown in Eq. (1). We can see that when the bottleneck situation is not too severe (i.e.,  $BWR \leq 0.9$ ), the dispatching rule prioritizes the operations with a large processing time of the next operation (i.e., “ $-NPT$ ” in  $a$ ). It implies that the rule intends to use a long-term strategy to process some tough operations before the bottleneck comes. When there is a severe bottleneck in the job shop, the dispatching rule prefers operations with a small processing time to finish more operations in a shorter time (i.e., both “ $PT$ ” in the main rule and “ $NPT$ ” in the conditional part are positively correlated to the heuristic value). By this means, the job shop improves the pipeline of the job shop as soon as possible.

$$RULE_{IF} = \max(PT, \frac{5a + 4NOR}{PT}) + (5a + 4NOR) \times PT$$

$$a = \begin{cases} WINQ - NPT + PT, & \text{if } BWR \leq 0.9 \\ NPT \times (NPT + NINQ), & \text{otherwise} \end{cases} \quad (1)$$

The example rule from G2LGP/input for  $\langle T_{mean}, 0.95 \rangle$  is shown in Eq. (2). The main rule is relatively simple, adding three simple terms together. However, we can see dimension inconsistency in its conditional part  $a$ . The first condition  $rFDR \leq WKRR$  and  $WINQ \leq 0.4$  compares the terminals with different physical meanings (i.e.,  $rFDR$  and  $WKRR$ ) and compares the workload in the next machine queue ( $WINQ$ ) with a meaningless constant 0.4. Note that  $WINQ$  is larger than 2 (the minimum workload unit for a single operation is 2) in most cases and is equivalent to 0 only at the beginning of the simulation where most machines are idle. Thus, the second condition is almost a tautological condition, and the third branch of  $a$  can be neglected in many cases.

$$RULE_{input} = NINQ + 2PT + \min(a, NPT)$$

$$a = \begin{cases} NOR, & \text{if } rFDR \leq WKRR \\ & \text{and } WINQ \leq 0.4 \\ NINQ^3, & \text{else if } WINQ > 0.4 \\ rFDD, & \text{otherwise} \end{cases} \quad (2)$$

2)  $\langle WF_{meanER}, 0.95 \rangle$ : When the problem considers three performance metrics, the example rule from G2LGP-IF still maintains a good interpretability. The rules from G2LGP-IF and G2LGP/input are shown in Eq. (3) and Eq. (4) respectively. For the rule from G2LGP-IF, we can see that if the user response time is important (i.e.,  $SFR > 0.2$  implying  $SFR = 1$  or  $SFR = 2.3$ ) or the number of operations in the next queue is relatively large ( $NNQR > 0.3$ ), the LGP rule emphasizes the next processing time  $NPT$ . Since a small  $NPT$  also implies that the next operation will be prioritized when it is available (i.e., replacing  $PT$  by  $NPT$  in Eq. (3)), the job shop prefers finishing those easy-to-process jobs so that it can response more jobs to reduce the overall response time. Otherwise, the job shop focuses on  $WKR$  to reduce the overall flowtime.

$$RULE_{IF} = ((4PT + a) * a)^2$$

$$a = \frac{b + PT + NINQ}{W}$$

$$b = \begin{cases} NPT, & \text{if } SFR > 0.2 \text{ or } NNQR > 0.3 \\ WKR, & \text{otherwise} \end{cases} \quad (3)$$

Contrarily, the example rule from G2LGP/input compares terminals with different physical meanings in its IF operation. The conditional part  $c$  compares the deviation of processing time ( $DPT$ ) with the number of remaining operations  $NOR$  and compares the number of operations in the next machine queue ( $NINQ$ ) with the processing time ( $PT$ ). These comparisons make the decisions from Eq. (4) hard to be interpreted and might lead to unexpected behaviors during optimization.

$$RULE_{input} = \frac{W \cdot PT \cdot a \cdot TIS}{b} + \frac{W \cdot PT \cdot b \cdot PT + WINQ \cdot NPT \cdot PT}{W^2} + \frac{2b}{W \cdot PT} + a + c + TIS + PT + SL$$

$$a = c + NOR$$

$$b = c + PT$$

$$c = \begin{cases} \max(NINQ, PT), & \text{if } DPT > NOR \\ PT, & \text{otherwise} \end{cases} \quad (4)$$

### D. Training Time

Our experiments run on Intel Broadwell (E5-2695v4, 2.1 GHz). With the same total number of fitness evaluations (i.e., 51200), the average training time of the five compared methods are 3.26, 5.39, 8.08, 9.06, and 9.18 hours respectively. They show a pattern of  $G2LGP < LGP+ < G2LGP/input \approx G2LGP/locnum \approx G2LGP-IF$  in terms of training time. G2LGP has the shortest training time since it uses neither the normalized terminals nor IF operations. Although LGP+ includes the normalized terminals and IF operations in its primitive set, there are no grammar rules to enforce each LGP rule to use these primitives. Contrarily, the last three compared methods that use grammar to enforce the use of the normalized terminals and IF operations, increase the computation time of each rule. The results show that the use of the normalized terminals and IF operations increases the computation time of dispatching rules.

However, we advocate that the increase in the computation time of dispatching rules here is acceptable in practice since

```

... /* other rules are the same as the proposed rules */
defset FLOWCTRL {IfLarge3,IfLessEq3};
...
condition(I\O\R) ::= <O\{FUNS}\R+I\R+I> :: branch :: <O\{FUNS}\R+I\R+I>
:: <O\{FUNS}\R+I\R+I> :: <O\{FUNS}\R+I\R+I>;
... /* other rules are the same as the proposed rules */

```

Fig. 9. The grammar rules of G2LGP-body3.

the training of GP methods is off-line and the decision time (i.e., the computation time of all candidate operations) of LGP rules is much smaller than the processing time in reality. Take the longest training time of G2LGP-IF (i.e., 9.18 hours) as an example. The 9.18 hours are composed of  $256 \times 200 = 51200$  fitness evaluations, each evaluation processes more than 6000 jobs, and each job with 6 operations on average. Thus, the average decision time for each operation is about  $\frac{9.18 \text{ hours}}{51200 \times 6000 \times 6} = 1.8E^{-5}$  seconds, which is much smaller than the operation processing time in many real-world applications. Once we obtain a dispatching rule from the off-line training, the rule can make decisions for unseen problem instances in a very short time. Furthermore, the normalized terminals and IF operation bring a significant performance gain in many DJSS scenarios.

#### E. Summary on Main Results

This section investigates the effectiveness and interpretability of the five compared methods. We found that existing LGPHH methods cannot effectively evolve rules with IF operations based on the inferior performance of LGP+. But without IF operations, state-of-the-art LGPHH cannot effectively solve complicated DJSS scenarios (see the inferior performance of G2LGP in the third scenario set). To make effective use of IF operations, we propose to restrict dispatching rules to only use a limited number of IF operations at the beginning of the rules, which substantially improves the effectiveness of LGPHH in solving complicated scenarios. Moreover, the proposed normalized terminals and input restriction improve the dimension consistency of IF operations in output rules, which can further improve interpretability. We advocate that evolving dispatching rules with IF operations by restricting the input features, number, and location of IF operations has a promising performance in terms of both effectiveness and interpretability.

### VI. FURTHER ANALYSES

#### A. Effectiveness of Simple IF Operations

LGP can naturally implement human-like programming styles of IF operations, including IF branches with different lengths and nested IF branches. In our proposed grammar rules, we restrict IF branches to only contain one instruction and avoid nested IF branches for simplicity, but have not verified the effectiveness of these simple designs. Therefore, this section investigates the effectiveness of long IF branches and nested IF branches.

The first variant of G2LGP, denoted as G2LGP-body3, forces IF branches to include three instructions. The grammar

```

... /* other rules are the same as the proposed rules */
defset FLOWCTRL {IfLarge1,IfLessEq1,IfLarge2,IfLessEq2,
IfLarge3,IfLessEq3};
...
condition(I\O\R) ::= <O\{FUNS}\R+I\R+I> :: branch :: <O\{FUNS}\R+I\R+I>*3;
... /* other rules are the same as the proposed rules */

```

Fig. 10. The grammar rules of G2LGP-nested.

TABLE VI  
THE TEST PERFORMANCE OF THE G2LGP VARIANTS IN THE SECOND SCENARIO SET.

Scenarios	G2LGP-IF	G2LGP-body3	G2LGP-nested
<TmaxE,0.85>	2078.4 (23.1)	2085.8 (22.9) ≈	2083.7 (23.9) ≈
<TmaxE,0.95>	3152.8 (65.9)	3170.7 (61.5) ≈	3164 (56.9) ≈
<TmeanE,0.85>	1330.4 (1.5)	1330.4 (1.4) ≈	1330.1 (1.7) ≈
<TmeanE,0.95>	1651.7 (9.6)	1651.8 (6.6) ≈	1650.4 (5.3) ≈
<WTmeanE,0.85>	1487.6 (3.7)	1487.6 (3.6) ≈	1487.6 (4.4) ≈
<WTmeanE,0.95>	1987.2 (15.2)	1988.8 (16.4) ≈	1989.6 (14.1) ≈
<FmaxE,0.85>	2369.7 (23.1)	2374.3 (19.9) ≈	2369.5 (18.5) ≈
<FmaxE,0.95>	3443.7 (56.3)	3440.8 (55.2) ≈	3454.4 (69.1) ≈
<FmeanE,0.85>	1553 (1.4)	1552.9 (1.5) ≈	1553.2 (1.5) ≈
<FmeanE,0.95>	1876.3 (5.7)	1874.9 (6.2) ≈	1876.2 (6.8) ≈
<WFmeanE,0.85>	1975.2 (4.1)	1976.2 (4.3) ≈	1976.7 (3.9) ≈
<WFmeanE,0.95>	2481.1 (14.7)	2483.8 (21.5) ≈	2486 (21.3) ≈
win/draw/lose		0-12-0	0-12-0
mean rank	1.71	2.08	2.21
p-values		1	0.633

of G2LGP-body3 is shown in Fig. 9, where dispatching rules use IF operations with three instructions (i.e., IfLarge3 and IfLessEq3), and the derivation rule of condition includes more instruction modules.

The second G2LGP variant is G2LGP-nested, which allows an IF branch to include other IF branches (i.e., nested IF branches). The grammar rules for G2LGP-nested are shown in Fig. 10, where dispatching rules use IF operations with one to three instructions, and each logical condition is followed by up to three instruction modules (see condition in Fig. 10).

We verify the effectiveness of these two G2LGP variants by comparing their test performance with G2LGP-IF in the second scenario set, as shown in Table VI. We use the second scenario set for verification because its configurations follow the popularly used settings of existing studies [66], which makes it comparable to existing studies. In addition, it is more challenging than the basic scenario set, which gives a better understanding of the proposed algorithm for readers. The Friedman test on the test performance of the three compared methods returns a p-value of 0.428, indicating a null hypothesis of no significant difference among these test performances. We can also see that both G2LGP-body3 and G2LGP-nested are very competitive with G2LGP-IF in Table VI. The results confirm that the proposed grammar rules for IF operations are effective enough to produce concise and effective rules. Increasing the complexity of IF branches by increasing the length of IF branches or nesting IF branches does not improve the effectiveness of IF-included dispatching rules in our problem.

#### B. Patterns of Normalized Terminals

This paper moves a step forward in effectively evolving IF-included dispatching rules for DJSS problems. To understand the relationship between IF operations and decision situations

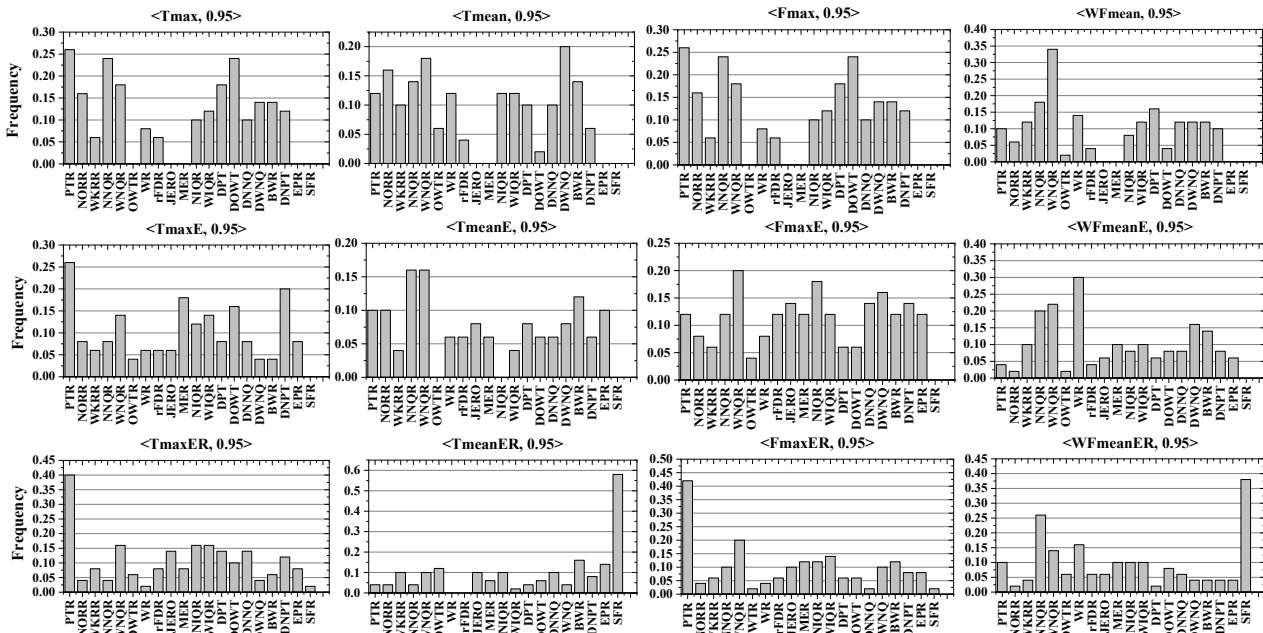


Fig. 11. Frequency of the normalized terminals over 50 independent runs in the example scenarios.

and inspire the future design of IF-included dispatching rules, this section investigates the distributions of input features of IF operations in the produced rules. Since we restrict that G2LGP-IF only uses the proposed normalized terminals as the input features of IF operations, we mainly analyze the frequency of normalized terminals in the best rules over 50 independent runs, as shown in Fig. 11. The frequently used normalized terminals imply crucial information in different decision situations. We select four scenarios respectively from the three scenario sets.

In the basic scenario set (the first row of Fig. 11), all the four example scenarios switch behaviors based on long-term information, such as the remaining operations and workload of a job (i.e., NORR and WKRR) and the machine that processes the next operation (i.e., NNQR and WNQR). Besides, the maximum objectives like Tmax and Fmax, clearly consider processing time (i.e., PTR and DPT) more than the mean objectives when switching behaviors. PTR and DPT are frequently used in their logical operations. To reduce the maximum values, maximum objectives also switch behaviors based on the deviation of operation waiting time (DOWT). When some operations have a large waiting time, those operations are likely to be delayed or have a long flowtime. Dispatching rules should prioritize these operations before it is too late. Contrarily, to improve global performance, Tmean and Fmean do not consider DOWT but consider more machines in the job shop. For example, Tmean emphasizes the deviation of workload in the next machine (DWNQ) and bottleneck workload ratio (BWR). Furthermore, WFmean has the highest rate of using the job weights (WR), with a frequency of nearly 0.15. It implies that switching dispatching behaviors based on the job weights can effectively improve the performance of weighted objectives.

In the energy-considered scenario set (the second row

of Fig. 11), we have two main observations. First, energy-related terminals are all highlighted in the second scenario set. For example, <TmaxE, 0.95> frequently considers MER, and <TmeanE, 0.95> frequently considers EPR. This implies that different energy prices need different dispatching behaviors. Second, the distributions of other normalized terminals are similar to those in the basic scenario set. For example, NNQR and WNQR are also frequently used in all four example scenarios, <TmaxE, 0.95>, and <FmaxE, 0.95> prefer processing time-related terminals such as PTR, and <WFmeanE, 0.95> changes behaviors based on WR. The two observations confirm that the G2LGP-IF dispatching rules simultaneously optimize tardiness- or flowtime-related objectives, and energy cost by dynamically adjusting the dispatching behaviors based on decision situations.

In the third scenario set that simultaneously optimizes tardiness/flowtime, energy cost, and response time, the response cost rate ratio SFR is extensively considered in TmeanER and WFmeanER, which means different response cost rates need different dispatching rules. However, the results show that changing behaviors based on SFR is not a good choice for optimizing maximum tardiness and maximum flowtime. <TmaxER, 0.95> and <FmaxER, 0.95> mainly change behaviors based on processing time (PTR) and the workload of the next machine queue (WNQR).

Based on the twelve example scenarios, we find that WNQR (and NNQR) and WIQR (and NIQR) are frequently used information in all different scenarios. WNQR and NNQR represent similar information indicating how large a work burden the next machine has. WIQR and NIQR represent similar information indicating how large a work burden the current machine has. Although these four normalized terminals might not be the most frequently used ones in the example scenarios, WNQR (and NNQR) and WIQR (and NIQR) have

a relatively high frequency (i.e., more than 0.1) in nearly all the cases. The observation implies that different machine situations likely need different dispatching rules.

The analyses for Fig. 11 have some new findings compared to existing feature analyses of DJSS [67], [68]. Some input features that did not show their importance in existing studies are highlighted by IF operations. For example, existing studies seldom see WIQ and NIQ as important features. But WIQR and NIQR which represent the same information as WIQ and NIQ are frequently used by IF operations. Existing studies seldom use the operation waiting time (OWT) in output rules. But our analyses show that operation waiting time (i.e., DOWT) is very useful in the IF operations of  $\langle T_{\max}, 0.95 \rangle$ ,  $\langle F_{\max}, 0.95 \rangle$ , and  $\langle T_{\max E}, 0.95 \rangle$ .

## VII. CONCLUSIONS

This paper aims to find a way to effectively evolve dispatching rules with IF operations for solving complicated DJSS problems. Specifically, we propose to evolve IF-included dispatching rules by G2LGP. To get rid of redundant and less effective IF branches, we first design a new set of normalized terminals for DJSS problems and further propose a set of grammar rules to restrict the available inputs, the number, and the locations of IF operations.

To comprehensively investigate the effectiveness of dispatching rules and our proposed method, we develop three scenario sets. The empirical results confirm that IF operations are crucial for dispatching rules in complex problems. The results have verified that using grammar rules to restrict the usage of IF operations in LGPHH is an effective way to harness IF operations. Armed with the proposed normalized terminals and grammar rules, the proposed method outperforms the state-of-the-art LGPHH method in terms of both effectiveness and interpretability. Further analyses highlight that IF operations greatly improve the flexibility of dispatching rules, performing different dispatching behaviors based on different decision situations. The analyses also find three important DJSS features that are missed by existing IF-excluded dispatching rules. This paper shows the great potential of IF-included dispatching rules in solving complex problems.

Standing on the achievement of this paper, we intend to include more advanced flow control operations such as FOR operations into GP's primitive set in the future and investigate the corresponding grammar rules for them. This will facilitate GP to evolve more concise and powerful programs.

## REFERENCES

- [1] L. Lamorgese and C. Mannino, "A noncompact formulation for job-shop scheduling problems in traffic management," *Operations Research*, vol. 67, no. 6, pp. 1586–1609, 2019.
- [2] A. D'Ariano, D. Pacciarelli, M. Pistelli, and M. Pranzo, "Real-time scheduling of aircraft arrivals and departures in a terminal maneuvering area," *Networks*, vol. 65, no. 3, pp. 212–227, 2015.
- [3] A. P. Vepsäläinen and T. E. Morton, "Priority rules for job shops with weighted tardiness costs," *Management Science*, vol. 33, pp. 1035–1047, 1987.
- [4] D. Jakobović and L. Budin, "Dynamic Scheduling with Genetic Programming," in *Proceedings of European Conference on Genetic Programming*, 2006, pp. 73–84.
- [5] 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.
- [6] J. R. Koza, *Genetic Programming: On the Programming of Computers By Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.
- [7] Y. Bi, B. Xue, and M. Zhang, "Genetic programming-based evolutionary deep learning for data-efficient image classification," *IEEE Transactions on Evolutionary Computation*, pp. 1–15, 2022, early access.
- [8] D. Magalhães, R. H. Lima, and A. Pozo, "Creating deep neural networks for text classification tasks using grammar genetic programming," *Applied Soft Computing*, vol. 135, p. 110009, 2023.
- [9] H. Zhang, A. Zhou, Q. Chen, B. Xue, and M. Zhang, "SR-Forest: A Genetic Programming-based Heterogeneous Ensemble Learning Method," *IEEE Transactions on Evolutionary Computation*, pp. 1–15, 2023, early access.
- [10] D. Wittenberg and F. Rothlauf, "Small Solutions for Real-World Symbolic Regression Using Denoising Autoencoder Genetic Programming," in *Proceedings of European Conference on Genetic Programming*, 2023, pp. 101–116.
- [11] E. Pantridge and T. Helmuth, "Solving Novel Program Synthesis Problems with Genetic Programming using Parametric Polymorphism," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2023, pp. 1175–1183.
- [12] S. Nguyen, Y. Mei, and M. Zhang, "Genetic programming for production scheduling: a survey with a unified framework," *Complex & Intelligent Systems*, vol. 3, pp. 41–66, 2017.
- [13] F. Zhang, S. Nguyen, Y. Mei, and M. Zhang, *Genetic Programming for Production Scheduling*. Springer Singapore, 2021.
- [14] M. Brameier and W. Banzhaf, *Linear Genetic Programming*. Springer US, 2007.
- [15] Z. Huang, Y. Mei, F. Zhang, and M. Zhang, "A further investigation to improve linear genetic programming in dynamic job shop scheduling," in *IEEE Symposium Series on Computational Intelligence*, 12 2022, pp. 496–503.
- [16] Z. Huang, Y. Mei, F. Zhang, and M. Zhang, "Multitask linear genetic programming with shared individuals and its application to dynamic job shop scheduling," *IEEE Transactions on Evolutionary Computation*, pp. 1–15, 2023, doi: 10.1109/TEVC.2023.3263871.
- [17] Z. Huang, Y. Mei, F. Zhang, and M. Zhang, "Grammar-guided Linear Genetic Programming for Dynamic Job Shop Scheduling," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2023, pp. 1137–1145.
- [18] P. Nordin, "A compiling genetic programming system that directly manipulates the machine code," *Advances in genetic programming*, vol. 1, pp. 311–331, 1994.
- [19] M. Đurasević, D. Jakobović, and K. Knežević, "Adaptive scheduling on unrelated machines with genetic programming," *Applied Soft Computing Journal*, vol. 48, pp. 419–430, 2016.
- [20] T. Hildebrandt, J. Heger, and B. Scholz-reiter, "Towards Improved Dispatching Rules for Complex Shop Floor Scenarios — a Genetic Programming Approach," in *Proceedings of the Annual Conference on Genetic and Evolutionary Computation*, 2010, pp. 257–264.
- [21] C. D. Geiger, R. Uzsoy, and H. Aytug, "Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach," *Journal of Scheduling*, vol. 9, pp. 7–34, 2006.
- [22] A. Masood, Y. Mei, G. Chen, and M. Zhang, "Many-Objective Genetic Programming for Job-Shop Scheduling," in *Proceedings of IEEE Congress on Evolutionary Computation*, 2016, pp. 209–216.
- [23] A. Masood, G. Chen, Y. Mei, H. Al-Sahaf, and M. Zhang, "Genetic Programming with Pareto Local Search for Many-Objective Job Shop Scheduling," in *Proceedings of Australasian Joint Conference on Artificial Intelligence*, 2019, pp. 536–548.
- [24] A. Masood, G. Chen, Y. Mei, H. Al-Sahaf, and M. Zhang, "A Fitness-based Selection Method for Pareto Local Search for Many-Objective Job Shop Scheduling," in *Proceedings of IEEE Congress on Evolutionary Computation*, 2020, pp. 1–8.
- [25] D. Karunakaran, Y. Mei, G. Chen, and M. Zhang, "Dynamic job shop scheduling under uncertainty using genetic programming," in *Intelligent and Evolutionary Systems*, 2017, pp. 195–210.
- [26] D. Karunakaran, G. Chen, Y. Mei, and M. Zhang, "Toward evolving dispatching rules for dynamic job shop scheduling under uncertainty," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2017, pp. 282–289.
- [27] J. Park, Y. Mei, S. Nguyen, G. Chen, and M. Zhang, "Investigating the generality of genetic programming based hyper-heuristic approach



- to dynamic job shop scheduling with machine breakdown,” in *Proceedings of Australasian Conference on Artificial Life and Computational Intelligence*, 2017, pp. 301–313.
- [28] J. Park, Y. Mei, S. Nguyen, G. Chen, and M. Zhang, “Investigating a machine breakdown genetic programming approach for dynamic job shop scheduling,” in *Proceedings of European Conference on Genetic Programming*, 2018, pp. 253–270.
- [29] K. Miyashita, “Job-shop scheduling with genetic programming,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2000, pp. 505–512.
- [30] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, “Evolving reusable operation-based due-date assignment models for job shop scheduling with genetic programming,” in *Proceedings of European Conference on Genetic Programming*, 2012, pp. 121–133.
- [31] W. Kantschik and W. Banzhaf, “Linear-graph GP – A new GP structure,” in *Proceedings of European Conference on Genetic Programming*, 2002, pp. 83–92.
- [32] P. Whigham, “Grammatically-based Genetic Programming,” in *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, 1995, pp. 33–41.
- [33] R. I. McKay, N. X. Hoai, P. A. Whigham, Y. Shan, and M. O’neill, “Grammar-based genetic programming: A survey,” *Genetic Programming and Evolvable Machines*, vol. 11, pp. 365–396, 2010.
- [34] S. Forstenechne, D. Fagan, M. Nicolau, and M. O’Neill, “A grammar design pattern for arbitrary program synthesis problems in genetic programming,” 2017, pp. 262–277.
- [35] S. Forstenechne, D. Fagan, M. Nicolau, and M. O’Neill, “Extending program synthesis grammars for grammar-guided genetic programming,” 2018, pp. 197–208.
- [36] L. Ingelse, J.-I. Hidalgo, J. M. Colmenar, N. Lourenço, and A. Fonseca, “Comparing Individual Representations in Grammar-Guided Genetic Programming for Glucose Prediction in People with Diabetes,” in *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, 2023, pp. 2013–2021.
- [37] M. Hughes, M. Goerigk, and T. Dokka, “Automatic generation of algorithms for robust optimisation problems using grammar-guided genetic programming,” *Computers and Operations Research*, vol. 133, 2021.
- [38] C. Ryan, J. Collins, and M. O. Neill, “Grammatical evolution: Evolving programs for an arbitrary language,” in *Proceedings of European Conference on Genetic Programming*, 1998, pp. 83–96.
- [39] M. O’Neill and C. Ryan, “Grammatical evolution,” *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 4, pp. 349–358, 2001.
- [40] N. Lourenço, F. Pereira, and E. Costa, “SGE: A Structured Representation for Grammatical Evolution,” in *Proceedings of International Conference on Artificial Evolution*, 2015, pp. 136–148.
- [41] N. Lourenço, F. B. Pereira, and E. Costa, “Unveiling the properties of structured grammatical evolution,” *Genetic Programming and Evolvable Machines*, vol. 17, no. 3, pp. 251–289, 2016.
- [42] W. M. Leung, L. K. Sak, M. L. Wong, and K. S. Leung, “Applying logic grammars to induce sub-functions in genetic programming,” in *Proceedings of the IEEE Conference on Evolutionary Computation*, 1995, pp. 737–740.
- [43] M. L. Wong and K. S. Leung, *Data Mining Using Grammar-Based Genetic Programming and Applications*. USA: Kluwer Academic Publishers, 2000.
- [44] B. J. Ross, “Logic-based genetic programming with definite clause translation grammars,” *New Generation Computing*, vol. 19, no. 4, pp. 313–337, 2001.
- [45] D. J. Montana, “Strongly typed genetic programming,” *Evolutionary Computation*, vol. 3, p. 199–230, 1995.
- [46] W. Pei, B. Xue, L. Shang, and M. Zhang, “A Threshold-free Classification Mechanism in Genetic Programming for High-dimensional Unbalanced Classification,” in *Proceedings of IEEE Congress on Evolutionary Computation*, 2020, pp. 1–8.
- [47] V. Manahov, R. Hudson, and P. Linsley, “New evidence about the profitability of small and large stocks and the role of volume obtained using strongly typed genetic programming,” *Journal of International Financial Markets, Institutions and Money*, vol. 33, pp. 299–316, 11 2014.
- [48] K. Michell and W. Kristjanpoller, “Generating trading rules on us stock market using strongly typed genetic programming,” *Soft Computing*, vol. 24, pp. 3257–3274, 3 2020.
- [49] E. Christodoulaki and M. Kampouridis, “Using strongly typed genetic programming to combine technical and sentiment analysis for algorithmic trading,” in *Proceedings of IEEE Congress on Evolutionary Computation*, 2022, pp. 1–8.
- [50] S. Wappler and J. Wegener, “Evolutionary unit testing of object-oriented software using strongly-typed genetic programming,” in *Proceedings of Annual Conference on Genetic and Evolutionary Computation*, 7 2006, pp. 1925–1932.
- [51] G. Dick and P. A. Whigham, “Initialisation and grammar design in grammar-guided evolutionary computation,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2022, pp. 534–537.
- [52] A. Fonseca and D. Poças, “Comparing the expressive power of strongly-typed and grammar-guided genetic programming,” in *Proceedings of Genetic and Evolutionary Computation Conference*, 2023, pp. 1–9.
- [53] L. F. D. P. Sotto and V. V. de Melo, “A probabilistic linear genetic programming with stochastic context-free grammar for solving symbolic regression problems,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2017, pp. 1017–1024.
- [54] R. Hunt, J. Richard, and M. Zhang, “Evolving Dispatching Rules with Greater Understandability for Dynamic Job Shop Scheduling. Technical report ECSTR-15-6.” Victoria University of Wellington, Tech. Rep., 2015.
- [55] T. P. Pawlak and M. O’Neill, “Grammatical evolution for constraint synthesis for mixed-integer linear programming,” *Swarm and Evolutionary Computation*, vol. 64, p. 100896, 7 2021.
- [56] F. Michael, L. David, K. Stepan, C. Holger, and O. Michael, “Evolving coverage optimisation functions for heterogeneous networks using grammatical genetic programming,” in *Proceedings of Applications of Evolutionary Computation*, 2016, pp. 219–234.
- [57] T. Saber, D. Lynch, D. Fagan, S. Kucera, H. Claussen, and M. O’Neill, “Hierarchical Grammar-Guided Genetic Programming Techniques for Scheduling in Heterogeneous Networks,” in *Proceedings of IEEE Congress on Evolutionary Computation*, 2020, pp. 1–8.
- [58] R. F. R. Correa, H. S. Bernardino, J. M. de Freitas, S. S. R. F. Soares, L. B. Gonçalves, and L. L. O. Moreno, “A Grammar-based Genetic Programming Hyper-Heuristic for Corridor Allocation Problem,” in *Proceedings of Brazilian Conference on Intelligent Systems*, 2022, pp. 504–519.
- [59] C. S. Pereira, D. M. Dias, M. A. C. Pacheco, M. M. Vellasco, A. V. Abs Da Cruz, and E. H. Hollmann, “Quantum-Inspired Genetic Programming Algorithm for the Crude Oil Scheduling of a Real-World Refinery,” *IEEE Systems Journal*, vol. 14, no. 3, pp. 3926–3937, 2020.
- [60] C. S. Pereira, D. M. Dias, L. Martí, and M. Vellasco, “A Multi-Objective Decomposition Optimization Method for Refinery Crude Oil Scheduling through Genetic Programming,” in *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, 2023, pp. 1972–1980.
- [61] L. Ingelse and A. Fonseca, “Domain-Aware Feature Learning with Grammar-Guided Genetic Programming,” in *Proceedings of European Conference on Genetic Programming*, 2023, pp. 227–243.
- [62] Y. Mei, S. Nguyen, and M. Zhang, “Evolving time-invariant dispatching rules in job shop scheduling with genetic programming,” in *Proceedings of European Conference on Genetic Programming*, 2017, pp. 147–163.
- [63] Z. Huang, Y. Mei, and M. Zhang, “Investigation of Linear Genetic Programming for Dynamic Job Shop Scheduling,” in *Proceedings of 2021 IEEE Symposium Series on Computational Intelligence*, dec 2021, pp. 1–8.
- [64] M. Đurasević and D. Jakobović, “A survey of dispatching rules for the dynamic unrelated machines environment,” *Expert Systems with Applications*, vol. 113, pp. 555–569, 2018.
- [65] A. Baykasoğlu, M. Göçken, and L. Özbakir, “Genetic Programming Based Data Mining Approach to Dispatching Rule Selection in a Simulated Job Shop,” *Simulation*, vol. 86, no. 12, pp. 715–728, 2010.
- [66] Y. Liu, H. Dong, N. Lohse, S. Petrovic, and N. Gindy, “An investigation into minimising total energy consumption and total weighted tardiness in job shops,” *Journal of Cleaner Production*, vol. 65, pp. 87–96, feb 2014.
- [67] Y. Mei, S. Nguyen, B. Xue, and M. Zhang, “An Efficient Feature Selection Algorithm for Evolving Job Shop Scheduling Rules with Genetic Programming,” *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 1, no. 5, pp. 339–353, 2017.
- [68] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, “Evolving Scheduling Heuristics via Genetic Programming With Feature Selection in Dynamic Flexible Job-Shop Scheduling,” *IEEE Transactions on Cybernetics*, vol. 51, no. 4, pp. 1797–1811, 2021.