A Novel Fitness Function for Genetic Programming in Dynamic Flexible Job Shop Scheduling

Gaofeng Shi, Fangfang Zhang✉, Yi Mei
School of Engineering and Computer Science, Victoria University of Wellington, New Zealand
shigaof@myvuw.ac.nz, fangfang.zhang@ecs.vuw.ac.nz, yi.mei@ecs.vuw.ac.nz

2022 IEEE Congress on Evolutionary Computation (CEC) | 978-1-6654-6708-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/CEC55065.2022.9870235

***Abstract*—Dynamic flexible job shop scheduling (DFJSS) is a complex and challenging combinatorial optimisation problem. In DFJSS, job operations have to be processed on a set of machines, and thus machine assignment and operation sequencing decisions need to be made simultaneously in dynamic situations. Genetic programming (GP), as a hyper-heuristic approach, has been widely used to learn scheduling heuristics for DFJSS automatically. However, the traditional GP parent selection method based on fitness value only may not be sufficiently effective, since not all the subtrees of a GP individual are meaningful and can contribute to the goodness of the individual. This paper proposes a new GP algorithm with a novel fitness function by incorporating the subtree importance into the parent selection method. Specifically, the subtree importance is measured by the correlation coefficient between the behaviour of subtrees and the GP individual. The proposed algorithm is expected to improve the effectiveness of GP by capturing more useful subtrees for producing offspring to the next generation. This paper uses nine DFJSS scenarios to examine the effectiveness of the proposed algorithm. The results show that the proposed algorithm achieves slightly better performance in some of the scenarios while no worse in all other scenarios. Further analyses, including the effect of the designed fitness function and sizes of the learned scheduling heuristics, are also conducted.**

***Index Terms*—Parent Selection, Fitness Function, Genetic Programming, Dynamic Flexible Job Shop Scheduling.**

I. Introduction

Job shop scheduling (JSS) [1] is an important but challenging optimisation problem in computer science. In JSS, the task is to process a number of jobs (e.g., each job has a sequence of operations) by a set of machines. Flexible JSS [2] is an extension of JSS, where there are two decisions (i.e., machine assignment and operation sequencing) need to be made simultaneously. Dynamic flexible job shop scheduling (DFJSS) [3] aims to optimise the machine resource under a dynamic environment with unpredicted events, such as new job arrivals [4]. DFJSS is an NP-hard problem [5].

Traditional solution optimisation methods such as mathematical programming [6] and meta-heuristics [7] cannot solve DFJSS effectively due to their high computational complexity and inability to react to dynamic events efficiently. Scheduling heuristics such as dispatching rules [8]–[10], on the other hand, are promising techniques to tackle the dynamic environment efficiently. Rather than optimising an inflexible solution in advance, scheduling heuristics make decisions (e.g., machine assignment and operation sequencing) on-the-fly based on the latest information that incorporates all the dynamic events that have occurred so far. This way, scheduling heuristics can generate the schedule in real time. In DFJSS, the routing rule (i.e., for machine assignment) and the sequencing rule (i.e., for operation sequencing) work collaboratively to make the schedules.

However, it is very challenging to manually design effective scheduling heuristics that consider all the complex interactions with the various job shop attributes. Genetic Programming (GP), as a hyper-heuristic approach [11]–[13], is an effective method to automatically evolve scheduling heuristics for DFJSS. In DFJSS, GP is used to learn a routing rule for machine assignment and a sequencing rule for operation sequencing simultaneously.

This paper focuses on improving the effectiveness of GP in evolving scheduling heuristics for DFJSS. Particularly, it has been known that the traditional GP approaches evolve individuals with a large number of redundant branches, which can negatively affect the search effectiveness (e.g., swapping redundant branches of two parents is highly likely to generate useless offspring). There are several studies [14]–[18] that develop different types of biased/guided genetic operators rather than crossing over or mutating parents randomly. Compared with designing genetic operators, guiding the search of an algorithm with fitness function is easy to be implemented. Furthermore, selecting proper parents that are led by the fitness function, is the prerequisite for the genetic operators to generate good offspring. This paper aims to address this issue from a different angle, i.e., the fitness perspective.

When defining the fitness function, in addition to the raw fitness (i.e., the DFJSS objectives), we also consider the potential of generating promising offspring as a parent. Specifically, in the tree-based crossover and mutation operators, a sub-tree of the parent is selected and replaced by another sub-tree from the other parent (crossover) or randomly generated sub-tree (mutation) [19]. Therefore, a more promising parent should be more likely to have its useful building blocks swapped to the other parent in crossover, and less likely to have its useful building blocks destroyed in crossover or mutation.

Based on the above considerations, we define a new measure of the potential of individuals to generate promising offspring in terms of the likelihood of reusing and destroying useful building blocks from a GP tree in genetic operators. Then, we define a new fitness function based on the raw fitness and the potential of individuals to generate promising offspring.

The overall goal of this paper is to develop a new fitness function for GP to evolve scheduling heuristics for DFJSS

978-1-6654-6708-7/22/$31.00 ©2022 IEEE

more effectively and efficiently. The specific research objectives of this paper are as follows.

1) We propose a scheme to identify the useful building blocks (sub-trees) of a tree based on the correlation of the phenotypic behaviours on a set of decision situations.
2) We develop a measure on the potential of generating good offspring based on the sub-tree correlation, position and size. Briefly speaking, a tree whose building blocks have higher correlations, and are smaller and at a position farther away from the root node tends to have a higher potential to generate good offspring.
3) We define a new fitness function based on the raw fitness and the new measure on the potential of individuals. The new fitness function is used for parent selection.
4) We propose a new GP with the new fitness function, and verify its effectiveness on a range of DFJSS scenarios.

II. Background

A. Dynamic Flexible Job Shop Scheduling

JSS aims to improve the production efficiency in a shop floor [20]. In DFJSS problem, m machines $M = \{M_1, M_2, ..., M_m\}$ are used to process n jobs $J = \{J_1, J_2, ..., J_n\}$. Each job J_j consists of a sequence of operations $O_j = (O_{j1}, O_{j2}, ..., O_{ji})$. An operation O_{ji} can be processed by one of its candidate machines $\pi(O_{ji})$ and its processing time $\delta(O_{ji})$ depends on the machine that processes it. In this paper, we focus on the dynamic event that job can arrive over time. That is, the information of a job is unknown until it arrives at the job shop. The following constraints must be satisfied in the problem.

- The order of operations for each job is predefined, and one cannot start processing an operation until all its precedent operations have been processed.
- Each operation can be processed only by one of its candidate machines.
- Each machine can process at most one operation at a time.
- The scheduling is non-preemptive, i.e., once start, the processing of an operation cannot be stopped or paused until it is completed.

We consider three common objectives, i.e., mean-flowtime (Fmean), mean-tardiness (Tmean), and mean-weighted-tardiness (WTmean), which are calculated as follows.

- Fmean $= \frac{\sum_{j=1}^{n} \{C_j \quad r_j\}}{n}$
- Tmean $= \frac{\sum_{j=1}^{n} Max\{0, C_j \quad d_j\}}{n}$
- WTmean $= \frac{\sum_{j=1}^{n} w_j * Max\{0, C_j \quad d_j\}}{n}$

Where C_j is the completion time of 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 of J_j, and n is the number of jobs to be processed.

B. Scheduling Heuristics For DFJSS

Scheduling heuristic consists of a routing rule and a sequencing rule in DFJSS [21]. Due to the DFJSS constraints, the allocation of machines is only performed for ready operations. There are two kinds of operations that will become ready operations. One is the first operation of a job arrived at the shop floor, and the other is the subsequent operation whose preceding operation is just finished. Once an operation becomes a ready operation, the operation will be allocated to the machine with the highest priority according to the routing rule. When a machine becomes idle, and the queue of the machine is not empty, at this point, the sequencing rule will be applied to calculate the priority value of each operation in its queue. The highest priority operation in the queue of the machine will be chosen for the next processing.

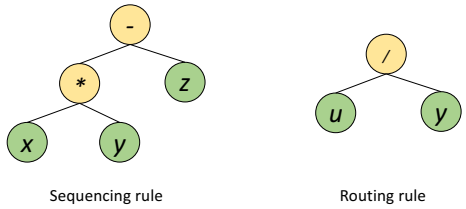

Fig. 1: An example of tree-based representation of sequencing and routing rule, where x, y, z and u are features, and -, *, / are functions.

C. Genetic Programming Hype-heuristics

GP has been widely used to handle the dynamic JSS problem due to it advantages of learning scheduling heuristics [13], [22]–[24]. One is its flexible representation, so we do not need to define the structure of rules in advance. The other is that the tree-based programs obtained by GP provide us with opportunities to understand the behaviour of the evolved rules. That is very important for real-world applications [17].

1) Individual Representation: Each GP individual is represented as a tree, which is essentially a combination of the job shop state attributes. Fig. 1 gives an example of the tree-based routing rule and sequencing rule. A routing rule (i.e., $x * y - z$) is applied to calculate the priority value of each candidate machine of a ready operation. The machine with the best priority value is selected to process the operation. In addition, a sequencing rule (i.e., u/y) is applied to calculate the priority value of each operation in the queue of an idle machine, if operations are waiting in its queue. The operation with the best priority is selected to be processed next.

2) Process of GP: GP improves the quality of individuals generation by generation. GP starts with a population of randomly initialised individuals, called *individual initialisation*. All individuals are evaluated with fitness function, named *fitness evaluation*. Then, parents are selected to generate offspring to the next generation (i.e., *parent selection*). During the *evolution* process, new offspring are produced based on the selected parents along with the genetic operators, i.e., crossover, mutation and reproduction. If the stopping criterion is not met, the newly generated individuals will be moved to

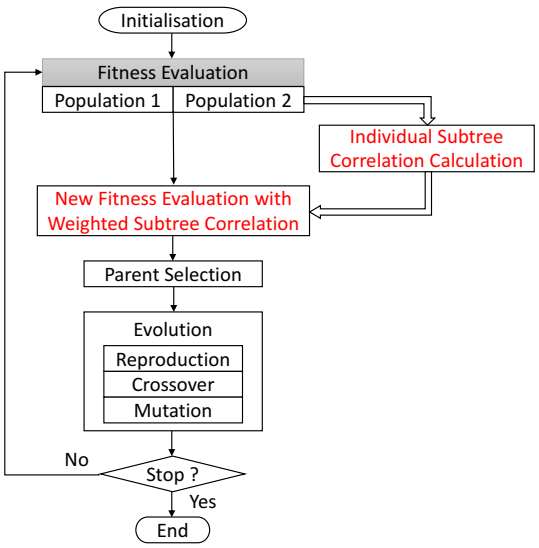

Fig. 2: The flowchart of the proposed method.

the next generation. Otherwise, the best learned scheduling heuristic will be reported as the output of a GP algorithm.

Cooperative coevolution based GP (CCGP) was investigated to evolve routing and sequencing rules simultaneously in [25]. The proposed GP algorithm in this paper was conducted based on CCGP, which will be described in the next section.

III. The Proposed Method

The proposed GP with the new fitness function is described in this section. First, we introduce the overall flowchart of the algorithm. Then, we describe the components in detail, especially the new fitness function.

A. The Overall Framework

Fig. 2 shows the flowchart of the proposed method, which is named CCGP with sub-tree importance (sCCGP). It follows the existing CCGP process, which maintains two sub-populations, one for the routing rule and the other for the sequencing rule. The details can be found in [25]. It starts with initialising the sub-populations randomly. Then, at each generation, it first evaluates the individuals in the sub-populations based on a set of training DFJSS simulations. In the CCGP framework, to evaluate a routing (sequencing) rule, it is paired with the best sequencing (routing) rule from the previous generation. For the first generation, we just randomly choose one sequencing or the routing rule from the current generation as the best ones. The raw fitness of an individual is calculated as the objective function value (e.g., mean-flowtime) of the generated schedule. Afterwards, for each individual represented as a tree, the correlation coefficient between the behaviour of each sub-tree and itself is calculated based on a set of decision situations. Then, the adjusted fitness of the individual is calculated as a weighted aggregation of the raw fitness and the correlation coefficient of its sub-trees. After the fitness evaluation, the parent selection is conducted to select parent individuals for breeding. The tournament selection is used. Specifically, k candidate individuals are randomly sampled from the population, and then the candidate with the best adjusted fitness is selected. Next, the offspring individuals are generated by applying the genetic operators (e.g., crossover, mutation and reproduction) to the parents. In addition, some top individuals in terms of *raw fitness* are selected as elites. The evolutionary process continues until the stopping criterion (i.e., maximal number of generations) is reached.

The main difference between the newly proposed algorithm and the existing GP methods is the fitness function (highlighted in Fig. 2). The existing GP methods typically use the raw fitness, while the proposed method uses the adjusted fitness, which is a weighted aggregation of the raw fitness and the sub-tree correlation.

B. New Fitness Evaluation

Given a GP tree $\mathbb{T}$, its fitness function is defined as follows.

$$fit(\mathbb{T}) = f_{raw}(\mathbb{T}) - \alpha \times C(\mathbb{T}), \tag{1}$$

where $f_{raw}(\mathbb{T})$ is the raw fitness of $\mathbb{T}$, which is the same as the fitness function of the existing GP methods. Specifically, given a training DFJSS simulation $\texttt{sim}$ and its paired individual rule $\mathbb{T}_{pair}$, the raw fitness is calculated as

$$f_{raw}(\mathbb{T}) = \texttt{obj}(\texttt{sim}(\mathbb{T}, \mathbb{T}_{pair})), \tag{2}$$

where $\texttt{sim}(\mathbb{T}, \mathbb{T}_{pair})$ is the schedule generated by applying $\mathbb{T}$ and $\mathbb{T}_{pair}$ to the simulation, and $\texttt{obj}(\cdot)$ is the objective function, such as mean flowtime or makespan.

The component $C(\mathbb{T})$ reflects the potential of $\mathbb{T}$ to generate promising offspring by genetic operators. Specifically, if a tree is more likely be swap its useful building blocks to other trees, and less likely to have its useful building blocks destroyed by the genetic operators, then it should have a higher $C(\mathbb{T})$ value.

To this end, we define $C(\mathbb{T})$ based on two aspects. The first is the identification of useful building blocks of the tree. Here, a sub-tree is considered a useful building block if it greatly contributes to the tree itself, i.e., it shows consistent behaviour with the entire tree. On the other hand, a sub-tree is a redundant branch if its behaviour is independent of the entire tree. We adopt the method in [17] to identify useful building blocks based on the behaviour correlation. The second aspect is the likelihood of a building block being transferred to other parents and destroyed by genetic operators. We propose a new measure to estimate such likelihood of a building block (sub-tree) based on its size and position in the tree.

1) Sub-tree Behaviour Correlation: First, we need to characterise the behaviour of a (sub-)tree. Here, we adopt the commonly used phenotypic characterisation, which is based on a list of *decision situations* [17]. A decision situation is defined as a decision point when a routing/sequencing decision is made by the rule. It can be represented as a matrix $\mathbf{D}$, where each row $\mathbf{D}_{i,*}$ stands for a candidate (machine/operation) and each column $\mathbf{D}_{*,j}$ indicates a feature. In other words, an element D_{ij} means the value of feature j of candidate i in this decision situation.

TABLE I: An example of behaviour calculation in a decision situation.

$\mathbb{T} = A_1 + A_2/A_3$					
D	A_1	A_2	A_3	$\mathbb{T}(\mathbf{D})$	**b**
Candidate 1	100	150	3	150	1
Candidate 2	130	110	2	185	2
Candidate 2	140	100	2	190	3

TABLE II: An example of correlation coefficient of sub-trees in the decision situation shown in Table I.

(Sub-)Tree	b	Correlation
$\mathbb{T} = A_1 + A_2/A_3$	$[1, 2, 3]$	N/A
A_1	$[1, 2, 3]$	1
A_2	$[3, 2, 1]$	-1
A_3	$[3, 1, 1]$	-0.87
A_2/A_3	$[1, 3, 1]$	0

Applying a tree $\mathbb{T}$ to a decision situation $\mathbf{D}$ results in a vector $\mathbb{T}(\mathbf{D})$, where $\mathbb{T}(\mathbf{D}_{i,*})$ is the priority value of the candidate i. Finally, we obtain the behaviour of $\mathbb{T}$ in $\mathbf{D}$ as the sorted index list of the candidates in the ascending order of the priority values. Table I shows an example of the behaviour calculation of a tree $\mathbb{T} = A_1 + A_2/A_3$ (A_1, A_2 and A_3 are job shop attributes) in a decision situation. It can be seen that in this decision situation, $\mathbb{T}(\mathbf{D}) = [150, 185, 190]$ and $\mathbf{b} = [1, 2, 3]$.

We can calculate the behaviour vector for each sub-tree. For example, in Table I, the tree has four sub-trees A_1, A_2, A_3 and A_2/A_3. We can calculate the behaviour vectors in this decision situation as $\mathbf{b}(A_1) = [1, 2, 3]$, $\mathbf{b}(A_2) = [3, 2, 1]$, $\mathbf{b}(A_3) = [3, 1, 1]$ and $\mathbf{b}(A_2/A_3) = [1, 3, 1]$. Then, we can calculate the Spearman correlation coefficient [26] between the behaviour vectors of each sub-tree and the entire tree as shown in Table II. For the sake of convenience, we simply call "Spearman correlation coefficient" as "correlation" without loss of generality. We can see that both A_1 and A_2 show completely consistent behaviour (one positive and the other negative) with $\mathbb{T}$. A_3 has slightly weaker correlation than A_1 and A_2, and A_2/A_3 is completely uncorrelated of $\mathbb{T}$.

2) Sub-tree Weights: In addition to the correlation on behaviour, another important aspect of sub-tree quality is its size and position in the tree. It is commonly known that a larger sub-tree that is closer to the root node tends to behave more consistently with the entire tree [17]. However, it also tends to contain more redundant branches and useless genetic materials. This could affect the effectiveness of the genetic operators in exchanging useful building blocks. In other words, if a sub-tree is smaller and at lower positions that are far away from the root node, then the sub-tree is more likely to be selected to exchange and less likely to be destroyed by the crossover/mutation operators.

Fig. 3 shows an example GP tree with four subtrees with at least a depth of 2 (ignoring the terminals). Assume that T_2, T_3 and T_5 have consistent behaviour (correlation of 1) with

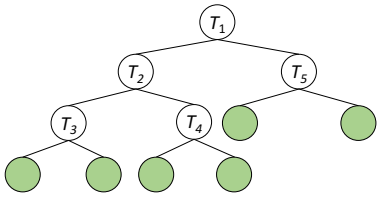

Fig. 3: An example of a tree with four sub-trees with at least a depth of 2.

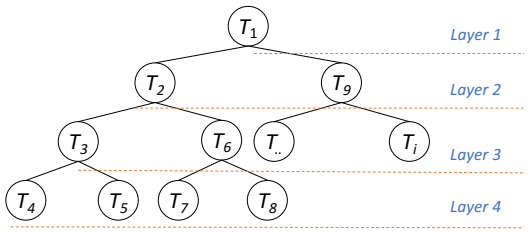

Fig. 4: An example of a tree and its layers.

the entire tree, T_3 is considered to be better than T_2 and T_5. Compared with T_2, T_3 has a smaller size, and its position is farther away from the root node. Thus, it has a higher probability of being swapped to other parents. Specifically, there are three nodes that can swap T_3 (T_3, T_2 and T_1), while only two nodes swapping T_2 (T_2 and T_1). T_3 also has a smaller probability of being destroyed than T_2. To destroy T_3, the cutting point must be one of the two child nodes of T_3 (probability of $2/11$). However, there are six child nodes that can destroy T_2 (probability of $6/11$). T_5 has the same size as T_3, thus having the same probability to be destroyed. However, it is closer to the root node than T_3. As a result, it is less likely to be swapped to other parents (only two possibilities, T_5 and T_1 versus three possibilities of T_3).

Based on this consideration, we divide a tree into different layers, as shown in Fig. 4. The root node is in layer 1, its child nodes are in layer 2, and so on. Obviously, for each node in the tree, we have

$$\texttt{layer}(node) = \texttt{depth}(node) + 1 \tag{3}$$

Giving the same correlation coefficient in behaviour, a sub-tree in a larger layer tends to be better to choose.

The sub-tree correlation component $C(\mathbb{T})$ is defined based on the correlation and layer of the sub-trees. Given a set of decision situations $\mathcal{D}$, the calculation of $C(\mathbb{T})$ is shown in Algorithm 1, $|\mathcal{D}|$ in line 12 indicates the number of decision situations. For each decision situation $\mathbf{D} \in \mathcal{D}$, it first calculates the behaviour vectors of the tree and all its sub-trees $\mathbb{T}_i \in \mathbb{T}$, and then calculates the correlation coefficient between each sub-tree and the entire tree. Then, for a decision situation $\mathbf{D}$, the component $C(\mathbb{T}; \mathbf{D})$ is calculated as the weighted sum of the absolute correlation coefficient and the layer of the sub-trees (line 8) which is normalised by the sum of the sub-tree layers. It repeats the process for all the decision situations,

Algorithm 1: Calculation of $C(\mathbb{T})$

Input : A GP tree $\mathbb{T}$, a set of decision situations $\mathcal{D}$
Output: $C(\mathbb{T})$

$C(\mathbb{T}) \leftarrow 0$;
for $\mathbf{D} \in \mathcal{D}$ **do**
  Calculate the behaviour vector $\mathbf{b}(\mathbb{T})$ of $\mathbb{T}$ in $\mathbf{D}$;
  Set $C(\mathbb{T};\mathbf{D}) = 0$, $L = 0$;
  foreach *sub-tree* $\mathbb{T}_i \in \mathbb{T}$ **do**
    Calculate the behaviour vector $\mathbf{b}(\mathbb{T}_i)$;
    Calculate the correlation coefficient $\theta(\mathbf{b}(\mathbb{T}_i), \mathbf{b}(\mathbb{T}))$;
    $C(\mathbb{T};\mathbf{D}) \leftarrow C(\mathbb{T};\mathbf{D}) + |\theta(\mathbf{b}(\mathbb{T}_i), \mathbf{b}(\mathbb{T}))| \cdot \texttt{layer}(\mathbb{T}_i)$;
    $L \leftarrow L + \texttt{layer}(\mathbb{T}_i)$;
  $C(\mathbb{T};\mathbf{D}) \leftarrow C(\mathbb{T};\mathbf{D})/L$;
  $C(\mathbb{T}) \leftarrow C(\mathbb{T}) + C(\mathbb{T};\mathbf{D})$;
$C(\mathbb{T}) \leftarrow C(\mathbb{T})/|\mathcal{D}|$;
return $C(\mathbb{T})$;

and finally set $C(\mathbb{T})$ as the average value over all the decision situations.

IV. Experiment Studies

To verify the effectiveness of the newly proposed fitness function and the resultant new sCCGP algorithm, we conduct experiments on a range of DJFSS scenarios and compare with the baseline CCGP algorithm [25] with the raw fitness function. We first give the experiment setup and parameter settings, and then show the results and discussions.

A. DFJSS Scenarios and Simulation Configuration

In the experiments, we consider the following DFJSS simulations. In the job shop, there are 10 machines. The jobs arrive at the shop floor over time. The job arrivals follow the Poisson process, i.e., the gaps between subsequent job arrivals follow an exponential distribution. For each job, the number of operations is randomly sampled from 1 to 10 (10 is the maximal number of machines). For each operation, the set of candidate machines is randomly sampled as a subset of the 10 machines, and the processing time on each candidate machine is randomly sampled from a uniform discrete distribution between 1 and 99. The job weights are randomly sampled from a distribution so that 20%, 60% and 20% of the jobs have a weight of 1, 2 and 4, respectively. The due date factor of each job is set to 1.5. We consider a simulation with a sufficiently large number of job arrivals to measure the steady-state performance. To this end, we start with an empty shop floor, run 1000 job arrivals as a warm-up stage, and then collect the information of the next 5000 job arrivals to measure the performance of the scheduling heuristics. This simulation configuration has been commonly used in the studies of dynamic scheduling [27], [28].

Here, we characterise different DFJSS scenarios by the utilisation level and objective function. The utilisation level is a parameter that reflects the percentage of time that the job shop is busy via controlling the frequency of job arrivals in the Poisson process. The objective function can affect the effectiveness of different scheduling heuristics. For example, the shortest processing time heuristic works well for the mean flowtime objective, while the first-come-first-serve heuristic can perform better for the maximal flowtime objective.

In the experiments, we consider three different utilisation levels (i.e., 0.75, 0.85 and 0.95) and three different objectives, i.e., Fmean, Tmean and WTmean. A utilisation level of 0.75 corresponds to the least busy shop floor (easiest scheduling problem), while 0.95 leads to the busiest shop floor (hardest scheduling problem). As a result, we have $3 \times 3 = 9$ different DFJSS scenarios. We denote each scenario as "$\texttt{obj}$-ul". For training, we use one training simulation per generation, and rotate its random seed at each generation. After the best scheduling heuristics are trained, they are tested on 50 unseen test simulations to calculate the test performance.

For each DFJSS scenario, each compared algorithm is run 30 times independently, and their results are compared under the statistical significance test, i.e., Wilcoxon test.

TABLE III: The parameter setting of GP.

Parameter	Value
Number of subpopulations	2
Subpopulation size	500
The number of generations	51
Number of elites for each subpopulation	5
Initialisation method	ramped-half-and-half
Initial minimum / maximum depth	2 / 6
Maximal tree depth	8
Crossover / Mutation / Reproduction rate	80% / 15% / 5%
Tournament parent selection size	7
Terminal / non-terminal selection rate	10% / 90%

TABLE IV: The terminal and function sets of GP.

	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, / is protected division

B. Parameter Settings

We adopt the common GP parameter settings, which is shown in Table III. Note that there are two subpopulations, one for routing rule and the other for sequencing rule. The terminal set and function set are shown in Table IV following the suggestions in [29], [30].

C. Parameter Sensitivity Analysis

In the new fitness function, the parameter α is important to balance the raw fitness and the new measure $C(\mathbb{T})$ of potential in offspring generation. For example, if $\alpha = 0$, then the new measure is completely ignored, and sCCGP is reduced to the

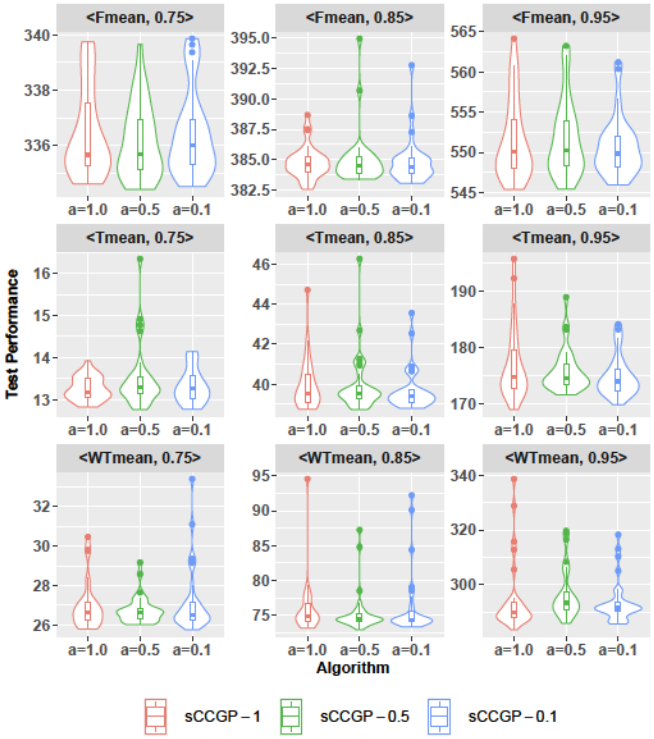

Fig. 5: Violin plots of the average objective values on test instances of sCCGP with different α values.

baseline CCGP. On the other hand, if α is sufficiently large, then the parent selection tends to be purely based on the new measure regardless of the raw fitness value. To find the best setting of α, we conduct the parameter sensitivity analysis experiment to compare sCCGP with different α values.

Fig. 5 shows the violin plot of the test performance of three different α values: 1.0, 0.5 and 0.1. As α decreases, the importance of the raw fitness becomes higher. From the figure, no statistical significance can be seen from the three different α values. $\alpha = 1.0$ tends to show slightly better results on some scenarios (e.g., <Fmean, 0.85> and <Tmean, 0.75>). However, it might also produce outliers with very poor test performance (e.g., on <WTmean, 0.85> and <WTmean, 0.95>). We conduct a Wilcoxon rank sum test to compare the results of the three α values, but find no significant difference. Thus, we simply select $\alpha = 0.5$ for subsequent experiments.

D. Results and Discussions

Table V shows the mean and standard deviation of the test performance of the 30 independent runs of CCGP and the baseline sCCGP ($\alpha = 0.5$) on the nine DFJSS scenarios. For each scenario, we conduct the Wilcoxon rank sum test with a significance level of 0.05 to compare the results of CCGP and sCCGP. In Table V, the notation "$\approx$" indicates that there is no statistical significance between the results of the two algorithms. From Table V, we can see that for all the scenarios, the two algorithms show statistically comparable test performance. This shows that the new fitness does not seem to significantly affect the search ability of CCGP, and

TABLE V: The mean (standard deviation) of the average objective values of the 30 independent runs of CCGP and sCCGP ($\alpha = 0.5$) for the DFJSS scenarios.

Scenario	CCGP	sCCGP
<Fmean, 0.75>	336.23(1.26)	336.13(1.35)($\approx$) (p=0.48)
<Fmean, 0.85>	384.69(1.63)	385.05(2.29)($\approx$) (p=0.71)
<Fmean, 0.95>	550.94(5.79)	551.67(4.77)($\approx$) (p=0.29)
<Tmean, 0.75>	13.28(0.40)	13.51(0.76)($\approx$) (p=0.37)
<Tmean, 0.85>	40.27(1.85)	39.97(1.48)($\approx$) (p=0.68)
<Tmean, 0.95>	175.49(2.85)	175.81(3.85)($\approx$) (p=0.65)
<WTmean, 0.75>	27.04(1.05)	26.77(0.72)($\approx$) (p=0.43)
<WTmean, 0.85>	75.82(3.83)	75.38(3.10)($\approx$) (p=0.85)
<WTmean, 0.95>	294.58(9.65)	296.61(9.26)($\approx$) (p=0.12)

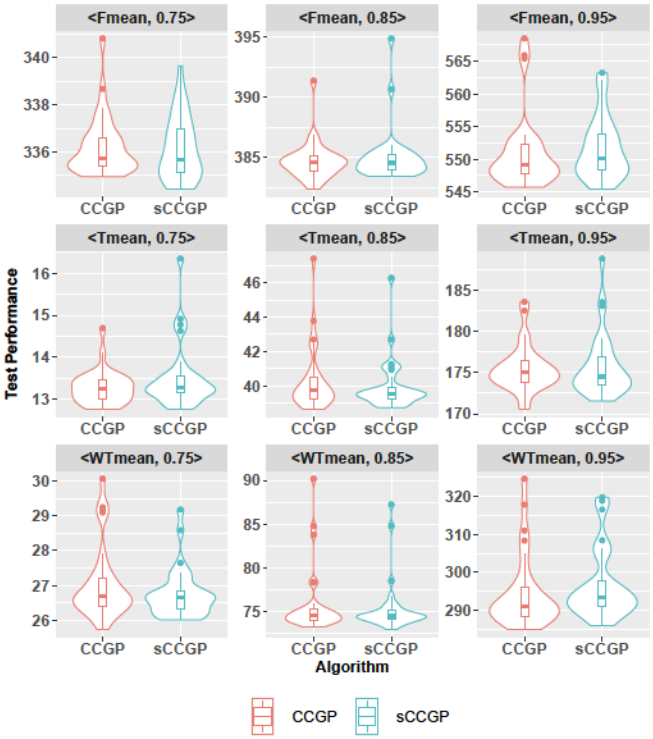

Fig. 6: Violin plots of the average objective values on test instances of CCGP and sCCGP ($\alpha = 0.5$).

sCCGP tends to select similar subsets (compared with CCGP) of individuals as parents to generate offspring.

Fig. 6 shows the violin plots of the test performance of CCGP and sCCGP. Again, the violin plots show no significant difference between CCGP and sCCGP. The mean/median of the two algorithms seems to be very close to each other. In terms of outliers, the figure shows that sCCGP seems to perform better in terms of outliers (on <Fmean, 0.75>, <Fmean, 0.95>, <Tmean, 0.85>, <WTmean, 0.75>, <WTmean, 0.85> and <WTmean, 0.95>). This suggests that sCCGP tends to perform more stably than CCGP.

Fig. 7 shows the convergence curves of the test performance of CCGP and sCCGP. From the figure, we can see that the new fitness function could help sCCGP converge faster than CCGP for some instances. For <Fmean, 0.75>, the test performance

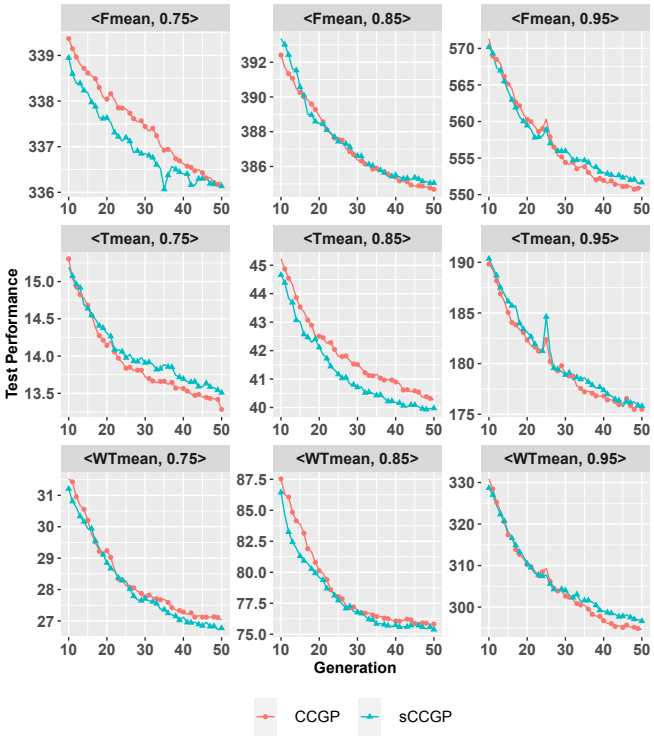

Fig. 7: Convergence curves of the average objective values on test instances of CCGP and sCCGP ($\alpha = 0.5$).

of sCCGP converges much faster than CCGP, although it is finally similar with CCGP. For <Tmean, 0.85>, the curve of sCCGP is always below that of CCGP, showing the advantage of sCCGP during the entire evolutionary process. For the remaining scenarios, the convergence curves of the two algorithms are very similar at the beginning. sCCGP tends to be stuck into local optima and show slightly worse performance than CCGP on some scenarios (e.g., <Tmean, 0.75> and <Fmean, 0.95>). We will investigate why the advantage of the new fitness function tends to disappear at the later stage of the evolution.

E. Further Analyses

1) Effect of New Fitness Function: To further understand how the new fitness function changes the behaviour of the algorithm, we select some populations in different generations of sCCGP, and draw the scatter plot of the raw fitness versus new fitness of the top 100 individuals in the population. We consider the top 100 individuals since they are most likely to be selected as parents. Fig. 8 shows a scatter plot including the fitness of subpopulations for both routing and sequencing rules in the <Fmean, 0.85> scenario. The red line $y = x$ is given as a reference.

From the figure, we can see that for all the populations in different generations (i.e., early, middle and late stages of the evolution) and for both sequencing and routing rules, the points in the scatter plot are distributed highly parallel with the $y = x$ line. This indicates that after subtracting the term $\alpha \times C(\mathbb{T})$, the fitness tends not to change significantly, and

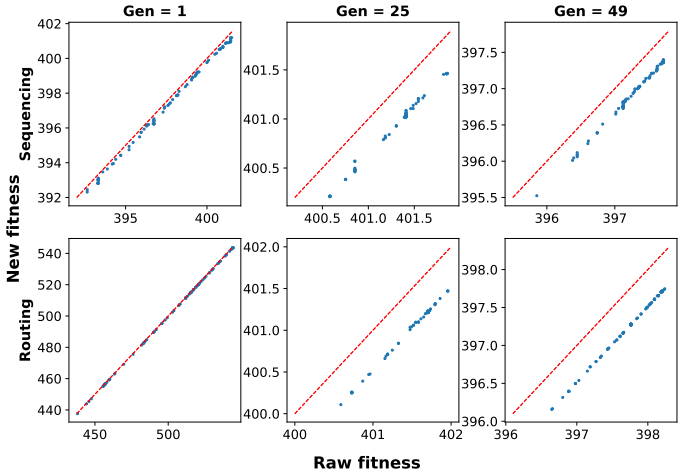

Fig. 8: The raw fitness versus new fitness of some populations in sCCGP based on top 100 individuals in <Fmean, 0.85>.

TABLE VI: The mean (standard deviation) of **sequencing** rule sizes of CCGP and sCCGP (α = 0.5) over 30 independent runs.

Scenario	CCGP	sCCGP
<Fmean, 0.75>	36.53(17.15)	36.67(18.40)(≈)
<Fmean, 0.85>	48.00(17.11)	43.40(17.78)(≈)
<Fmean, 0.95>	44.00(17.41)	41.00(17.76)(≈)
<Tmean, 0.75>	46.67(15.65)	29.13(14.20)(–)
<Tmean, 0.85>	50.53(15.57)	40.07(17.53)(–)
<Tmean, 0.95>	48.87(16.14)	42.20(23.43)(–)
<WTmean, 0.75>	50.80(20.01)	43.53(19.64)(≈)
<WTmean, 0.85>	51.73(15.51)	43.60(15.84)(–)
<WTmean, 0.95>	45.60(17.02)	48.47(13.04)(≈)

the relative orders of the individuals often stay the same after subtracting the new term. In other words, in the tournament selection, we will often select the same individual as the parent, no matter whether the raw fitness or the new fitness is used. This explains why sCCGP shows very similar performance with CCGP. Other scenarios show similar patterns.

Although most of the time the new fitness changes little, we can still see the effect in some places. For example, in generation 25, the new fitness can better distinguish the individuals whose raw fitness are around 400.8 and those around 401.4.

2) Rules Size: Tables VI and VII show the size of the best sequencing and routing rules obtained by CCGP and sCCGP, where notations "≈" and "–" indicate similar and smaller sizes, respectively. From the table, we can see that sCCGP manages to obtain much smaller sequencing rules for a number of scenarios (<Tmean, 0.75>, (<Tmean, 0.85>, <Tmean, 0.95> and <WTmean, 0.85>). For the remaining scenarios, the sequencing rules also tend to be smaller. This shows that the new fitness function can help sCCGP generate more compact offspring for the sequencing rules. The routing rules have roughly the same size.

TABLE VII: The mean (standard deviation) of **routing** rule size of CCGP and sCCGP (α = 0.5) over 30 independent runs.

Scenario	CCGP	sCCGP
<Fmean, 0.75>	62.00(19.10)	59.93(16.29)(≈)
<Fmean, 0.85>	64.13(18.05)	56.60(15.95)(≈)
<Fmean, 0.95>	58.80(14.32)	63.67(14.78)(≈)
<Tmean, 0.75>	56.47(16.63)	53.87(16.58)(≈)
<Tmean, 0.85>	63.33(18.86)	56.67(14.64)(≈)
<Tmean, 0.95>	56.40(13.75)	60.27(15.36)(≈)
<WTmean, 0.75>	61.00(16.62)	61.27(15.41)(≈)
<WTmean, 0.85>	61.20(19.08)	58.33(19.80)(≈)
<WTmean, 0.95>	60.87(12.18)	64.27(16.06)(≈)

V. Conclusions and Future work

This paper aims to improve the effectiveness of CCGP to evolve scheduling heuristics for DFJSS. This goal has been successfully achieved by proposing a new fitness function that incorporates the potential of an individual to retain its useful building blocks. The experimental results show that the proposed sCCGP algorithm with the new fitness function can obtain smaller rule sizes, especially for the sequencing rule, while keeping comparable test performance. This shows the effectiveness of the new fitness function in generating more compact offspring.

In future, we will investigate adaptive fitness functions to achieve a better balance between the raw fitness and the potential of the tree structure, so that the new fitness function can distinguish more individuals with similar raw fitness. We will also extend the current sub-tree contribution mechanism to other methods, such as feature construction in DFJSS.

References

[1] C. A. Brizuela and N. Sannomiya, "A diversity study in genetic algorithms for job shop scheduling problems," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 1999, pp. 75–82.

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

[3] F. Zhang, Y. Mei, S. Nguyen, K. C. Tan, and M. Zhang, "Multitask genetic programming-based generative hyper-heuristics: A case study in dynamic scheduling," *IEEE Transactions on Cybernetics*, 2021, Doi: 10.1109/TCYB.2021.3065340.

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

[5] 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.

[6] M. C. Gomes, A. P. Barbosa-Póvoa, and A. Q. Novais, "Reactive scheduling in a make-to-order flexible job shop with re-entrant process and assembly: a mathematical programming approach," *International Journal of Production Research*, vol. 51, no. 17, pp. 5120–5141, 2013.

[7] E. Guzman, B. Andres, and R. Poler, "Matheuristic algorithm for job-shop scheduling problem using a disjunctive mathematical model," *Computers*, vol. 11, no. 1, p. 1, 2022.

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

[9] R. Braune, F. Benda, K. F. Doerner, and R. F. Hartl, "A genetic programming learning approach to generate dispatching rules for flexible shop scheduling problems," *International Journal of Production Economics*, vol. 243, p. 108342, 2022.

[10] 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, 2020.

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

[12] N. Pillay and R. Qu, "Assessing hyper-heuristic performance," *Journal of the Operational Research Society*, vol. 72, no. 11, pp. 2503–2516, 2021.

[13] X. Chen, R. Bai, R. Qu, H. Dong, and J. Chen, "A data-driven genetic programming heuristic for real-world dynamic seaport container terminal truck dispatching," in *Proceedings of the IEEE Congress on Evolutionary Computation*. IEEE, 2020, pp. 1–8.

[14] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Guided subtree selection for genetic operators in genetic programming for dynamic flexible job shop scheduling," in *Proceedings of the European Conference on Genetic Programming*. Springer, 2020, pp. 262–278.

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

[16] R. Poli and N. F. McPhee, "General schema theory for genetic programming with subtree-swapping crossover: Part ii," *Evolutionary Computation*, vol. 11, no. 2, pp. 169–206, 2003.

[17] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Correlation Coefficient-Based Recombinative Guidance for Genetic Programming Hyperheuristics in Dynamic Flexible Job Shop Scheduling," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 3, pp. 552–566, 2021.

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

[19] R. Poli, W. B. Langdon, N. F. McPhee, and J. R. Koza, *A field guide to genetic programming*. Lulu. com, 2008.

[20] 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.

[21] 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.

[22] K. Jaklinović, M. Durasević, and D. Jakobović, "Designing dispatching rules with genetic programming for the unrelated machines environment with constraints," *Expert Systems with Applications*, p. 114548, 2021.

[23] T. Hildebrandt and J. Branke, "On Using Surrogates with Genetic Programming," *Evolutionary Computation*, vol. 23, no. 3, pp. 343–367, 2015.

[24] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Collaborative multi-fidelity based surrogate models for genetic programming in dynamic flexible job shop scheduling," *IEEE Transactions on Cybernetics*, 2021. Doi: 10.1109/TCYB.2021.3050141.

[25] D. Yska, Y. Mei, and M. Zhang, "Genetic Programming Hyper-Heuristic with Cooperative Coevolution for Dynamic Flexible Job Shop Scheduling," in *Proceedings of the European Conference on Genetic Programming*. Springer, 2018, pp. 306–321.

[26] W. W. Daniel and Others, *Applied nonparametric statistics*. Houghton Mifflin, 1978.

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

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

[29] F. Zhang, Y. Mei, and M. Zhang, "A Two-Stage Genetic Programming Hyper-heuristic Approach with Feature Selection for Dynamic Flexible Job Shop Scheduling," in *Proceedings of the Genetic and Evolutionary Computation Conference*. IEEE, 2019, pp. 347–355.

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