





Guided Subtree Selection for Genetic Operators in Genetic Programming for Dynamic Flexible Job Shop Scheduling

Fangfang Zhang¹ , Yi Mei¹ , Su Nguyen² , and Mengjie Zhang¹ 

¹ School of Engineering and Computer Science, Victoria University of Wellington,
PO BOX 600, Wellington 6140, 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

Abstract. Dynamic flexible job shop scheduling (DFJSS) has been widely studied in both academia and industry. Both machine assignment and operation sequencing decisions need to be made simultaneously as an operation can be processed by a set of machines in DFJSS. Using scheduling heuristics to solve the DFJSS problems becomes an effective way due to its efficiency and simplicity. Genetic programming (GP) has been successfully applied to evolve scheduling heuristics for job shop scheduling automatically. However, the subtrees of the selected parents are randomly chosen in traditional GP for crossover and mutation, which may not be sufficiently effective, especially in a huge search space. This paper proposes new strategies to guide the subtree selection rather than picking them randomly. To be specific, the occurrences of features are used to measure the importance of each subtree of the selected parents. The probability to select a subtree is based on its importance and the type of genetic operators. This paper examines the proposed algorithm on six DFJSS scenarios. The results show that the proposed GP algorithm with the guided subtree selection for crossover can converge faster and achieve significantly better performance than its counterpart in half of the scenarios while no worse in all other scenarios without increasing the computational time.

Keywords: Guided subtree selection · Scheduling heuristic · Dynamic flexible job shop scheduling · Genetic programming.

1 Introduction

Job shop scheduling (JSS) [1] is an important combinatorial optimisation problem that can be applied to almost all areas of our lives such as manufacturing [2] and cloud computing [3]. The task in JSS is to process a number of jobs by a set of machines. Each job has a sequence of operations. The goal of JSS is to find a good schedule to complete the processing task. An effective scheduling

decision-making scheme is the key to enhancing the competitiveness of a modern enterprise. Flexible JSS (FJSS) [4], as a variant of JSS, better reflects the real-world applications than ordinary JSS. In FJSS, one operation can be processed on a set of machines. Except for choosing an operation as the next operation to be processed by an idle machine (*operation sequencing*), we need to assign an operation to a particular machine (*machine assignment*). These two decisions need to be made simultaneously. In addition, many practical scheduling problems are changing over time, for example, due to new job arrivals [5,6,7]. Dynamic FJSS (DFJSS) is to consider flexible JSS under dynamic environments.

Scheduling heuristics such as dispatching rules [8] are widely used to handle DFJSS. A scheduling heuristic is a heuristic that works like a priority function to evaluate the priorities of operations and machines. To be specific, in DFJSS, a machine that has the highest priority value based on the *routing rule* (i.e. routing scheduling heuristic) will be assigned a job to be processed. An operation with the highest priority value based on the *sequencing rule* (i.e. sequencing scheduling heuristic) will be chosen as the next operation to be processed. There are some rules such as SPT (i.e. shortest processing time) and WIQ (i.e. the workload in the queue of a machine) which have been identified as effective rules for JSS. However, they are manually designed by experts, which is time-consuming. In practice, it is hard to manually design effective rules due to the complexity and diversity of the investigated job shop environments.

Genetic programming (GP) [9], as a hyper-heuristic (GPHH) method, has been successfully applied to automatically evolve scheduling heuristic for JSS [10,11]. GP uses crossover, mutation and reproduction to generate offspring for the next generation. In a typical subtree-based crossover, offspring are created by *swapping the subtrees* of the parents. On the other hand, mutation is generally to maintain diversity within the population and prevent premature convergence. In the common subtree-based mutation, an individual (i.e. parent) is selected, and an offspring is generated by *replacing one of its subtrees* with a new randomly generated subtree.

In traditional GP, subtrees (i.e. function nodes) are randomly chosen to generate individuals. However, the importance of subtrees in each individual can be different. Some subtrees are redundant or less important and removing them might not affect the fitness of an individual too much. On the other hand, some subtrees play important roles for an individual, and losing them will cause considerable loss to the fitness. It may not be an effective way to randomly select subtrees without considering the importance of subtrees. To this end, this paper proposes subtree selection strategies for crossover and mutation to help GP improve the effectiveness of generating new offspring.

The overall goal of this paper is to develop novel guided subtree selection strategies based on the occurrence of features for crossover and mutation to help GP find more effective scheduling heuristics for DFJSS efficiently. The proposed algorithms are expected to speed up the convergence of GP and find effective rules in a shorter time. In particular, this paper has the following research objectives:

- Develop guided subtree selection strategies both for crossover and mutation with the information of the occurrence of features to improve the effectiveness of the evolutionary process.
- Verify the effectiveness and efficiency of the proposed GP algorithm with the guided subtree selection strategy by comparing its performance and convergence curve with the baseline GP counterpart.
- Analyse how the subtree selection strategy affects the evolutionary process of GP.

2 Background

2.1 Dynamic Flexible Job Shop Scheduling

In FJSS problem [12], n jobs $J = \{J_1, J_2, \dots, J_n\}$ need to be processed by m machines $M = \{M_1, M_2, \dots, M_m\}$. Each job J_j has an arrival time $at(J_i)$ and a sequence of operations $O_j = (O_{j1}, O_{j2}, \dots, O_{ji})$. Each operation O_{ji} can only be processed by one of its optional machines $\pi(O_{ji})$ and its processing time $\delta(O_{ji})$ depends on the machine that processes it. It indicates that there are two decisions which are routing decision and sequencing decision in FJSS. In DFJSS, not only the two decisions need to be made simultaneously, but also the dynamic events are necessary to be taken into account when making schedules. This paper focuses on one dynamic event (i.e. continuously arriving new jobs). That is, the information of a job is unknown until its arrival time.

2.2 Genetic Programming Hyper-heuristic for DFJSS

A hyper-heuristic [13] is a heuristic search method that seeks to select or generate heuristics to efficiently solve hard computational search problems. The unique characteristic is that the search space of hyper-heuristic is heuristics instead of solutions. Hyper-heuristic is often incorporated with machine learning techniques to achieve its goal.

GP, as a hyper-heuristic method [14], has been successfully applied to more informative scheduling heuristics for combinatorial optimisation problems such as packing [15,16], timetabling [17], arc routing [18], and JSS [19,20,21,22]. Scheduling heuristics, including routing and sequencing rules, are needed in DFJSS in our research. To follow the sequence constraint of operations of a job, we only start to allocate an operation when it becomes a *ready operation*. There are two sources of ready operations. One is the first operation of a job. The second is the operation that its proceeding operation is just finished. Once an operation becomes a ready operation (*routing decision point*), it will be allocated to the machine by the routing rule. When a machine becomes idle, and its queue is not empty (*sequencing decision point*), the sequencing rule will be triggered to choose the next operation to be processed.

Although GP has been successfully applied to DFJSS [19,20], to the best of our knowledge, little research has been conducted on genetic operators to

improve the effectiveness of generating offspring. To this end, this paper aims to propose subtree selection strategies for both crossover and mutation to help GP evolve more effective scheduling heuristics for DFJSS.

3 The Proposed GP with Subtree Selection

Fig. 1 shows the flowchart of the proposed algorithm. The main process is the same as the traditional GP. There are three different parts. After evaluating all the individuals in the population, the occurrence of each feature is counted based on promising individuals. The occurrence information of features is used to calculate the importance of subtrees of the selected parents. During the evolutionary process, crossover and mutation are conducted based on the proposed corresponding subtree selection strategies. In this way, when generating new offspring by crossover and mutation, the subtrees are selected with guidance.

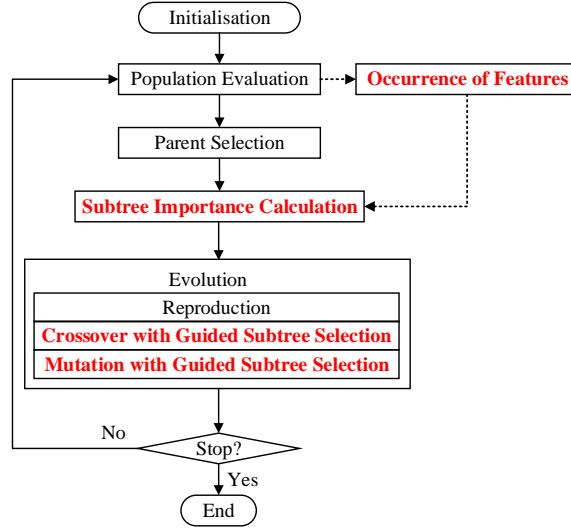


Fig. 1: The flowchart of the proposed algorithm.

According to the proposed algorithm framework, the three research questions in this paper are how to extract feature information to assess the importance of subtrees, how to measure the importance of subtrees, and how to apply the subtree importance information to crossover and mutation. These three questions are studied in the following three sections, separately.

3.1 The Occurrences of Features

An advantage of GP is that it can automatically select important features to build individuals. The features of individuals with good fitness are more likely to

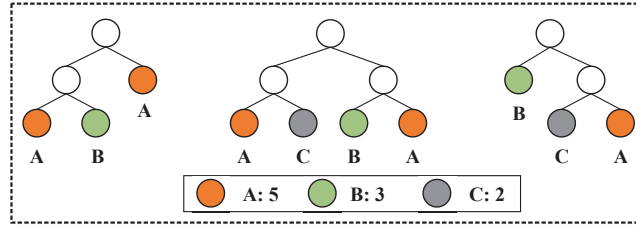


Fig. 2: The occurrence of features in the top three individuals.

be important features. On the other hand, the individuals that contain important features are more likely to be promising individuals. This means that the features involved in promising individuals can be used to measure the importance of subtrees.

In this paper, the occurrence of features in the top ten individuals is further used to assess the importance of subtrees. Our preliminary studies show that top ten individuals tend to have promising fitness which is good for detecting feature characteristics. Another advantage of using the occurrence information of features is that we do not need to put too much extra effort to obtain useful information since the information is already generated during the evolutionary process.

Fig. 2 shows an example of how to extract feature occurrence information based on three individuals. These three individuals contain different numbers of features and have different structures. Assuming that they are top three individuals in the population based on the fitness. According to the three individuals, the occurrence of features in all three individuals is counted. The occurrences of feature A, B, and C are 5, 3, and 2, respectively. This information will be used to measure the importance of subtrees in an individual.

3.2 The Importance of Subtrees

An individual (i.e. a tree) can be considered to be composed of multiple subtrees. After a function node is selected, the subtree is determined. The importance of subtrees is measured from bottom to top, and this paper uses the concept *score* to indicate the importance of a subtree. Each feature has its occurrence information at the bottom level of an individual, and the score of their parent node (i.e. the importance of subtree) is set as the average occurrence number of its child nodes. Assuming that importance (i.e. occurrence) of feature A, B and C are ranked as $A > B > C$. If only considering the simplest subtrees (i.e. depth is two) and only take two features, there will be three possible combinations for the subtree which are A and B, A and C, and B and C. The importance of the subtrees should be ranked as $subtree(A, B) > subtree(A, C) > subtree(B, C)$.

Fig. 3 shows an example of how to measure the importance of each subtree for an individual. For example, the $subtree_1$ (i.e. in the bottom-left corner) contains two features (i.e. A and B), the score of their parent node is set as 4 (i.e. $5 +$

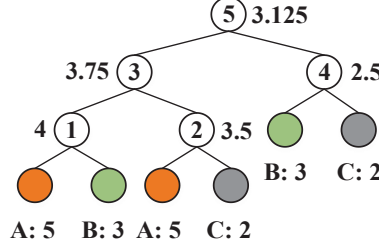


Fig. 3: The importance (i.e. score) of subtrees in an individual.

3) / 2). The importance of *subtree*₃ is assigned as the average scores of its two subtrees (i.e. *subtree*₁ and *subtree*₂). By analogy, all score of subtrees will be assigned, as shown in Fig. 3.

Taking the subtrees that the depth is two into consideration, there are three subtrees (i.e. indicated by subtree 1, 2 and 4), whose importance are marked as 4, 3.5 and 2.5, respectively. The importance of subtree 1, 2 and 4 are ranked as *subtree*₁ > *subtree*₂ > *subtree*₄, which is consistent with the importance measurement design. When looking at all the subtrees, the importance rank of all subtrees in this individual is *subtree*₁ > *subtree*₃ > *subtree*₂ > *subtree*₅ > *subtree*₄.

3.3 Subtree Selection

Based on the importance of subtrees, the probability of the subtrees that will be selected can be calculated. There are two different techniques to calculate the probability for different purposes (i.e. one for mutation, and the other for both crossover and mutation). The probability is designed proportionally to the scores.

Fig. 4 shows the two different techniques to calculate the probability of each subtree in an individual. Fig. 4 (a) shows the technique that tends to choose the important subtree (i.e. the subtree with a larger score, the higher the probability it will be selected). Let us continue with the previous example in Fig. 3, there are five subtrees with score [3.125, 3.75, 2.5, 4, 3.5]. If we prefer to choose the important subtree, the larger the score of the subtree, the higher the probability it will have. First, we sum up the total score (i.e. $16.875 = 3.125 + 3.75 + 2.5 + 4 + 3.5$). Then, the probability of subtrees is assigned as [0.185, 0.222, 0.148, 0.237, 0.207] (i.e. $[3.125/16.875, 3.75/16.875, 2.5/16.875, 4/16.875, 3.5/16.875]$). The rank of probability of subtrees is *subtree*₁ > *subtree*₃ > *subtree*₂ > *subtree*₅ > *subtree*₄.

Fig. 4 (b) shows the technique that tends to choose the unimportant subtree (i.e. the subtree with a larger score, the lower the probability it will be selected). If we prefer to choose the unimportant subtree, the larger the score of the subtree, the lower the probability it will have. Thus, the score is converted to $[1/3.125, 1/3.75, 1/2.5, 1/4, 1/3.5]$ first. Then, we sum up the total score and get the final

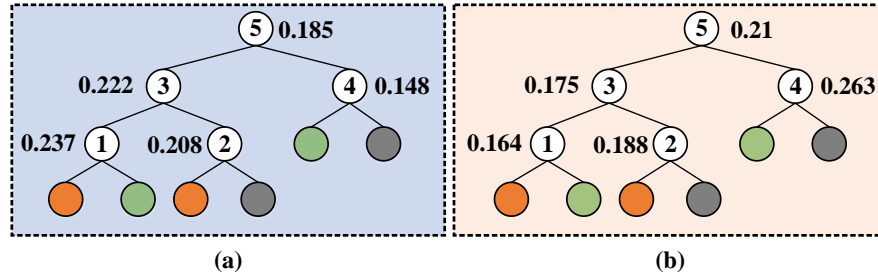


Fig. 4: Two different ways to calculate the probability of each subtree for an individual. (a) tends to choose important subtree while (b) tends to choose unimportant subtree.

probability as we just mentioned. The probabilities of subtrees are shown beside the function nodes. The rank of probability of subtrees is $subtree_4 > subtree_5 > subtree_2 > subtree_3 > subtree_1$.

Crossover with Guided Subtree Selection. For the crossover, there are two parents (i.e. $parent_1$ and $parent_2$) which are both promising individuals that are selected as parents. Without loss of generality, this paper assumes that $parent_1$ is no worse than $parent_2$. The unimportant subtree from $parent_1$ is expected to be swapped with an important subtree from $parent_2$ to make $parent_1$ an even better individual. Therefore, for $parent_1$, the larger the score of the subtree, the lower the probability it will have. For $parent_2$, the larger the score of the subtree, the higher the probability it will have.

Mutation with Guided Subtree Selection. For mutation, we expect to make the parent produce a better individual by replacing unimportant subtree with a newly generated subtree. We prefer to choose an unimportant subtree, and a larger score of the subtree leads to a lower probability it will be chosen.

3.4 Summary

The purpose of the proposed algorithms is to improve the effectiveness of crossover and mutation by introducing subtree selection strategies instead of choosing subtrees randomly. The occurrence information of features is utilised to measure the importance of subtrees. Then, the subtrees importance information is used to determine the probability that the subtrees will be selected along with the characteristics of crossover and mutation.

4 Experiment Design

To investigate the *effectiveness* of the proposed subtree selection strategies for crossover and mutation, a set of experiments have been conducted. In this section, the experiment design is shown in detail.

4.1 Simulation Model

Assuming that there are 5000 jobs need to be processed by ten machines. The importance of jobs might be different, which are indicated by weights. The weights of 20%, 60%, and 20% of jobs are set as one, two and four, respectively. The number of operations of each job varies by 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]. The number of candidate machines for an operation follows a uniform discrete distribution between one and ten.

In each problem instance, jobs arrive stochastically according to a Poisson process with rate λ . To improve the generalisation ability of the evolved rules for DFJSS problems, the seeds used to stochastically generate the jobs are rotated in the training process at each generation. In addition, in order to make sure the accuracy of the collected data, a warm-up period of 1000 jobs is used.

4.2 Parameter Settings

In our experiment, the terminal and function set are shown in Table 1. The “/” operator is protected division, returning one if divided by zero. The other parameter settings of GP are shown in Table 2.

Table 1: 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	The waiting time of an operation
Job-related	WKR	The 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

4.3 Comparison Design

Four algorithms are taken into the comparison in this paper. The cooperative co-evolution genetic programming (CCGP) [5] which can be used to evolve routing rule and sequencing rule simultaneously, is selected as the baseline algorithm. Our proposed algorithm, which incorporates subtree selection strategy into the

Table 2: The parameter setting of GP.

Parameter	Value
Number of subpopulations	2
Subpopulation size	512
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%

crossover, is named as CCGP^c (i.e. choose subtrees for crossover). The algorithm that incorporates subtree selection into the mutation (i.e. choose subtrees for mutation) is called CCGP^m. The proposed algorithm, which incorporates subtree selection by both crossover and mutation, is named as CCGP^{cm}. CCGP^c, CCGP^m and CCGP^{cm} are compared with CCGP, respectively.

The proposed algorithms are tested on *six different scenarios*. The scenarios consist of three objectives (i.e. max flowtime, mean flowtime, and mean weighted flowtime) and two utilisation levels (i.e. 0.85 and 0.95) [20]. For the sake of convenience, Fmax, Fmean, and WFmean are used to indicate max flowtime, mean flowtime, and mean weighted flowtime, respectively. The evolved best rule at each generation is tested on 50 different test instances, and the mean objective value of them is reported as the objective value of this best rule. This aims to guarantee the accuracy of measuring the performance.

5 Results and Discussions

Thirty independent runs are conducted for the comparison. Wilcoxon rank-sum test with a significance level of 0.05 is used to verify the performance of proposed algorithms. In the following results, “-” and “+” indicate the corresponding result is significantly better or worse than its counterpart. If there is no mark there, that means the performance between them is similar.

5.1 Performance of Evolved Rules

Table 3 shows the mean and standard deviation of the objective value of the four algorithms over 30 independent runs for six DFJSS scenarios. CCGP^c performs significantly better than CCGP for three scenarios (i.e. <Fmean,0.85>, <WFmean,0.85> and <WFmean,0.95>). For the remaining three scenarios, CCGP^c performs as well as the CCGP. In scenario <Fmax,0.85>, although CCGP^c does not achieve significantly better performance than that of CCGP,

Table 3: The mean (standard deviation) of the objective value of CCGP, CCGP^c, CCGP^m, and CCGP^{cm} over 30 independent runs for six DFJSS scenarios.

Scenario	CCGP	CCGP ^c	CCGP ^m	CCGP ^{cm}
<Fmax,0.85>	1211.84(35.27)	1211.68(30.21)	1217.81(28.24)	1215.76(26.41)
<Fmax,0.95>	1942.06(31.70)	1944.84(31.52)	1936.62(23.11)	1955.04(56.65)
<Fmean,0.85>	386.07(3.53)	384.80(1.67)(-)	384.40(2.02)(-)	384.88(1.60)(-)
<Fmean,0.95>	550.99(5.28)	551.94(4.94)	551.20(4.70)	551.12(3.87)
<WFmean,0.85>	832.46(7.25)	829.70(4.83)(-)	832.03(7.33)	830.14(4.26)
<WFmean,0.95>	1110.04(10.82)	1107.59(12.49)(-)	1109.44(12.33)	1107.76(8.08)(-)

the mean and standard deviation are smaller than that of CCGP (i.e. still better). However, CCGP^m performs significantly better than that of CCGP only in scenario <Fmean,0.85> and achieves better performance in scenario <Fmax,0.95>. In general, it seems like it is not that effective to apply subtree selection strategy to mutation as it does not get better results in most scenarios. The effectiveness of CCGP^{cm} is similar to CCGP^c. Its performance might be mainly due to the role played by applying subtree selection strategy into the crossover.

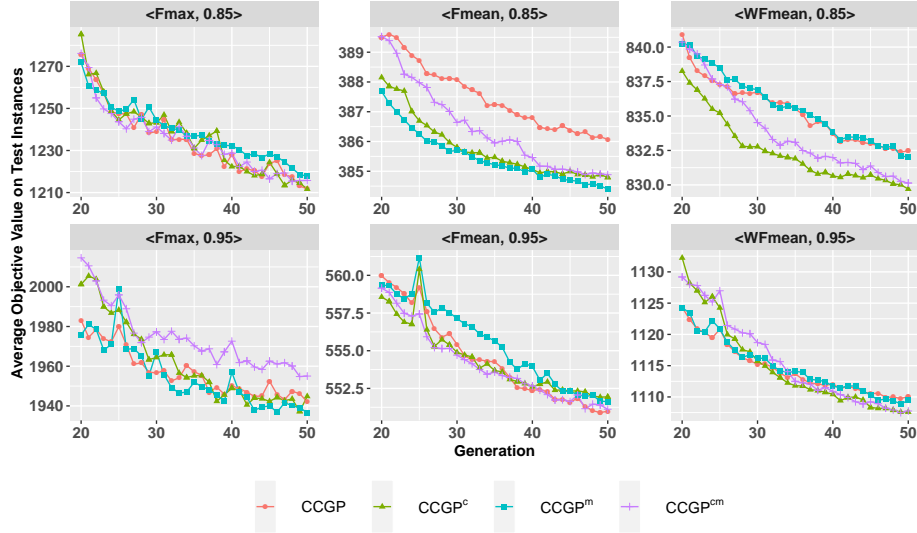
Fig. 5: The convergence curves of CCGP, CCGP^c, CCGP^m, and CCGP^{cm} from generation 20 to generation 50 in six scenarios.

Fig. 5 shows the convergence curves of the average objective value on the test instances of the four algorithms. To better show the performance of the proposed algorithms, only the curves between generation 20 and 50 are shown in Fig. 5. Except for max-flowtime related scenarios (i.e. <Fmax,0.85> and

$\langle F_{\max}, 0.95 \rangle$) and scenario $\langle F_{\text{mean}}, 0.95 \rangle$, the three proposed algorithms (i.e. CCGP^c , CCGP^m , and CCGP^{cm}) can achieve better performance than that of CCGP. In $\langle F_{\text{mean}}, 0.85 \rangle$, all the proposed three algorithms show their advantages in both convergence speed and final performance, especially CCGP^m . In scenario $\langle \text{WFmean}, 0.85 \rangle$, CCGP^c has the best convergence speed and performance. In scenario $\langle \text{WFmean}, 0.95 \rangle$, CCGP^m convergence faster than CCGP^c before generation 30 roughly, however, it loses to CCGP^c after generation 30. Finally, CCGP^c achieves better performance than that of CCGP^m . For minimising max-flowtime, the proposed three algorithms have no obvious advantages. It might be because max-flowtime is more sensitive to the worst case, which is more complex and hard to optimise.

Summary. Based on the results, CCGP^c is the most promising algorithm which shows the effectiveness of improving the crossover operator by subtree selection strategy. CCGP^m is not as promising as CCGP^c . One possible reason is that the mutation rate is low and can not affect the evolutionary process too much. That might also be the reason why the performance of CCGP^{cm} is similar to that of CCGP^c . The other possible reason is that mutation aims to maintain the diversity of the population, and it is better not to guide its direction.

5.2 The Probability Difference

The main idea in this paper is to differentiate the probability of subtrees to be chosen instead of choosing subtrees randomly. The probability difference ($P_s - P_u$) is defined as the difference between the assigned probability (P_s) and the uniform probability (P_u) of the selected subtree. The probability difference can be positive, negative, and zero. If the probability is a positive number, that would mean the current subtree is selected with a higher chance compared with uniform probability. If the probability is a negative number, that means the current subtree is selected with a lower chance compared with uniform probability. If the probability is zero, that means the assigned probability is the same as the uniform one, which will not affect the crossover and mutation operators.

This paper takes CCGP^c in scenario $\langle \text{WFmean}, 0.85 \rangle$ as an example to show how the proposed subtree selection strategy affects the selection probability on crossover since CCGP^c performs significantly better than other algorithms in this scenario. Fig. 6 shows the histogram plot of the probability difference in early generation (i.e. generation 1), middle generation (i.e. generation 25) and late generation (i.e. generation 45) of CCGP^c in scenario $\langle \text{WFmean}, 0.85 \rangle$ based on 30 independent runs. The small in the subtitles means the smaller the score of the subtree, the higher probability it will be chosen (i.e. for parent_1 in the crossover). The big in the subtitles means the larger the score of the subtree, the higher probability it will be chosen (i.e. for parent_2 in the crossover). In general, most of the probability differences are positive numbers and much larger (i.e. more than 0.5) than uniform probability. At the early state (i.e. generation 1), the probability difference is not that higher than that of in the late generation (i.e. generation 25 and 45). This means that the proposed subtree selection strategy for crossover can successfully influence the selection of nodes of individuals.

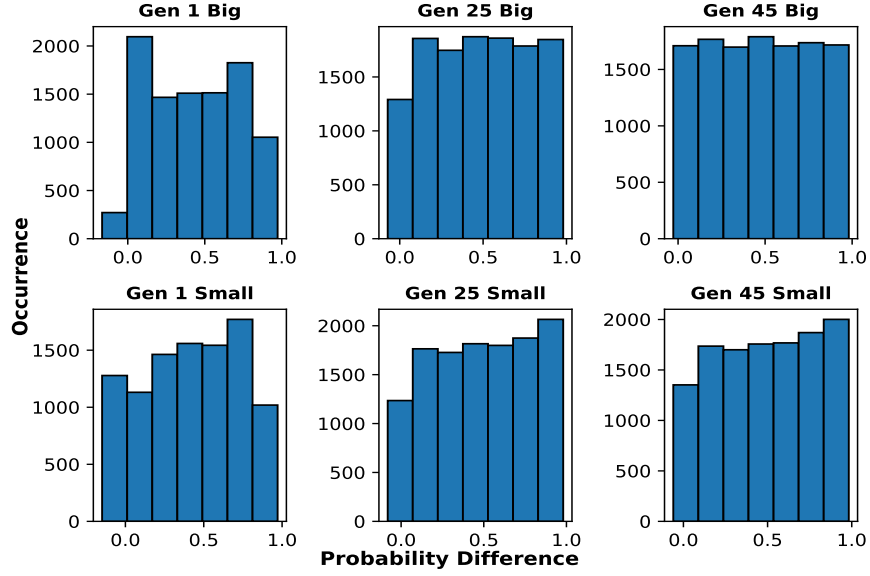


Fig. 6: The histogram plot of probability difference of $CCGP^c$ in generation 1, 25, and 45 in scenario $\langle WFmean, 0.85 \rangle$ based on 30 independent runs.

Fig. 7 shows the histogram plot of the probability difference in generation 1, 25, and 45 of $CCGP^m$ in scenario $\langle Fmean, 0.85 \rangle$ based on 30 independent runs. There are only three blocks because $CCGP^m$ only works on the mutation to choose the unimportant subtree (i.e. the smaller the score of the subtree, the higher probability it will be chosen). It is obvious that the number of subtree selections is not that high as that in Fig. 6, because the mutation rate is lower than the crossover rate. The same trend is shown in Fig. 6, the probability difference is becoming larger and larger as the number of generations increases.

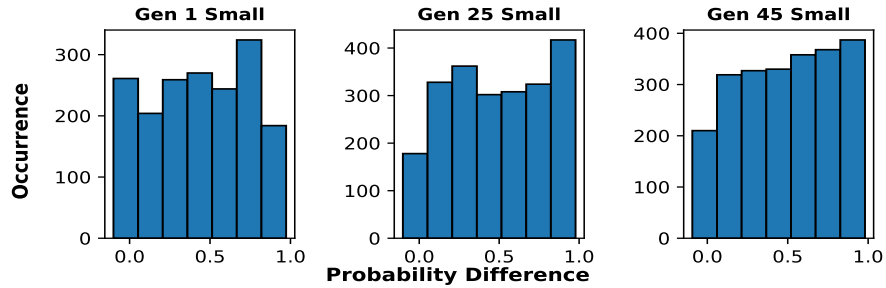


Fig. 7: The histogram plot of probability difference of $CCGP^m$ in generation 1, 25, and 45 in scenario $\langle Fmean, 0.85 \rangle$ based on 30 independent runs.

5.3 The Occurrences of Features

It is interesting to see the trend of the feature occurrence that carries the information. Fig. 8 shows the curves of the occurrence of features in routing rules during the evolutionary process of CCGP^c. The MWT (i.e. machine waiting time) is the most important feature for the routing rules in all scenarios. The importance of MWT is much higher than other features. In the scenarios whose utilisation levels are 0.85, WIQ (i.e. the workload in the queue) also plays a second important role. In the scenarios whose have a higher utilisation level (i.e. 0.95), NIQ (i.e. the number of operations in the queue) plays a significant role. Intuitively, both WIQ and NIQ are important indicators for measuring the workload for machines, they might have the same functions, and one might take over another one. However, we do not know how they work in different scenarios. It is interesting to see that the role of NIQ is significantly higher than that of WIQ in the scenarios that have higher utilisation level. One possible reason is that NIQ is an important factor in busy scenarios, which is an important finding.

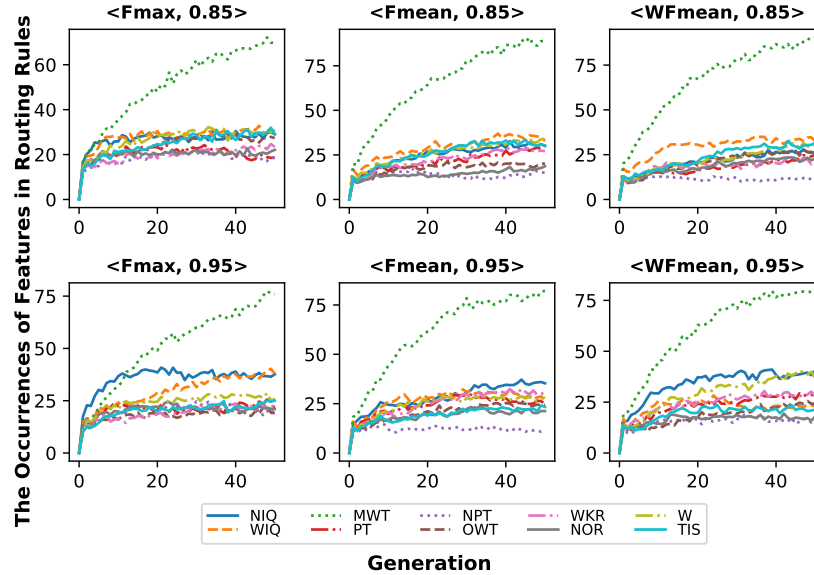


Fig. 8: The curves of the occurrence of features in *routing rules* during the evolutionary process of CCGP^c.

Fig. 9 shows the curves of the occurrence of terminals in sequencing rules during the evolutionary process of CCGP^c. Different from routing rules, there are three terminals (i.e. WKR, TIS, and PT) play a vital role in minimising max-flowtime. PT and WKR also are two important terminals in minimising mean-flowtime and weighted mean-flowtime. Except for them, W plays a dominant role

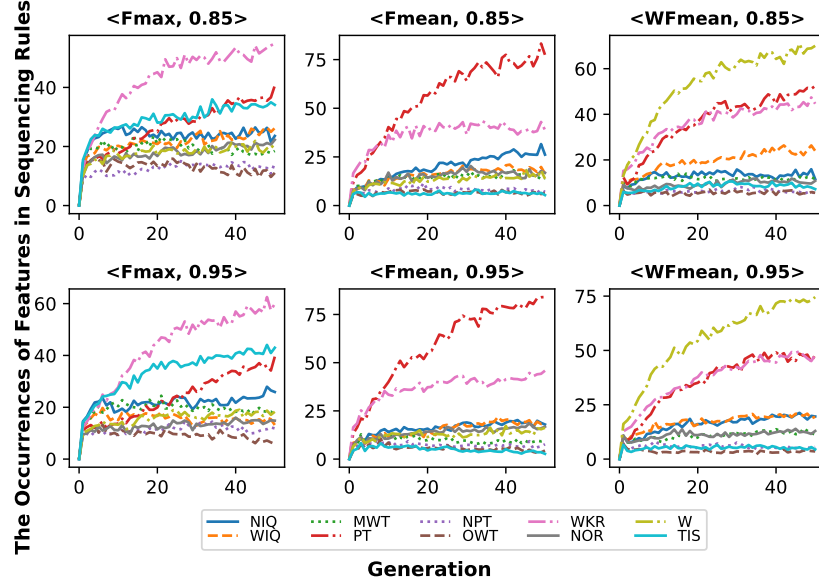


Fig. 9: The curves of the occurrence of features in *sequencing rules* during the evolutionary process of CCGP^c.

in weighted mean-flowtime, which is consistent with our intuition. In addition, W plays its role mainly in sequencing rules instead of routing rules.

5.4 Training time

Table 4 shows the mean and standard deviation of training time of CCGP, CCGP^c, CCGP^m, and CCGP^{cm} in six different scenarios. There is no significant difference between the four algorithms. It means the proposed subtree selection strategies do not need extra computational cost. This verifies the advantages of utilising the information generated during the evolutionary process of GP.

Table 4: The mean (standard deviation) of **training time** (in minutes) obtained by the involved four algorithms over 30 independent runs for six scenarios.

Scenario	CCGP	CCGP ^c	CCGP ^m	CCGP ^{cm}
<Fmax,0.85>	74(10)	75(11)	83(36)	75(11)
<Fmax,0.95>	86(16)	84(13)	84(12)	83(12)
<Fmean,0.85>	73(10)	73(11)	72(11)	75(13)
<Fmean,0.95>	79(11)	78(14)	77(13)	83(14)
<WFmean,0.85>	73(13)	70(10)	70(10)	73(11)
<WFmean,0.95>	82(13)	81(14)	80(14)	82(13)

6 Conclusions and Future Work

The goal of this paper was to develop subtree selection strategies to improve the effectiveness of crossover and mutation operators to guide GP to improve its convergence speed and evolve more effective scheduling heuristics for DFJSS. The goal was achieved by proposing the guided subtree selection strategy that can utilise the information of the occurrence of features information obtained during the evolutionary process.

The results show that using the proposed guided subtree selection in crossover can speed up the convergence and achieve better performance in half scenarios while no worse in all other scenarios without increasing the computational cost. The proposed subtree selection can successfully guide GP to select important or unimportant subtrees according to the need of genetic operators. The evolved rules have better test performance of given complex job shop scenarios, especially for minimising mean-flowtime and weighted mean-flowtime. An advantage of the proposed algorithms is that incorporating the occurrence of features information needs no extra computational cost. This shows the benefits of making better use of the information during the evolutionary process. In addition, this paper discovered that although both NIQ and WIQ can be used to measure the workload of a machine, NIQ has an important role in busy scenarios while WIQ has a significant role in less busy scenarios.

Some interesting directions can be further investigated in the near future. This work already shows the potential to improve the effectiveness of crossover by choosing subtrees. We would like to find more promising ways to select the subtrees for crossover to further improve its effectiveness.

References

1. Manne, A.S.: On the job-shop scheduling problem. *Operations Research* 8(2), 219–223 (1960)
2. Tay, J.C., Ho, N.B.: Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering* 54(3), 453–473 (2008)
3. Nguyen, S.B.S., Zhang, M.: A hybrid discrete particle swarm optimisation method for grid computation scheduling. In: *2014 IEEE Congress on Evolutionary Computation (CEC)*. pp. 483–490. IEEE (2014)
4. Brucker, P., Schlie, R.: Job-shop scheduling with multi-purpose machines. *Computing* 45(4), 369–375 (1990)
5. Yska, D., Mei, Y., Zhang, M.: Genetic programming hyper-heuristic with cooperative coevolution for dynamic flexible job shop scheduling. In: *European Conference on Genetic Programming*. pp. 306–321. Springer (2018)
6. Zhang, F., Mei, Y., Zhang, M.: Genetic programming with multi-tree representation for dynamic flexible job shop scheduling. In: *Proceedings of the Australasian Joint Conference on Artificial Intelligence (AI)*. pp. 472–484. Springer (2018)
7. Zhang, F., Mei, Y., Zhang, M.: Evolving dispatching rules for multi-objective dynamic flexible job shop scheduling via genetic programming hyper-heuristics. In: *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*. pp. 1366–1373. IEEE (2019)

8. Durasevic, M., Jakobovic, D.: A survey of dispatching rules for the dynamic unrelated machines environment. *Expert Systems with Applications* 113, 555–569 (2018)
9. Koza, J.R., Poli, R.: Genetic programming. In: *Search Methodologies*, pp. 127–164. Springer (2005)
10. Miyashita, K.: Job-shop scheduling with genetic programming. In: *Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation*. pp. 505–512. Morgan Kaufmann Publishers Inc. (2000)
11. Nguyen, S., Zhang, M., Johnston, M., Tan, K.C.: Genetic programming for evolving due-date assignment models in job shop environments. *Evolutionary Computation* 22(1), 105–138 (2014)
12. Maccarthy, B.L., Liu, J.: Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling. *The International Journal of Production Research* 31(1), 59–79 (1993)
13. Branke, J., Nguyen, S., Pickardt, C.W., Zhang, M.: Automated design of production scheduling heuristics: A review. *IEEE Transactions on Evolutionary Computation* 20(1), 110–124 (2016)
14. Burke, E.K., Hyde, M.R., Kendall, G., Ochoa, G., Ozcan, E., Woodward, J.R.: Exploring hyper-heuristic methodologies with genetic programming. In: *Computational Intelligence*, pp. 177–201. Springer (2009)
15. Burke, E.K., Hyde, M.R., Kendall, G., Woodward, J.R.: A genetic programming hyper-heuristic approach for evolving 2-d strip packing heuristics. *IEEE Trans. Evolutionary Computation* 14(6), 942–958 (2010)
16. Hyde, M.R.: A genetic programming hyper-heuristic approach to automated packing. Ph.D. thesis, University of Nottingham, UK (2010)
17. Bader-El-Den, M.B., Poli, R., Fatima, S.: Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework. *Memetic Computing* 1(3), 205–219 (2009)
18. Ardeh, M.A., Mei, Y., Zhang, M.: A novel genetic programming algorithm with knowledge transfer for uncertain capacitated arc routing problem. In: *Pacific Rim International Conference on Artificial Intelligence*. pp. 196–200. Springer (2019)
19. Zhang, F., Mei, Y., Zhang, M.: A new representation in genetic programming for evolving dispatching rules for dynamic flexible job shop scheduling. In: *Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP)*. pp. 33–49. Springer (2019)
20. Zhang, F., Mei, Y., Zhang, M.: A two-stage genetic programming hyper-heuristic approach with feature selection for dynamic flexible job shop scheduling. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*. pp. 347–355. IEEE (2019)
21. Durasevic, M., Jakobovic, D.: Evolving dispatching rules for optimising many-objective criteria in the unrelated machines environment. *Genetic Programming and Evolvable Machines* 19(1-2), 9–51 (2018)
22. Hildebrandt, T., Heger, J., Scholz-Reiter, B.: Towards improved dispatching rules for complex shop floor scenarios: a genetic programming approach. In: *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*. pp. 257–264. ACM (2010)