Crossover Operators Between Multiple Scheduling Heuristics with Genetic Programming for Dynamic Flexible Job Shop Scheduling

Luyao Zhu[1], Fangfang Zhang[2], Mengyuan Feng[1], Ke Chen[1]✉, Xiaodong Zhu[1], Mengjie Zhang[2]

[1] *School of Electrical and Information Engineering*, *Zhengzhou University*, Zhengzhou, China
zhuluyao58@163.com, fmy1727980547@126.com, chenkezixf@zzu.edu.cn, Zhu_xd@zzu.edu.cn

[2] *Centre for Data Science and Artificial Intelligence & School of Engineering and Computer Science*, *Victoria University of Wellington*, Wellington, New Zealand
fangfang.zhang@ecs.vuw.ac.nz, mengjie.zhang@ecs.vuw.ac.nz

2024 IEEE Congress on Evolutionary Computation (CEC) | 979-8-3503-0836-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/CEC60901.2024.10612212

***Abstract*—Dynamic flexible job shop scheduling (DFJSS) is an important combinatorial optimisation problem that aims to optimise machine resources to improve production efficiency. Multi-tree genetic programming (MTGP) has been widely used to learn the routing rule and the sequencing rule for DFJSS simultaneously. Unlike traditional genetic programming that only operates crossover on a single tree, MTGP has various cases to conduct crossover since a genetic programming individual consists of more than one tree. Different crossover operators may affect the performance of MTGP for DFJSS. However, the investigation into different crossover operators in MTGP for DFJSS is rare. Specifically, it is not clear what influence will have on MTGP if involving both the routing rule and the sequencing rule for crossover. To this end, this paper provides a comprehensive investigation of four possible crossover cases between multiple scheduling heuristics with MTGP for DFJSS. The four operators are designed according to the number of trees/rules that crossover operator works on, and whether swapping full trees between parents. The results show that although the compared algorithms have comparable results in most scenarios, MTGP with both rules for crossover and the swapping strategy is ranked as the best one. Further analyses show that the sizes of learned rules are highly related to the crossover operators, and crossover involving more rules can increase the rule sizes, and vice versa. In addition, the population diversity and the number of unique features in the learned rules of MTGP with both rules for crossover are increased to learn effective rules.**

***Index Terms*—genetic programming, dynamic flexible job shop scheduling, crossover operators, scheduling heuristics**

I. Introduction

Dynamic flexible job shop scheduling (DFJSS) [1], [2] has received extensively attention due to its practical value, such as in supply chain management [3], transportation [4] and network communication [5]. The goal of DFJSS is to optimise resource allocation to improve production efficiency. In DFJSS, a number of jobs are awaiting handling by a set of machines in dynamic environments such as the continuous arrival of new jobs [6]. Each job consists of a sequence of operations, and each operation can be processed by more than one machine. Specifically, there are two decisions that need to be made simultaneously in DFJSS. One is *machine assignment* to assign a machine to handle a ready operation. The other is *operation sequencing* to select which operation to be processed next when a machine becomes idle.

DFJSS is an NP-hard problem [7], which cannot be handled efficiently by *exact optimisation approaches*, such as integer linear programming [8] and dynamic programming [9]. Although *approximate solution optimisation approaches*, such as genetic algorithms [10], can find the near-optimal solution to get some effective scheduling plans, they are not efficient to cope with dynamic situations for real-time scheduling. *Scheduling heuristics* [11], such as dispatching rules [12] denoted as priority functions, can rank machines or operations at decision points to generate real-time solutions for DFJSS efficiently. However, these scheduling rules need to be manually designed by domain experts, and the process is normally time-consuming and the designed scheduling heuristics can only be used in particular scenarios. Genetic programming [13] (GP) has been successfully applied to learn scheduling heuristics for DFJSS automatically. In DFJSS, a scheduling heuristic consists of a *routing rule* for machine assignment and a *sequencing rule* for operation sequencing. GP with a multi-tree (MTGP) representation is widely used to learn the routing rule and the sequencing rule simultaneously in DFJSS [14] with two trees. Specifically, one MTGP individual consists of two trees, one tree representing the routing rule and the other for the sequencing rule.

Crossover is an essential genetic operator in MTGP for generating offspring to the next generation [15]. Regarding to MTGP for DFJSS, since the routing rule and the sequencing rule are designed for different purposes, the crossover is usually conducted between routing rules or sequencing rules, separately. A crossover operator with a swapping strategy was proposed in [14] for DFJSS to generate more diverse offspring, which is randomly selecting one tree for crossover

This work was supported in part by the National Natural Science Foundation of China (62206255), the China Postdoctoral Science Foundation (2022M712878 and 2022TQ0298), and the Young Talents Lifting Project of Henan Association for Science and Technology (2024HYTP023).

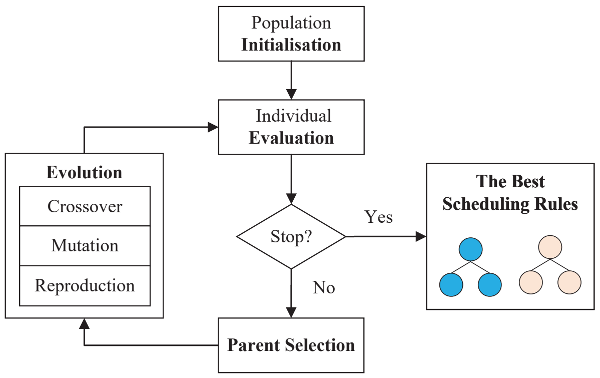

Fig. 1. The flowchart of MTGP to learn scheduling heuristics for DFJSS.

and swapping the other full tree. As in [14], most existing studies [16] only focus on the crossover in a single type of tree to generate scheduling heuristics for DFJSS. However, the multi-tree representation can increase the possible crossover cases since more than one tree can do crossover. The investigation into different crossover cases of MTGP is rare, and it is not clear whether conducting crossover on routing rules and sequencing rules simultaneously will affect the effectiveness and rule sizes of MTGP or not. In addition, it is still not clear why the learned scheduling heuristics with MTGP are smaller than using cooperative coevolution GP to learn the routing rule and the sequencing rule simultaneously with multiple populations [14]. To fill this gap, this paper aims to answer the following research questions:

- How will crossover with both routing and sequencing rules simultaneously affect the effectiveness of MTGP?
- How will the swapping strategy affect the effectiveness of MTGP?
- Why are the learned scheduling heuristics by MTGP smaller than obtained by using cooperative coevolution GP algorithm? This is a remaining question in [14].

II. Background

A. Dynamic Flexible Job Shop Scheduling

DFJSS aims to optimise the allocation of machine resources to process jobs in the job shop. In DFJSS, m machines $\mathcal{M} = \{M_1, M_2, ..., M_m\}$ need to process n jobs $\mathcal{J} = \{J_1, J_2, ..., J_n\}$. Each job has a number of operations $\mathcal{O}_j = \{O_{j1}, O_{j2}, ..., O_{jl_j}\}$ that are required to be handled in a fixed order. Each operation O_{ji} can be handled on several machines $M(O_{ji}) \subseteq \pi(O_{ji})$. The routing rule determines which specific machine can process an operation, and the sequencing rule determines which operation can be first handled on an idle machine. This paper focuses on a common dynamic event, i.e., new job arrival, and the information regarding a new job remains unknown until it is delivered to the shop floor. The principal constraints in DFJSS are presented below.

- A machine can process at most one operation at a time.
- Each operation is exclusive to one of its candidate machines for processing purposes.
- The processing of an operation cannot begin until all its preceding operations have been completed.
- Once started, the processing of an operation cannot be stopped or paused until it is completed.

This paper aims to minimize three commonly used objectives, respectively. Their calculations are demonstrated as follows.

- Mean-flowtime: $\frac{\sum_{j=1}^{n}(C_j - r_j)}{n}$
- Mean-tardiness: $\frac{\sum_{j=1}^{n} max\{0, C_j - d_j\}}{n}$
- Mean-weighted-tardiness: $\frac{\sum_{j=1}^{n} max\{0, C_j - d_j\} * w_j}{n}$

where n is the number of jobs, r_j denotes the release time of J_j, C_j is the completion time of a job J_j, d_j is the due date of J_j and w_j represents the weight of J_j.

B. Genetic Programming for DFJSS

Fig. 1 shows the flowchart of MTGP to learn scheduling heuristics for DFJSS problems. It starts with a randomly generated population, and all individuals in the population are then evaluated to get their fitness values. If the stopping condition is not met, parent selection method will be used to select individuals as parents to generate new offspring via the genetic operators (i.e., crossover, mutation, and reproduction) to form a new population. Otherwise, the MTGP algorithm will output the best individual (scheduling rules) in the current generation. Note that each MTGP individual consists of two trees, the first tree is the sequencing rule and the second tree is the routing rule.

III. Different Crossover Operators of Multi-tree Genetic Programming

In terms of the number of trees/rules for crossover, and whether involving swapping strategy, there are four possible cases for crossover, which will be described one by one. Note that crossover only performs in the same type of trees due to different trees/rules for different purposes.

A. Only One Tree for Crossover

In traditional MTGP, the crossover operator is applied to only one tree in a parent (individual) at a time, and other trees remain unchanged and directly move to the offspring [14], which is shown in Fig. 2. First, we randomly select one type tree (i.e., the first tree T_1) for crossover, and then a subtree is randomly chosen from each parent's T_1, which is highlighted in green and blue. Then, we exchange them to generate two new offspring. For DFJSS, this indicates either the routing rule or the sequencing rule will do crossover at a time of using the crossover operator.

B. Both Trees for Crossover

Instead of doing crossover for one type of tree only, the crossover operator is conducted in both types of trees, separately. As shown in Fig. 3, we choose a subtree on each type of tree from each parent for crossover, and then exchange subtrees with the same type to produce two offspring. Compared with only using one tree for crossover, using both

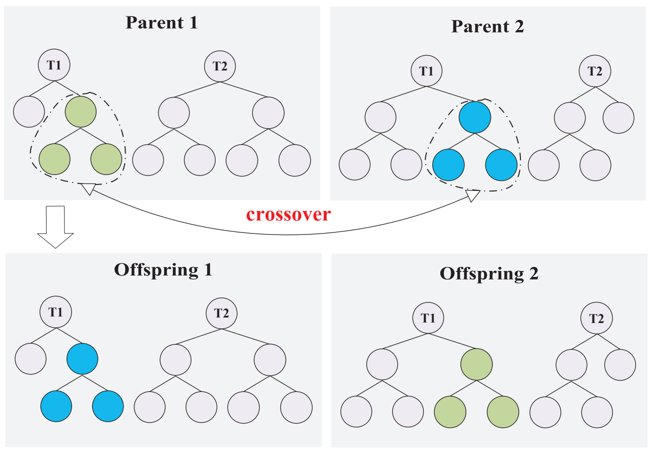

Fig. 2. An example of using a single tree for crossover in MTGP.

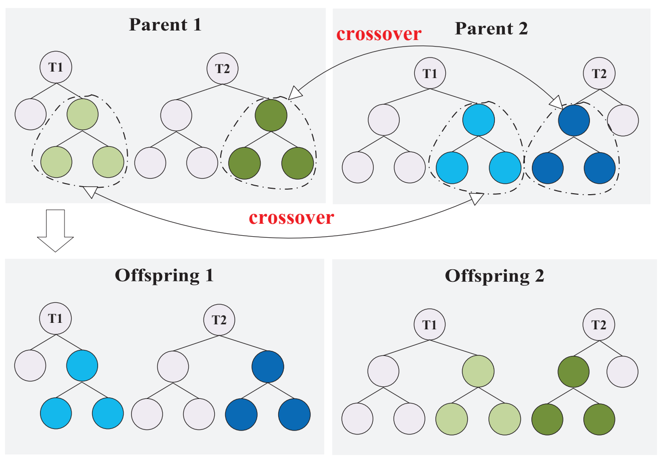

Fig. 3. An example of using both trees for crossover in MTGP.

trees for crossover can share more genetic information to get more diverse individuals. For DFJSS, this means that both the routing rule and the sequencing rule in a MTGP individual will do crossover at a time of using the crossover operator.

C. One Tree for Crossover and One Tree for Swap

A new crossover operator with swapping was proposed in [14] to improve the diversity of newly generated offspring. In addition to selecting a tree for crossover, one random full tree in a MTGP individual is swapped. Fig. 4 shows an example of one tree for crossover and one tree for swap in MTGP. The first step is the same as section III-A, after randomly selecting one type tree (i.e., T_1) for crossover, we exchange the chosen subtrees of T_1 in each parent. The next step is to swap the other full tree (i.e., T_2). Note that it does not matter whether the swapped tree is T_1 or T_2 since either of them generates the same two offspring. For DFJSS, this strategy indicates either the routing rule or the sequencing rule will do crossover, and the other of them will be swapped at a time of using the crossover operator.

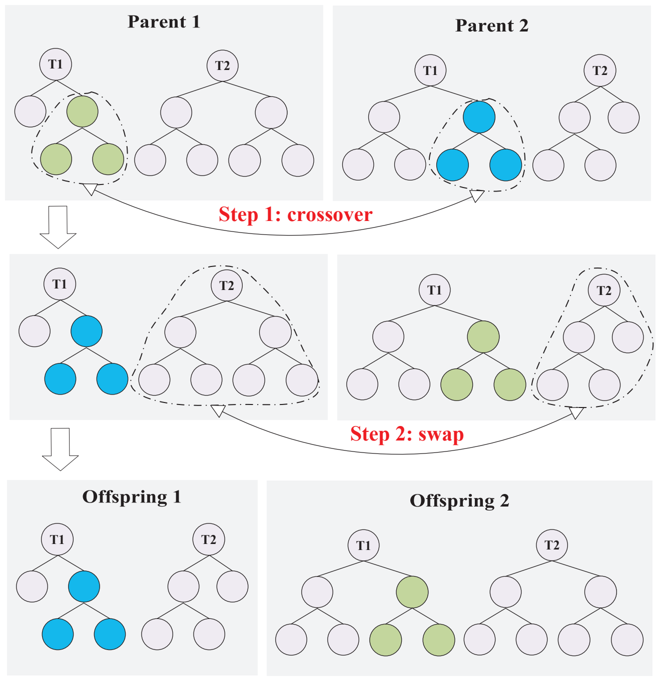

Fig. 4. An example of using a single tree for crossover and swapping one tree in MTGP.

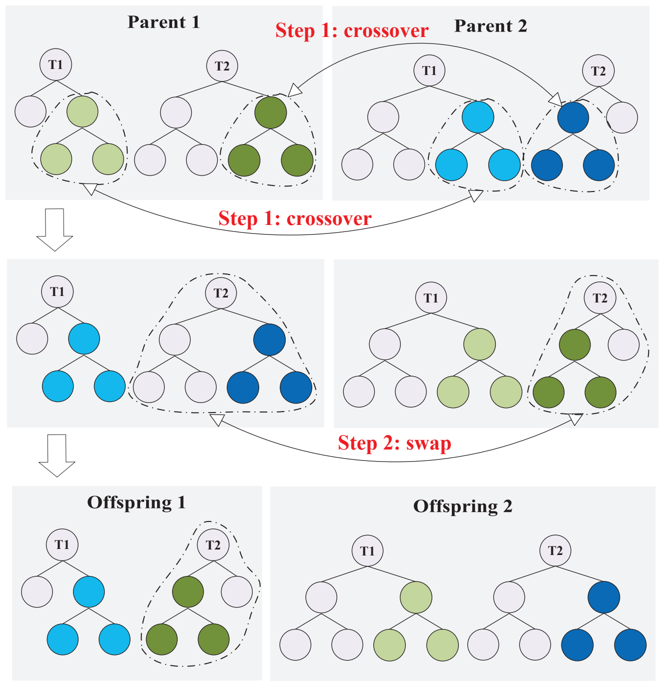

Fig. 5. An example of using both trees for crossover and swapping one tree in MTGP.

D. Both Trees for Crossover and One Tree for Swap

Motivated by the studies in sections III-B and III-C, we combine the idea of both trees for crossover and one tree for swap together to further increase the potential to produce diverse offspring. Fig. 5 shows an example of using both trees for crossover and one tree for swap in MTGP. After doing

TABLE I
THE TERMINAL SET.

Notation	Description
MWT	A machine's waiting time
WIQ	Current work in the queue
NIQ	The number of operations in the queue
NPT	Median processing time for the next operation
OWT	The waiting time of an operation
PT	Processing time of an operation on a specified machine
WKR	Median amount of work remaining for a job
NOR	The number of operations remaining for a job
TIS	Time in system
W	Weight of a job

crossover in both trees, we randomly swap one full tree (i.e., T_2) to get two offspring. For DFJSS, both the routing rule and the sequencing rule will do crossover, and either of them will be swapped at a time of using the crossover operator.

It is worth noting that the involved crossover and swap strategies in the four operators are only performed on the same type of tree, i.e., either the routing rule or the sequencing rule, due to the difference of routing and sequencing decisions.

IV. EXPERIMENT DESIGN

A. Simulation Model

We use simulation techniques to mimic DFJSS situations. Referring to the widely used DFJSS simulation [17], we assume that ten machines need to process 5000 jobs. Jobs have different weights, and a larger weight indicates a more important job. The weights of 20%, 60%, and 20% jobs are set as 1, 2, and 4. The due date of a job is set to be 1.5 times its processing time. In addition, the number of operations of each job differs, which is randomly distributed between 1 and 10 following a uniform distribution. The processing time for each operation is obtained from a uniform discrete distribution with values ranging from 1 to 99. The number of candidate machines for each operation follows a uniform distribution between 1 and 10. New jobs will arrive at the job shop over time following a Poisson process with the rate λ. *Utilisation level* (p) is a key factor in simulating job shop scenarios, which represents how busy the job shop is. The λ is controlled with the following equation. $\lambda = \mu * P_M/p$, λ is the expected utilisation rate of a machine in a Poisson process, μ denotes the average processing time of machines, and P_M refers to the probability of a job entering a machine. To obtain steady-state performance, we regard the initial 1000 jobs as warm-up jobs and collect data from the subsequent 5000 jobs. The simulation will end when the 6000th job is completed.

B. Design of Comparisons

MTGP with different crossover operators are tested in various DFJSS scenarios to verify their effectiveness. Two utilisation levels (i.e., 0.85 and 0.95) and three commonly used objectives, i.e., mean-flowtime (Fmean), mean-tardiness (Tmean) and mean-weighted-tardiness (WTmean), are used to form six examined scenarios. Each scenario is denoted as <objective, utilisation level> such as <Fmean, 0.85>.

TABLE II
THE PARAMETER SETTINGS IN GP.

Parameter	Value
⋆ Number of subpopulations	2
⋆ Subpopulation size	250
⋆ The number of elites for each subpopulation	5
● Population size	500
● The number of elites for population	10
The number of generations	100
Parent selection	Tournament selection with size 5
Initial minimum / maximum depth	2 / 6
Maximal depth of programs	8
Crossover / Mutation / Reproduction rate	80% / 15% / 5%
Terminal / non-terminal selection rate	10% / 90%
Method for initialising population	ramped-half-and-half

⋆: for CCGP only; ●: for MTGP

Five algorithms are involved in this paper, including MTGP with the four crossover operators, and the crossover operator with cooperative coevolution GP for comparison.

1) CCGP: Unlike MTGP, a *cooperative cooperative* framework [18] uses two subpopulations to evolve the routing rule and the sequencing rule separately. An individual in CCGP has only one tree, and the crossover can only occur within the same subpopulation.
2) MTGP-C: MTGP only randomly selects either the routing rule or the sequencing rule for *crossover*.
3) MTGP-C^2: MTGP selects *both* the routing rule and the sequencing rule for *crossover*.
4) MTGP-CS: MTGP randomly selects either the routing rule or the sequencing rule for *crossover*, and then either the routing rule or the sequencing rule is selected for *swap*.
5) MTGP-C^{2}S: MTGP selects *both* the routing rule and the sequencing rule for *crossover*, and then either the routing rule and the sequencing rule is selected for *swap*.

C. Parameter Settings

GP individuals consist of terminals and functions. Following the setting in [2], the details of terminals are shown in Table I. The function set is set to $\{+, -, *, \text{protected } /, max, min\}$. The protected "/" returns one if divided by zero. The other parameter settings of the GP algorithm are shown in Table II.

V. RESULTS AND ANALYSES

We use Friedman's test and Wilcoxon rank-sum test with a significance level of 0.05 to verify the performance of the algorithms in 30 independent runs. "Average Rank" is the average rank of the algorithm on all the examined scenarios. In the following tables, "↑", "↓", and "≈" indicate the corresponding result is significantly better than, worse than or similar to the compared algorithm. One algorithm will be compared with all algorithms on its left one by one.

TABLE III
THE MEAN (STANDARD DEVIATION) OF **OBJECTIVE VALUES** OF CCGP, MTGP-C, MTGP-C^2, MTGP-CS AND MTGP-C^{2}S IN SIX SCENARIOS ACCORDING TO 30 INDEPENDENT RUNS.

Scenarios	**CCGP**	**MTGP-C**	**MTGP-C^2**	**MTGP-CS**	**MTGP-C^{2}S**
<Fmean, 0.85>	386.22(3.86)	385.62(2.74)(≈)	**384.91(1.97)**(≈)(≈)	385.84(2.83)(≈)(≈)(≈)	385.52(3.08)(≈)(≈)(≈)(≈)
<Fmean, 0.95>	551.78(5.65)	554.54(8.05)(≈)	551.73(5.69)(≈)(≈)	551.90(6.42)(≈)(≈)(≈)	**551.25(4.65)**(≈)(≈)(≈)(≈)
<Tmean, 0.85>	40.21(2.13)	40.25(1.49)(≈)	39.98(2.08)(≈)(≈)	39.95(1.67)(≈)(≈)(≈)	**39.38(1.10)**(↑)(↑)(≈)(↑)
<Tmean, 0.95>	**175.51(3.03)**	178.57(6.62)(≈)	177.08(4.58)(≈)(≈)	177.54(4.52)(↓)(≈)(≈)	175.98(5.15)(≈)(≈)(≈)(↑)
<WTmean, 0.85>	76.35(5.13)	75.51(2.03)(≈)	76.33(5.27)(≈)(≈)	**75.26(2.07)**(≈)(≈)(≈)	76.02(4.03)(≈)(≈)(≈)(≈)
<WTmean, 0.95>	299.19(14.69)	296.91(10.74)(≈)	296.08(10.11)(≈)(≈)	**293.39(6.66)**(≈)(≈)(≈)	296.44(11.90)(≈)(≈)(≈)(≈)
Win / Draw / Lose	1 / 5 / 0	1 / 5 / 0	0 / 6 / 0	2 / 4 / 0	N/A
Average Rank	2.96	3.33	2.91	2.97	**2.83**

* The number of Win/Draw/Lose means the number of scenarios in which MTGP-C^{2}S is better/similar/worse than the corresponding algorithm.

TABLE IV
THE MEAN (STANDARD DEVIATION) OF **THE BEST PAIR RULE SIZE** OF CCGP, MTGP-C, MTGP-C^2, MTGP-CS AND MTGP-C^{2}S IN SIX SCENARIOS ACCORDING TO 30 INDEPENDENT RUNS.

Scenarios	**CCGP**	**MTGP-C**	**MTGP-C^2**	**MTGP-CS**	**MTGP-C^{2}S**
<Fmean, 0.85>	113.27(29.82)	**79.33(21.31)**(↑)	123.27(27.22)(≈)(↓)	82.13(21.62)(↑)(≈)(↑)	129.47(29.89)(≈)(↓)(≈)(↓)
<Fmean, 0.95>	120.40(22.86)	**91.33(21.16)**(↑)	118.87(30.02)(≈)(↓)	92.20(21.50)(↑)(≈)(↑)	138.07(28.65)(↓)(↓)(↓)(↓)
<Tmean, 0.85>	110.93(24.35)	**87.33(22.38)**(↑)	118.87(27.78)(≈)(↓)	89.47(26.59)(↑)(≈)(↑)	130.47(34.57)(↓)(↓)(≈)(↓)
<Tmean, 0.95>	112.33(23.42)	**89.53(24.87)**(↑)	123.33(27.58)(≈)(↓)	92.20(19.64)(↑)(≈)(↑)	129.47(32.66)(↓)(↓)(≈)(↓)
<WTmean, 0.85>	118.73(21.29)	94.53(18.93)(↑)	114.93(29.16)(≈)(↓)	**88.27(22.99)**(↑)(≈)(↑)	121.20(27.42)(≈)(↓)(≈)(↓)
<WTmean, 0.95>	111.20(21.89)	**92.27(23.25)**(↑)	121.67(30.09)(≈)(↓)	104.60(24.69)(≈)(≈)(↑)	132.06(25.96)(↓)(↓)(≈)(↓)
Win / Draw / Lose	0 / 2 / 4	0 / 0 / 6	0 / 5 / 1	0 / 0 / 6	N/A
Average Rank	3.47	**2.05**	3.28	2.48	3.72

A. *Quality of Learned Scheduling Heuristics*

Table III shows the mean and standard deviation of objective values of CCGP, MTGP-C, MTGP-C^2, MTGP-CS and MTGP-C^{2}S according to 30 independent runs in six scenarios. The results show that all examined algorithms perform similarly in four out of six scenarios (i.e., <Fmean, 0.85>, <Fmean, 0.95>, <WTmean, 0.85>, and <WTmean, 0.95>). Among them, MTGP-C^{2}S obtains the best performance with the smallest average rank, where both the routing rule and the sequencing rule do crossover and either of them is swapped at a time of using crossover operator. Specifically, MTGP-C^{2}S achieves significantly better performance than CCGP, MTGP-C and MTGP-CS in <Tmean, 0.85>, and MTGP-CS in <Tmean, 0.95>. This indicates that exchanging more information among the parents may be beneficial to generate promising MTGP individuals for DFJSS.

Fig. 6 shows the curves of average objective values of CCGP, MTGP-C, MTGP-C^2, MTGP-CS and MTGP-C^{2}S in six different scenarios according to 30 independent runs. We can see although the performance of all algorithms is similar, the algorithms with both trees for crossover (i.e., MTGP-C^2 and MTGP-C^{2}S) can converge faster in most scenarios, i.e., <Fmean, 0.85>, <Fmean, 0.95>, <Tmean, 0.85>, <WTmean, 0.85> and <WTmean, 0.95>), which shows exchanging more information between selected parents has more potential to improve the exploration ability.

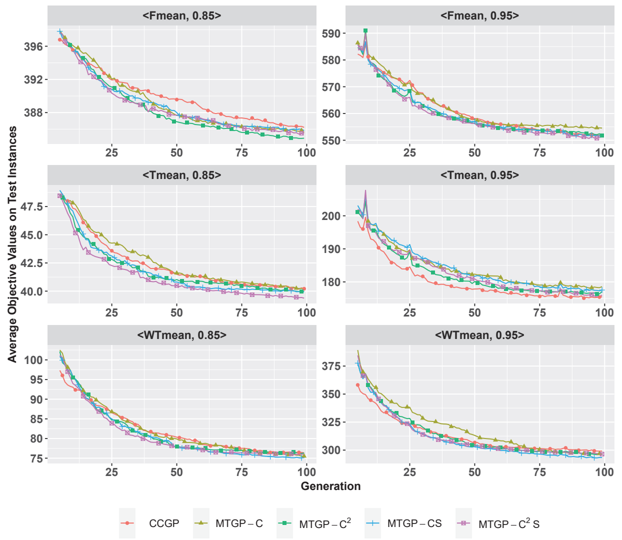

Fig. 6. Curves of average objective values on test instances of CCGP, MTGP-C, MTGP-C^2, MTGP-CS and MTGP-C^{2}S in six different scenarios according to 30 independent runs.

B. *Rule Size Analyses*

Rule size can be defined as the number of nodes in a GP individual. In DFJSS, the rule size is defined as the size of sequencing rule plus the size of routing rule since they work together to make the final decisions [14]. Smaller rules are preferred due to three reasons. First, the evaluation of smaller rules is less time-consuming which can reduce the computational cost to learn scheduling heuristics. Second, small rules tend to have better generalisation ability. Finally, small rules are more interpretable for decision-makers.

Table IV presents the mean (standard deviation) of the size

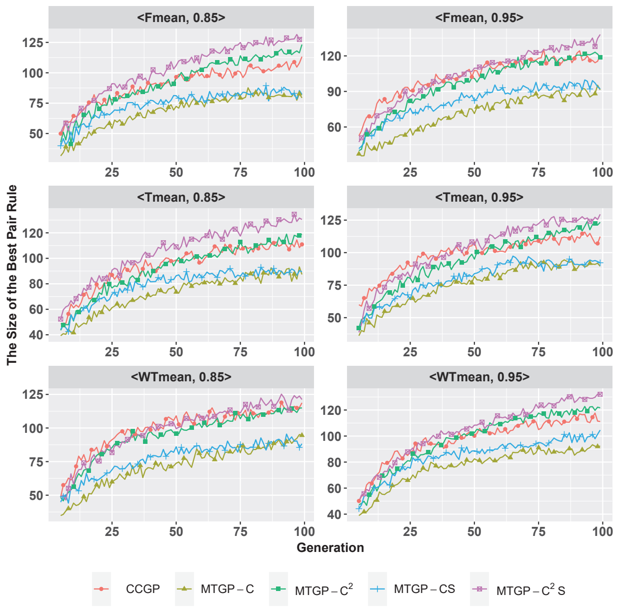

Fig. 7. Curves of pair rule size of the best individual of CCGP, MTGP-C, MTGP-C^2, MTGP-CS and MTGP-C^{2}S in six different scenarios according to 30 independent runs.

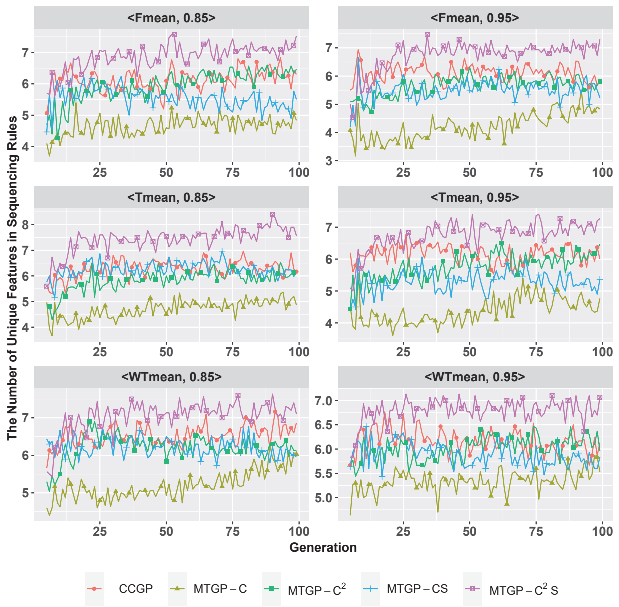

Fig. 8. Curves of mean number of unique features in the best sequencing rule evolved by CCGP, MTGP-C, MTGP-C^2, MTGP-CS and MTGP-C^{2}S in six different scenarios according to 30 independent runs.

of the best pair of routing rule and sequencing rule of all algorithms according to 30 independent runs in six scenarios. The rule size of MTGP-C is the smallest in five out of six scenarios and it ranks first based on the average rank. Comparing MTGP-C and MTGP-C^2, we can see that MTGP-C gets the smaller rule than MTGP-C^2 in all examined scenarios. This suggests that adopting crossover in more trees can increase the rule size by exchanging more genetic materials. The same findings can be obtained by comparing MTGP-CS and MTGP-C^{2}S. In addition, comparing MTGP-C and MTGP-CS, the results show the rule size of MTGP-CS is slightly bigger than that of MTGP-C in five out of six scenarios, but there is no significantly difference in all examined scenarios. This indicates swapping one full tree between parents can potentially increase the rule size, however, the affect on rule size is marginal. The comparison of MTGP-C^2 and MTGP-C^{2}S can also find this phenomenon.

The results can also explain a remaining question in the papers introducing different GP representations [14], why the rule size in CCGP is bigger than MTGP? This is because the crossover operators of MTGP in [14] select only one tree for crossover, which corresponds to MTGP-C and MTGP-CS in this paper. However, both rules in two subpopulations of CCGP do crossover. If both trees do crossover in MTGP (i.e., MTGP-C^2), we can find the rule size between CCGP and MTGP-C^2 becomes similar in all examined scenarios.

Fig. 7 shows the curves of the sizes of pair rule of the best individual of all algorithms in six different scenarios along with generations. The results show that the rule sizes changes along with generations are stable, where the relative rule size comparison among algorithms at different generations is the same. We can see that MTGP-C and MTGP-CS achieve smaller rule sizes among with the generations than other algorithms, and the rule size of MTGP-CS is slightly larger than MTGP-C. This is consistent with the observations in [14]. In addition, MTGP-C^{2}S starts to have larger rule size than others algorithms from an early stage. This further vitrifies that the rule size changes is highly relative to the crossover operators. In general, crossover with more rules or swapping rules between parents, is more likely to increase the rule size.

C. The Number of Unique Features in the Best Rules

The number of unique features denotes how many different terminals are used in evolving the scheduling rules. If the number of unique features in a rule is small, this rule can be simplified easier to become smaller, which is more interpretable to end-users [19].

1) The Number of Unique Features in the Best ***Sequencing*** *Rule:* Fig. 8 shows the curves of average number of unique features in the best sequencing rule evolved by all algorithms. We can find some similar patterns in the curve of rule size in Fig. 7. The number of unique terminals in the sequencing rule of MTGP-C is almost the smallest. In addition, the curve of MTGP-C^{2}S is always the largest across generations. This indicates that the number of unique terminals in the best sequencing rule is related to how much the genetic information is exchanged between parents. Crossover on both routing and sequencing rules or swapping rules, can increase the number of unique features in the best sequencing rules. This is consistent with our intuitive that exchanging more genetic materials can increase the possibility to have more features from other individuals.

2) The Number of Unique Features in the Best ***Routing*** *Rule:* Fig. 9 shows the curves of average number of unique

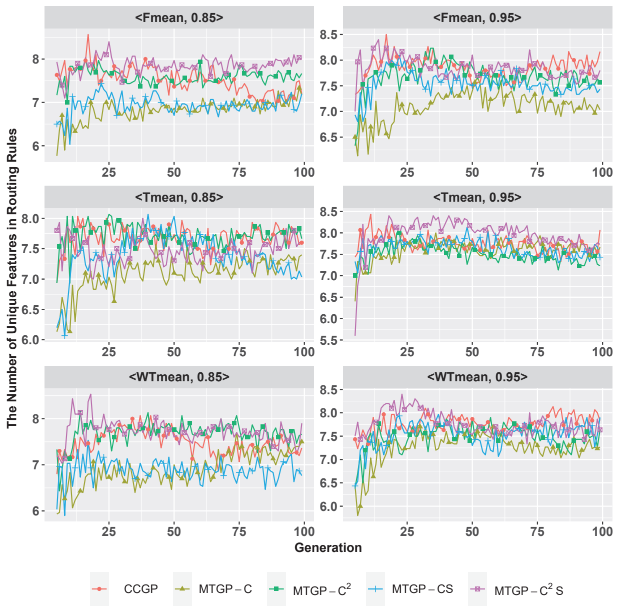

Fig. 9. Curves of average number of unique features in the best routing rule evolved by CCGP, MTGP-C, MTGP-C^2, MTGP-CS and MTGP-C^{2}S in six different scenarios according to 30 independent runs.

features in the best sequencing rule evolved by all algorithms. We find the same pattern as for sequencing rules, the number of MTGP-C and MTGP-C^{2}S is still smaller and larger than other algorithms in most scenarios. However, the patterns of the unique number of features in the best routing rule is not clear as that in the sequencing rules, since the number of unique features among different algorithms is similar, especially in the scenarios with utilisation level of 0.95. From literature, the routing rule is more important than the sequencing rule and the size of the routing rule is bigger than the sequencing rules [20]. It is possible that the routing rules require more features to form. Thus, the influence of different crossover operators is less obvious than the curve in Fig. 8.

D. Population Diversity

Diversity is an important indicator of population, which plays a role in avoiding premature convergence. We measure diversity with entropy. Its calculation is as follows. $entropy = -\sum_{c\epsilon C}(\frac{|c|}{|inds|})log(\frac{|c|}{|inds|})$, where C denotes the number of clusters generated by the DBScan clustering algorithm [21] with a cluster radius of 0 and the phenotypic distance measure. The bigger the entropy, the better the diversity.

Fig. 10 presents the curve of population diversity of all MTGP algorithms in six different scenarios according to 30 independent runs. Although the diversities of all algorithms are similar at the end of evolution, at the earlier stage of evolution, i.e., before generation 25, MTGP-C^{2}S has the largest diversity, followed by MTGP-C^2. The diversity of MTGP-CS is ranked as three, followed by MTGP-C. Compared with MTGP-C and MTGP-C^2, and MTGP-CS and MTGP-C^{2}S, we can conclude that crossover on both routing rules and sequencing rules can increase the diversity of population for better exploration.

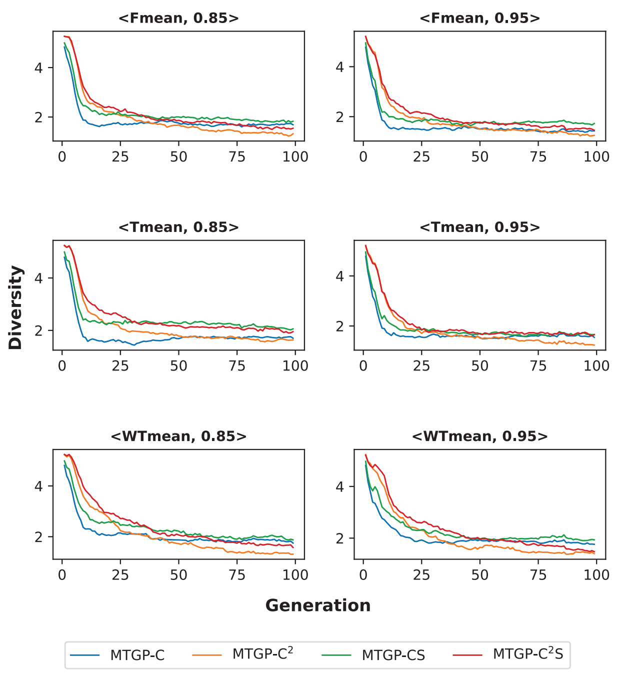

Fig. 10. Curves of population diversity of MTGP-C, MTGP-C^2, MTGP-CS and MTGP-C^{2}S in six different scenarios according to 30 independent runs.

Compared with MTGP-C and MTGP-CS, and MTGP-C^2 and MTGP-C^{2}S, we can see that swapping rules can increase the population diversity of GP as well. This might be a reason why MTGP-C^{2}S achieves the best performance among all algorithms as discussed in V-A.

VI. Discussions

According to the results in the last section, we can summarise the response to the research questions of this paper. In addition, some advice are given for guidance on using different crossover operators.

A. How will crossover with both routing rules and sequencing rules simultaneously affect the effectiveness of MTGP?

Crossover on both rules can increase the effectiveness of MTGP for DFJSS, i.e., comparing with MTGP-C with a rank of 3.33 and MTGP-C^2 with a rank of 2.91 ($2.91<3.33$), and MTGP-CS with a rank of 2.97 and MTGP-C^{2}S with a rank of 2.83 ($2.83<2.97$). The population diversity is increased for better exploration ability leading to a fast convergence. This still suggests considering both types of rules for crossover with MTGP that gives different trees better chance to evolve, especially there are many trees such as five trees involved for different functions. In addition, this operator can increase the sizes of learned rules, and the number of unique features of learned best routing rule and the sequencing rule.

B. How will the swapping strategy affect the effectiveness of MTGP?

Swapping strategy can improve the quality of learned scheduling heuristics, i.e., comparing with MTGP-C with a rank of 3.33 and MTGP-CS with a rank of 2.97 (2.97<3.33), and MTGP-C^2 with a rank of 2.91 and MTGP-C^{2}S with a rank of 2.83 (2.83<2.91). The swapping strategy can increase the population diversity and the number of unique features in the learned routing rule and the sequencing rule as well. However, comparing with crossover with both the routing rule and the sequencing rule, swapping strategy has a smaller effect on all these indicators.

C. Why the learned scheduling heuristics by MTGP are smaller than obtained by using cooperative coevolution GP algorithm? This is a remaining question in [14].

The main reason is that traditional crossover operators of MTGP select only one tree for crossover [14], while both rules in two subpopulations of CCGP do crossover. As discussed before, crossover on multiple trees can increase the rule size. MTGP with crossover of both the routing rule and the sequencing rule for crossover, has similar rule sizes to CCGP.

D. Advice for Using Different Crossover Operators

Exchanging more genetic information between parents of MTGP can potentially improve the effectiveness of learned scheduling heuristics and increase the population diversity. However, it can also increase the sizes of learned scheduling heuristics, which may weaken the interpretability of rules. If focusing mainly on performance, MTGP-C^{2}S which uses both routing rules and sequencing rules for crossover and swapping rules, might be the most suitable one. If the interpretability is important to consider, MTGP-C that applies a single tree crossover is a good candidate. Finally, MTGP-CS that uses a single tree crossover and swaps rules is a trade-off between effectiveness and rule size.

VII. Conclusions and Future Work

The goal of this paper is to investigate different crossover operators between multiple scheduling heuristics with GP for DFJSS. The goal has been successfully achieved by comparing four MTGP algorithms with different crossover operators, and CCGP algorithm that learns the routing rule and the sequencing rule separately.

The results show that although there is no big difference among all crossover operators, MTGP-C^{2}S including both rules for crossover and swapping strategy achieves the best performance with a fast converge speed. Crossover on more trees and swapping trees between parents can also increase the population diversity, the number of unique features, and the sizes of learned scheduling heuristics. In addition, the effect of crossover with more scheduling heuristics is larger than swapping whole trees directly on all these indicators. The investigation and findings in this paper can provide guidance to design crossover operators with MTGP, especially for the problems with multiple trees/rules for different functions.

Some interesting directions can be further investigated in the near future. This work shows sharing more genetic information between parents can increase population diversity of MTGP to enhance exploration ability, thus, we can design an adaptive crossover operators to balance the exploration and exploitation.

References

[1] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, “Survey on genetic programming and machine learning techniques for heuristic design in job shop scheduling,” *IEEE Transactions on Evolutionary Computation*, vol. 28, no. 1, pp. 147–167, 2024.

[2] L. Zhu, F. Zhang, X. Zhu, K. Chen, and M. Zhang, “Sample-aware surrogate-assisted genetic programming for scheduling heuristics learning in dynamic flexible job shop scheduling,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2023, pp. 384–392.

[3] J. B. Houlihan, “International supply chain management,” *International Journal of Physical Distribution & Materials Management*, vol. 15, no. 1, pp. 22–38, 1985.

[4] L. R. Ford Jr and D. R. Fulkerson, “Solving the transportation problem,” *Management Science*, vol. 3, no. 1, pp. 24–32, 1956.

[5] H. Fattah and C. Leung, “An overview of scheduling algorithms in wireless multimedia networks,” *IEEE Wireless Communications*, vol. 9, no. 5, pp. 76–83, 2002.

[6] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, “Multitask multiobjective genetic programming for automated scheduling heuristic learning in dynamic flexible job-shop scheduling,” *IEEE Transactions on Cybernetics*, vol. 53, no. 7, pp. 4473–4486, 2023.

[7] 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.

[8] 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.

[9] 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.

[10] F. Pezzella, G. Morganti, and G. Ciaschetti, “A genetic algorithm for the flexible job-shop scheduling problem,” *Computers & Operations Research*, vol. 35, no. 10, pp. 3202–3212, 2008.

[11] A. Muhlemann, A. Lockett, and C. K. Farn, “Job shop scheduling heuristics and frequency of scheduling,” *The International Journal of Production Research*, vol. 20, no. 2, pp. 227–241, 1982.

[12] 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.

[13] 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.

[14] 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.

[15] E. Semenkin and M. Semenkina, “Self-configuring genetic programming algorithm with modified uniform crossover,” in *Proceedings of the IEEE Congress on Evolutionary Computation*. IEEE, 2012, pp. 1–6.

[16] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, “Correlation coefficient-based recombinative guidance for genetic programming hyperheuristics in dynamic flexible job shop scheduling,” *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 3, pp. 552–566, 2021.

[17] F. Zhang, S. Nguyen, Y. Mei, and M. Zhang, “Genetic programming for production scheduling: An evolutionary learning approach,” in *Machine Learning: Foundations, Methodologies, and Applications*. Springer, 2021, DOI: 10.1007/978-981-16-4859-5, pp. XXXIII+338 pages.

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

[19] 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.

[20] ——, “Importance-aware genetic programming for automated scheduling heuristics learning in dynamic flexible job shop scheduling,” in *International Conference on Parallel Problem Solving from Nature*. Springer, 2022, pp. 48–62.

[21] M. Ester, H. P. Kriegel, J. Sander, X. Xu *et al.*, “A density-based algorithm for discovering clusters in large spatial databases with noise,” in *kdd*, vol. 96, no. 34, 1996, pp. 226–231.