2025 IEEE Congress on Evolutionary Computation (CEC) | 979-8-3315-3431-8/25/$31.00 ©2025 IEEE | DOI: 10.1109/CEC65147.2025.11043103

Investigation of Decision Making with Scheduling Rules Learned via Genetic Programming for Dynamic Flexible Job Shop Scheduling

Luyao Zhu , Fangfang Zhang✉ , Yi Mei , Mengjie Zhang
Centre for Data Science and Artificial Intelligence & School of Engineering and Computer Science,
Victoria University of Wellington, Wellington, New Zealand
{luyao.zhu, fangfang.zhang, yi.mei, mengjie.zhang}@vuw.ac.nz

***Abstract*—Genetic programming (GP) has been popularly used to learn scheduling rules for dynamic flexible job shop scheduling. These scheduling rules serve as priority functions to prioritise candidate machines or operations at decision points. In the implementation level, a high prioritised machine and operation can be the ones with the highest or lowest priority value calculated with a scheduling rule in the decision making process. In theory, GP can adaptively evolve scheduling rules to accommodate different priority settings. However, research exploring the possible hidden differences during the evolutionary process remains limited. To fill this gap, this paper presents a comprehensive investigation into scheduling rules learned by GP under varying priority settings. The results show that while GP achieves similar performance in most investigated scenarios with different priority settings, GP-low where candidates with the lowest priority values are selected, can learn effective scheduling rules faster. Specifically, GP-low can learn smaller sequencing rules. Furthermore, visualisations of the scheduling rules illustrate how GP adaptively adjusts node positions to evolve effective rules. The study also highlights an inherent bias in initialised scheduling rules, which tend to prefer candidates with lower priority values in the examined scenarios. Moreover, GP-low exhibits a broader distribution of priority values. These findings can provide deeper insights into GP's adaptive learning mechanisms and offer valuable guidance for decision-making of using scheduling rules.**

***Index Terms*—scheduling rules, decision making, genetic programming, dynamic flexible job shop scheduling**

I. Introduction

Dynamic flexible job shop scheduling (DFJSS) is a critical combinatorial optimisation problem that has been widely applied in various domains, including manufacturing [1], [2] and cloud computing [3]. DFJSS aims to optimise resource allocation efficiently. In DFJSS, multiple jobs need to be processed by several machines. Each job consists of a predefined sequence of operations. Due to the flexibility of the job shop, each operation can be processed by more than one machine. Consequently, two decisions need be made simultaneously. The first is machine assignment (i.e., assign a ready operation to a particular machine). The second is operation sequencing (i.e., select a specific operation processed first in an idle machine). Furthermore, decision-making must be conducted in some dynamic environments, such as job arrivals [4], [5].

Scheduling rules [6] have been extensively employed to address DFJSS. These rules operate as priority functions, assigning priority values to candidates at some decision points and selecting specific options based on obtained priority values. However, the design process of manual rules typically requires domain experts to rely on prior experience and costly trial-and-error processes. Furthermore, these rules are highly sensitive to the scenarios in which they are applied, often resulting in good performance only within certain specific situations [1]. Genetic programming (GP) [7], a hyper-heuristic approach [8], has been successfully applied to automatically learning scheduling rules for DFJSS [9]. Specially, only few domain knowledge is needed in the process of learning rules with GP. Scheduling rules learned by GP tend to have better performance and generalisation ability compared to manually designed rules.

A crucial aspect of decision-making in scheduling rules is priority settings. Priority settings typically follows two approaches. One approach assigns the highest priority to the candidate with the **lowest priority value**. For example, the commonly used dispatching rule *Shortest Processing Time* (SPT) [10] prioritises jobs with the lowest processing time. Conversely, the other approach assigns the highest priority to the candidate with the **highest priority value**. A typical example is the *Weighted Shortest Processing Time* (WSPT) rule [11], which selects the job with the highest weight-to-processing-time ratio. Theoretically, GP can adaptively learn effective scheduling rules under different priority settings. Therefore, whether GP is designed to select candidates with the lowest or highest priority values during training, it can autonomously evolve effective scheduling rules for DFJSS. Although GP ultimately achieves comparable performance under different priority settings, certain differences may arise during the evolutionary process. These differences may result from GP's adaptive learning of scheduling rules. It is worth investigating these differences and their underlying causes. However, little research has been conducted to explore these aspects. To fill this gap, this paper compares and analyses scheduling rules learned under different decision-making strategies from multiple perspectives, aiming to answer the following questions:

- Although different priority settings may lead to similar

final performance, which setting enables faster discovery of superior solutions during evolution?
- Will different priority settings influence rule size (i.e., the number of nodes within a GP tree) and selected features?
- How does GP adaptively learn effective scheduling rules in response to changes in priority settings?

II. Background

A. Dynamic Flexible Job Shop Scheduling

DFJSS aims to optimise the allocation of machine resources for processing jobs in the job shop. In DFJSS, m machines $\mathcal{M} = \{M_1, M_2, ..., M_m\}$ are required 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 must be handled in a fixed order. Each operation O_{ji} can be processed by more than one machine $M(O_{ji}) \subseteq \pi(O_{ji})$. In addition, DFJSS involves two distinct scheduling rules. One is routing rule which determines which specific machine can process an operation, and the other is the sequencing rule which decides which operation should be first handled on an idle machine. A common dynamic event, i.e., new job arrival, is considered, where the details of a new job remain unknown until it is released to the shop floor. The main constraints in DFJSS are outlined below.

- A machine can process only one operation at a time.
- Each operation can only be processed by one of its candidate machines at a time.
- An operation cannot begin processing until all of its preceding operations have been completed.
- Once an operation has started, its processing cannot be stopped or paused until it is completed.

This paper aims to minimise two time-based objectives and maximise one profit-based objective, respectively. Their calculations are demonstrated as follows.

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

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, R_j is the total revenue obtained from the completion of J_j before d_j, α_j represents unit time delay penalty of J_j beyond d_j.

Simulation techniques is used to mimic DFJSS situations. Referring to the widely used DFJSS simulation [12], we assume that ten machines need to process 5000 jobs. Jobs have different weights w, 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 a due date factor (e.g., 1.5) multiplied by its processing time PT plus the arrival time.

In addition, the number of operations in each job varies uniformly between 1 and 10. The processing time for each operation is drawn from a discrete uniform distribution ranging from 1 to 99. Each operation is assigned between 1 and 10 candidate machines, also based on a uniform distribution. The revenue R of each job follows a lognormal distribution based on a standard normal distribution with a mean of 0 and a standard deviation of 1 [13]. The unit time delay penalty beyond the due date, denoted as α, is calculated as $\alpha = w \cdot R/(4PT)$, where w is the job weight and PT is the total processing time. New jobs arrive over time according to a Poisson process with an arrival rate λ, which is determined by the system utilisation level p. Specifically, λ is computed using the equation $\lambda = \mu \cdot P_M/p$, where μ is the average processing time and P_M represents the probability of a job visiting a given machine. To obtain steady-state performance, we collect data of 5000 completed jobs after 1000 warm-up jobs arrival.

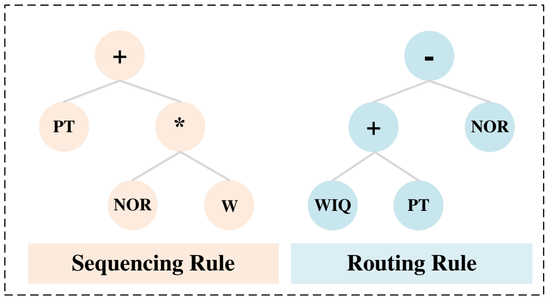

Fig. 1. An example of a multi-tree GP individual representation.

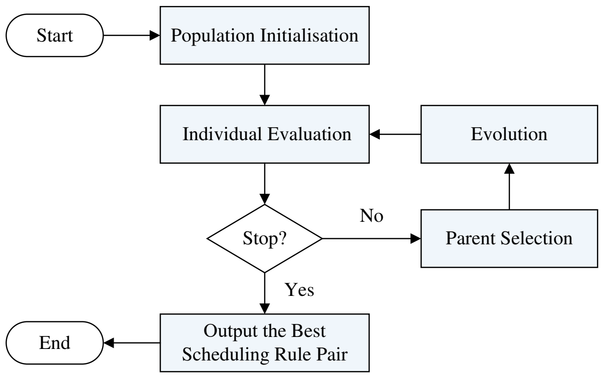

Fig. 2. The flowchart of MTGP to learn scheduling rules for DFJSS.

B. Genetic Programming for DFJSS

GP has been successfully used to learn scheduling rules for DFJSS problems. The multi-tree GP structure [14] is a commonly used representation for evolving both the sequencing rule and the routing rule simultaneously. Each GP individual consists of two trees, where one tree corresponds to the sequencing rule and the other to the routing rule, forming a scheduling rule pair. An example of a multi-tree GP individual representation is shown in Fig. 1. PT, NOR, W, WIQ are features related to the job shop, representing the processing time of an operation, the number of remaining operations for a job, the weight of a job, and the total work in the machine's waiting queue, respectively. The sequencing rule is a priority function PT + NOR * W, which assigns priority values to its

TABLE I
AN EXAMPLE OF THE DECISION MAKING OF THE SEQUENCING RULE PT + NOR * W AT A SEQUENCING DECISION POINT WITH THREE CANDIDATE OPEARTIONS.

Opeartion Number	(PT	Feature NOR	W)	Priority Value	Chosen Operation
O_1	30	3	2	36	
O_2	10	8	4	$\underline{42}$	O_2 or O_3
O_3	20	1	1	$\underline{21}$	

candidate operations. The operation with the highest or lowest priority value is processed first.

Fig. 2 illustrates the flowchart of GP to learn scheduling heuristics for DFJSS problems. The process begins 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 based on their fitness values. These parents generate new offspring through genetic operators (i.e., crossover, mutation, and reproduction), forming a new population. Otherwise, the GP algorithm will output the best individual (scheduling rule pair) in the current generation.

C. Decision Making of Scheduling Rules

Taking the sequencing rule PT + NOR * W as an example, assume that three operations (i.e., O_1, O_2 and O_3) are waiting to be processed in an idle machine, which is shown in Table I. Given the feature values of three operations, the priority values of O_1, O_2 and O_3 are 93, 58 and 121, respectively. According to the priority values, this machine will handle O_3 first if we define sequencing rule select candidate operation with the lowest priority value. Conversely, if selecting the operation with the highest priority, O_2 will be chosen for processing first. Different decision making would affect resource allocation, which directly impacts the optimisation objective values in this instance. Thus, a scheduling rule may exhibit different fitness values depending on the priority settings. GP will adaptively evolve scheduling rules based on their fitness values.

III. EXPERIMENT STUDIES

A. Design of Comparisons

We simulate various DFJSS scenarios to evaluate the quality of learned rules. Specifically, we use three utilisation levels (i.e., 0.75, 0.85 and 0.95), three commonly used minimised objectives i.e., mean-flowtime (Fmean), mean-tardiness (Tmean) and mean-weighted-tardiness (WTmean), and one fix due date factor (i.e., 1.5), resulting in nine examined minimisation scenarios. Additionally, for maxmisation scenarios, we combine three utilisation levels (i.e., 0.75, 0.85 and 0.95), three due date factor (i.e., 1.0, 1.2, and 1.5) and one maximised objective (i.e., profit), forming nine maximisation scenarios. Each scenario is represented as <objective, utilisation level, due date factor> such as <Fmean, 0.85, 1.5>. The settings remain consistent between training and test instances. The rules obtained after training are evaluated on 50 unseen instances, and the average objective value across these instances is reported as the rule's test performance. This paper aims to explore potential hidden differences in the evolutionary process of GP under different priority settings. Since including other methods would not directly support this investigation, only GP with different priority settings are included for comparison.

1) GP-low: the highest priority represented by the **lowest priority values** in scheduling rules learned by GP.
2) GP-high: the highest priority represented by the **highest priority values** in scheduling rules learned by GP.

TABLE II
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
$R^\star$	Revenue of a job

$\star$: for profit-objective only;

TABLE III
THE PARAMETER SETTINGS IN GP.

Parameter	Value
Population size	500
The number of elites for population	10
The number of generations	100
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
Parent selection	Tournament selection with size 5

B. Parameter Settings

GP individuals are composed of terminals and functions. Following the configuration in [15], the details of the terminal set are provided in Table II. The function set is set to $\{+, -, *,$ protected $/, max, min\}$. The protected "/" returns one if divided by zero. The other parameter settings of the GP algorithm are shown in Table III.

C. Quality of Learned Scheduling Heuristics

To evaluate the performance of the algorithms over 30 independent runs, we utilise Friedman's test and the Wilcoxon rank-sum test with a significance level of 0.05. The "Average Rank" reflects the mean ranking of each algorithm across all tested scenarios. In the following tables, "$\uparrow$", "$\downarrow$", and "$\approx$" indicate the corresponding result is significantly better than, worse than or similar to the compared algorithm.

TABLE IV
THE MEAN (STANDARD DEVIATION) OF **TIME-BASED OBJECTIVE VALUES** OF GP-HIGH AND GP-LOW IN NINE SCENARIOS ACCORDING TO 30 INDEPENDENT RUNS.

Scenarios	GP-high	GP-low
<Fmean, 0.75, 1.5>	**336.21(1.48)**	336.31(1.57)(≈)
<Fmean, 0.85, 1.5>	386.78(4.03)	**385.84(2.83)**(≈)
<Fmean, 0.95, 1.5>	553.18(7.18)	**551.90(6.42)**(≈)
<Tmean, 0.75, 1.5>	13.49(0.80)	**13.27(0.59)**(≈)
<Tmean, 0.85, 1.5>	40.50(1.95)	**39.95(0.16)**(≈)
<Tmean, 0.95, 1.5>	**177.10(5.27)**	177.54(4.52)(≈)
<WTmean, 0.75, 1.5>	28.03(2.35)	**27.58(1.95)**(≈)
<WTmean, 0.85, 1.5>	79.01(6.65)	**75.26(2.07)**(↑)
<WTmean, 0.95, 1.5>	297.57(8.36)	**293.39(6.66)**(↑)
Win / Draw / Lose	0 / 7 / 2	N/A
Average Rank	1.58	**1.42**

TABLE V
THE MEAN (STANDARD DEVIATION) OF **PROFIT-BASED OBJECTIVE VALUES** OF GP-HIGH AND GP-LOW IN NINE SCENARIOS ACCORDING TO 30 INDEPENDENT RUNS.

Scenarios	GP-high	GP-low
<Profit, 0.75, 1.0>	7217.70(28.03)	**7221.79(42.96)**(≈)
<Profit, 0.85, 1.0>	6802.44(56.92)	**6829.18(39.81)**(≈)
<Profit, 0.95, 1.0>	**5900.23(101.57)**	5874.53(131.92)(≈)
<Profit, 0.75, 1.2>	**7696.57(20.78)**	7695.01(23.17)(≈)
<Profit, 0.85, 1.2>	**7360.34(43.56)**	7357.00(51.41)(≈)
<Profit, 0.95, 1.2>	6518.90(104.06)	**6586.66(66.92)**(↑)
<Profit, 0.75, 1.5>	8039.90(14.84)	**8040.81(15.78)**(≈)
<Profit, 0.85, 1.5>	7828.52(43.37)	**7842.45(30.97)**(≈)
<Profit, 0.95, 1.5>	7186.38(94.84)	**7234.09(56.02)**(↑)
Win / Draw / Lose	0 / 7 / 2	N/A
Average Rank	1.55	**1.45**

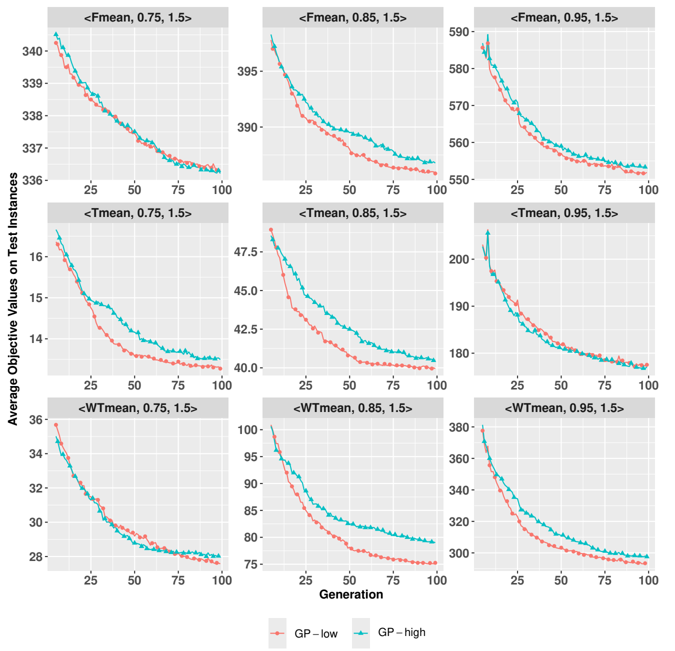

Fig. 3. Curves of average **time-based objective values** on test instances of GP-low and GP-high in nine time-based scenarios according to 30 independent runs.

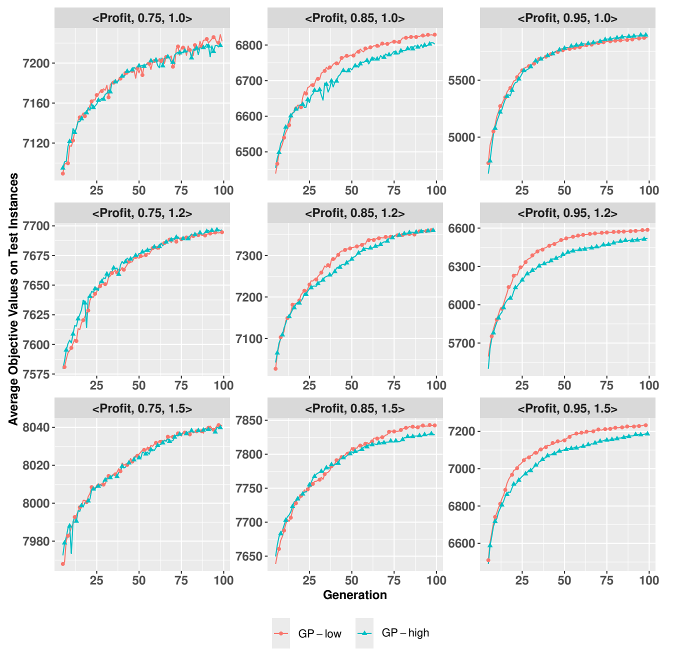

Fig. 4. Curves of average **profit-based objective values** on test instances of GP-low and GP-high in nine profit-based scenarios according to 30 independent runs.

Table IV and Table V shows the mean and standard deviation of time-based and profit-based objective values of GP-low and GP-high according to 30 independent runs, separately. The results show that two examined algorithms perform similarly in 7 out of 9 time-based scenarios and 7 out of 9 profit-based scenarios. This indicates that GP can adaptively learn suitable scheduling rules as different candidate selection changes. Although GP-low achieves slightly better average rank across scenarios, overall, their performance is similar. This is consistent to our expectation.

Fig. 3 and Fig. 4 shows the curves of average objective values of GP-low and GP-high in different scenarios according to 30 independent runs. Although the performance of two algorithms is similar, GP-low, which selects candidates with the lowest priority values, can find better solutions faster in both minimisation and maximisation scenarios, (i.e., <Fmean, 0.85, 1.5>, <Tmean, 0.75, 1.5>, <Tmean, 0.85, 1.5>, <WTmean, 0.85, 1.5>, <WTmean, 0.95, 1.5> and <Profit, 0.85, 1.0>), which suggests maximisation or minimisation objectives may be not related to priority settings. A possible reason for GP-low's ability to find better solutions faster is that the majority of features utilised in GP are designed to minimise (e.g., time and workload), which is more suitable for selecting candidates with the lowest priority values.

In summary, GP-low and GP-high achieve comparable performance in both time-based and profit-based scenarios, demonstrating the adaptive capability of GP to learn scheduling rules. Moreover, GP-low can generate better scheduling rules faster in the evolutionary process, which shows there are indeed differences between different priority settings during

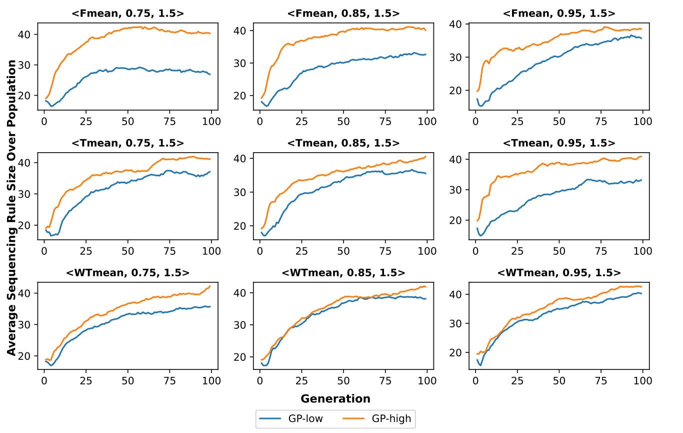

Fig. 5. Curves of average **sequencing rule** size over population on test instances of GP-low and GP-high in nine time-based scenarios according to 30 independent runs.

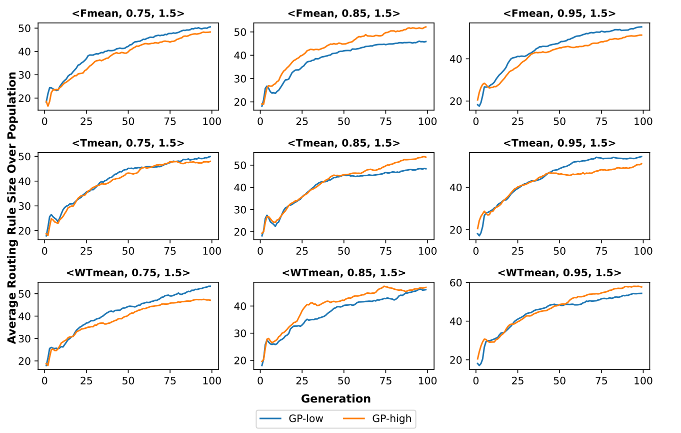

Fig. 6. Curves of average **routing rule** size over population on test instances of GP-low and GP-high in nine time-based scenarios according to 30 independent runs.

the process of evolution.

IV. Further Analyses

A. Rule Size Analyses

The size of a rule refers to the number of nodes within a GP individual. In DFJSS, there are two rules, i.e., the sequencing rule and the routing rule. Smaller rules are generally preferred for three main reasons. First, evaluating smaller rules is faster, which helps reduce the computational cost. Second, smaller rules often exhibit better generalisation capabilities. Lastly, they are easier for decision-makers to interpret compared with large rules.

Fig. 5 and Fig. 6 illustrate the curves of average sequencing and routing rule size over population on test instances of GP-low and GP-high in time-based scenarios according to 30 independent runs, respectively. An interesting observation is that, in almost all scenarios, the sequencing rule size of GP-low is consistently smaller than that of GP-high. However, the pattern does not extend to the routing rules. According to the [16], routing rules play a more significant role than sequencing rules and are inherently larger in size. As a result, routing rules tend to remain relatively unaffected. In addition, the increase in sequencing rule size for GP-high can be attributed to feature design. Most features are related to time and workload, are intended to be minimised. If these features contribute to GP-high, they should be inverted accordingly. For instance, operations often prefer machines with the shortest processing time (PT). A direct scheduling rule in such cases is PT, corresponding to selecting a machine with the lowest priority value (i.e., PT). However, if GP-high aims to achieve a similar result while selecting candidates with the highest priority values, the rule needs to be adjusted. This could involve transforming the PT feature by subtraction or division, leading to a more complex rule like $1 - PT$. The size of this new rule would increase to 3 (i.e., incorporating one node 1 and one subtraction/division function) compared to the original

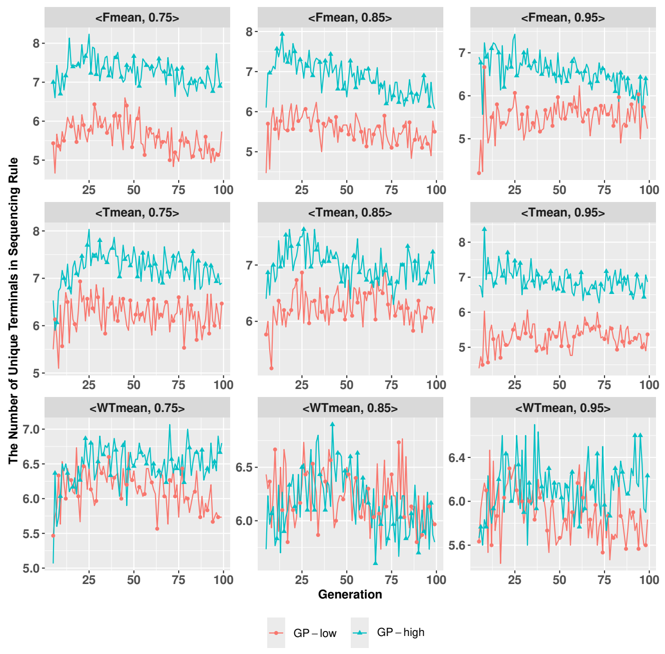

Fig. 7. Curves of mean number of unique features in the best sequencing rule evolved by GP-low and GP-high in nine time-based scenarios according to 30 independent runs.

size of 1. The increase in the number of unrelated features in sequencing rule in Section IV-B also verifies this hypothesis.

In summary, although GP-low and GP-high can achieve similar performance, GP-low can learn smaller sequencing rules for DFJSS.

B. The Number of Unique Features in the Best Sequencing 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. According to [17], we know routing

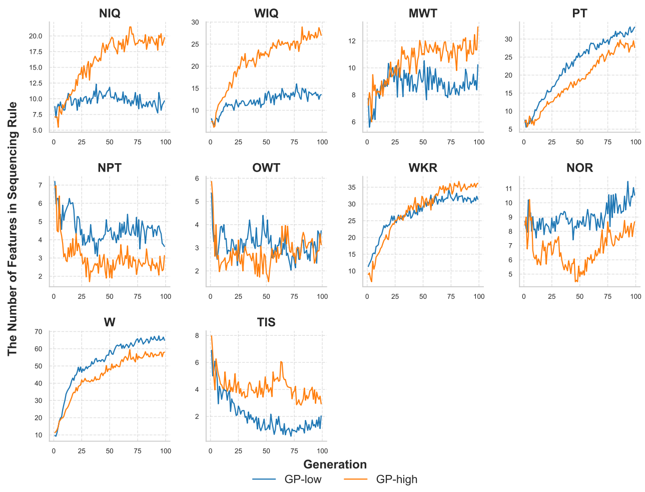

Fig. 8. Curves of mean occurrence of features in five best sequencing rules evolved by GP-low and GP-high in the scenario <WTmean, 0.95, 1.5> according to 30 independent runs.

rules typically have a larger number of unique features, as routing decisions are more critical and require a broader set of features to construct effectively. Thus, the number of unique features in routing rule is often unaffected, always large, we only choose sequencing rule for analyses.

Fig. 7 shows the curve of average number of unique features in the best sequencing rule in time-based scenarios. Specifically, the number of unique terminals in the sequencing rules of GP-low is smaller than that of GP-high. The underlying reason for this phenomenon aligns with the explanation provided in Section IV-A. To achieve comparable performance when selecting candidates with the highest priority values, a rule should adapt by incorporating functions such as subtraction or division. This adaptation increases the usage of unique features, contributing to the observed differences.

Fig. 8 shows the curve of mean occurrence of features in five best sequencing rules evolved by GP-low and GP-high in the scenario <WTmean, 0.95, 1.5>. The results show that some features, such as NIQ (i.e., the number of operations in a machine's waiting queue), WIQ (i.e., the workload in a machine's waiting queue), and MWT (i.e., a machine's waiting time), appear more frequently in GP-high compared to GP-low. These features are related to machines and can be considered as constants in sequencing rules because all the operations prioritised by a sequencing rule are in a queue of an idle machine and have the same values for machine related features. This indicates that GP-high incorporates some unimportant features for distinguishing the priorities of operations, which is one reason for the increase of the number of sequencing rules, as shown in Fig. 5, and the rise in the number of unique features observed in Fig. 7. This finding aligns with our earlier assumption: GP-high needs to adaptively adjust node positions, such as changing PT to $1 - PT$, where these features effectively act like the constant 1.

In brief, different priority settings influence the number of unique and utilized features. The scheduling rules learned by GP-low incorporate fewer unique and used features, while GP-high involves more unimportant features.

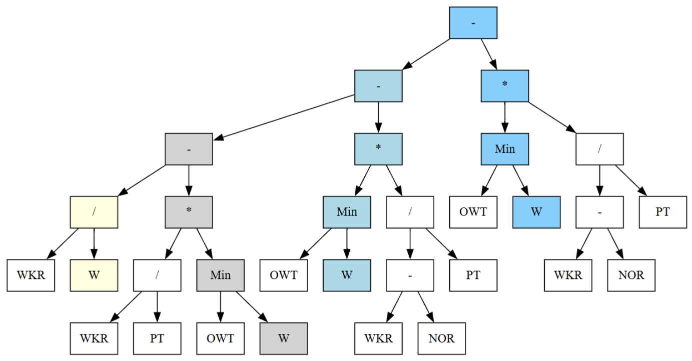

Fig. 9. One of best sequencing rule learned by GP-low in the scenario <WTmean, 0.95, 1.5>.

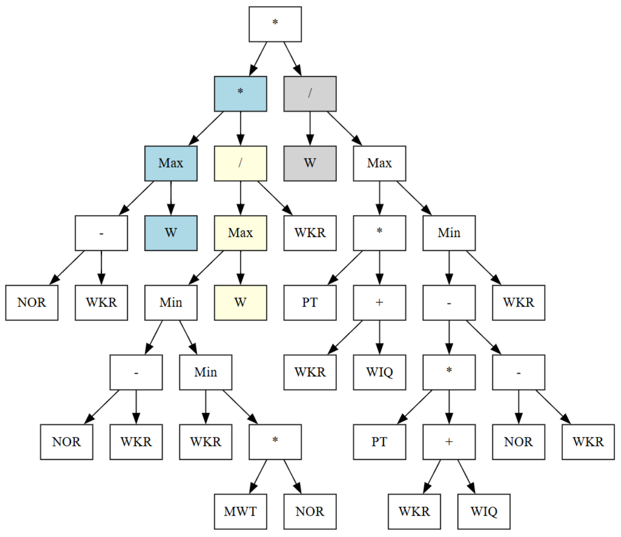

Fig. 10. One of best sequencing rule learned by GP-high in the scenario <WTmean, 0.95, 1.5>.

C. Insight of Learned Scheduling Rules

In order to further analyse how GP-low and GP-high adaptively learn scheduling rules, this section examines the sequencing rules evolved by GP-low and GP-high. Since all scenarios have the same pattern, so we randomly choose one (i.e., <WTmean, 0.95, 1.5>) for analyses.

Fig. 9 and Fig. 10 show one of the best sequencing rules evolved by GP-low and GP-high, respectively. Focusing on W, an interesting pattern emerges, all terminal nodes W appear on the right side of subtrees in GP-low. Furthermore, the parent nodes of terminal W tend to be division or subtraction. Even when the immediate parent node is not division or subtraction, one of the parent nodes in the hierarchy will always include division or subtraction. This occurs because W is inversely proportional to the priority function when selecting candidates with the lowest priority values. Conversely, weight should be directly proportional to the priority function when selecting

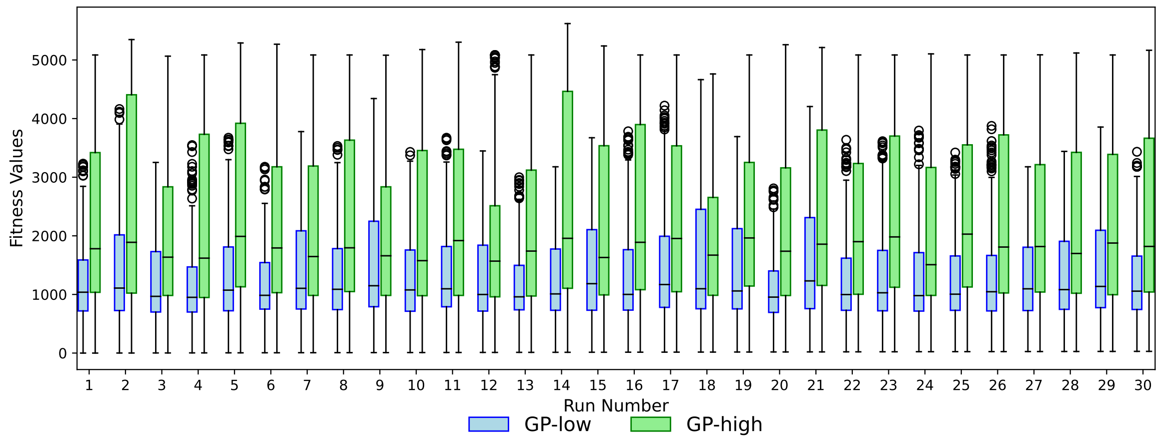

Fig. 11. Fitness values of top 200 individuals in the initialised population in the scenario <WTmean, 0.95, 1.5> across 30 runs during training stage.

candidates with the highest priority values. Consequently in GP-high, W often appears on the left side of function nodes. For instance, after the Max function, W is positioned to the left of multiplication or division. These observations demonstrate that GP adjusts the positions of terminal nodes and function nodes to evolve effective scheduling rules based on different priority settings.

Additionally, sequencing rules evolved by GP-low and GP-high are simplified for further analysis. The sequencing rule learned by GP-low as shown in Fig. 9 can be expressed as:

$$S1 = WKR * \left(\frac{1}{W} - \frac{W}{PT}\right) \tag{1}$$

Since GP-low selects candidates with the lowest priority value, this sequencing rule prioritises operations with a smaller amount of work remaining for a job (WKR), a higher weight (W), and a smaller processing time (PT).

The corresponding sequencing rule learned by GP-high as shown in Fig. 10 can be simplified as:

$$S2 = \frac{W^3}{WKR * PT * (WKR + WIQ)} \tag{2}$$

GP-high, which selects candidates with the highest priority value, tends to prioritise operations with smaller WKR, smaller PT, and higher W, which is the same as the rule learned by GP-low. It is worth noting that WIQ (i.e., workload in a machine's queue) acts as a constant in this context. This observation aligns with the findings that some unimportant features appear in GP-high as shown in Fig. 8.

Overall, while the scheduling rules learned by GP-low and GP-high appear to differ, they ultimately prioritise operations with similar features. These visualisations show that GP can adjust the positions of terminals to adaptively learn scheduling rule learning process of GP.

D. *Fitness Values of Initialised Individuals*

Since one scheduling rule may have different fitness due to different priority settings, and the scheduling rules learned by GP-low and GP-high are only the same in the initialised population, we analyse the fitness values of initialised population to observe whether scheduling rules exhibit bias toward a particular priority settings.

In this section, we analyse the fitness values of top 200 individuals in the initialised population in the scenario <WTmean, 0.95, 1.5> across 30 independent runs during training stage, which in shown in Fig. 11.

Notably, scheduling rules learned by GP-low achieve better fitness values in the initialised population across all runs. This indicates that selecting candidates with the lowest priority values is more suitable for randomly generated scheduling rules. A possible reason for the initial bias toward lower priority values is that most of the features used tend to be minimised, making the initialised individuals more suitable for selecting lower priority values. Consequently, their performance is better in GP-low than in GP-high. Moreover, this phenomenon explains why GP-low exhibits faster convergence in Fig. 3, as it benefits from a better-initialised population. However, as the evolution progresses, the performance of GP-low and GP-high tends to become similar.

E. *Priority Value Distribution*

Due to different range of terminals, GP-low and GP-high may assign varying priority values to machines or operations. A wide range of priority distribution can better distinguish candidates to select the suitable option. Fig. 12 and Fig. 13 present some operations' priority value distribution assigned by one of the best sequencing rule evolved by GP-low and GP-high in one training instance under the scenario <WTmean, 0.95, 1.5>.

Both distributions exhibit a long-tail pattern. For GP-low, most priority values are concentrated at smaller values, whereas for GP-high, they are skewed toward larger values. Although most data points are close to zero, the range of priority values assigned by GP-low is significantly larger than that of GP-high. A larger range provides a better ability to distinguish candidates, this might be a possible reason for the faster convergence observed in GP-low.

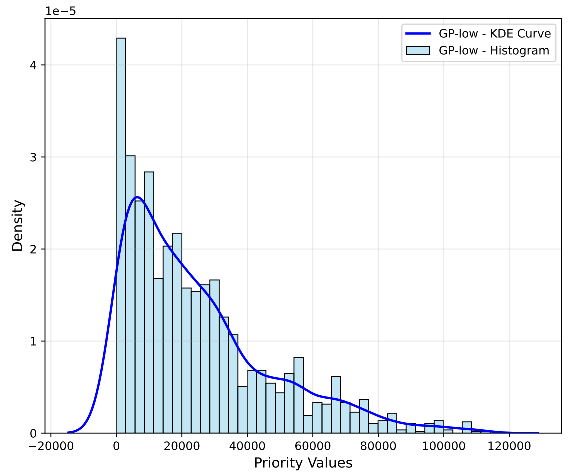

Fig. 12. Priority value distribution assigned by one of the best sequencing rule evolved by GP-low in one training instance under the scenario <WTmean, 0.95, 1.5>.

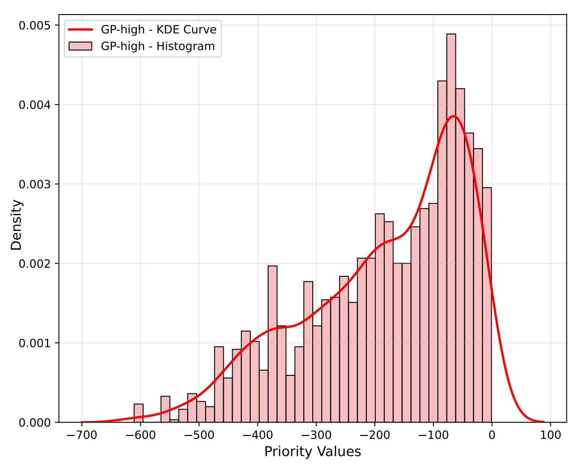

Fig. 13. Priority value distribution assigned by one of the best sequencing rule evolved by GP-high in one training instance under the scenario <WTmean, 0.95, 1.5>.

V. Conclusions and Future Work

The goal of this paper is to investigate and analyse differences between scheduling rules learned by GP when selecting candidates with either the highest or lowest priority values during the evolutionary stage. Additionally, it aims to understand how GP adaptively learns scheduling rules. The goal has been successfully achieved by comparing GP-evolved rules with different priority settings from various perspectives.

The results show that, first, consistent with consensus, GP-low and GP-high perform similarly in the end, but GP-low can generate better scheduling rules faster during evolution. Second, sequencing rules in GP-low are smaller and use fewer unique features. Third, rule visualisation shows how GP adjusts node positions to evolve promising rules. Furthermore, initialised scheduling rules have the bias to select candidates with the lowest priority values. Finally, GP-low exhibits a wider distribution of priority values. These findings suggest GP-low is the better choice for our simulated DFJSS problem.

Some interesting directions can be further investigated in the near future. It seems terminals appear to influence results across different priority settings. More investigations to terminal can be conducted.

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, 2023.

[2] L. Zhu, F. Zhang, X. Zhu, K. Chen, and M. Zhang, "Phenotype and genotype based sample aware surrogate-assisted genetic programming in dynamic flexible job shop scheduling," *IEEE Transactions on Artificial Intelligence*, 2025, DOI:10.1109/TAI.2025.3562161.

[3] S. A. Murad, A. J. M. Muzahid, Z. R. M. Azmi, M. I. Hoque, and M. Kowsher, "A review on job scheduling technique in cloud computing and priority rule based intelligent framework," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 6, pp. 2309–2331, 2022.

[4] J.-P. Huang, L. Gao, and X.-Y. Li, "A hierarchical multi-action deep reinforcement learning method for dynamic distributed job-shop scheduling problem with job arrivals," *IEEE Transactions on Automation Science and Engineering*, 2024.

[5] K. Akram, M. U. Bhutta, S. I. Butt, S. H. I. Jaffery, M. Khan, A. Z. Khan, and Z. Faraz, "A pareto-optimality based black widow spider algorithm for energy efficient flexible job shop scheduling problem considering new job insertion," *Applied Soft Computing*, vol. 164, p. 111937, 2024.

[6] S. S. Panwalkar and W. Iskander, "A survey of scheduling rules," *Operations research*, vol. 25, no. 1, pp. 45–61, 1977.

[7] 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.

[8] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A classification of hyper-heuristic approaches: revisited," *Handbook of metaheuristics*, pp. 453–477, 2019.

[9] H. Guo, J. Liu, Y. Wang, and C. Zhuang, "An improved genetic programming hyper-heuristic for the dynamic flexible job shop scheduling problem with reconfigurable manufacturing cells," *Journal of Manufacturing Systems*, vol. 74, pp. 252–263, 2024.

[10] O. Rose, "The shortest processing time first dispatch rule and some variants in semiconductor manufacturing," in *Proceeding of the Winter Simulation Conference*, vol. 2. IEEE, 2001, pp. 1220–1224.

[11] N. Ye, Z. Yang, Y.-C. Lai, and T. Farley, "Enhancing router qos through job scheduling with weighted shortest processing time-adjusted," *Computers & operations research*, vol. 32, no. 9, pp. 2255–2269, 2005.

[12] 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.

[13] S. Geramipour, G. Moslehi, and M. Reisi-Nafchi, "Maximizing the profit in customer's order acceptance and scheduling problem with weighted tardiness penalty," *Journal of the Operational Research Society*, vol. 68, no. 1, pp. 89–101, 2017.

[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] 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.

[16] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Importance-aware genetic programming for automated scheduling heuristics learning in dynamic flexible job shop scheduling," in *Proceedings of the International Conference on Parallel Problem Solving from Nature*. Springer, 2022, pp. 48–62.

[17] L. Zhu, F. Zhang, M. Feng, K. Chen, X. Zhu, and M. Zhang, "Crossover operators between multiple scheduling heuristics with genetic programming for dynamic flexible job shop scheduling," in *IEEE Congress on Evolutionary Computation*. IEEE, 2024, pp. 1–8.