Correlation Coefficient based Recombinative Guidance for Genetic Programming Hyper-heuristics in Dynamic Flexible Job Shop Scheduling

Fangfang Zhang, *Graduate Student Member, IEEE,* Yi Mei, *Senior Member, IEEE,*
Su Nguyen, *Member, IEEE,* and Mengjie Zhang, *Fellow, IEEE*

***Abstract*—Dynamic flexible job shop scheduling is a challenging combinatorial optimisation problem due to its complex environment. In this problem, machine assignment and operation sequencing decisions need to be made simultaneously under the dynamic environments. Genetic programming, as a hyper-heuristic approach, has been successfully used to evolve scheduling heuristics for dynamic flexible job shop scheduling. However, in traditional genetic programming, recombination between parents may disrupt the beneficial building-blocks by choosing the crossover points randomly. This paper proposes a recombinative mechanism to provide guidance for genetic programming to realise effective and adaptive recombination for parents to produce offspring. Specifically, we define a novel measure for the importance of each subtree of an individual, and the importance information is utilised to decide the crossover points. The proposed recombinative guidance mechanism attempts to improve the quality of offspring by preserving the promising building-blocks of one parent and incorporating good building-blocks from the other. The proposed algorithm is examined on six scenarios with different configurations. The results show that the proposed algorithm significantly outperforms the state-of-the-art algorithms on most tested scenarios, in terms of both final test performance and convergence speed. In addition, the rules obtained by the proposed algorithm have good interpretability.**

***Index Terms*—Recombinative Guidance, Correlation Coefficient, Genetic Programming, Hyper-heuristics, Dynamic Flexible Job Shop Scheduling.**

I. Introduction

Job shop scheduling (JSS) [1] is an important but challenging optimisation problem in computer science and operations research in which jobs are assigned to machines at particular times. For JSS, the task is to process a number of jobs (e.g., each job has a sequence of operations) by a set of machines. Flexible JSS (FJSS) [2] is a relaxation of JSS where each operation can be processed on a set of candidate machines. In FJSS, we need to make two decisions simultaneously. One is *machine assignment* (i.e., assign an operation to a particular machine) and the other is *operation sequencing* (i.e., choose an operation as the next operation to be processed by an idle machine). Dynamic FJSS (DFJSS) [3] aims to optimise the machine resources under a dynamic environment with unpredicted events, such as new job arrivals [4], [5], [6] and machine breakdown [7], [8]. DFJSS is an NP-hard problem [9]. Job shop environments are crucial in many industries, such as manufacturing processes [10], [11] and cloud computing [12]. As a result, the ability to create efficient production schedules for job shops can be a key value added propositions for manufacturers. It is hardly possible to find effective solutions by hand, especially for large scale problems or in dynamic environments.

Exact optimisation methods such as dynamic programming [13] and integer linear programming [14] are usually not applicable for real-world large problem instances. *Approximate solution optimisation methods*, such as simulated annealing [15], tabu search [16], particle swarm optimisation [17] and genetic algorithms [18], which aim to find a near-optimal solution within a reasonable time budget, have been widely applied for JSS. However, most of them can hardly handle dynamic problems efficiently because the re-optimisation process is still too slow to be able to react in real-time. *Scheduling heuristics* such as dispatching rules [19], [20], [21], might be the most popularly used heuristics for dynamic JSS. Scheduling heuristics make decisions based on the priorities of machines or operations at the decision points. There are two main reasons for the success of scheduling heuristics in dynamic JSS. One is their ability to handle large scale problems. The other is their efficiency to make real-time decisions with dynamic events. A scheduling heuristic in DFJSS consists of a *routing rule* (i.e., for machine assignment) and a *sequencing rule* (i.e., for operation sequencing). There are several rules such as SPT (shortest processing time) and some composite rules [22] which have been identified as effective rules for JSS. However, they are manually designed by experts, which

Manuscript received XXX; revised XXX; revised XXX; revised November 8, 2020; accepted January 28, 2021. Date of publication XXX; date of current version XXX. This work was supported in part by the Marsden Fund of New Zealand Government under Contracts VUW1509 and VUW1614, the Science for Technological Innovation Challenge (SfTI) fund under grant E3603/2903, and the MBIE SSIF Fund under Contract VUW RTVU1914. This work of Fangfang Zhang was supported by the China Scholarship Council (CSC)/Victoria University Scholarship. *(Corresponding author: Fangfang Zhang.)*

Fangfang Zhang, Yi Mei, and Mengjie Zhang are with the Evolutionary Computation Research Group, School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand (e-mail: fangfang.zhang@ecs.vuw.ac.nz; yi.mei@ecs.vuw.ac.nz; mengjie.zhang@ecs.vuw.ac.nz).

Su Nguyen is with the Centre for Data Analytics and Cognition, La Trobe University, Melbourne, VIC 3086, Australia (e-mail: P.Nguyen4@latrobe.edu.au).

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

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

Digital Object Identifier XXX

highly relies on domain knowledge, especially for complex scenarios. In addition, many potential rules have never been investigated [18]. For complex DFJSS problems, on the other hand, human experts are often unable to identify all the subtle and interrelated conditions between different types of attributes to create and evaluate rules, or the use of highly experienced experts is too expensive.

Tree-based genetic programming (GP) [23], as a hyper-heuristic approach (GPHH), has been successfully applied to evolve scheduling heuristics automatically for JSS [24], [25], [26], [27], [28], [29], [30]. In an evolutionary computation algorithm, genetic operators play important roles for generating offspring. The crossover is an essential genetic operator for GP to produce offspring during the evolutionary process. In essence, the crossover is a recombination of different materials from the parents. In traditional GP, subtrees are randomly chosen from two parents to swap with each other to produce two offspring. However, the importance of subtrees in each individual can be different. Some subtrees are redundant or less important, and removing them might not affect the fitness of an individual too much. On the other hand, some subtrees play essential roles for an individual and losing them will cause considerable loss to the fitness. The random way of recombination may disrupt beneficial building-blocks.

To the best of our knowledge, little is yet known to improve the recombinative effectiveness of GPHH via the crossover for JSS. Riccardo et al. provided a comprehensive general schema theory for GP with subtree-swapping crossover in [31], [32]. This theory suggests that the biases of GP operators can be beneficial for different purposes, such as improving the quality of the offspring and controlling the size or shape of the offspring. However, it is challenging when we apply bias in GP in practice, such as the DFJSS problem. *One critical challenge* is how to measure the subtrees based on the desired purpose. *The other challenge* is how to apply the expected "biases" in GP for a specific problem. The guided subtree selection strategy proposed in [33] is the first attempt to improve the quality of offspring by guiding the behaviour of genetic operators of GP for solving the DFJSS problem. The importance of a subtree is measured by a simple average score based on the occurrences of features. However, using the occurrences of features to measure the importance of subtrees may not be accurate due to the redundant branches in GP.

To address the above issues, this paper proposes to use correlation coefficient based recombinative guidance to improve the quality of offspring for GPHH in DFJSS via the crossover operator. The developed importance measure reflects the degree of relationship between the behaviour of the subtree and the entire tree. The probability of a subtree to be chosen is then set based on its importance. An offspring is generated by replacing an unimportant subtree from one parent with an important subtree from the other.

The goal of this paper is to *develop an effective correlation coefficient based recombination guidance for GPHH to evolve effective scheduling heuristics in the DFJSS problem automatically.* The proposed algorithm is expected to help GPHH find better scheduling heuristics more efficiently by improving the quality of the produced offspring. Specifically, this work has the following research objectives:

1) Develop an effective way to measure the importance of subtrees of an individual according to the characteristics of the investigated DFJSS problem.
2) Propose a novel recombinative guidance mechanism for the crossover operator in GPHH.
3) Analyse the effectiveness and efficiency of the proposed algorithm in terms of the performance of evolved rules, the convergence speed, and training time.
4) Analyse how the proposed algorithm influences the behaviour of GPHH to select crossover points.
5) Analyse the evolved scheduling heuristics in terms of size and rule structure.

The major contribution of this paper is to propose a new effective recombinative guidance for GP to generate offspring by measuring the importance of the subtrees. The way of measuring the importance of subtrees can provide guidance for developing subtree importance measures for other problems. In addition, the algorithm analyses provide us with a better understanding of the mechanism of GP based algorithms from the perspective of building-blocks recombination.

The rest of this paper is organised as follows. Section II gives a background introduction. Detailed descriptions of the proposed algorithm are given in Section III. The experiment design is shown in Section IV, followed by results and discussions in Section V. Further analyses are conducted in Section VI. Finally, Section VII concludes the paper.

II. Background

This section provides a brief introduction of JSS with a focus on DFJSS, scheduling heuristics for DFJSS, and how to use GPHH for solving the DFJSS problem. In addition, related studies on genetic operators of GP are reviewed.

A. Dynamic Flexible Job Shop Scheduling

Job shop scheduling focuses on improving production efficiency in a shop floor. In FJSS problem, n jobs $J = \{J_1, J_2, ..., J_n\}$ need to be processed by m machines $M = \{M_1, M_2, ..., M_m\}$. Each job J_j has an arrival time $at(J_j)$ and a sequence of operations $O_j = (O_{j1}, O_{j2}, ..., O_{ji})$. Each operation O_{ji} can only be processed by one of its candidate machines $\pi(O_{ji})$ and its processing time $\delta(O_{ji})$ depends on the machine that processes it. It implies that there are two types of decisions in FJSS, i.e.,, routing decision and sequencing decision. DFJSS aims to make two decisions simultaneously under dynamic environment with unpredicted events. In this paper, we focus on the dynamic job arrivals. That is, the information of a job is unknown until it arrives at the job shop. The following constraints must be satisfied in the problem.

- The order of operations for each job is predefined, and one cannot start processing an operation until all its precedent operations have been processed.
- Each operation can be processed only by one of its candidate machines.
- Each machine can process at most one operation at a time.

- The scheduling is non-preemptive, i.e., once start, the processing of an operation cannot be stopped or paused until it is completed.

The objective of the scheduling is to assign the operations to proper machines and sequence the operations in the queue of the machines so as to optimise some objective functions while satisfying all the above constraints. In this paper, we consider three commonly used flowtime-related objective functions, which are calculated as follows. It is noted that the due dates of jobs are not considered in this paper.

- Max-flowtime $= max\{C_1 - r_1, C_i - r_i, ..., C_n - r_n\}$
- Mean-flowtime $= \frac{\sum_{i=1}^{n}\{C_i - r_i\}}{n}$
- Mean-weighted-flowtime $= \frac{\sum_{i=1}^{n} w_i * \{C_i - r_i\}}{n}$

where C_i is the completion time of job J_i, r_i is the release time of J_i, and w_i is the weight of J_i.

B. Scheduling Heuristics for DFJSS

Two decisions need to be made simultaneously in DFJSS. Scheduling heuristics (i.e., routing rules and sequencing rules) are needed in DFJSS. Tay et al. [11] proposed to use GP to evolve the sequencing rule by fixing the routing rule as a manually designed rule for FJSS. It is a simple way to solve the FJSS problems with scheduling heuristics. Yska et al. [4] introduced a cooperative coevolution framework with GP (CCGP) to evolve routing and sequencing rules simultaneously. The proposed method shows its superiority due to the coevolution mechanism for evolving two rules simultaneously. Zhang et al. [5] introduced GP with multi-tree representation for evolving two rules together. The proposed method is promising in terms of the effectiveness, efficiency, and the sizes of evolved rules. This paper adopts the CCGP framework. Since the crossover operation of routing and sequencing is independent, CCGP is suitable for validating the effectiveness of the proposed crossover operator.

Due to the precedent constraint, only *ready operations* are allowed to be allocated to machines. Two kinds of operations will become ready operations. One is the first operation of a job arrived at the shop floor. The other is the subsequent operation whose preceding operation is just finished.

Fig. 1 shows an example of the decision making processes of DFJSS with scheduling heuristics. There are three machines, each with several operations waiting in its queue. The operation O_{81} is being processed on *Machine 3*.

a) Routing decision: Once an operation becomes a ready operation (*a routing decision situation is encountered*), it will be allocated to the machine with the highest priority according to the routing rule. For example, when a new job (J_9) arrives the job shop, its first operation O_{91} is allocated to *Machine 2* which has the highest priority value among the three machines according to the routing rule. In addition, as O_{81} is just finished, its next operation (O_{82}) becomes a ready operation and is allocated to *Machine 1* by the routing rule.

b) Sequencing decision: When a machine (e.g., *Machine 1*) becomes idle, and its queue is not empty (*a sequencing decision situation is encountered*), the sequencing rule will be used to calculate the priority value of each operation in its queue. The operation with the highest priority is then chosen as the next operation to be processed (e.g., O_{32} is selected in this case to be processed on *Machine 1*).

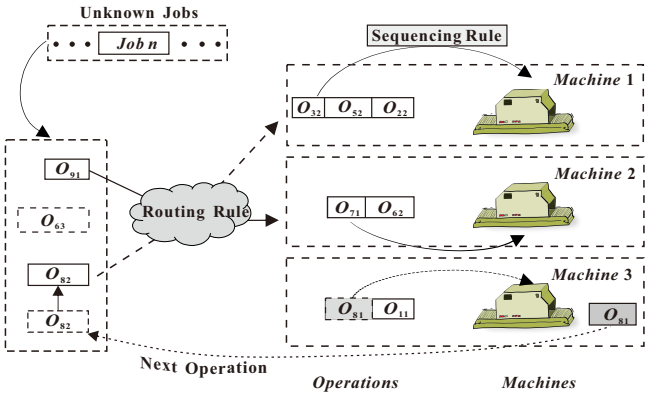


Fig. 1. An example of decision making processes of dynamic flexible job shop scheduling with scheduling heuristics.

C. Genetic Programming Hyper-heuristics for DFJSS

A hyper-heuristic [34] seeks to select or generate heuristics to solve hard computational search problems efficiently. The unique characteristic is that hyper-heuristic works on heuristic space instead of the solution space. There are two types of hyper-heuristics methods [35]. One is *heuristic selection* which aims to choose existing heuristics for different scenarios. The other is *heuristic generation* which aims to generate new high-level heuristic using existing low-level heuristics. In JSS, heuristic generation is commonly used to evolve scheduling heuristics from the basic job shop state features.

GP, as a hyper-heuristic method [36], has been successfully applied to evolve scheduling heuristics for combinatorial optimisation problems such as packing [37], [38], timetabling [39], [40], arc routing [41], and scheduling [42], [43], [44], [45], [46], [47]. GP can automatically generate computer programs to solve problems without needing much domain knowledge. There are some advantages of using GPHH for JSS. One is its flexible representation. This implies that we do not need to define the structure of rules in advance. The other is that the tree-based programs obtained by GP provide us with opportunities to understand the behaviour of the evolved rules, which is very important for real-world applications.

Fig. 2 shows the overall research process of GPHH for DFJSS in this paper. In the training phase, GPHH is used to train heuristics based on a set of training instances. The outputs of the training phase are heuristics (routing and sequencing rules) rather than solutions (schedules). In the test phase, the evolved heuristics obtained in the training phase are tested on unseen instances to generate the final schedules. Based on the final schedules, the processing information of jobs, such as the starting and finishing time of each operation, can be confirmed. Finally, the performance of evolved scheduling heuristics can be measured along with the objectives such as flowtime.

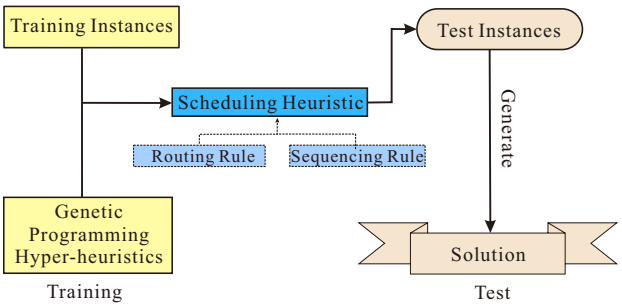


Fig. 2. The overall process of genetic programming hyper-heuristic for dynamic flexible job shop scheduling.

D. Related Work on Genetic Operators of GP

The flexibility of GP makes it stand out among lots of evolutionary computation algorithms. However, GP still has some limitations. For example, an individual is likely to behave very differently and become much worse even after small changes. It is not fully clear what kinds of genetic operators can make the performance of GP better. In terms of the way to enhance the effectiveness of the genetic operators, we group the related studies into three categories. In this section, we review the related studies on genetic operators of GP with a focus on the crossover.

Adaptive rate for genetic operators. Changing the rates of genetic operators is a simple way to improve the effectiveness of producing offspring. Adaptive operator selection rates with designed reward policies were proposed in [48] for GP. The results show that adaptive rate selection is an effective way to improve the performance of GP. Different methods of adapting the probabilities of genetic operators were proposed in [49] based on population-level, fitness, or individual-level information of GP. In [50], an adaptive decreasing mutation rate was proposed for GP to solve the truss structure optimisation problem. These methods succeed by balancing exploration and exploitation during the evolutionary process.

Depth-dependent crossover. Intuitively, the depth of crossover point is an important factor for the quality of offspring because the performance of subtree is related to the depth to some extent. A general heuristic that can be used to guide the development of the most effective depth-control strategy for any given problem was discussed in [51]. A "height-fair" crossover operator that only allowed to swap subtrees with the same depth was proposed in [52]. A depth selection probability was defined in [53] to ensure the node towards the root of an individual has a higher probability of being chosen as a crossover point than the ones towards leaves. These methods aim to bias the crossover depth to improve the performance of GP. However, it is not straightforward to apply them to DFJSS, since the optimal depth is not known.

Semantic crossover. The information of GP individuals can be used to produce offspring that bias to some semantics. Two new geometric search operators were developed in [54] to fulfil precise semantic requirements for symbolic regression. A novel crossover operator was proposed in [55] to address the exponential growth in the size of the individuals. The constrained dimensionally aware GP was designed in [56] to ensure only semantically correct individuals can be generated to improve the interpretability of evolved rules for JSS. The crossover bias for having the more fit parent as the root parent was presented in [57]. These methods tend to achieve the goal by utilising the semantics of GP individuals during the evolutionary process.

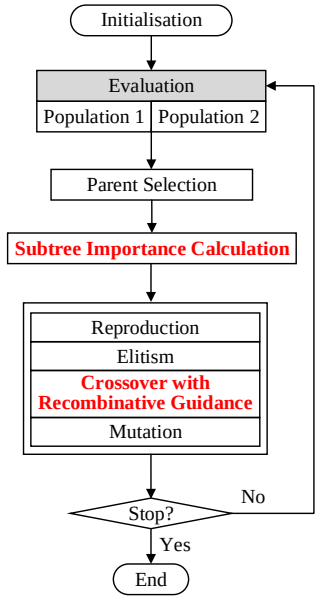


Fig. 3. The flowchart of the proposed algorithm.

Although there are some studies [58], [59], [60] on genetic operators of GP, little research has been conducted on the crossover to improve the quality of offspring by investigating the importance of subtrees directly. To this end, this paper aims to improve the effectiveness of crossover by proposing an effective and adaptive recombinative guidance mechanism based on the importance of subtrees.

III. The Proposed Algorithm

This paper proposes a correlation coefficient based recombinative guidance mechanism to improve the quality of the produced offspring based on the importance of subtrees. The framework of the algorithm is described first, followed by the key components of the algorithm.

A. The Framework of the Proposed Algorithm

Fig. 3 shows the flowchart of the proposed algorithm. The main processes are the same as the traditional GP. It starts with initialising the population randomly, and then evaluates the individuals in the population. It is noted that there are two subpopulations. One subpopulation is designed for evolving routing rules, and the other for evolving sequencing rules. However, there are two new components that are different from the traditional GP, which are highlighted in red in Fig. 3. First, the importance of each subtree of parents is calculated before the mating process. Second, during the mating process, the crossover is conducted based on the proposed recombinative guidance mechanism.

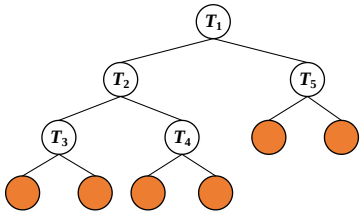


Fig. 4. An example of a labelled tree-based GP individual.

TABLE I
AN EXAMPLE OF THE CALCULATION FOR **DECISION VECTOR** OF THE SUBTREES IN AN INDIVIDUAL.

Subtree (T_i)	M_1	M_2	M_3	Decision Vector (d_i)
T_1	100 ①	150 ②	200 ③	(1, 2, 3)
T_2	300 ①	320 ②	350 ③	(1, 2, 3)
T_3	140 ③	120 ②	110 ①	(3, 2, 1)
T_4	100 ①	160 ③	130 ②	(1, 3, 2)

According to the framework of the proposed algorithm, the two research questions in this paper are how to measure the importance of subtrees, and how to apply the subtree importance information to guide the recombination between parents via the crossover.

B. Calculation of the Importance of Subtrees

An individual (i.e., a tree) in GP consists of multiple subtrees. Fig. 4 shows an example of a GP individual with five subtrees. Each subtree can be considered as an independent "individual", which has its own decision-making ability. To characterise the behaviour of a subtree T_i under a decision situation, this paper uses a decision vector $\vec{d_i}$ which is the list of the ranks of the candidates (i.e., machines for routing decision situations, or operations for sequencing decision situations) decided by T_i.

Table I shows an example of how to calculate the decision vectors of subtrees. The individual is a routing rule, and it has four subtrees. For simplicity, the decision situation is to allocate a ready operation to one of the three candidate machines. The numbers in the machine columns are the priority values (i.e., real numbers) based on the corresponding subtrees (routing rules) and the ranks of the machines based on the priority values. A machine with a smaller priority value has a better priority than other machines. Finally, the decision vectors are composed of the ranks. It shows that different subtrees can have the same decisions (T_1 and T_2), opposite decisions (T_1 and T_3) or partially same decisions (T_1 and T_4). Since a decision is made solely based on the ranks rather than the exact priority values of the candidates, this paper focuses on the relationship in terms of the ranks rather than the priority values.

Pearson and Spearman correlation coefficients [61] are two commonly used measures of the relationship between two variables. Pearson's correlation coefficient assesses linear relationships [62], while Spearman's correlation coefficient assesses monotonic relationships (regardless of whether they are linear or not). Specifically, the Spearman correlation coefficient measures the statistical dependence between the rank values of two variables. The decision making processes of subtrees in DFJSS are based on the ranks of machines or operations, therefore the Spearman correlation coefficient is a natural candidate for measuring the correlation between the behaviour of subtrees. This paper uses correlation c_i between the decisions (i.e., $\vec{d_i}$ and $\vec{d_1}$) made by T_i and T_1 (i.e., the whole tree) to measure the importance of a subtree T_i. The values range between -1 and 1. If $|c_i|$ is close to 1, the behaviour of T_i is highly consistent with T_1 (either positively or negatively), and T_i is an important subtree for an individual. If $|c_i|$ is close to 0, the behaviour of T_i is almost irrelevant with the behaviour of T_1, and thus T_i is not important subtree for T_1.

TABLE II
AN EXAMPLE OF THE CALCULATIONS FOR **CORRELATION** OF SUBTREES IN AN INDIVIDUAL IN A DECISION SITUATION.

Subtree (T_i)	Decision Vector (d_i)	Correlation (c_i)
T_1	(1, 2, 3, 4, 5, 6)	1
T_2	(1, 2, 3, 4, 5, 6)	1
T_3	(1, 3, 2, 6, 4, 5)	0.77
T_4	(6, 5, 1, 2, 3, 4)	-0.43
T_5	(6, 5, 4, 3, 2, 1)	-1

Algorithm 1: Calculation of the importance of a subtree

Input : An individual T, a subtree T_i of T, and a set of decision situations
Output: The importance of the subtree T_i
$S(T_i) \leftarrow null$, $\vec{d_i} \leftarrow null$
$c_i \leftarrow 0$, $sum(c_i) \leftarrow 0$
for *j = 1 to* $|decisionSituations|$ **do**
  Calculate the priority values of machines or operations based on the subtree T_i
  Rank machines or operations based on the priority values
  $\vec{d_i} \leftarrow$ get the decision vector of subtree T_i based on the ranks
  $c_i \leftarrow$ calculate the correlation of $\vec{d_i}$ and $\vec{d_1}$
  $sum(c_i) \leftarrow sum(c_i) + |c_i|$
end
$S(T_i) \leftarrow \frac{sum(c_i)}{|decisionSituations|}$
return $S(T_i)$

Table II shows an example of the calculations for the correlation of subtrees of the individual shown in Fig. 4. Different subtrees have different correlations (i.e., either positive or negative values). T_2 makes exactly the same decisions with T_1, and thus is a very important subtree of T_1. On the other hand, T_5 has a correlation of -1, which means its behaviour is completely reverse as the behaviour of T_1. In this case, T_5 is also a very important subtree of T_1, since its behaviour can be converted to be the same as that of T_1 by a slight modification, e.g., "$0 - T_1$". In contrast, T_3 and T_4 showed relatively weaker relationship with T_1, and thus are considered to be less important than T_2 and T_5.

The pseudo-code of measuring the importance of a subtree is shown in Algorithm 1. An absolute value of the correlation closer to 1 leads to a more important subtree. It is noted that the correlation between the behaviours of two trees can vary across different decision situations since the characteristics of

jobs (e.g., processing time) and machines (e.g., the workload) can be different. To have a reliable measure on the relationship, we sample a set of representative decision situations [63], and define the relationship between the behaviours of two trees to be the average correlation values over all the sampled decision situations. To sample a set of representative decision situations, this paper uses the WIQ (work in the queue) rule for routing and the SPT (shortest processing time) rule for sequencing, and runs a preliminary simulation with 5000 jobs on 10 machines, which generate about 50,000 routing and 50,000 sequencing decision situations. In [63], decision situations are created randomly containing between 2 and 20 jobs, which have been proven to have good performance in dynamic JSS. Taking the complexity of DFJSS into consideration, for both routing and sequencing decisions, the number of candidates, either machines or operations, is 7 in this paper. Then, we randomly select 50 routing and 50 sequencing decision situations from the generated routing and sequencing decisions with a length of 7. This means that each subtree has a decision vector with a dimension of 7. The fixed dimension length aims to get feasible correlation value. In each decision situation, the priority values of machines or operations are calculated (line 4) to get their ranks (line 5). A vector $\vec{d_i}$ denotes the decision made by T_i (line 6). The correlation c_i between T_i and T_1 is used to measure the importance of T_i (line 7). The final importance of subtree T_i is the average c_i over all decision situations (line 10).

C. Crossover with Recombinative Guidance

A GP crossover operator is typically conducted on two parents ($parent_1$ and $parent_2$) which are both considered to be promising individuals in the population (e.g., selected by tournament selection). For each parent, it is reasonable to choose the unimportant subtree and replace it with an important subtree from the other. Based on the importance of subtrees $S(T)$, two probability calculations are designed for different purposes. One is designed for selecting important subtrees while the other for selecting unimportant subtrees. Then, we design the crossover with recombinative guidance according to the probabilities.

The probability for each subtree. Based on the subtree importance information, this paper uses the idea of roulette wheel selection to choose the desired subtrees. We continue to use the example shown in Table II. We assume that there is only one decision situation, and the calculated importance (using Algorithm 1) of subtrees from T_1 to T_5 are 1, 1, 0.77, 0.43, and 1.

Fig. 5 shows an example of the two different ways to calculate the probability of each subtree in an individual for crossover. Fig. 5 (a) shows the way that tends to choose unimportant subtrees. The larger the score of a subtree, the lower the probability it has "↓" in the caption. Fig. 5 (b) shows the way that tends to choose important subtrees. The larger the score of a subtree, the higher the probability it has "↑" in the caption. In other words, the way to calculate the probabilities of subtrees follows the strategy in roulette-wheel selection, but according to the correlation values rather than fitness.

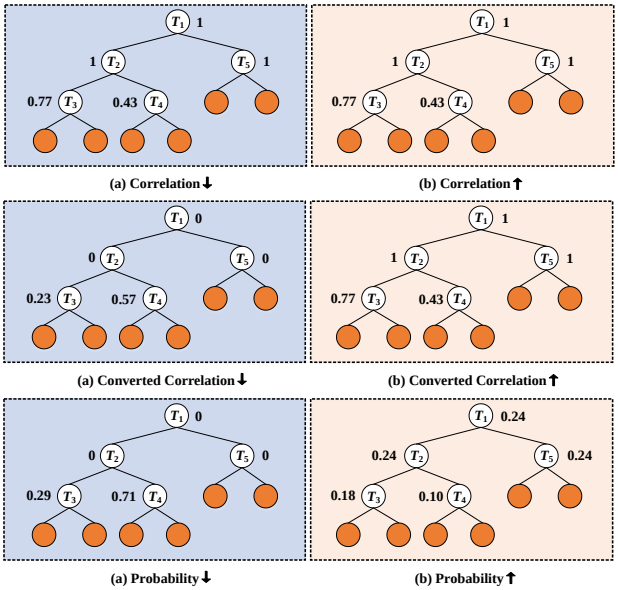


Fig. 5. An example of calculating the probabilities for subtrees. Fig. 5 (a) tends to choose unimportant subtrees while Fig. 5 (b) tends to choose important subtrees.

As shown in Fig. 5, at the beginning, the correlation values of subtrees are the same, as shown in Fig. 5 "(a) Correlation ↓" and Fig. 5 "(b) Correlation ↑". However, different from Fig. 5 (b), the correlation value of each subtree will be converted to $1 - S(T)$, as shown in Fig. 5 "(a) Converted Correlation ↓" since we tend to choose unimportant subtrees. The probabilities of subtrees are shown beside the function nodes in the last row of Fig. 5. The rank of the probability of subtrees in Fig. 5 (a) is $T_4 > T_3 > T_1 = T_2 = T_5$, and in Fig. 5 (b) is $T_1 = T_2 = T_5 > T_3 > T_4$. In this way, this paper can make sure that important and unimportant subtrees can be selected in accordance with the requirements.

The recombinative guidance mechanism. The pseudo-code of the proposed crossover operator is shown in Algorithm 2. The importance of subtrees of an individual is calculated before choosing important and unimportant subtrees based on roulette wheel selection (line 2 to line 8 for $parent_1$, line 9 to line 13 for $parent_2$). Finally, one offspring is produced by replacing the unimportant subtree $parent_1(T^*)^n$ from $parent_1$ with the important subtree $parent_2(T^*)^p$ from $parent_2$ (line 14). The other offspring is produced by replacing the unimportant subtree $parent_2(T^*)^n$ from $parent_2$ with the important subtree $parent_1(T^*)^p$ from $parent_1$ (line 15).

Continuing the example in Fig. 5, Fig. 6 shows an example of produced offspring with the proposed recombinative guidance. For $parent_1$ ($parent_2$), the unimportant subtree with an unhappy face from $parent_1$ ($parent_2$) is expected to be replaced by the important subtree with a happy face from $parent_2$ ($parent_1$), aiming to produce an even better offspring. The produced offspring are expected to preserve the promising building-blocks of one parent and incorporating good building-blocks from the other parent (i.e., produce offspring with more happy faces).

Algorithm 2: Crossover with recombinative guidance

Input : Two parents for the crossover ($parent_1$ and $parent_2$)
Output: The generated offspring ($offspring$)

1: set $offspring \leftarrow$ null
2: **if** $parent_1$ **then**
3: $S(T) \leftarrow$ Calculate the importance of subtrees (**Algorithm 1**)
4: $S(T)^p \leftarrow |S(T)|$
5: $S(T)^n \leftarrow 1 - |S(T)|$
6: $parent_1(T^*)^p \leftarrow$ Selected important subtree based on roulette wheel selection with $S(T)^p$
7: $parent_1(T^*)^n \leftarrow$ Selected unimportant subtree based on roulette wheel selection with $S(T)^n$
8: **end**
9: **if** $parent_2$ **then**
10: repeat from line 3 to line 5
11: $parent_2(T^*)^p \leftarrow$ Selected important subtree based on roulette wheel selection with $S(T)^p$
12: $parent_2(T^*)^n \leftarrow$ Selected unimportant subtree based on roulette wheel selection with $S(T)^n$
13: **end**
14: $offspring_1 \leftarrow$ produce offspring by replacing the subtree chosen from $parent_1(T^*)^n$ with $parent_2(T^*)^p$
15: $offspring_2 \leftarrow$ produce offspring by replacing the subtree chosen from $parent_2(T^*)^n$ with $parent_1(T^*)^p$
16: $offspring \leftarrow offspring_1 \cup offspring_2$
17: **return** $offspring$

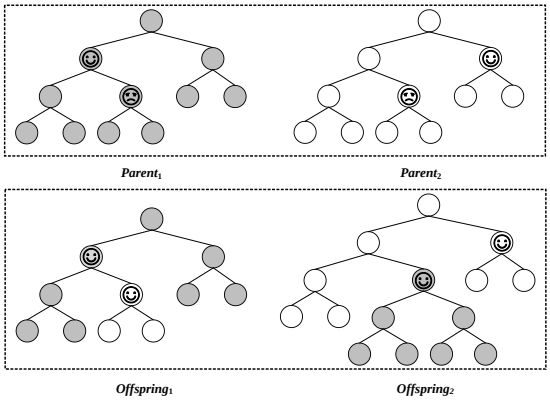


Fig. 6. An example of produced offspring from two parents with the proposed recombinative guidance mechanism.

D. Summary

The proposed algorithm aims to improve the effectiveness of crossover by introducing recombinative guidance mechanism rather than choosing subtrees randomly. We assume removing an unimportant subtree from an individual does not make a big difference to its fitness. However, introducing an important subtree to the position of the removed unimportant subtree has a high probability of making the individual better. It is noted that the idea in this paper is not limited to DFJSS but can benefit GP in general. An important issue is to design a proper measure for the subtree importance based on the specific problem to be solved. Taking the symbolic regression problem as an example, the subtree importance can be measured with sampling semantics [64].

IV. Experiment design

In this section, the simulation model, parameter setting, and the comparison design, are presented in detail.

A. Simulation Model

Simulation is widely used to investigate complex real-world problems [65]. A problem instance is an instantiation of the problem with a particular pseudo-random number generator seed [34]. Multiple different instances will be used to train and test the scheduling heuristics. At each generation, we only use one instance to evaluate the quality of evolved rules. However, the instance will be changed at each generation during the training process by assigning a different random seed to improve the generalisation of the GP algorithm. This strategy has been shown to be useful to improve the effectiveness and generalisation of evolved rules of GP [66], [67].

The simulation model contains 5000 jobs that need to be processed by 10 machines. Each job has a different number of operations that are randomly generated from a uniform discrete distribution between 1 and 10. The importance of jobs might be different, which are indicated by weights. The weights of 20%, 60%, and 20% jobs are set as 1, 2, and 4 following the setting in [63]. The processing time of each operation is sampled from a uniform discrete distribution with the range [1, 99]. The number of candidate machines for an operation follows a uniform discrete distribution between 1 and 10.

To verify the effectiveness and efficiency of the proposed algorithm, scenarios with different settings (i.e., different objectives and utilisation levels) are examined. New jobs will arrive over time according to a Poisson process with rate λ. The *utilisation level* (p) is an essential factor to simulate different scenarios. It indicates the proportion of time that a machine is expected to be busy. The expression is shown in Eq. (1), where μ is the average processing time of the machines. P_M is the probability of a job visiting a machine. For example, P_M is 2/10 if each job has two operations. A larger utilisation level leads to a busier and more complex job shop scenario.

$$\lambda = \mu * P_M / p \tag{1}$$

The first 1000 jobs are treated as warm-up jobs to get typical situations occurring in a long-term simulation of a dynamic job shop system, and jobs arrive as a continuous arrival process. We collect data from the next 5000 jobs. The simulation stops when the 6000th job is finished.

B. Parameter Setting

The terminals of GP serve as features of the problem to capture sufficient information about the problem. The terminal set of GP in this paper consists of a number of basic features of machines, jobs and operations in the job shop following the suggestions in [28], [34], [68].

Machine-related features: the states of machines such as workload are key factors for allocating operations to machines. A good schedule should not overload or underload a machine.

- NIQ is the number of operations in the machine's queue. It is designed to capture the workload of a machine by counting the number of operations in its queue.
- WIQ is the total processing time of the operations in the machine's queue. It is used to capture the workload of a machine by calculating the total processing time required for a machine to finish all the operations in its queue.

TABLE III
THE PARAMETER SETTING OF GP.

Parameter	Value
Number of subpopulations	2
Subpopulation size	512
The number of elites for each subpopulation	5
Method for initialising population	ramped-half-and-half
Initial minimum / maximum depth	2 / 6
maximal depth of programs	8
Crossover / Mutation / Reproduction rate	80% / 15% / 5%
Parent selection	Tournament selection with size 7
Terminal / non-terminal selection rate	10% / 90%
The number of generations	51

- MWT indicates the waiting time for the machine to become idle again, i.e., the completion time of the current processing on the machine minus the current time.

Job-related features: the states of jobs have a significant effect on deciding which job has a better priority to be processed earlier. A good schedule is expected to process important jobs earlier, and take the current and look-ahead job information into account.

- W is the weight of a job. A larger weight indicates a more important job.
- NOR is the number of remaining operations for a job. It reflects the current processing stage of the job.
- WKR is the median processing time needed for the remaining operations. The median time is an estimation of the processing time, since the exact processing time of the operation in DFJSS depends on the machine, and is unknown in advance as the machine is not decided yet. This feature estimates the processing stage of the job in terms of processing time.
- TIS is the time that the job has stayed in the job shop since its arrival.

Operation-related features: the characteristics and states of operations are important factors for choosing the next operation to be processed. A good schedule is supposed to consider the time cost of processing the operation and its waiting time properly.

- PT is the processing time of the operation on the candidate machine.
- NPT is the median processing time of the next operation of the candidate operation (0 if the candidate operation is the last one of the job)
- OWT is the time that the operation has waited in the machine's queue.

GPHH can automatically select proper simple features from the terminals and construct high-level features that are appropriate for a particular problem. The function set is set to $\{+, -, *, /, Max, Min\}$ [68], [69]. The arithmetic operators take two arguments. The "$/$" operator is a protected division, returning one if divided by zero. The Max and Min functions take two arguments and return the maximum and minimum of their arguments, respectively. The other parameter settings of GP are shown in Table III.

C. Comparison Design

The goal of this paper is to improve the effectiveness and efficiency of the crossover operator of GP with correlation coefficient based recombinative guidance mechanism to evolve effective scheduling heuristics for DFJSS. Three algorithms are taken into comparison in this paper. The cooperative coevolution genetic programming (CCGP) [4] is selected as the baseline algorithm with the uniform crossover operator. The goal of this paper is to improve the effectiveness of crossover by selecting subtrees to exchange building-blocks in GP individuals to generate offspring rather than the problem itself. In order to verify the effectiveness of $\mathrm{CCGP^c}$, it is suitable to compare with the same kind of technique, and the state-of-the-art algorithm (i.e., we name it as $\mathrm{CCGP^f}$) [33] that chooses crossover points by calculating the scores of subtrees based on the occurrences of features. The evolved rules of the proposed algorithm are also compared with the widely used manually designed rules by human experts, which can be found in the supplementary materials. In addition, to further verify the proposed subtree importance measure and recombinative guidance mechanism, we compare with a reverse algorithm named $\mathrm{CCGP^{!c}}$ that uses unimportant subtrees to replace important subtrees.

In this paper, we focus on the objective function and the utilisation level to construct multiple problems because the performance of evolved rules is influenced significantly by these two factors. In order to verify their effectiveness and efficiency, the proposed algorithm is tested on *six scenarios*. The scenarios consist of three objectives (e.g., max-flowtime, mean-flowtime, and mean-weighted-flowtime) and two utilisation levels (e.g., 0.85 and 0.95). For the sake of convenience, Fmax, Fmean, and WFmean are used to indicate max-flowtime, mean-flowtime, and mean-weighted-flowtime, respectively. All the evolved rules are tested on the same set of 50 different unseen test instances, and the average objective value across the 50 test instances is reported as the test performance of a rule, which is a good approximation of the true performance of the rule [66], [67].

V. RESULTS AND DISCUSSIONS

Wilcoxon rank-sum test with a significance level of 0.05 is used to verify the performance of the proposed algorithm. Fifty independent runs are conducted in this paper. Note that this paper works on minimisation problems. In the following results, "-", "+", and "≈" indicate the corresponding result is significantly better, worse or similar than (with) its counterpart.

A. The Performance of Evolved Rules

Table IV shows the mean (standard deviation) of the objective values of the four compared algorithms on unseen instances based on 50 independent runs in six scenarios. It can be seen that $\mathrm{CCGP^f}$ shows similar performance with CCGP in all scenarios. $\mathrm{CCGP^c}$ is significantly better than CCGP in half of the scenarios (e.g., <Fmean, 0.85>, <WFmean, 0.85> and <WFmean, 0.95>) and no worse in all other scenarios. Although $\mathrm{CCGP^c}$ is not significantly better than that of CCGP in scenario <Fmax, 0.85> and <Fmean, 0.95>, it

TABLE IV
THE MEAN (STANDARD DEVIATION) OF THE OBJECTIVE VALUES OF CCGP, CCGP^{f}, CCGP^{c} AND CCGP$^{!c}$ ON UNSEEN INSTANCES OVER 50 INDEPENDENT RUNS FOR SIX SCENARIOS.

Sce.	CCGP	CCGP^{f}	CCGP^{c}	CCGP$^{!c}$
1	1212.05(34.68)	1215.55(32.62)(≈)	1211.83(27.45)(≈)	1291.96(48.23)(+)
2	1941.98(29.93)	1939.84(32.97)(≈)	1942.09(29.16)(≈)	2026.88(80.15)(+)
3	385.95(3.22)	384.66(1.19)(≈)	384.68(1.92)(–)	389.79(3.96)(+)
4	551.18(5.78)	551.11(3.81)(≈)	550.30(3.72)(≈)	563.76(10.14)(+)
5	831.41(6.08)	829.89(4.76)(≈)	828.98(3.57)(–)	841.22(9.78)(+)
6	1111.01(12.02)	1109.52(11.27)(≈)	**1105.84(7.21)**(–)	1141.54(23.04)(+)

* 1: <Fmax, 0.85> 2: <Fmax, 0.95> 3: <Fmean, 0.85>
* 4: <Fmean, 0.95> 5: <WFmean, 0.85> 6: <WFmean, 0.95>

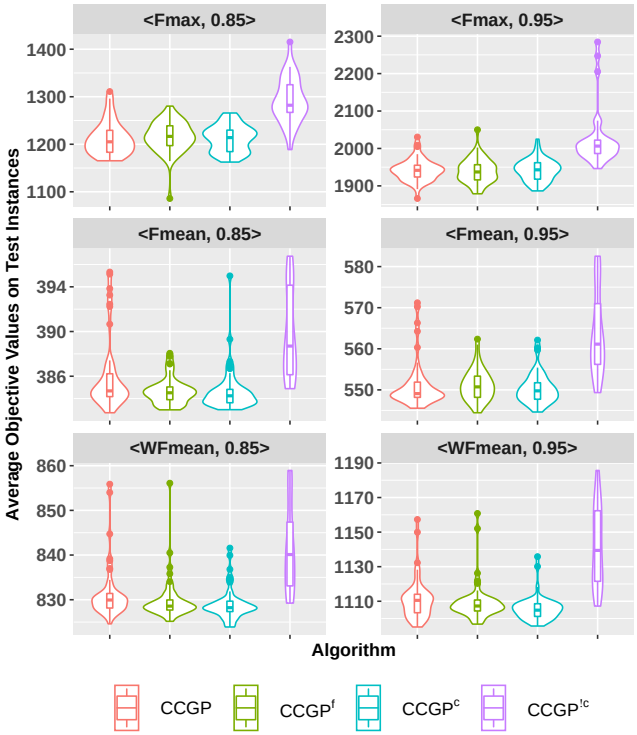


Fig. 7. The violin plot of the average objective values of CCGP, CCGP^{f}, CCGP^{c} and CCGP$^{!c}$ on unseen instances over 50 independent runs.

still shows its superiority in terms of the mean and standard deviation values obtained. In addition, CCGP^{c} is significantly better than CCGP^{f} in the most complex scenario (<WFmean, 0.95>), which is shown in bold. CCGP$^{!c}$ is significantly worse than all other algorithms, which is as expected. This verifies the effectiveness of proposed subtree importance measure and recombinative guidance from an opposite perspective.

Fig. 7 shows the violin plot of the test objective values of CCGP, CCGP^{f}, CCGP^{c} and CCGP$^{!c}$ over 50 independent runs in six different scenarios. It shows that although CCGP^{f} is not significantly better than CCGP in any scenario, it achieves better performance (i.e., smaller objective values) than CCGP in most scenarios (e.g., <Fmean, 0.85>, <Fmean, 0.95>, <WFmean, 0.85> and <WFmean, 0.95>). For CCGP^{c}, most obtained test objective values distribute at a lower position (i.e., the smaller, the better) than that of CCGP^{f} and CCGP in scenario <Fmean, 0.85>, <Fmean,

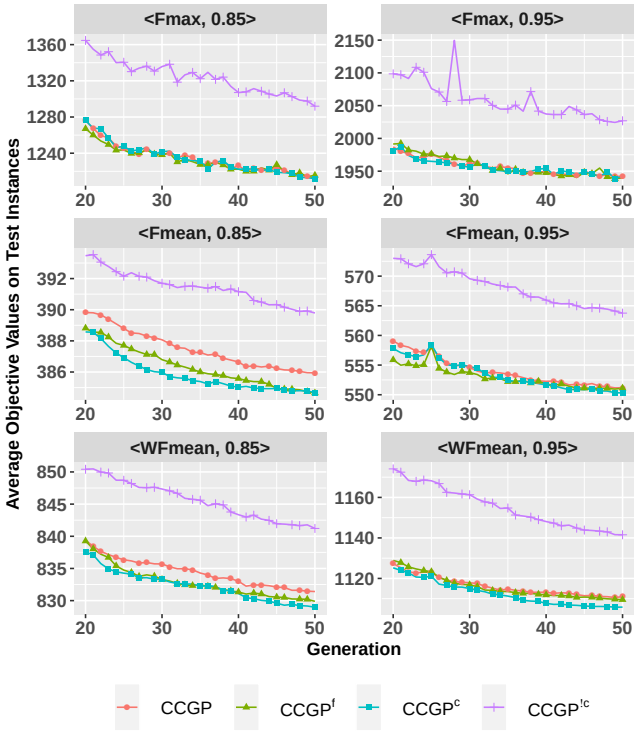


Fig. 8. The curves of the average objective values of CCGP, CCGP^{f}, CCGP^{c} and CCGP$^{!c}$ on unseen instances over 50 independent runs.

0.95>, <WFmean, 0.85> and <WFmean, 0.95>. In addition, the obtained objectives of CCGP$^{!c}$ are larger than other algorithms as expected, since the idea of CCGP$^{!c}$ is the opposite of that of the proposed algorithm CCGP^{c}. This verifies the effectiveness of the proposed subtree importance measure and the proposed recombinative guidance based on replacing unimportant subtrees with important ones.

Fig. 8 shows the convergence curves of the average objective values based on 50 independent runs on the unseen instances of CCGP, CCGP^{f}, CCGP^{c} and CCGP$^{!c}$. In most scenarios (e.g., <Fmean, 0.85>, <Fmean, 0.95>, <WFmean, 0.85> and <WFmean, 0.95>), CCGP^{c} achieves better performance than its counterparts. In half of the scenarios (e.g., <Fmean, 0.85>, <WFmean, 0.85> and <WFmean, 0.95>), CCGP^{c} convergence much faster than CCGP and CCGP^{f}. In addition, the individuals evolved by CCGP$^{!c}$ are much worse than other algorithms over generations, which also demonstrates the effectiveness of CCGP^{c}. For max-flowtime related scenarios (e.g., <Fmax, 0.85> and <Fmax, 0.95>), the performance of the involved three algorithms do not have obvious difference. This may be because max-flowtime is not easy to be optimised due to its sensitivity to the worst case.

B. The Depth Ratio of Selected Subtree

Both CCGP^{c} and CCGP^{f} tend to choose proper crossover points, however, CCGP^{c} shows its superiority. It is interesting to analyse the different behaviours of CCGP^{c} and CCGP^{f}. We define the *depth ratio* to measure the location of the selected subtrees of a tree. The depth ratio is the division of the number of depth where a selected subtree on and the depth

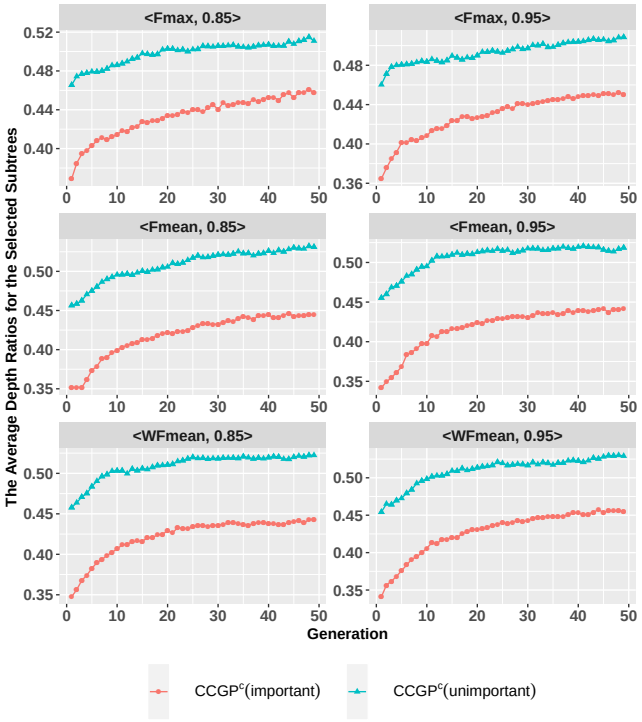


Fig. 9. The curves of the average depth ratios for the selected important and unimportant subtrees of $\mathrm{CCGP^c}$ over 50 independent runs in six scenarios.

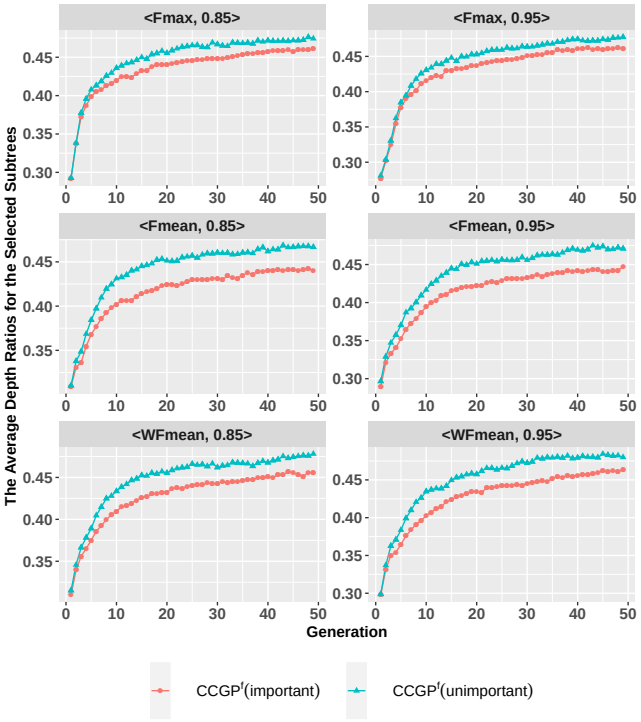


Fig. 10. The curves of the average depth ratios for the selected important and unimportant subtrees of $\mathrm{CCGP^f}$ over 50 independent runs in six scenarios.

of the tree. A smaller (larger) ratio lends to a closer location of the selected subtree to the root node (terminals) of a tree.

Fig. 9 and Fig. 10 show the average depth ratios for the selected important and unimportant subtrees of $\mathrm{CCGP^c}$ and $\mathrm{CCGP^f}$, respectively. For both $\mathrm{CCGP^c}$ and $\mathrm{CCGP^f}$, the depth ratios of important subtrees are smaller than that of unimportant subtrees. This is consistent with our intuition that the subtrees closer to the root are more likely to be important subtrees because they contain more comprehensive components. The gaps in depth ratios between important subtrees and unimportant subtrees of $\mathrm{CCGP^c}$ is much bigger than that of $\mathrm{CCGP^f}$. In addition, it can be seen that $\mathrm{CCGP^c}$ can detect important and unimportant subtrees better than that of $\mathrm{CCGP^f}$ in the early stage (i.e., before generation 10).

Fig. 11 shows the curves of average depth ratios of important subtrees obtained by the 50 independent runs of CCGP, $\mathrm{CCGP^f}$ and $\mathrm{CCGP^c}$ in different DFJSS scenarios. It shows that the depth ratios of the selected important trees of CCGP, $\mathrm{CCGP^f}$ and $\mathrm{CCGP^c}$ are similar to each other. The average depth ratios of important subtrees of CCGP, $\mathrm{CCGP^f}$, and $\mathrm{CCGP^c}$ are consistently between 0.4 and 0.45 after generation 10, which means we do not usually select the important subtrees towards the root. In general, the depth ratios of the selected important subtrees of $\mathrm{CCGP^c}$ are slightly smaller than its counterparts. However, the main difference is that $\mathrm{CCGP^c}$ prefers to choose the subtrees which are further away from root node with a larger depth ratio while CCGP and $\mathrm{CCGP^f}$ tend to select the subtrees that are closer to the root node with a smaller depth ratio in the early stage (i.e., from generation 1 to generation 5 roughly). It implies that the ability of $\mathrm{CCGP^f}$ to detect promising subtrees is limited at the early

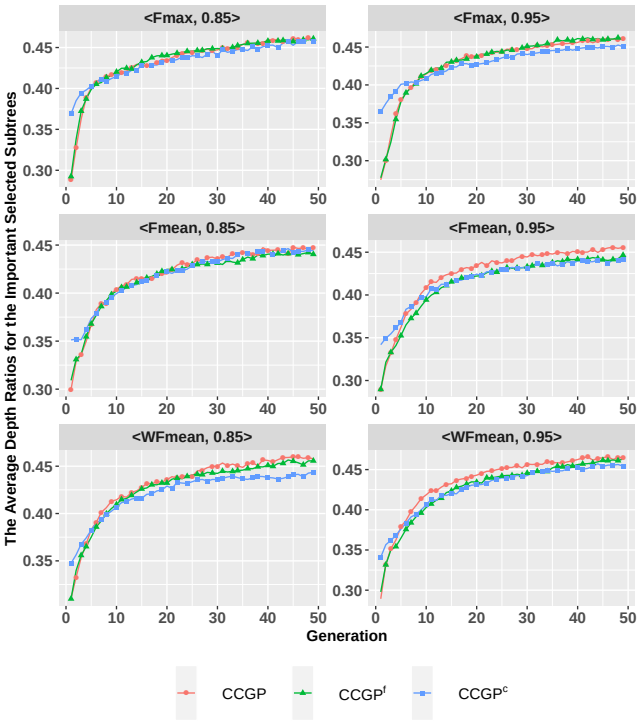


Fig. 11. The curves of the average depth ratios of *important subtrees* obtained by CCGP, $\mathrm{CCGP^f}$ and $\mathrm{CCGP^c}$ over 50 independent runs in six scenarios.

stage. One possible reason is that the occurrences of features are not accurate to measure the importance of features. This shortcoming is more obvious at the early stage because the individuals have not evolved well yet, and the occurrence

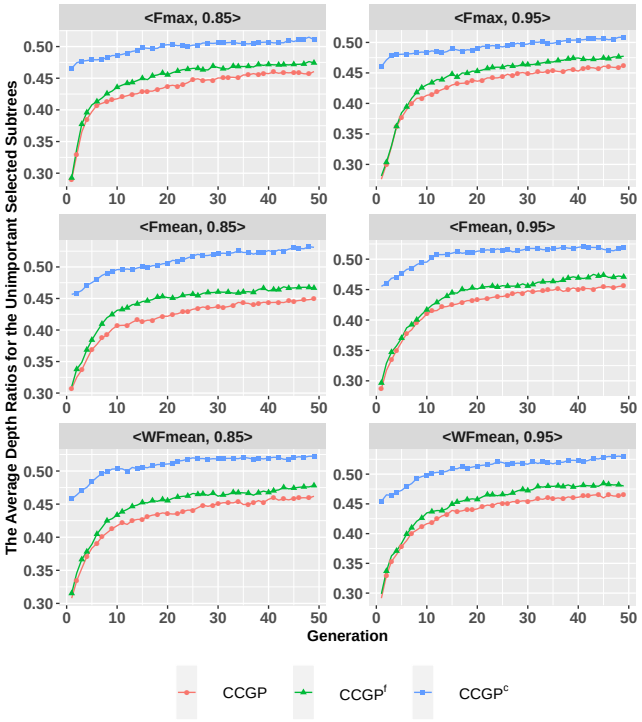


Fig. 12. The curves of the average depth ratios of *unimportant subtrees* of CCGP, CCGP^{f} and CCGP^{c} over 50 independent runs in six scenarios.

information of features is not reliable.

Fig. 12 shows the curves of average depth ratios of unimportant subtrees obtained by CCGP, CCGP^{f} and CCGP^{c} based on 50 independent runs in six scenarios. Fig. 12 shows that CCGP, CCGP^{f} and CCGP^{c} make clearly different decisions when selecting unimportant subtrees. On the one hand, both CCGP^{f} and CCGP^{c} tend to choose the subtrees with larger tree depths, i.e., on the lower parts of an individual. On the other hand, compared with CCGP^{f}, the depth ratios of the unimportant subtrees of CCGP^{c} are much larger. At the later stage (i.e., from generation 10 to generation 50 roughly), the depth ratios of CCGP^{f} fluctuate around 0.45 while the depth ratios of CCGP^{c} show a trend of fluctuation around 0.5. This means that CCGP^{c} treats the subtrees that are closer to the leaf nodes as unimportant subtrees.

C. The Correlations of Selected Subtrees

The correlations of subtrees determine the subtree selection probabilities. Fig. 13 shows the histogram plot for correlations of the selected subtrees of CCGP^{c} at early, middle and late stages over 50 independent runs in scenario <WFmean, 0.95>. The "Gen X Large (Small)" in the subtitle indicates that the subtree with a larger (smaller) score has a higher (lower) chance of being chosen. All the correlations are between -1 and 1. From the sub-figures in the first row (i.e., for selecting important subtrees), we find that the subtrees with correlations as zero are seldom selected and the absolute values of the correlations of the selected subtrees are close to 1. This is in line with our expectation because we tend to choose important subtrees, which have larger absolute correlation values.

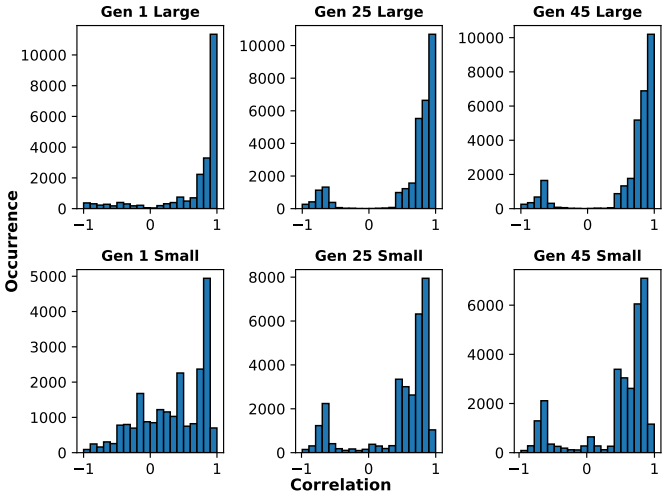


Fig. 13. The histogram plot for **correlations** of the selected subtrees of CCGP^{c} at generation 1, 25, and 45 in scenario <WFmean, 0.95> over 50 independent runs.

For selecting unimportant subtrees, as shown in the sub-figures of the second row, many selected subtrees have their correlations close to zero, especially at the early stage (generation 1). However, it is inconsistent with our intuition that there are still lots of correlations of the selected unimportant subtrees between 0.5 and 1 at generation 25 and generation 45. When we further look at the correlations during the process of selecting unimportant subtrees, we find that it can occur that all of the subtrees in an individual are important with correlations range between 0.5 and 1, especially in the middle and late stages. This is the reason why the correlations of the selected unimportant subtrees show in such a distribution. In other words, the proposed algorithm still chooses relatively unimportant subtrees.

D. The Probability Difference

The basic idea in this paper is to differentiate the probabilities of subtrees to be chosen instead of choosing subtrees randomly. We use *probability difference* to measure how the proposed algorithm influences the chance of subtrees to be selected. The probability difference is defined as the difference (i.e., subtraction) between the assigned probability by the proposed recombinative guidance mechanism and the uniform probability of the selected subtree. It is noted that the probability difference can be positive, negative, and zero. A positive probability difference indicates that the current subtree is selected with a higher chance compared with uniform probability. A negative probability difference means that the current subtree is selected with a lower chance compared with uniform probability. If the probability difference is zero, the assigned probability is the same as the uniform probability. This means that the proposed algorithm does not have effective guidance on choosing the crossover point for producing offspring.

We take CCGP^{c} in scenario <WFmean, 0.95> as an example to investigate how CCGP^{c} affects the choice of a subtree since CCGP^{c} performs significantly better than the other two algorithms in this scenario. Fig. 14 shows the histogram

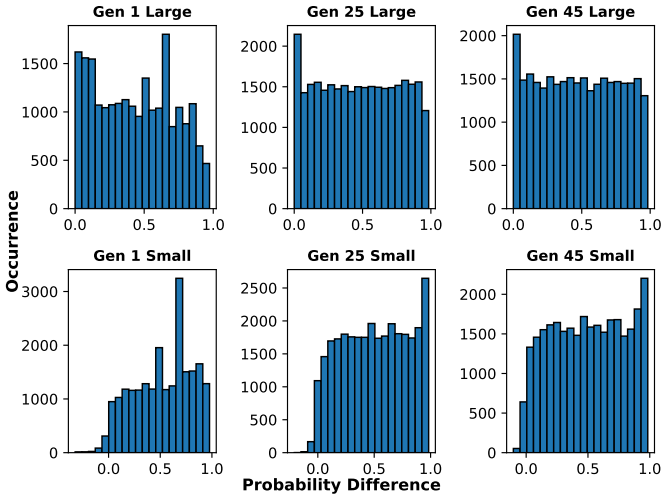


Fig. 14. The histogram plot of **probability difference** of the selected subtrees of CCGP^{c} at generation 1, 25, and 45 in scenario <WFmean, 0.95> over 50 independent runs.

TABLE V
THE MEAN (STANDARD DEVIATION) OF TRAINING TIME (IN MINUTES) OF CCGP, CCGP^{f}, AND CCGP^{c} OVER 50 INDEPENDENT RUNS FOR SIX DIFFERENT DFJSS SCENARIOS.

Scenario	CCGP	CCGP^{f}	CCGP^{c}
<Fmax, 0.85>	73(9)	74(13)(≈)	74(11)(≈)
<Fmax, 0.95>	87(15)	88(13)(≈)	89(12)(≈)
<Fmean, 0.85>	71(10)	72(10)(≈)	72(9)(≈)
<Fmean, 0.95>	80(13)	81(11)(≈)	81(12)(≈)
<WFmean, 0.85>	73(13)	75(16)(≈)	74(15)(≈)
<WFmean, 0.95>	82(13)	82(12)(≈)	83(13)(≈)

plot of the probability difference in the early (generation 1), middle (generation 25) and late (generation 45) stages of the evolutionary process in scenario <WFmean, 0.95> over 50 independent runs. Overall, most of the probability differences are positive numbers. This indicates that the proposed algorithm increases the probabilities of both the selected important and unimportant subtrees. This is in line with our expectation that CCGP^{c} can successfully guide GPHH to choose important or unimportant subtrees for crossover as required.

E. Training Time

Table V shows the mean and standard deviation of the training time (in minutes) of CCGP, CCGP^{f} and CCGP^{c} over 50 independent runs in six scenarios. It is obvious that there is no significant difference among CCGP, CCGP^{f} and CCGP^{c} in terms of training time. In other words, although more information calculations are involved in CCGP^{f} and CCGP^{c}, both CCGP^{f} and CCGP^{c} are efficient algorithms for solving the DFJSS problems.

For CCGP^{f}, it verifies the advantages of taking the information such as the occurrences of terminals during the evolutionary process of GP to improve the algorithm further. For CCGP^{c}, it verifies the advantages of taking some techniques such as correlation coefficient that can be quickly utilised along with the information during the evolutionary process of GP to enhance the performance of the algorithm.

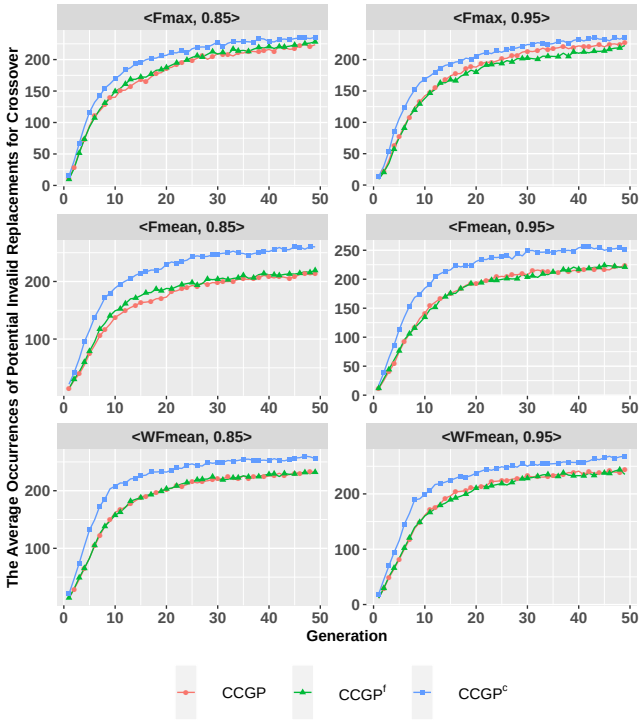


Fig. 15. The curve of average occurrences of *potential invalid replacements* of CCGP^{f} and CCGP^{c} over 50 independent runs in six scenarios.

VI. FURTHER ANALYSES

To deeply understand the effect of the proposed algorithm, the occurrences of the potential unsuccessful crossover whose offspring exceed the maximum depth limit, the sizes of evolved rules, and the evolved heuristics are further analysed.

A. The Occurrences of Potential Invalid Crossover

The sizes of offspring highly depend on the depth ratios of selected subtrees from parents. If a subtree with large depth ratio of a parent replaces with a subtree with a small depth ratio of the other parent, the produced offspring tends to have a large size. We are interested in how the proposed algorithm affects the size of offspring, since the offspring whose depths are larger than eight will be ignored during the crossover.

We record the number of "invalid" crossover which generates an offspring whose depth is larger than eight. We name the "invalid" crossover as *potential invalid replacements* since the produced offspring are ignored, and the crossover actually does not happen. The number of potential invalid replacements can be used to investigate how the proposed algorithm influences the process of generating offspring. Fig. 15 shows the average potential invalid replacements for the crossover of CCGP, CCGP^{f} and CCGP^{c} at each generation over 50 independent runs in six scenarios. In all scenarios, CCGP^{c} leads to more potential invalid replacements than that of CCGP^{f} along with the generations. It is consistent with the analyses in subsection V-B. In CCGP^{c}, the unimportant subtrees with larger depth ratios are more likely to be replaced by the important subtrees with smaller depth ratios, which leads to more potential invalid

TABLE VI
THE MEAN (STANDARD DEVIATION) OF THE SIZES OF EVOLVED THE BEST ROUTING AND SEQUENCING RULES OF CCGP, CCGP^{f}, AND CCGP^{c} OVER 50 INDEPENDENT RUNS FOR SIX DIFFERENT DFJSS SCENARIOS.

	Routing Rule			Sequencing Rule		
Scenario	CCGP	CCGP^{f}	CCGP^{c}	CCGP	CCGP^{f}	CCGP^{c}
<Fmax, 0.85>	61.48(18.30)	68.68(18.68)(≈)	62.32(19.07)(≈)	54.40(18.12)	51.36(15.01)(≈)	53.72(18.23)(≈)
<Fmax, 0.95>	59.28(19.20)	66.28(20.30)(≈)	60.68(18.87)(≈)	51.32(16.34)	53.92(16.77)(≈)	50.08(19.32)(≈)
<Fmean, 0.85>	59.84(15.05)	61.52(16.21)(≈)	59.40(18.09)(≈)	46.64(19.98)	47.32(18.43)(≈)	45.32(15.22)(≈)
<Fmean, 0.95>	64.16(19.42)	65.28(16.45)(≈)	59.60(16.52)(≈)	44.92(16.04)	44.80(15.47)(≈)	42.12(19.94)(≈)
<WFmean, 0.85>	59.00(17.35)	63.12(17.99)(≈)	64.52(17.20)(≈)	46.44(18.77)	50.92(13.93)(≈)	46.32(18.32)(≈)
<WFmean, 0.95>	63.88(15.95)	61.44(15.30)(≈)	65.20(18.11)(≈)	47.04(18.45)	52.92(20.33)(≈)	51.68(19.11)(≈)

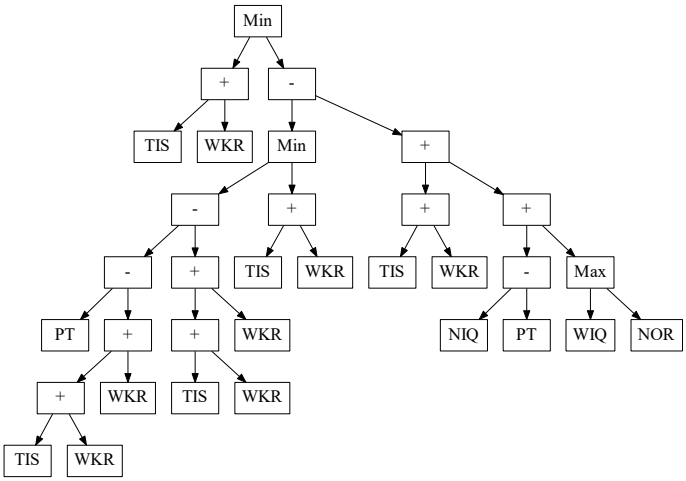


Fig. 16. One of the best evolved sequencing rules evolved by CCGP^{c} in scenario <Fmax, 0.95>.

replacements. Fortunately, it does not have a significant impact on the efficiency of the proposed algorithm.

B. The Sizes of Evolved Rules

The size (i.e., the number of nodes) can be a measure for the "interpretability" [70] of the evolved rules. A small rule can be more easily interpreted than a large rule. In this subsection, we investigate how the proposed algorithm influences the sizes of the evolved rules in terms of the sizes of the evolved best rules. Table VI shows the mean and standard deviation of the sizes of the evolved best routing rule and sequencing rules in different scenarios. Compared with CCGP, for both routing rules and sequencing rules, there is no statistical significant difference between the sizes of evolved rules obtained by CCGP^{f} and CCGP^{c}. We can conclude that the proposed algorithm CCGP^{c} with recombinative guidance achieves better performance without having impact on the sizes of the evolved rules.

C. Insight on the Evolved Scheduling Heuristics

To study the behaviours of the evolved rules obtained by CCGP^{c}, this subsection conducts structural analyses on the evolved sequencing rules. Specifically, the best sequencing rules obtained by CCGP^{c} for minimising max-flowtime and mean-weighted-flowtime with utilisation level of 0.95 are further investigated, respectively. It is noted that a small value calculated by the rule leads to a better priority for the candidate operation.

Evolved Rule for Max-flowtime. Fig. 16 shows one of the best evolved sequencing rules by CCGP^{c} in scenario <Fmax, 0.95>. It is observed that the rule is a combination of six simple terminals (TIS, WKR, PT, NIQ, WIQ, and NOR), and TIS and WKR are the most frequently used terminals for building this rule. In addition, "TIS + WKR" might be an effective constructed building block for this sequencing rule, since it appears five times in this rule.

This paper simplifies the rule by calculating different components. To make analysis easy, the rule in Fig. 16 is further simplified, as shown in Eq. (2).

$$\begin{aligned} S_1 =& Min\{TIS + WKR, \\ & Min\{PT - 2TIS - 4WKR, TIS + WKR\} - \\ & (TIS + WKR + NIQ - PT + Max\{WIQ, NOR\})\} \\ \approx& Min\{TIS + WKR, \\ & 2PT - 3TIS - 5WKR - NIQ - WIQ\} \\ \approx& 2PT - 3TIS - 5WKR - NIQ - WIQ \\ =& 2PT - 3TIS - 5WKR \end{aligned} \tag{2}$$

From step 1 to step 2, "Min{PT - 2TIS - 4WKR, TIS + WKR}" is simplified as "PT - 2TIS - 4WKR", since "PT - 2TIS - 4WKR" is almost always smaller than "TIS + WKR". In addition, "Max{WIQ, NOR}" is represented as WIQ, since WIQ (time) tends to be larger than NOR (between 1 and 10). Similarly, the rule in step 2 can be mostly replaced by the rule in step 3. Finally, NIQ and WIQ are the same for all operations in the same queue, and can be safely ignored, since they do not affect the final decision of choosing an operation. This rule suggests that when a machine is idle, the machine should process the operation with small processing time first. In addition, the jobs that arrive at the job shop earlier or have more remaining work should be processed earlier. Otherwise, if they are completed too late, the max-flowtime will be increased. It is consistent with our intuition for minimising max-flowtime due to its sensitivity to the worst case. The weights of the terminals may require domain knowledge and many rounds of trial-and-error if manually designed.

Evolved Rule for Mean-weighted-flowtime. Fig. 17 shows one of the best evolved sequencing rules obtained by CCGP^{c} in scenario <WFmean, 0.95>. This rule consists of four

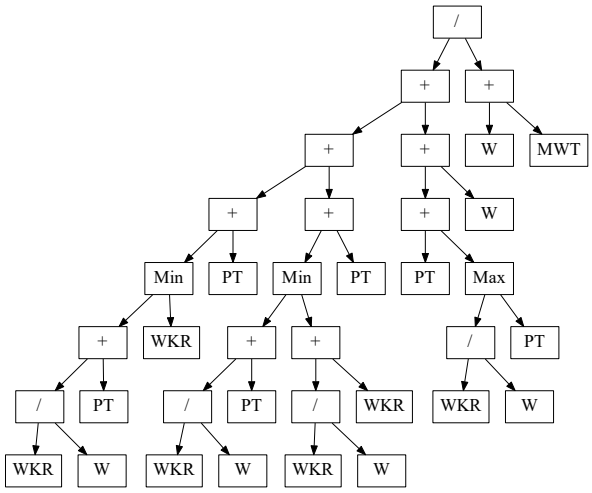


Fig. 17. One of the best evolved sequencing rules evolved by CCGP^c in scenario <WFmean, 0.95>.

simple terminals (WKR, W, PT, MWT), and four functions (+, /, Max, Min). "WKR / W" is an important learned component in this rule, and it appears four times. W tends to play its role as a denominator. PT is also an important terminal which mainly plays its role as a component for addition.

The simplification of the sequencing rule in Fig. 17 is shown in Eq. (3).

$$\begin{aligned} S_2 =&(Min\{WKR/W + PT, WKR\} + PT \\ &+ Min\{WKR/W + PT, WKR/W + WKR\} + PT \\ &+ PT + Max\{WKR/W, PT\} + W)/(W + MWT) \\ =&(Min\{WKR/W + PT, WKR\} + 3PT \\ &+ Min\{PT, WKR\} \\ &+ Max\{WKR/W, PT\} + W)/(W + MWT) \end{aligned} \tag{3}$$

This rule suggests to process the important operation with a large W earlier. In addition, the operations with short processing time and the jobs with small remaining work are preferred to be processed as soon as possible. Otherwise, the weighted-flowtime will increase.

In summary, this paper shows the advantage of evolving scheduling heuristics with the proposed algorithm. The evolved scheduling heuristics consist of simple heuristics but are combined in an effective way, which is not easy to be designed manually. In addition, the evolved scheduling heuristics have good interpretability, which is important for real-world applications.

VII. Conclusions and Future Work

The goal of this paper was to develop an effective recombinative guidance strategy for GP to evolve effective scheduling heuristics by improving the quality of produced offspring for DFJSS automatically. The goal was achieved by proposing an effective way to measure the importance of subtrees of an individual based on the characteristics of DFJSS with correlation coefficient technique and a properly designed recombinative guidance mechanism for crossover in GP.

The results showed that the evolved rules by the proposed algorithm with correlation coefficient based recombinative guidance have better performance in most scenarios while no worse in all other scenarios due to its effectiveness for producing offspring. This is also verified by the analyses in terms of the depth ratios of selected subtrees, the correlations of selected subtrees, and the probability difference during the evolutionary process. In terms of training time, the proposed algorithm does not need extra computational time compared with its counterparts. This verifies the advantages of utilising the information produced by GP during the evolutionary process and the efficient information calculation techniques such as correlation coefficient. In addition, the involved scheduling heuristics by the proposed algorithm have a better interpretability, which is easily accepted for solving practical problems.

Some interesting directions can be further investigated in the future. We will carry out more investigations about the effect on the evolved scheduling heuristics such as the rule size when applying crossover bias. We would also like to do a comprehensive study on the representation of GP for DFJSS, including multi-tree representation. This work focuses on flowtime-based objective functions and no due dates are considered for the jobs. We will consider the due date assignment in our future work. In addition, the proposed algorithm could be applicable to other problems such as the knapsack problem, if the importance of subtrees can be measured properly according to the characteristics of the examined problems. This will be further investigated in the future.

References

[1] A. S. Manne, "On the job-shop scheduling problem," *Operations Research*, vol. 8, no. 2, pp. 219–223, 1960.

[2] P. Brucker and R. Schlie, "Job-shop scheduling with multi-purpose machines," *Computing*, vol. 45, no. 4, pp. 369–375, 1990.

[3] 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*, 2020. Doi: 10.1109/TCYB.2020.3024849.

[4] D. Yska, Y. Mei, and M. Zhang, "Genetic programming hyper-heuristic with cooperative coevolution for dynamic flexible job shop scheduling," in *Proceedings of the European Conference on Genetic Programming*. Springer, 2018, pp. 306–321.

[5] F. Zhang, Y. Mei, and M. Zhang, "Genetic programming with multi-tree representation for dynamic flexible job shop scheduling," in *Proceedings of the Australasian Joint Conference on Artificial Intelligence*. Springer, 2018, pp. 472–484.

[6] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "A preliminary approach to evolutionary multitasking for dynamic flexible job shop scheduling via genetic programming," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2020, pp. 107–108.

[7] J. Xiong, L.-n. Xing, and Y.-w. Chen, "Robust scheduling for multi-objective flexible job-shop problems with random machine breakdowns," *International Journal of Production Economics*, vol. 141, no. 1, pp. 112–126, 2013.

[8] 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 the European Conference on Genetic Programming*. Springer, 2018, pp. 253–270.

[9] Y. N. Sotskov and N. V. Shakhlevich, "Np-hardness of shop-scheduling problems with three jobs," *Discrete Applied Mathematics*, vol. 59, no. 3, pp. 237–266, 1995.

[10] C. D. Geiger, R. Uzsoy, and H. Aytuğ, "Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach," *Journal of Scheduling*, vol. 9, no. 1, pp. 7–34, 2006.

[11] J. C. Tay and N. B. Ho, "Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems," *Computers & Industrial Engineering*, vol. 54, no. 3, pp. 453–473, 2008.

[12] S. Bennett, S. Nguyen, and M. Zhang, "A hybrid discrete particle swarm optimisation method for grid computation scheduling," in *Proceedings of the IEEE Congress on Evolutionary Computation*. IEEE, 2014, pp. 483–490.

[13] H. Chen, C. Chu, and J.-M. Proth, "An improvement of the lagrangean relaxation approach for job shop scheduling: a dynamic programming method," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 5, pp. 786–795, 1998.

[14] F. Y.-P. Simon *et al.*, "Integer linear programming neural networks for job-shop scheduling," in *Proceedings of the IEEE International Conference on Neural Networks*. IEEE, 1988, pp. 341–348.

[15] P. J. Van Laarhoven and E. H. Aarts, "Simulated annealing," in *Simulated annealing: Theory and applications*. Springer, 1987, pp. 7–15.

[16] F. Glover and M. Laguna, "Tabu search," in *Handbook of combinatorial optimization*. Springer, 1998, pp. 2093–2229.

[17] K. Chen, B. Xue, M. Zhang, and F. Zhou, "An evolutionary multitasking-based feature selection method for high-dimensional classification," *IEEE Transactions on Cybernetics*, 2020, Doi: 10.1109/TCYB.2020.3042243.

[18] G. V. Conroy, "Handbook of genetic algorithms," *Knowledge Eng. Review*, vol. 6, no. 4, pp. 363–365, 1991.

[19] M. Durasevic and D. Jakobovic, "A survey of dispatching rules for the dynamic unrelated machines environment," *Expert Systems with Applications*, vol. 113, pp. 555–569, 2018.

[20] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Genetic programming with adaptive search based on the frequency of features for dynamic flexible job shop scheduling," in *Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer, 2020, pp. 214–230.

[21] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Learning iterative dispatching rules for job shop scheduling with genetic programming," *The International Journal of Advanced Manufacturing Technology*, vol. 67, no. 1-4, pp. 85–100, 2013.

[22] M. Jayamohan and C. Rajendran, "New dispatching rules for shop scheduling: a step forward," *International Journal of Production Research*, vol. 38, no. 3, pp. 563–586, 2000.

[23] J. R. Koza, *Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems*. Stanford University, Department of Computer Science Stanford, CA, 1990, vol. 34.

[24] F. Zhang, Y. Mei, and M. Zhang, "Can stochastic dispatching rules evolved by genetic programming hyper-heuristics help in dynamic flexible job shop scheduling?" in *Proceedings of the IEEE Congress on Evolutionary Computation*. IEEE, 2019, pp. 41–48.

[25] S. Nguyen, M. Zhang, and K. C. Tan, "Surrogate-assisted genetic programming with simplified models for automated design of dispatching rules," *IEEE Transactions on Cybernetics*, vol. 47, no. 9, pp. 2951–2965, 2017.

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

[27] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Genetic programming for evolving due-date assignment models in job shop environments," *Evolutionary Computation*, vol. 22, no. 1, pp. 105–138, 2014.

[28] F. Zhang, Y. Mei, and M. Zhang, "A two-stage genetic programming hyper-heuristic approach with feature selection for dynamic flexible job shop scheduling," in *Proceedings of the Genetic and Evolutionary Computation Conference*. IEEE, 2019, pp. 347–355.

[29] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Collaborative multi-fidelity based surrogate models for genetic programming in dynamic flexible job shop scheduling," *IEEE Transactions on Cybernetics*, 2020. Doi: 10.1109/TCYB.2021.3050141.

[30] S. Nguyen, D. Thiruvady, M. Zhang, and K. C. Tan, "A genetic programming approach for evolving variable selectors in constraint programming," *IEEE Transactions on Evolutionary Computation*, 2021, Doi: 10.1109/TEVC.2021.3050465.

[31] R. Poli and N. F. McPhee, "General schema theory for genetic programming with subtree-swapping crossover: Part I," *Evolutionary Computation*, vol. 11, no. 1, pp. 53–66, 2003.

[32] ——, "General schema theory for genetic programming with subtree-swapping crossover: Part II," *Evolutionary Computation*, vol. 11, no. 2, pp. 169–206, 2003.

[33] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Guided subtree selection for genetic operators in genetic programming for dynamic flexible job shop scheduling," in *Proceedings of the European Conference on Genetic Programming*. Springer, 2020, pp. 252–267.

[34] J. Branke, S. Nguyen, C. W. Pickardt, and M. Zhang, "Automated design of production scheduling heuristics: A review," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 1, pp. 110–124, 2016.

[35] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A classification of hyper-heuristic approaches," in *Handbook of metaheuristics*. Springer, 2010, pp. 449–468.

[36] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. R. Woodward, "Exploring hyper-heuristic methodologies with genetic programming," in *Computational Intelligence*. Springer, 2009, pp. 177–201.

[37] E. K. Burke, M. R. Hyde, G. Kendall, and J. R. Woodward, "A genetic programming hyper-heuristic approach for evolving 2-d strip packing heuristics," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 6, pp. 942–958, 2010.

[38] M. R. Hyde, "A genetic programming hyper-heuristic approach to automated packing," Ph.D. dissertation, Computer Science, University of Nottingham, UK, 2010.

[39] M. B. Bader-El-Den, R. Poli, and S. Fatima, "Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework," *Memetic Computing*, vol. 1, no. 3, pp. 205–219, 2009.

[40] N. Pillay and W. Banzhaf, "A genetic programming approach to the generation of hyper-heuristics for the uncapacitated examination timetabling problem," in *Proceedings of the Portuguese Conference on Aritficial Intelligence*. Springer, 2007, pp. 223–234.

[41] M. A. Ardeh, Y. Mei, and M. Zhang, "Genetic programming hyper-heuristics with probabilistic prototype tree knowledge transfer for uncertain capacitated arc routing problems," in *Proceedings of the IEEE Congress on Evolutionary Computation*. IEEE, 2020, pp. 1–8.

[42] F. Zhang, Y. Mei, and M. Zhang, "A new representation in genetic programming for evolving dispatching rules for dynamic flexible job shop scheduling," in *Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer, 2019, pp. 33–49.

[43] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Automatic programming via iterated local search for dynamic job shop scheduling," *IEEE Transactions on Cybernetics*, vol. 45, no. 1, pp. 1–14, 2015.

[44] F. Zhang, Y. Mei, and M. Zhang, "Evolving dispatching rules for multi-objective dynamic flexible job shop scheduling via genetic programming hyper-heuristics," in *Proceedings of the IEEE Congress on Evolutionary Computation*. IEEE, 2019, pp. 1366–1373.

[45] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Hybrid evolutionary computation methods for quay crane scheduling problems," *Computers & Operations Research*, vol. 40, no. 8, pp. 2083–2093, 2013.

[46] M. Durasevic and D. Jakobovic, "Evolving dispatching rules for optimising many-objective criteria in the unrelated machines environment," *Genetic Programming and Evolvable Machines*, vol. 19, no. 1-2, pp. 9–51, 2018.

[47] F. Zhang, Y. Mei, and M. Zhang, "Surrogate-assisted genetic programming for dynamic flexible job shop scheduling," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2018, pp. 766–772.

[48] M. H. Kim, R. I. B. McKay, N. X. Hoai, and K. Kim, "Operator self-adaptation in genetic programming," in *Proceedings of the European Conference on Genetic Programming*. Springer, 2011, pp. 215–226.

[49] J. Niehaus and W. Banzhaf, "Adaption of operator probabilities in genetic programming," in *Proceedings of the European Conference on Genetic Programming*, 2001, pp. 325–336.

[50] H. Assimi, A. Jamali, and N. Nariman-Zadeh, "Multi-objective sizing and topology optimization of truss structures using genetic programming based on a new adaptive mutant operator," *Neural Computing and Applications*, vol. 31, no. 10, pp. 5729–5749, 2019.

[51] H. Xie, M. Zhang, and P. Andreae, "An analysis of depth of crossover points in tree-based genetic programming," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2007, pp. 4561–4568.

[52] U.-M. O'Reilly and F. Oppacher, "Program search with a hierarchical variable length representation: Genetic programming, simulated annealing and hill climbing," in *Proceedings of the International Conference on Parallel Problem Solving from Nature*. Springer, 1994, pp. 397–406.

[53] T. Ito, H. Iba, and S. Sato, "A self-tuning mechanism for depth-dependent crossover," *Advances in Genetic Programming*, vol. 3, p. 377, 1999.

[54] Q. Chen, M. Zhang, and B. Xue, "Geometric semantic genetic programming with perpendicular crossover and random segment mutation for symbolic regression," in *Proceedings of the Simulated Evolution and Learning*, 2017, pp. 422–434.

[55] Q. U. Nguyen, T. A. Pham, X. H. Nguyen, and J. McDermott, "Subtree semantic geometric crossover for genetic programming," *Genetic Programming and Evolvable Machines*, vol. 17, no. 1, pp. 25–53, 2016.

[56] Y. Mei, S. Nguyen, and M. Zhang, "Constrained dimensionally aware genetic programming for evolving interpretable dispatching rules in dynamic job shop scheduling," in *Proceedings of the Simulated Evolution and Learning*, 2017, pp. 435–447.

[57] N. F. McPhee, M. K. Dramdahl, and D. Donatucci, "Impact of crossover bias in genetic programming," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2015, pp. 1079–1086.

[58] H. Iba and H. de Garis, "Extending genetic programming with recombinative guidance," *Advances in Genetic Programming*, vol. 2, pp. 69–88, 1996.

[59] M. Zhang, X. Gao, and W. Lou, "A new crossover operator in genetic programming for object classification," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 37, no. 5, pp. 1332–1343, 2007.

[60] T. P. Pawlak and K. Krawiec, "Synthesis of constraints for mathematical programming with one-class genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 1, pp. 117–129, 2018.

[61] W. W. Daniel *et al.*, *Applied nonparametric statistics*. Houghton Mifflin, 1978.

[62] K. Neshatian and M. Zhang, "Unsupervised elimination of redundant features using genetic programming," in *Proceedings of the Australasian Joint Conference on Artificial Intelligence*. Springer, 2009, pp. 432–442.

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

[64] N. Q. Uy, N. X. Hoai, M. O'Neill, R. I. McKay, and E. G. López, "Semantically-based crossover in genetic programming: application to real-valued symbolic regression," *Genetic Programming and Evolvable Machines*, vol. 12, no. 2, pp. 91–119, 2011.

[65] J. P. Davis, K. M. Eisenhardt, and C. B. Bingham, "Developing theory through simulation methods," *Academy of Management Review*, vol. 32, no. 2, pp. 480–499, 2007.

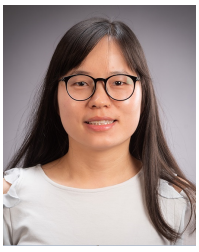
[66] T. Hildebrandt, J. Heger, and B. Scholz-Reiter, "Towards improved dispatching rules for complex shop floor scenarios: a genetic programming approach," in *Proceedings of the Annual Conference on Genetic and Evolutionary Computation*. ACM, 2010, pp. 257–264.

[67] S. Nguyen, Y. Mei, B. Xue, and M. Zhang, "A hybrid genetic programming algorithm for automated design of dispatching rules," *Evolutionary Computation*, vol. 27, no. 3, pp. 467–496, 2019.

[68] 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.

[69] Y. Mei, M. Zhang, and S. Nguyen, "Feature selection in evolving job shop dispatching rules with genetic programming," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2016, pp. 365–372.

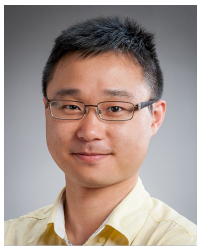
[70] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal, "Explaining explanations: An overview of interpretability of machine learning," in *Proceedings of the International Conference on Data Science and Advanced Analytics*, 2018, pp. 80–89.



Fangfang Zhang (Graduate Student Member, IEEE) received the B.Sc. and M.Sc. degrees from Shenzhen University, Shenzhen, China, in 2014 and 2017, respectively. She is currently pursuing the Ph.D. degree in computer science with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand.

She has over 20 journal and conference papers. Her current research interests include evolutionary computation, hyper-heuristic, job shop scheduling, and multitask optimization.

Ms. Zhang is a member of the IEEE Computational Intelligence Society and Association for Computing Machinery, and has been severing as reviewers for top international journals such as the IEEE Transactions on Evolutionary Computation and the IEEE Transactions on Cybernetics, and conferences including the Genetic and Evolutionary Computation Conference and the IEEE Congress on Evolutionary Computation. She is also a committee member of the IEEE NZ Central Section.



Yi Mei (M'09-SM'18) received the B.Sc. and Ph.D. degrees from the University of Science and Technology of China, Hefei, China, in 2005 and 2010, respectively.

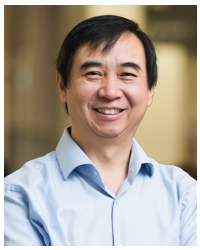
He is ~~currently~~ a Senior Lecturer with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. He has more than 100 fully referred publications, including the top journals in EC and Operations Research, such as IEEE Transactions on Evolutionary Computation, IEEE Transactions on Cybernetics, Evolutionary Computation, European Journal of Operational Research, and ACM Transactions on Mathematical Software. His research interests include evolutionary scheduling and combinatorial optimization, machine learning, genetic programming, and hyperheuristics.

Dr. Mei ~~serves as~~ a Vice-Chair of the IEEE CIS Emergent Technologies Technical Committee and a member of the Intelligent Systems Applications Technical Committee. He is an Editorial Board Member/Associate Editor of three international journals, and a Guest Editor of a special issue of the Genetic Programming and Evolvable Machines journal. He serves as a reviewer of over 30 international journals.

Su Nguyen (M'13) received his Ph.D. degree in Artificial Intelligence and Data Analytics from Victoria University of Wellington, New Zealand, in 2013.

He is a Senior Research Fellow and an Algorithm Lead with CDAC, La Trobe University, Melbourne, VIC, Australia. His expertise includes evolutionary computation (EC), simulation optimization, automated algorithm design, interfaces of AI/OR, and their applications in logistics, energy, and transportation. He has more than 70 publications in top EC/OR peer-reviewed journals and conferences. His current research focuses on novel people-centric artificial intelligence to solve dynamic and uncertain planning tasks by combining the creativity of evolutionary computation and power of advanced machine-learning algorithms.

Dr. Nguyen was the Chair of IEEE Task Force on Evolutionary Scheduling and Combinatorial Optimisation from 2014 to 2018 and is a member of the IEEE CIS Data Mining and Big Data Technical Committee. He delivered technical tutorials about EC and AI-based visualization at Parallel Problem Solving from Nature Conference in 2018 and IEEE World Congress on Computational Intelligence in 2020. He served as an Editorial Member of Complex and Intelligence Systems and the Guest Editor of the special issue on Automated Design and Adaption of Heuristics for Scheduling and Combinatorial Optimization in Genetic Programming and Evolvable Machines journal.



Mengjie Zhang Mengjie Zhang (M'04-SM'10-F'19) received the B.E. and M.E. degrees from Artificial Intelligence Research Centre, Agricultural University of Hebei, Baoding, China, and the Ph.D. degree in computer science from RMIT University, Melbourne, Australia, in 1989, 1992, and 2000, respectively.

He is currently a Professor of Computer Science, the Head of the Evolutionary Computation Research Group, and the Associate Dean (Research and Innovation) with the Faculty of Engineering, Victoria University of Wellington, New Zealand. His current research interests include AI and machine learning, particularly genetic programming, image analysis, feature selection and reduction, job shop scheduling, and transfer learning. He has published over 600 research papers in refereed international journals and conferences.

Prof. Zhang is a Fellow of the Royal Society of New Zealand, Fellow of IEEE and an IEEE Distinguished Lecturer. He was the Chair of the IEEE CIS Intelligent Systems and Applications Technical Committee, the IEEE CIS Emergent Technologies Technical Committee, and the Evolutionary Computation Technical Committee, and a member of the IEEE CIS Award Committee. He is a Vice-Chair of the Task Force on Evolutionary Computer Vision and Image Processing, and the Founding Chair of the IEEE Computational Intelligence Chapter in New Zealand.