Learning Strategies on Scheduling Heuristics of Genetic Programming in Dynamic Flexible Job Shop Scheduling

Fangfang Zhang[1]✉ , Yi Mei[1] , Su Nguyen[2] , Mengjie Zhang[1]
[1] School of Engineering and Computer Science, Victoria University of Wellington, New Zealand
{fangfang.zhang, yi.mei, mengjie.zhang}@ecs.vuw.ac.nz
[2] Centre for Data Analytics and Cognition, La Trobe University, Victoria 3086, Melbourne, Australia
P.Nguyen4@latrobe.edu.au

2022 IEEE Congress on Evolutionary Computation (CEC) | 978-1-6654-6708-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/CEC55065.2022.9870243

***Abstract*—Dynamic flexible job shop scheduling is an important combinatorial optimisation problem that covers valuable practical applications such as order picking in warehouses and service allocation in cloud computing. Machine assignment and operation sequencing are two key decisions to be considered simultaneously in dynamic flexible job shop scheduling. Genetic programming has been successfully and widely used to learn scheduling heuristics, including a routing rule for machine assignment and a sequencing rule for operation sequencing simultaneously. There are mainly two types of learning strategies to evolve scheduling heuristics, i.e., learning one rule by fixing the other rule, and learning the routing rule and the sequencing rule simultaneously. However, there is no guidance on which learning strategy to use in specific cases. To fill this gap, this paper provides a comprehensive study of learning strategies on scheduling heuristics of genetic programming in dynamic flexible job shop scheduling by comparing five learning strategies, including two strategies that are extended from the existing studies. The results show that learning two rules simultaneously, either using cooperative coevolution or multi-tree representation, is more effective than only learning one type of rule. Cooperative coevolution is recommended if an algorithm aims to handle a problem by dividing it into small sub-problems, and focuses on the characteristics of routing rule and sequencing rule. Genetic programming with multi-tree representation that treats the routing rule and the sequencing rule as an individual, is preferred to reduce the complexities of algorithms.**

***Index Terms*—Surrogate, Instance Rotation, Genetic Programming, Brood Recombination, Dynamic Job Shop Scheduling.**

I. Introduction

Job shop scheduling (JSS) is an important combinatorial optimisation problem that aims to allocate machine resources effectively [1]. It has attracted attention for both academics and industries due to its practical values, such as scheduling the gate resources in airports [2] and the order picking in warehouses [3]. In traditional JSS, a job consists of a sequence of operations, and the jobs need to be processed by a set of machines. Each operation can be processed by a predefined machine. Flexible JSS relaxes the machine resources accessibility constraint, and a set of machines can process an operation [4]. Two decisions need to be made simultaneously in flexible JSS, i.e., *machine assignment* and *operation sequencing*. Dynamic flexible JSS (DFJSS) considers flexible JSS under dynamic events in the job shop such as continuously new jobs arrival [5]. DFJSS is closer to real-world applications than other types of JSS such as traditional JSS, flexible JSS and dynamic JSS.

DFJSS is an NP-hard problem [6]. Exact approaches such as dynamic programming [7] aim to find the optimal solution for a problem. However, exact approaches are not efficient for DFJSS, and are typically used for static and small scale problems. Heuristic approaches such as genetic algorithms [8] can find promising solutions in a reasonable time. However, they are not suitable for DFJSS because they face rescheduling problems that cannot react to real-time scheduling efficiently. Dispatching rules (i.e., sequencing rules) [9], which can be regarded as priority functions, have been widely used to prioritise operations for JSS. However, they are usually manually designed by experts who are not always available. In addition, the designed rules are specific to a limit of scenarios, which are not effective to handle different scenarios. Genetic programming (GP) has been successfully used to automatically learn scheduling heuristics for JSS [10]–[13]. In DFJSS, the scheduling heuristics contain a routing rule for machine assignment and a sequencing rule for operation sequencing. The learning strategy of GP for evolving the routing rule and the sequencing rule is a key for its success in DFJSS.

We group the existing studies related to DFJSS with GP into two categories according to the strategies of learning routing rules and sequencing rules, i.e., learning one rule by fixing the other rule, and learning two rules simultaneously. For the first category, we can fix the routing/sequencing rule only to learn the sequencing/routing rule [14]. For the second category, there are mainly two kinds of methods, i.e., based on cooperative coevolution [15] or multi-tree representation [16]. However, there is no study on providing guidance on which learning strategy to use for a specific problem. To fill this gap, this paper compares the effectiveness of five learning strategies, including three existing ones [14]–[16], and two new strategies, to learn scheduling heuristics. In addition, this paper gives guidance for selecting the learning strategy to evolve the routing rule and the sequencing rule for DFJSS.

The overall goal of this paper is to provide a comprehensive study of learning strategies on scheduling heuristics of GP in

978-1-6654-6708-7/22/$31.00 ©2022 IEEE
Authorized licensed use limited to: Victoria University of Wellington. Downloaded on September 11,2022 at 03:51:31 UTC from IEEE Xplore. Restrictions apply.

DFJSS. This paper has the following research objectives:

- Develop GP algorithms with different learning strategies for evolving scheduling heuristics in DFJSS.
- Verify the effectiveness of the GP algorithms with different learning strategies of evolving the routing rule and the sequencing rule.
- Analyse the rule size and training time of the algorithms.
- Provide guidance for using a proper learning strategy to evolve scheduling heuristics for GP in DFJSS according to the characteristics of investigated problems.

II. Background

A. Dynamic Flexible Job Shop Scheduling

In the FJSS problem, n jobs $\mathcal{J} = \{J_1, J_2, ..., J_n\}$ need to be processed by m machines $\mathcal{M} = \{M_1, M_2, ..., M_m\}$. Each job J_j has a sequence of operations $O_j = (O_{j1}, O_{j2}, ..., O_{ji})$. Each operation O_{ji} can only be processed by one of its eligible machines $M(O_{ji}) \subseteq \pi(O_{ji})$ and its processing time $\delta(O_{ji}, M(O_{ji}))$ depends on the machine that processes it. This paper focuses on one dynamic event (i.e., continuously arriving new jobs). This means that the information of a job is unknown until it arrives. The goal of DFJSS is to optimise the machine resources to achieve descried objectives such as:

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

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

B. Genetic Programming for DFJSS

GP, as a hyper-heuristic learning approach [17], has been widely and successfully used to learn scheduling heuristics for DFJSS. Fig. 1 shows the flowchart of GP to learn scheduling heuristics for DFJSS. It starts with a randomly initialised population with a number of GP individuals. The individuals are then evaluated with a DFJSS simulation to measure the quality of individuals. If the stopping criterion is met, the GP algorithm will report the best individual found so far as the final output. Otherwise, parents are selected based on the parent selection method, and the selected parents are used to generate offspring with genetic operators, i.e., reproduction, crossover and mutation. All the generated offspring will build a new population and be evaluated in the next generation. It is noted that the output of GP for DFJSS is a routing rule and a sequencing rule. The routing rule prioritises machines at the routing decision points (i.e., when there is a ready operation). The sequencing rule prioritises operations at the sequencing decision points (i.e., when a machine becomes idle and operations are waiting in its queue).

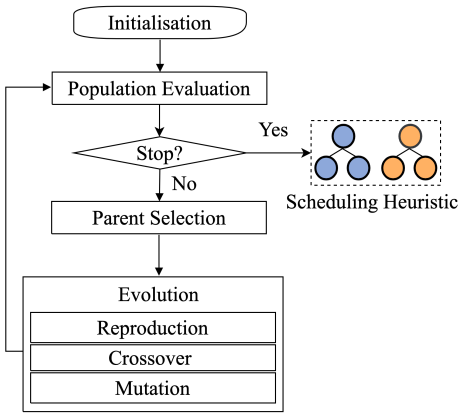

Fig. 1: The flowchart of GP to learn scheduling heuristics for DFJSS.

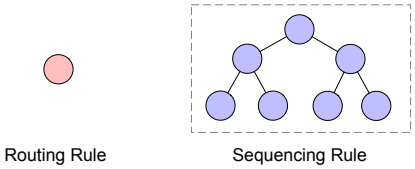

Fig. 2: Fixed routing rule and learn only sequencing rule.

III. Learning Strategies on Scheduling Heuristics in Dynamic Flexible Job Shop Scheduling

We describe five learning strategies for evolving scheduling heuristics in DFJSS. Note that the learning strategies in sections 3.1, 3.4 and 3.5 have been studied in [14]–[16], while the learning strategies described in 3.2 and 3.3 have not been investigated yet. We use dotted rectangles to highlight the learned rules. The pink and blue individuals represent routing rules and sequencing rules, respectively.

A. Fixed Routing Rule and Learn Only Sequencing Rule

A flexible JSS problem was studied in [14] by fixing the routing rule and learning only the sequencing rule. Specifically, the fixed routing rule is normally a manually designed rule (i.e., in this paper, we use WIQ [18], the total needed processing time of operations in the queue of machines). During the whole evolutionary process of GP, only the sequencing rule is learned, as shown in Fig. 2.

B. Fixed Sequencing Rule and Learn Routing Rule Only

Motivated by the study in [14], we can also fix the sequencing rule (i.e., PT, the processing time of operations) and learn only the routing rule, as shown in Fig. 3. It is noted that there is no study to investigate this learning strategy.

Authorized licensed use limited to: Victoria University of Wellington. Downloaded on September 11,2022 at 03:51:31 UTC from IEEE Xplore. Restrictions apply.

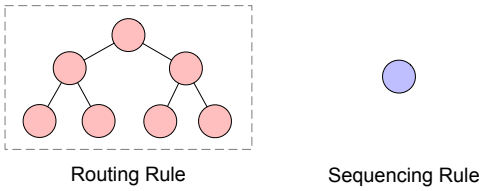

Fig. 3: Fixed sequencing rule and learn routing rule only.

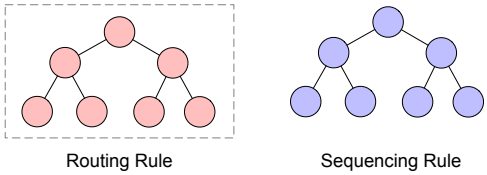

Fig. 4: Same rule as the routing rule and sequencing rule.

C. Same Rule as the Routing Rule and the Sequencing Rule

Another possible learning strategy is to use one rule to make decisions for both routing and sequencing, as shown in Fig. 4. The advantage is that the number of rules is reduced (i.e., only one), which might make it easy for decision makers to understand. The disadvantage is that the rules may not be effective. This is because the feature importance of routing and sequencing rules are not the same [18]. It is noted that this has not been investigated in the existing studies.

D. Learning Rules with Cooperative Coevolution

The routing rule and sequencing rule was learned simultaneously with a cooperative coevolution framework [15]. Two subpopulations are used to learn the routing rule and the sequencing rule, respectively, as shown in Fig. 5. For the evaluations of the routing/sequencing rules, the best learned sequencing/routing rule (denoted with stars) at the previous generation is combined to evaluate the corresponding individuals. At the first generation, we randomly select a routing rule and a sequencing rule as the best learned rules for evaluation.

E. Learning Rules with Multi-tree Representation

GP with multi-tree representation was proposed to learn the routing rule and the sequencing rule simultaneously by using one GP individual [16]. A GP individual contains two trees, one for the routing rule and the other for the sequencing rule, as shown in Fig. 6. The advantage is that each individual is a combination of the routing rule and the sequencing rule, and it can be evaluated independently (i.e., do not need to get the other rule by setting a manually designed rule or finding the best rule from the previous generation).

IV. Experiment Design

A. Simulation Model

We use simulation to mimic the environment of DFJSS to investigate the effectiveness of strategies to represent scheduling heuristics for GP. We assume 5000 jobs need to be processed by ten machines [19]. The importance of jobs differ,

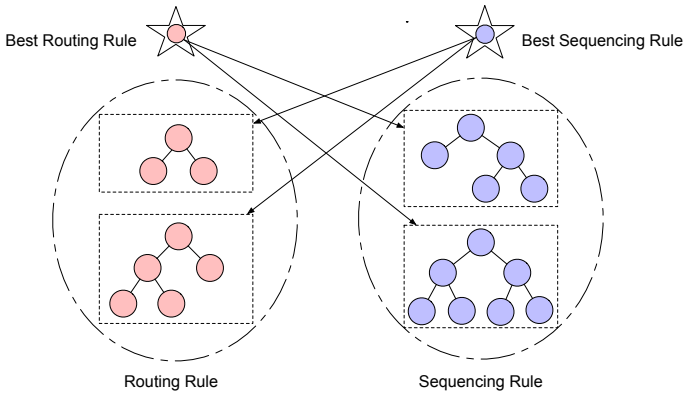

Fig. 5: Learn routing and sequencing rules with cooperative coevolution.

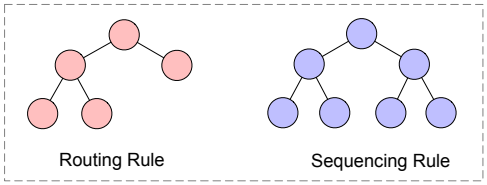

Fig. 6: Learn routing and sequencing rules with multi-tree representation.

and we use weight (i.e., one, two and four) to indicate the importance of jobs. A large weight indicates that a job is important. Each job has a number of operations that follows a uniform discrete distribution between one and ten. Each operation has a number of candidate machines, which follows a uniform discrete distribution between one and ten. The processing time of each operation is set by uniform discrete distribution with the range [1, 99].

In order to verify the effectiveness of the involved algorithms, scenarios with different settings (i.e., different objectives and utilisation levels) are examined. *Utilisation level* (p) is an essential factor to simulate different scenario environments. It is the proportion of time that a machine is busy. Jobs arrive continuously in the simulation according to a Poisson process with a rate λ. The expression is shown in Eq. (1), where μ represents the machines's average processing time, and P_M represents the likelihood of a job visiting a machine. For example, if each job has two operations, P_M will be 2/10. We can see that the utilisation level is managed by adjusting the rate λ in the Poisson process. A high utilisation level is realised by enlarging the value of λ.

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

To improve the generalisation ability of the learned scheduling heuristics for DFJSS problems, the simulation used at each generation is rotated by setting a new random seed. A warm-up period of 1000 jobs is used to improve the accuracy of collected data. We collect data from the following 5000 jobs. The simulation keeps running until the 6000th job is finished.

Authorized licensed use limited to: Victoria University of Wellington. Downloaded on September 11,2022 at 03:51:31 UTC from IEEE Xplore. Restrictions apply.

TABLE I: The involved rule types of GPs, GPr, GPrs, CCGPrs, and MTGPrs.

Algorithms	Routing Rule	Sequencing Rule	Same
GPs	✗	✓	No
GPr	✓	✗	No
GPrs	✓	✓	Yes
CCGPrs	✓	✓	No
MTGPrs	✓	✓	No

TABLE II: The terminal and function sets.

	Terminals	Description
Machine-related	NIQ	The number of operations in the queue
	WIQ	Current work in the queue
	MWT	Waiting time of a machine
Operation-related	PT	Processing time of an operation
	NPT	Median processing time for next operation
	OWT	Waiting time of an operation
Job-related	WKR	Median amount of work remaining of a job
	NOR	The number of operations remaining of a job
	W	Weight of a job
	TIS	Time in system
functions	$+, \ , *, /$ max, min	as usual meaning

TABLE III: The parameter setting of GP.

Parameter	Value
*Number of subpopulations	2
*Subpopulation size	500
Population size	1000
Method for initialising population	ramped-half-and-half
Initial minimum/maximum depth	2 / 6
Maximal depth of programs	8
The number of elites	10
Crossover/Mutation/Reproduction rate	80% / 15% / 5%
Parent selection	Tournament selection with size 7
Number of generations	51
Terminal/non-terminal selection rate	10% / 90%

* is for CCGPrs only.

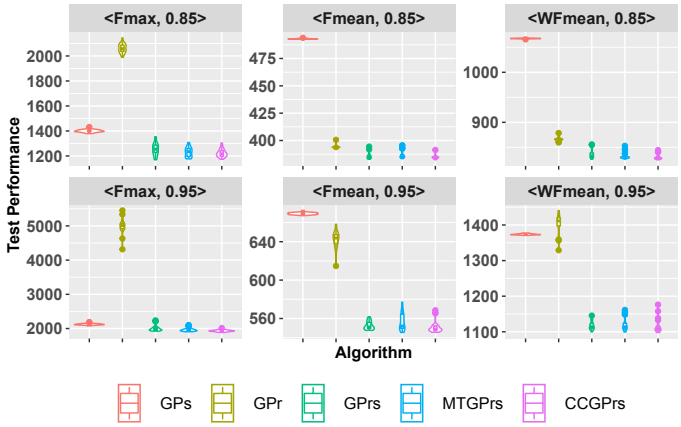

Fig. 7: The violin plots of GPs, GPr, GPrs, MTGPrs and CCGPrs over 30 independent runs in six flowtime related DFJSS scenarios.

B. Comparison Design

Five algorithms are involved in this paper. The algorithms that only learn sequencing/routing rule by fixing the routing/sequencing rule are named GPs and GPr, respectively. The algorithm that uses the same rule as both the routing rule and the sequencing rule is named GPrs. GP with cooperative coevolution to learn the routing rule and the sequencing rule simultaneously, is named CCGPrs in this paper. The algorithm with multi-tree representation to learn the routing rule and the sequencing rule is named MTGPrs. Table I shows the characteristics of the involved five algorithms.

The proposed algorithms are tested on 12 scenarios. The scenarios consist of six objectives (i.e., max-flowtime (Fmax), mean-flowtime (Fmean), and mean-weighted-flowtime (WFmean), max-tardiness (Tmax), mean-tardiness (Tmean) and mean-weighted-tardiness (WTmean)) and two utilisation levels (i.e., 0.85 and 0.95). Note that the learned best rule is tested on 50 unseen instances, and the mean objective value is reported as its objective value.

C. Parameter Settings

In our experiment, the terminal and function set are shown in Table II, following the setting in [20]. The "/" operator is a protected division, returning one if divided by zero. The other parameter settings of GP are shown in Table III.

V. Results and Discussions

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

A. Quality of Learned Scheduling Heuristics

Table IV shows the mean and standard deviations of the objective values of GPs, GPr, GPrs, MTGPrs and CCGPrs on 12 scenarios. The results show that CCGPrs performs the best with the smallest rank, and is significantly better than the algorithms in most of the scenarios. MTGPrs is the second-best among all other algorithms. However, the algorithms that learn only one rule have the worst performance. We can conclude that learning two rules simultaneously is more effective than learning one rule only.

Fig. 7 shows the violin plots of the test performance of GPs, GPr, GPrs, MTGPrs and CCGPrs on the scenarios with flowtime related objectives. Taking the algorithms that only learn one rule into consideration (i.e., GPs and GPr), GPr is significantly better than GPs in 4 out of 6 scenarios (i.e., <Fmean, 0.85>, <Fmean, 0.95>, <WFmean, 0.85> and <WFmean, 0.95>). However, GPr is worse than GPs in two max-flowtime scenarios (i.e., <Fmax, 0.85> and <Fmax, 0.95>). One possible reason is that the importance of the routing rule and the sequencing rule is sensitive to the op-

Authorized licensed use limited to: Victoria University of Wellington. Downloaded on September 11,2022 at 03:51:31 UTC from IEEE Xplore. Restrictions apply.

TABLE IV: The mean (standard deviation) of the objective values of GPs, GPr, GPrs, MTGPrs, and CCGPrs over 30 independent runs for 12 DFJSS scenarios.

Scenario	GPs	GPr	GPrs	MTGPrs	CCGPrs
<Fmax, 0.85>	1399.96(11.00)	2060.52(35.77)(↓)	1253.12(49.29)(↑)(↑)	1230.97(38.00)(↑)(↑)(≈)	1223.12(31.23)(↑)(↑)(↑)(≈)
<Fmax, 0.95>	2124.99(23.16)	4999.75(201.09)(↓)	2007.37(85.52)(↑)(↑)	1969.66(52.81)(↑)(↑)(↑)	1934.40(27.84)(↑)(↑)(↑)(↑)
<Fmean, 0.85>	493.02(0.29)	393.98(1.57)(↑)	385.66(2.69)(↑)(↑)	386.76(3.52)(↑)(↑)(≈)	384.69(1.63)(↑)(↑)(≈)(↑)
<Fmean, 0.95>	669.31(1.13)	642.50(8.69)(↑)	552.62(4.25)(↑)(↑)	556.53(8.98)(↑)(↑)(≈)	550.94(5.79)(↑)(↑)(↑)(↑)
<WFmean, 0.85>	1067.55(0.63)	865.91(3.55)(↑)	836.18(8.96)(↑)(↑)	832.18(6.10)(↑)(↑)(↑)	830.03(4.63)(↑)(↑)(↑)(↑)
<WFmean, 0.95>	1373.79(1.71)	1405.72(26.68)(↓)	1117.81(11.94)(↑)(↑)	1119.90(17.67)(↑)(↑)(≈)	1113.78(17.39)(↑)(↑)(↑)(↑)
<Tmax, 0.85>	864.73(9.16)	1373.38(36.14)(↓)	777.86(37.64)(↑)(↑)	759.78(27.91)(↑)(↑)(≈)	748.62(24.03)(↑)(↑)(↑)(≈)
<Tmax, 0.95>	1690.73(24.93)	4361.76(257.20)(↓)	1591.44(52.45)(↑)(↑)	1553.64(39.29)(↑)(↑)(↑)	1533.23(31.53)(↑)(↑)(↑)(↑)
<Tmean, 0.85>	102.58(0.20)	47.11(0.72)(↑)	40.46(1.14)(↑)(↑)	40.66(1.99)(↑)(↑)(≈)	40.27(1.85)(↑)(↑)(≈)(≈)
<Tmean, 0.95>	266.47(1.05)	262.10(9.88)(≈)	180.32(4.51)(↑)(↑)	177.91(5.13)(↑)(↑)(↑)	175.49(2.85)(↑)(↑)(↑)(↑)
<WTmean, 0.85>	209.26(0.60)	78.39(3.17)(↑)	78.39(3.17)(↑)(↑)	76.45(4.06)(↑)(↑)(↑)	75.82(3.83)(↑)(↑)(↑)(↑)
<WTmean, 0.95>	488.25(2.27)	565.74(26.39)(↓)	302.50(11.46)(↑)(↑)	306.18(18.08)(↑)(↑)(≈)	294.58(9.65)(↑)(↑)(↑)(↑)
Win / Draw / Lose	12 / 0 / 0	12 / 0 / 0	10 / 2 / 0	9 / 3 / 0	N/A
Average Rank	4.46	4.52	2.34	2.09	**1.61**

timised objectives. In the mean flowtime related scenarios (i.e., <Fmean, 0.85>, <Fmean, 0.95>, <WTmean, 0.85> and <WTmean, 0.95>), it might be true that the routing rule is more important than the sequencing rule, and learning an effective routing rule contributes more to the performance. On the contrary, the sequencing rule might be more important for the max-flowtime related scenarios (i.e., <Fmax, 0.85> and <Fmax, 0.95>), and learning an effective sequencing rule contributes more to the performance.

Learning two rules simultaneously (i.e., GPrs, MTGPrs, CCGPrs) shows their superiority compared with the ones that learn only one type of rules (i.e., GPs and GPr). This confirms that learning the routing rule and the sequencing rule simultaneously is effective for GP in DFJSS. In addition, MTGPrs and CCGPrs, can achieve better performance with smaller objective values than GPrs. This indicates that using one unique rule for both machine assignment and operation sequencing rule is not effective. This is consistent with our previous studies that the machine assignment and operation sequencing decisions have different characteristics, which leads to various requirements for learning scheduling heuristics [18].

B. Average Sizes of the Learned Routing Rules

A smaller rule size (i.e., the number of nodes) makes it easier for the job shop operators to understand the rules. In addition, algorithms with smaller programs are less complex (i.e., smaller search space). Fig. 8 shows the violin plots of the average sizes of routing rules of GPs, GPr, GPrs, MTGPrs, and CCGPrs over 30 independent runs in six flowtime related scenarios. We can see that the sizes of routing rules of GPs are 1 (i.e., fixed routing rule, WIQ) in all the scenarios. Among the algorithms that learn routing rules, MTGPrs has the smallest routing rules, while GPr gets the largest routing rules. In addition, GPrs and CCGPrs have similar routing rule sizes in the scenarios with utilisation levels of 0.85. However, CCGPrs has smaller routing rules than GPrs in the scenarios with utilisation levels of 0.95. Taking the test performance of the algorithms that learn two rules simultaneously (i.e., GPrs, MTGPrs and CCGPrs) into consideration (as discussed in Section V-A), we can see that although GPrs has larger routing rule sizes than MTGPrs and CCGPrs, it does not help itself achieve better performance. In general, MTGPrs can get effective rules with small rule sizes.

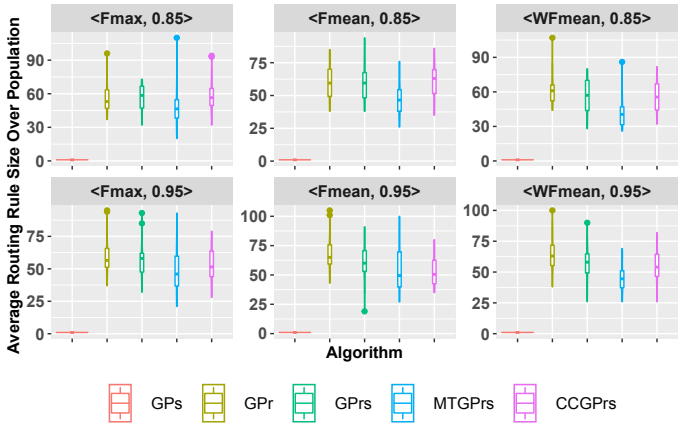

Fig. 8: Violin plots of the average sizes of *routing rules* of GPs, GPr, GPrs, MTGPrs, and CCGPrs over 30 runs in six flowtime related scenarios.

C. Average Sizes of the Learned Sequencing Rules

Fig. 9 shows the violin plots of the average sizes of GPs, GPr, GPrs, MTGPrs, and CCGPrs according to 30 independent runs in six flowtime related scenarios. For GPr, we can see that the sizes of sequencing rules are 1 (i.e., fixed sequencing rule, PT). The same as routing rules, MTGPrs achieves the smallest sequencing rules in all the examined scenarios. This is consistent with the finding in [16].

D. Average Sizes of Pairs of Routing and Sequencing Rules

We know that the routing rule and the sequencing rule work as a pair to make two decisions in DFJSS. An effective routing rule with an ineffective sequencing rule may have similar performance with an effective sequencing rule with an ineffective routing rule. Thus, it is reasonable to take the size

Authorized licensed use limited to: Victoria University of Wellington. Downloaded on September 11,2022 at 03:51:31 UTC from IEEE Xplore. Restrictions apply.

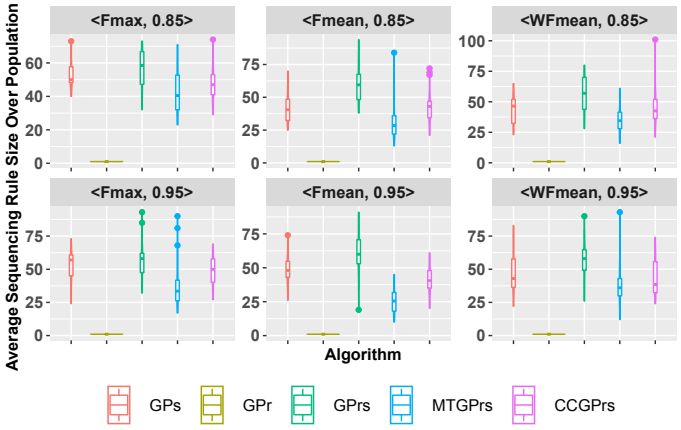

Fig. 9: Violin plots of the average sizes of *sequencing rules* of GPs, GPr, GPrs, MTGPrs, and CCGPrs over 30 runs in six flowtime related scenarios.

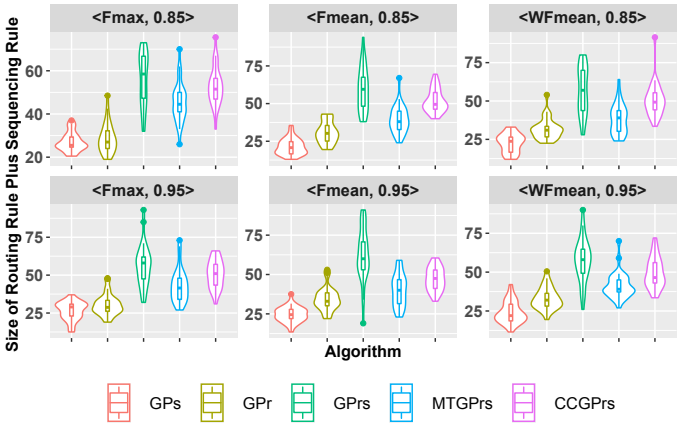

Fig. 10: Violin plots of the average sizes of routing rules plus sequencing rules of GPs, GPr, GPrs, MTGPrs, and CCGPrs over 30 independent runs in six flowtime related scenarios.

of the routing rule and sequencing rule together to investigate the rule size. Fig. 10 shows the violin plots of the average sizes of routing rules plus sequencing rules of GPs, GPr, GPrs, MTGPrs, and CCGPrs according to 30 independent runs in six flowtime related scenarios. The sizes of scheduling heuristics of GPs are smaller than GPr in all scenarios. This indicates that the routing rule might contribute more to the performance, and its size is increased by GP to find effective rule for DFJSS. The smallest rule sizes are achieved by MTGPrs, which is consistent with the findings in sections V-B and V-C.

E. Training time

Fig. 11 shows the training time (in minutes) of GPs, GPr, GPrs, MTGPrs, and CCGPrs over 30 independent runs in six flowtime related scenarios. The results show that although it is the same that GPs and GPr only learn one type of the rules, their training time is different. The training time of GPr is much longer than GPs in all the examined scenarios. It is consistent with the findings of routing sizes (i.e., GPr has larger rule sizes than GPs) in section V-D, and the

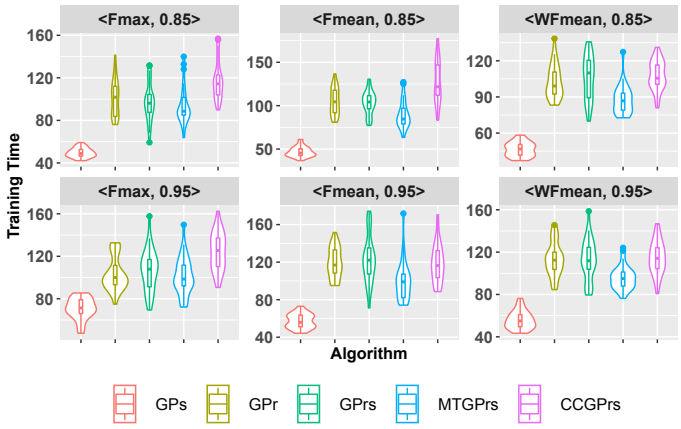

Fig. 11: Violin plots of the training time (in minutes) of GPs, GPr, GPrs, MTGPrs, and CCGPrs over 30 independent runs in six flowtime related scenarios.

algorithms with larger rule sizes are time-consuming. Among GPrs, MTGPrs and CCGPrs, MTGPrs is the most efficient one. In addition, although CCGPrs achieves better performance than MTGPrs as shown in section V-A, CCGPrs is more time-consuming than MTGPrs.

VI. Discussions and Further Analysis

A. Discussions

Considering the effectiveness and efficiency, MTGPrs and CCGPrs are promising algorithms to learn the routing rule and the sequencing rule simultaneously. However, they have different characteristics that affect the choose of them.

- MTGPrs and CCGPrs are promising algorithms to learn routing and sequencing rules simultaneously. However, implementing cooperative coevolution is more complex than using multi-tree representation. The implementation with multi-tree representation is easier for beginners.
- If we do not consider the complexity of algorithms, GP with cooperative coevolution and multi-tree representation are suitable for GP in DFJSS. Which one to use depends on users' preferences.
- If more than two rules need to be learned [19], [21], GP with multi-tree representation is a good choice to manage individuals to keep a simple framework. The interaction between rules can be easily managed.
- If the algorithm focuses on investigating the characteristics of routing rule and sequencing rule such as feature importance [18] and rule importance, using GP with cooperative coevolution is suitable to keep them separate from each other. This makes it easier to investigate them.
- Although using the same rule for routing and sequencing is not as good as using cooperative coevolution and multi-tree representation, its performance is not as bad compared with learning one type of rule. We know that the feature importance for effective routing rule and sequencing differs [18]. Using the same rule for routing and sequencing may not be effective. To investigate

Authorized licensed use limited to: Victoria University of Wellington. Downloaded on September 11,2022 at 03:51:31 UTC from IEEE Xplore. Restrictions apply.

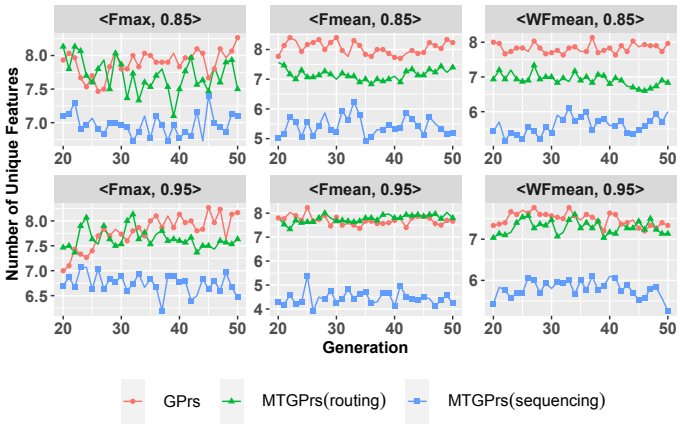

Fig. 12: Curves of the number of unique features of GPrs and MTGPrs over 30 independent runs in six flowtime scenarios.

this, Fig. 12 shows the curves of the number of unique features of GPrs and MTGPrs over 30 independent runs in six DFJSS scenarios. For GPrs, its routing rule and sequencing rule are the same, and there is only one line to indicate both the number of unique features of the routing rule and sequencing rule. The results show that the learned rules by GPrs involve a larger number of features than both the routing and sequencing rules obtained by MTGPrs in most scenarios. Further looking at MTGPrs, we can see that the routing rules contain more features than the sequencing rule. We can conclude that GPrs do manage to learn comprehensive rules for both routing and sequencing by involving a larger number of unique features. These findings can also verify the effectiveness of the strategy of learning two rules simultaneously, which makes it possible for the routing rule and the sequencing rule to focus on their own characteristics.

B. Learned Scheduling Heuristics

The algorithms investigated in this paper can be categorised into two groups based on the number of used rules for DFJSS, i.e., learning one rule for both routing and sequencing (GPrs), and learning a routing rule and a sequencing rule (GPs, GPr, MTGPrs and CCGPrs). It is interesting to know how these two kinds of rules work. In this section, we take the best learned rules obtained by GPrs and CCGPrs in scenario <WTmean, 0.95> to analyse the characteristics of rules. Based on our preliminary work [22], we know that in scenario <WTmean, 0.95>, MWT (marked in pink) is the most important feature for routing rule, and W (marked in blue) is the most important feature for sequencing rule. Taking the most important feature for each rule as an example, Fig. 13 shows one of the best rules obtained by GPrs that is used for making both routing and sequencing decisions in DFJSS. We can see that this rule aims to be comprehensive to cover the important features for routing and sequencing. Fig. 14 shows one of the best routing rules learned by CCGPrs. Different from the rule in Fig. 13, this routing rule pays more attention to important features (e.g., MWT) for making routing decisions only. Fig. 15 shows one of the best sequencing rules learned by CCGPrs. This sequencing rule focuses more on important features (e.g., W) for making sequencing decisions only. As we discussed earlier, the performance of CCGPrs is better than GPrs. We can conclude that using a single rule is hard to take the role for both routing and sequencing. This is because routing and sequencing decisions have their own characteristics which is hard for a single rule to cover all of them. This is consistent with our findings in Section V-A.

VII. Conclusions and Future Work

The goal of this paper is to study different learning strategies for evolving scheduling heuristics, i.e., routing rule and sequencing rule, for GP in DFJSS. The goal has been successfully achieved by investigating five learning strategies to evolve the routing rule and the sequencing rule.

The results show that GP with cooperative coevolution and GP with multi-tree representation are effective strategies for learning the routing rule and the sequencing rule simultaneously. GP with cooperative coevolution achieves the best performance. Compared with GP algorithm with cooperative coevolution, GP with multi-tree representation is more efficient, and the learned scheduling heuristics are more interpretable. We conclude that both GP with cooperative coevolution and multi-tree representation are effective strategies to learn scheduling heuristics for DFJSS. If an algorithm is designed to learn more than two rules, it is better to use GP with multi-tree representation to make the framework of the algorithms tidy. If an algorithm pays attention to the characteristics of routing rules and sequencing rules separately, using GP with cooperative coevolution is a good choice. In addition, we find that using the same rule for routing and sequencing is not effective. However, it does manage to cover the features for both the routing rule and the sequencing rule. This learning strategy is first explored in this paper.

Some interesting directions can be further investigated in the near future. This work already shows the importance of routing rule and sequencing rule differs in different scenarios. We would like to find more promising ways to allocate the computational resources properly to improve its performance.

References

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

[2] U. Dorndorf, A. Drexl, Y. Nikulin, and E. Pesch, "Flight gate scheduling: State-of-the-art and recent developments," *Omega*, vol. 35, no. 3, pp. 326–334, 2007.

[3] R. De Koster, T. Le-Duc, and K. J. Roodbergen, "Design and control of warehouse order picking: A literature review," *European journal of operational research*, vol. 182, no. 2, pp. 481–501, 2007.

[4] I. A. Chaudhry and A. A. Khan, "A research survey: review of flexible job shop scheduling techniques," *International Transactions in Operational Research*, vol. 23, no. 3, pp. 551–591, 2016.

[5] F. Zhang, Y. Mei, S. Nguyen, M. Zhang, and K. C. Tan, "Surrogate-assisted evolutionary multitask genetic programming for dynamic flexible job shop scheduling," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 4, pp. 651–665, 2021.

[6] 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.

Authorized licensed use limited to: Victoria University of Wellington. Downloaded on September 11,2022 at 03:51:31 UTC from IEEE Xplore. Restrictions apply.

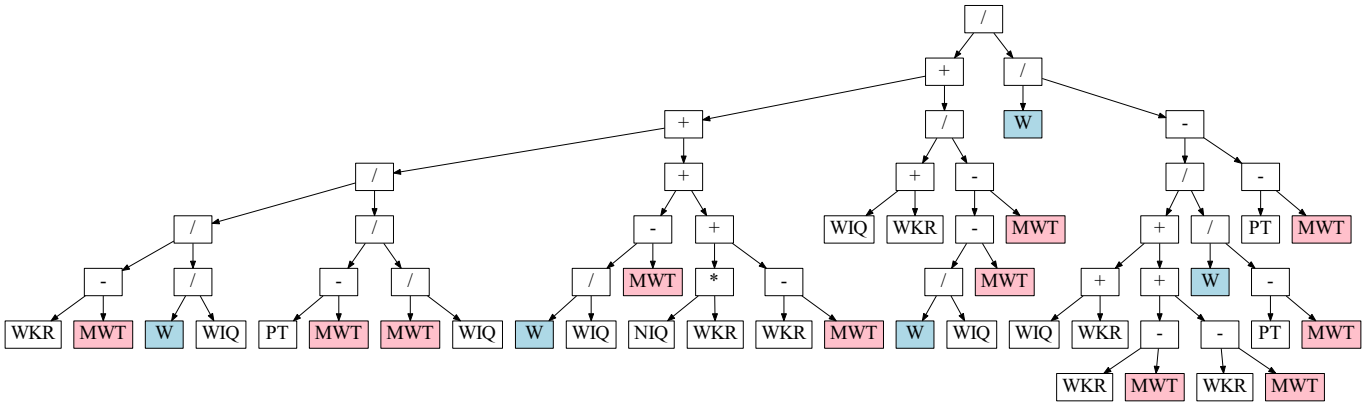

Fig. 13: A best rule (i.e., both for marking routing and sequencing decisions) learned by GPrs in scenario <WTmean, 0.95>.

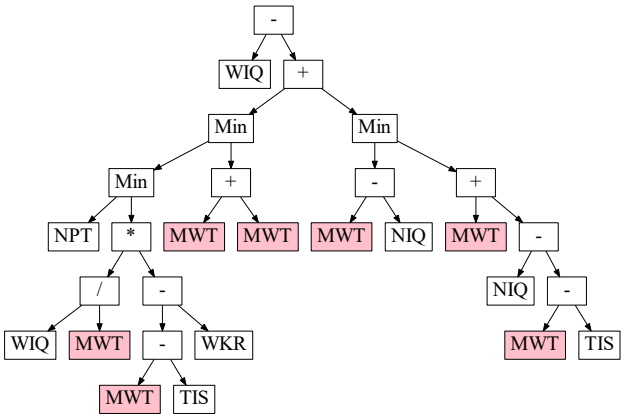

Fig. 14: An example of the best routing rule learned by CCGPrs in scenario <WTmean, 0.95>.

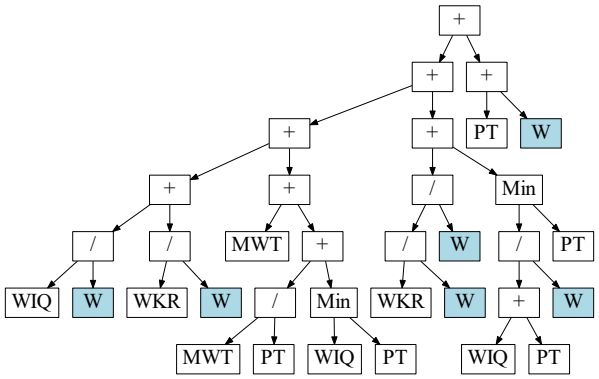

Fig. 15: An example of the best sequencing rule learned by CCGPrs in scenario <WTmean, 0.95>.

[7] H. Chen, C. Chu, and J. 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.

[8] J. F. Gonçalves, J. J. de Magalhães Mendes, and M. G. Resende, "A hybrid genetic algorithm for the job shop scheduling problem," *European journal of Operational Research*, vol. 167, no. 1, pp. 77–95, 2005.

[9] 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.

[10] 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 Genetic and Evolutionary Computation Conference*. ACM, 2010, pp. 257–264.

[11] 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.

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

[13] M. DJurasević and D. Jakobović, "Comparison of schedule generation schemes for designing dispatching rules with genetic programming in the unrelated machines environment," *Applied Soft Computing*, vol. 96, p. 106637, 2020.

[14] 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.

[15] 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.

[16] 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.

[17] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, "Hyper-heuristics: A survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, no. 12, pp. 1695–1724, 2013.

[18] 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.

[19] F. Zhang, Y. Mei, S. Nguyen, K. C. Tan, and M. Zhang, "Multitask genetic programming based generative hyper-heuristics: A case study in dynamic scheduling," *IEEE Transactions on Cybernetics*, 2020, Doi: 10.1109/TCYB.2021.3065340.

[20] 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.

[21] Y. Chen, J. Zhong, L. Feng, and J. Zhang, "An adaptive archive-based evolutionary framework for many-task optimization," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 4, no. 3, pp. 369–384, 2019.

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

Authorized licensed use limited to: Victoria University of Wellington. Downloaded on September 11,2022 at 03:51:31 UTC from IEEE Xplore. Restrictions apply.