Genetic Programming with Lexicase Selection for Large-scale Dynamic Flexible Job Shop Scheduling

Meng Xu, *Student Member, IEEE,* Yi Mei, *Senior Member, IEEE,*
Fangfang Zhang, *Member, IEEE,* and Mengjie Zhang, *Fellow, IEEE*

***Abstract*—Dynamic flexible job shop scheduling is a prominent combinatorial optimisation problem with many real-world applications. Genetic programming has been widely used to automatically evolve effective scheduling heuristics for dynamic flexible job shop scheduling. A limitation of genetic programming is the premature convergence due to the loss of population diversity. To overcome this limitation, this work considers using lexicase selection to improve population diversity, which has achieved success on regression and program synthesis problems. However, it is not trivial to apply lexicase selection to genetic programming for dynamic flexible job shop scheduling, since a fitness case (training scheduling simulation) is often large-scale, making the fitness evaluation very time-consuming. To address this issue, we propose a new multi-case fitness scheme, which creates multiple cases from a single scheduling simulation. Based on the multi-case fitness, we develop a new genetic programming algorithm with lexicase selection, which uses a single simulation for fitness evaluation, thus achieving a better balance between the number of cases for lexicase selection and evaluation efficiency. The experiments on a wide range of dynamic scheduling scenarios show that the proposed algorithm can achieve better population diversity and final performance than the current genetic programming parent selection methods and a state-of-the-art deep reinforcement learning method.**

***Index Terms*—dynamic flexible job shop scheduling, genetic programming, lexicase selection.**

I. Introduction

Job shop scheduling (JSS) [1] is an important combinatorial optimisation problem that has received extensive attention from scholars and industries. JSS aims to determine the best sequence for processing the job operations by a set of machines. Flexible JSS [2], [3] is an extension of JSS that considers the flexibility of machines. That is, each operation can be processed by a set of candidate machines rather than a fixed one. Flexible JSS needs to handle machine assignment (routing) and operation ordering (sequencing) simultaneously. It is known to be NP-hard [4]. In reality, scheduling is often with a large number of (e.g., thousands of) jobs and the environment changes dynamically by unpredictable real-time events, such as new job arrival [5] and machine breakdown [6], which is known as *dynamic flexible JSS* (DFJSS) [7]–[10]. In practice, DFJSS is usually large scale with many new job arrivals. Therefore, in this paper, we focus on such large-scale DFJSS problem with dynamic job arrivals.

Exact methods, such as mathematical programming [11]–[13] can guarantee solution optimality. However, their computational complexities are exponential to problem size, and are unsuitable for solving large-scale DFJSS problems, which require obtaining a solution in a very short time. The (meta-)heuristic solution optimisation methods, such as genetic algorithm [14]–[17] and tabu search [18], [19], are not suitable for solving DFJSS either, since it takes time for them to improve the solutions iteratively. Thus they cannot react to the dynamic environment in real-time. Scheduling heuristics are the most widely used methods in industries due to their advantages of being simple to implement and reacting quickly to dynamic environments [20]. The first-in-first-out, shortest processing time first, and most operations remaining first rules are some of the classical scheduling heuristics [21].

The effectiveness of scheduling heuristics varies from one scheduling scenario to another, and manually designing scheduling heuristics is time-consuming and highly relies on domain knowledge. Therefore, hyper-heuristic is proposed to automatically learn effective scheduling heuristics [22]. Genetic programming (GP) [23] is a promising hyper-heuristic approach for evolving effective scheduling heuristics for DFJSS [7], [24]. GP has many advantages over other optimisation algorithms. Firstly, GP has flexible representation and can explore various possible program structures [25]. Secondly, GP has good potential interpretability and allows users to analyse the program structure directly. Finally, the GP-evolved scheduling heuristics have good generalisability, thus can be reused in different situations [26].

An important limitation of GP for evolving scheduling heuristics is the premature convergence due to the lack of population diversity. It has been observed that GP tends to converge in the early stage of the evolution, and most promising individuals in the population have very similar program structures with each other [27], [28]. To overcome this limitation, *lexicase selection (LS)* [29] has been proposed to select parents that specialise in different fitness cases. On one hand, an individual is likely to be selected to be a parent regardless of its overall fitness (e.g., average error over all the fitness cases), as long as it performs well on some fitness cases. This can increase the diversity of the selected parents and thus the generated offspring. On the other hand, since each parent must specialise on some fitness cases, it is expected to have some promising building blocks that can be inherited by its

Manuscript received July 11, 2022; revised September 19, 2022, and December 26, 2022; accepted February 04, 2023. This work is supported in part by the Marsden Fund of New Zealand Government under Contract MFP-VUW1913 and by the MBIE SSIF Fund under Contract VUW RTVU1914. (Corresponding author: Fangfang Zhang.)

Meng Xu, Yi Mei, Fangfang Zhang, and Mengjie Zhang are with the Evolutionary Computation Research Group, School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand (e-mail: meng.xu@ecs.vuw.ac.nz; yi.mei@ecs.vuw.ac.nz; fangfang.zhang@ecs.vuw.ac.nz; mengjie.zhang@ecs.vuw.ac.nz).

offspring. In other words, LS selects a wider range of parents with different promising building blocks rather than randomly reducing the selection pressure. LS has been successfully applied to improve GP diversity and achieve better models for program synthesis [30], [31] and symbolic regression problems [29]. Thus, it is reasonable to extend the GP with LS to evolve scheduling heuristics as well.

However, it is non-trivial to extend the GP with LS from its successful domains such as program synthesis and symbolic regression to evolving scheduling heuristics, since the evaluation of a fitness case in DFJSS is very time-consuming. In program synthesis and symbolic regression, a fitness case typically consists of a given input and its target output. Evaluating a case fitness requires applying the model only once to generate the predicted outputs given the inputs. However, in DFJSS, a fitness case is usually a large-scale DFJSS simulation with thousands of jobs. Evaluating such a fitness case requires applying the GP model many times (each at a decision situation when a machine becomes idle or a job operation becomes ready to be processed) to generate a schedule given the dynamic job arrivals in the simulation. As a result, we cannot afford a large number of DFJSS fitness cases (i.e., simulations). On the other hand, if we use too few fitness cases, then the effectiveness of LS is negatively affected.

To address the above issues, we develop a GP with LS (GPLS) that applies LS effectively in GP without increasing complexity. Our contributions are summarised as follows:

- The newly proposed GPLS addresses the time-consuming evaluation of each case (simulation) by a new definition of case-fitness, which creates multiple cases from a single simulation. This way, it can use the LS efficiently without increasing computational cost.
- The newly developed multi-case fitness definition is not restricted to DFJSS, but can also provide more general guidelines on efficiently using LS for solving other complex problems with time-consuming fitness evaluations, such as dynamic vehicle routing and cloud resource allocation.
- The advantage of the proposed GPLS algorithm over the other GP algorithms and a state-of-the-art deep reinforcement learning algorithm[1] on a wide range of DFJSS scenarios demonstrates the efficacy of using LS in GP to increase the population diversity and better balance between exploration and exploitation.

The rest of this paper is organised as follows. Section II introduces the background. The proposed GPLS is described in Section III. Section IV gives the experiment design. The experimental results and analysis are shown in Section V. Then, further analyses are provided in Section VI. Finally, Section VII concludes the paper.

II. Background

A. Dynamic Flexible Job Shop Scheduling

In DFJSS, a set of jobs $\mathcal{J} = \{J_1, J_2, ..., J_n\}$ are to be processed by a set of machines $\mathcal{M} = \{M_1, ..., M_m\}$. Each job J_i has an arrival time r_i, a due date d_i, a weight w_i, and a sequence of operations $[O_{i,1}, O_{i,2}, ..., O_{i,p_i}]$ that must be processed in order, where p_i denotes the number of operations required for job J_i. Each operation $O_{i,j}$ has a workload $\pi_{i,j}$, and can be processed by an optional machine in $\mathcal{M}_{i,j} \subseteq \mathcal{M}$. The processing time $t_{i,j,k} = \pi_{i,j}/\gamma_k$ depends on the machine $M_k \in \mathcal{M}_{i,j}$ that processes the operation $O_{i,j}$, where γ_k is the processing speed/rate of the machine M_k. The machines are distributed, and there is a transport time τ_{k_1,k_2} to transport a job from machine M_{k_1} to M_{k_2}.

A solution (schedule) χ is an allocation of starting time $S_{i,j}$ for each operation $O_{i,j}$ on its selected machine $m_{i,j} \in \mathcal{M}_{i,j}$. The goal is to optimise some objectives. In this paper, we consider four objectives to be minimised, which are maximal flowtime ($Fmax$), mean flowtime ($Fmean$), maximal tardiness ($Tmax$), and maximal weighted tardiness ($WTmax$), which are defined as follows.

$$Fmax = \max_{i=1}^{n}\{C_i - r_i\},\ Fmean = \frac{1}{n}\sum_{i=1}^{n}(C_i - r_i)$$
$$Tmax = \max_{i=1}^{n}\{T_i\},\ WTmax = \max_{i=1}^{n}\{w_i T_i\}$$

where C_i and $T_i = \max\{C_i - d_i, 0\}$ are the completion time and tardiness of the job J_i.

Then, the problem can be formulated as a mixed integer linear programming model as follows [32]:

$$\begin{gathered}\min\ obj(\chi),\\ obj(\cdot) \in \{Fmax, Fmean, Tmax, WTmax\}\end{gathered} \tag{1}$$

s.t.:

$$S_{i,1} \geq r_i, \forall i = 1, \ldots, n \tag{2}$$

$$\begin{gathered}C_{i,j} \geq S_{i,j} + t_{i,j,k}x_{i,j,k}, \forall i = 1, \ldots, n,\\ \forall j = 1, \ldots, p_i, \forall k = 1, \ldots, m\end{gathered} \tag{3}$$

$$\begin{gathered}S_{i,j+1} \geq C_{i,j} + \tau_{k_1,k_2}z_{i,j,k_1,k_2}, \forall i = 1, \ldots, n,\\ \forall j = 1, \ldots, p_i - 1, \forall k_1, k_2 = 1, \ldots, m\end{gathered} \tag{4}$$

$$\begin{gathered}S_{i,j} \leq S_{a,b} - t_{i,j,k} + D \cdot (1 - y_{i,j,a,b,k}),\\ \forall i, a = 1, \ldots, n, \forall j, b = 1, \ldots, p_i, \forall k = 1, \ldots, m\end{gathered} \tag{5}$$

$$\sum_{k=1}^{m_{i,j}} x_{i,j,k} = 1, \forall i = 1, \ldots, n, \forall j = 1, \ldots, p_i \tag{6}$$

$$\begin{gathered}z_{i,j,k_1,k_2} \leq x_{i,j,k_1}, \forall i = 1, \ldots, n,\\ \forall j = 1, \ldots, p_i - 1, \forall k_1, k_2 = 1, \ldots, m\end{gathered} \tag{7}$$

$$\begin{gathered}z_{i,j,k_1,k_2} \leq x_{i,j+1,k_2}, \forall i = 1, \ldots, n,\\ \forall j = 1, \ldots, p_i - 1, \forall k_1, k_2 = 1, \ldots, m\end{gathered} \tag{8}$$

$$\begin{gathered}z_{i,j,k_1,k_2} \geq x_{i,j,k_1} + x_{i,j+1,k_2} - 1, \forall i = 1, \ldots, n,\\ \forall j = 1, \ldots, p_i - 1, \forall k_1, k_2 = 1, \ldots, m\end{gathered} \tag{9}$$

$$\begin{gathered}z_{i,j,k_1,k_2} \geq 0, \forall i = 1, \ldots, n, \forall j = 1, \ldots, p_i - 1,\\ \forall k_1, k_2 = 1, \ldots, m\end{gathered} \tag{10}$$

$$x_{i,j,k} = \begin{cases}1, \text{if } O_{i,j} \text{ is assigned to } M_k,\\ 0, \text{otherwise},\end{cases} \tag{11}$$

$$y_{i,j,a,b,k} = \begin{cases}1, \text{if } O_{i,j} \text{ is processed on } M_k \text{ before } O_{a,b},\\ 0, \text{otherwise},\end{cases} \tag{12}$$

[1]Due to page limit, the comparison with deep reinforcement learning is shown in the supplementary file.

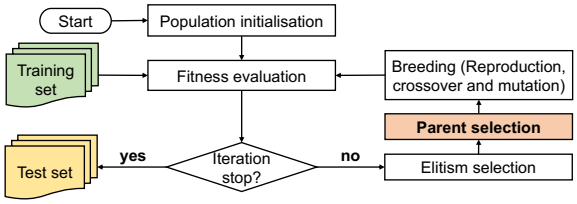


Fig. 1. The flowchart of the traditional GP training and test process.

$$z_{i,j,k_1,k_2} = \begin{cases} 1, \text{if } O_{i,j} \text{ is processed on } M_{k_1} \\ \quad \text{and } O_{i,j+1} \text{ is processed on } M_{k_2}, \\ 0, \text{otherwise}, \end{cases} \tag{13}$$

where D in Eq. (5) a sufficiently large enough constant. Eq. (2) means that the first operation of each job cannot start until the job is released. Eq. (3) indicates the relationship between the start time $S_{i,j}$ and completion time $C_{i,j}$ of $O_{i,j}$. Eq. (4) specifies that an operation $O_{i,j+1}$ cannot be started until its precedent operation $O_{i,j}$ has been completed and it has been transported to the allocated machine. Eq. (5) indicates that each machine can process at most one operation at a time. Eq. (6) requires each operation to be processed by one of its optional machines. Eqs. (7) - (10) defines the domain of the auxiliary variable z_{i,j,k_1,k_2}.

B. Related Work

Due to the dynamic characteristic of DFJSS, the common solution optimisation methods for classical JSS, e.g., exact methods, (meta-)heuristics, are no longer applicable to DFJSS due to their high computational cost. In contrast, scheduling heuristics (e.g., routing rule plus scheduling rule) are widely used for solving DFJSS, as they are easy to implement and can react in real time. However, manually designing scheduling heuristics needs domain knowledge, and is time-consuming.

To address this issue, there have been many GP algorithms proposed to automatically evolve scheduling heuristics. To evolve the routing rule and sequencing rule simultaneously, a multi-tree GP was proposed in [33]. Then, advanced machine learning and optimisation techniques were incorporated into GP to further improve the quality of the evolved scheduling heuristics. In [34], [35], surrogate models were used to help GP evaluate scheduling heuristics more efficiently. In [36], a niching-based feature selection strategy was designed to extract useful features for GP to evolve more effective scheduling heuristics. Some other studies focus on multi-task DFJSS [37], [38]. GP has received much attention in both single-objective and multi-objective DFJSS problems [39]–[42].

1) Genetic Programming Hyper-heuristics for DFJSS: GP hyper-heuristics can evolve high-quality heuristics automatically, and have achieved success in many problems such as bin packing [43], cloud computing [44], [45] and vehicle routing [46], [47]. GP has been widely used for evolving DFJSS scheduling heuristics (i.e., routing and sequencing rules) [33], [34], [36], [39], [40]. Unlike traditional solution optimisation approaches, GP hyper-heuristics follow a machine learning paradigm, consisting of *training* and *test* phases. In the training phase, GP evolves a population of scheduling heuristics,

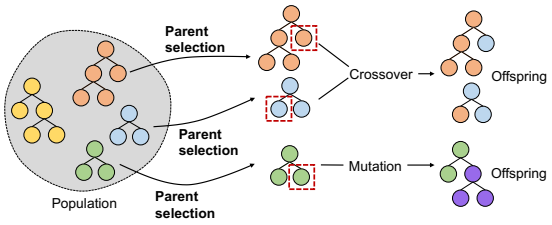


Fig. 2. The illustration of how GP generates offspring via crossover and mutation.

where the fitness evaluation is based on a set of training DFJSS simulations. In the test phase, a set of *unseen* test DFJSS simulations is used to evaluate the test/out-of-sample performance of the scheduling heuristics evolved by GP. Fig. 1 gives the flowchart of the GP training and test process.

Fig. 2 illustrates how GP generates offspring from the population at each generation. Specifically, it first selects individuals from the population as the parents (i.e., *parent selection*), and then applies the genetic operators such as crossover and mutation to the parents to generate offspring. The parent selection is a critical step to control the selection pressure and balance between exploration and exploitation of the GP process.

2) Parent Selection Schemes: In standard GP, tournament selection [48] is the most frequently used parent selection method. In tournament selection, firstly k (the tournament size parameter) individuals are randomly sampled from the population. Then, the individual with the best fitness among the k samples is selected as the parent. One can adjust the selection pressure through the k parameter value. A larger k value leads to a higher selection pressure. If $k = 1$, the tournament selection is reduced to random selection.

There have been many studies on tournament selection to analyse its behaviour and improve its effectiveness. Most of the early studies focused on sampling and selection pressure.

Tuning tournament size is the most straightforward strategy to control selection pressure in tournament selection [49]. In [50], a mathematical analysis of tournament selection was conducted to give an insight into the behaviour of tournament selection and the influence of the tournament size. However, changing tournament size can only affect the selection pressure at a coarse level. In [51], an alternative tournament selection is proposed. It employs an extra probability to be able to control the selection pressure at a fine level. In [49], the selection pressure of tournament selection under noisy environments is studied and the strong effect of selection pressure on population diversity and convergence rate is emphasised. In [52], an unbiased tournament selection is proposed to reduce sampling bias in tournament selection, which can eliminate the loss of diversity in tournament selection. A comprehensive study on selection pressure control in GP is conducted in [53], where an automatic selection pressure control approach for GP is proposed to dynamically adjust the individuals used as candidates for tournament selection. The results show that this selection pressure control approach can improve both the effectiveness and efficiency of GP. Xie et al. [54] investigate the impact of selection pressure on performance

by comparing the performance of standard parent selection with/without selection pressure, and it shows that giving a level of selection pressure in the parent selection process can significantly improve the performance of the algorithm. In [55], Xie et al. study the impact of the tournament selection with no replacement, and the results show that tournament selection with no replacement does not make the selection behaviour significantly different from that in the standard one. Furthermore, Xie et al. [56] present a novel approach that integrates the knowledge of the fitness rank distribution of a population into tournament selection to dynamically adjust the selection pressure. It shows that the new approach is effective and using the knowledge of the fitness rank distribution is a promising way to adjust the selection pressure dynamically.

There are other variations of selection schemes based on tournament selection. Cooperative selection [57] and rank-based selection [58] are two extensions of tournament selection, which select parents based on not only fitness but also some rank principles. The use of semantics in tournament selection [59] is also proposed to promote semantic diversity. The semantic-based tournament selection selects the individuals with good performance but with semantics substantially different from each other. This new approach has shown promising results on some regression problems. In [60], the use of statistical analysis of GP error vectors is introduced to create novel forms of tournament selection. The results show that the new selection schemes help to improve the semantic diversity and achieved significantly better performance than tournament selection and semantic-based selection [59] on regression problems.

Some studies focus on parent selection for crossover. The comparative partner selection [61] is proposed to select a pair of complementary parents, i.e., a pair of parents with strengths on different training instances. However, this method might ignore some high-quality pairs of parents due to the under-thought selection strategy. Diverse partner selection [62] is then proposed to overcome the defects of the comparative partner selection by proposing a smarter strategy. The above methods tend to select *generalist* parents that perform relatively well on all the training instances. However, the *specialist* individuals that perform very well on some training instances but with poor overall fitness are unlikely to be selected, and their special promising building blocks (e.g., sub-trees) will be lost in the subsequent generations.

To retain the promising information in the specialist individuals, Helmuth and Matheson [29] proposed the LS. The basic principle of LS is to select parents based on each training case separately rather than an aggregated function such as mean squared error. For example, given an order of the training cases, LS selects the individuals in the population that perform the best on the first case and moves on to the next case. This step is continued until only a single individual is selected. This way, a specialist individual has a chance to be selected if its specialised cases are ordered first. LS has shown its effectiveness on program synthesis problems [30], [31] and regression problems [63]. In the following sub-section, we focus on the review of the related work of LS.

3) Lexicase Selection: LS [64] has outperformed tournament selection across some benchmark problems [30], [31], [63]. The principle of LS is to remove the poor individuals from the whole population one by one based on a random sequence of the fitness cases/training instances, until only one individual remains (ties are broken randomly), which is the selected parent. For each fitness case, the individuals performing worse than the best individual in this case will be removed. Based on this, many different variants of LS methods are proposed, including ϵ-LS, random threshold LS, MADCAP ϵ-LS, and truncated LS [64]. They use different strategies to remove poor individuals.

Among them, ϵ-LS [63] is the most effective LS strategy for parent selection. In ϵ-LS, the requirements for removing individuals are somewhat relaxed compared to traditional LS methods. That is, an individual can still be retained if its fitness $fit(x)$ is not much worse than the best individual with fitness fit^* in the current fitness case, i.e., $fit(x) < fit^* + \epsilon$. The parameter ϵ is calculated based on the whole population, and can vary cross generations.

In summary, ϵ-LS is the method that can obtain the best solutions among the different variants of LS methods and is often used as a basis to propose new variations. Therefore, in this paper, we also consider to use ϵ-LS for solving DFJSS. In [65], ϵ-LS was used to solve the DFJSS problem for the first time, in which the training set is designed with many different instances by using different numbers of jobs and machines. However, they only consider small-scale instances, where the maximum number of jobs is no more than 100, and they obtain similar results with tournament selection. For evaluating the *steady-state* performance, a large-scale simulation with thousands of job arrivals is often required. In this case, the fitness evaluation for an instance becomes very time-consuming, making it not affordable to use a large number of training instances as fitness cases. Therefore, a new strategy needs to be developed to make ϵ-LS more suitable for solving the DFJSS problem.

III. Genetic Programming Hyper-heuristic with Lexicase Selection

A. Overall Framework

Fig. 3 shows the overall framework of GPLS. It starts with initialising a population of individuals, each as a scheduling heuristic. Then, in each generation, the newly proposed *multi-case fitness evaluation* is first applied to each individual to obtain both the standard fitness (that standard GP uses) and a case-fitness vector. The multi-case fitness evaluation will be described in detail in Section III-C. Afterwards, the elitism selection selects the elite individuals based on the standard fitness. Then, the *new parent selection* is applied to select parents to go through the breeding process and generate offspring by genetic operators. The new parent selection considers both tournament selection and LS. Specifically, for the first h generations, it uses tournament selection. Then, it switches to LS.

The use of the switching criterion is to tackle the hyper-selection issue [66] of LS, i.e., if an individual performs

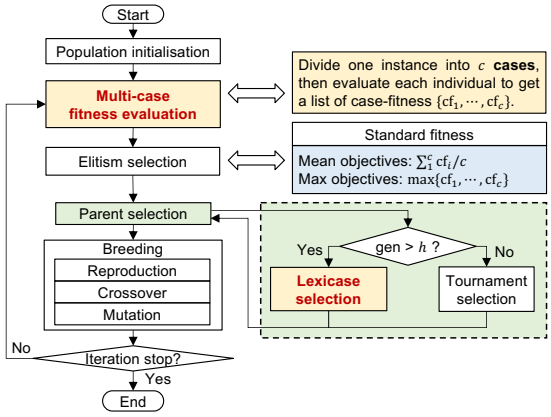


Fig. 3. The flowchart of the proposed GPLS algorithm.

significantly better than others on most cases, then it will be selected much more often than other individuals. This is more likely to occur at the early stage of the evolutionary process, where most of the individuals are randomly generated and the population is very diverse. In this case, it is likely that only a few elite individuals dominate most other individuals on most cases. LS tends to always select these elite individuals regardless of the order of the case, and the population diversity can be lost very fast. Therefore, we propose to use tournament selection in the first h generations, and then switch to LS when most individuals are good enough and can specialise on different cases.

B. Encoding and Decoding

Given a DFJSS instance I, a scheduling heuristic h can be used to obtain a solution/schedule χ. A scheduling heuristic consists of a routing rule and a sequencing rule, each encoded as a tree. The leaf nodes of the trees are sampled from the terminal set, while the non-leaf/inner nodes are sampled from the function set. The routing rule and the sequencing rule are priority functions to make routing and sequencing decisions during the simulation.

The decoding scheme used in this paper is based on the event trigger mechanism [67]. That is, when the preceding operation of an operation is completed, this operation becomes ready. Then, we calculate the priority of all the optional machines of the operation by the routing rule, and select the one with the highest priority to process this operation (routing decision). When a machine completes an operation, it becomes idle. We calculate the priority of the operations in its queue by the sequencing rule, and select the operation with the highest priority to process next (sequencing decision). Fig. 4 shows an example of a scheduling heuristic and the solution obtained by the decoding process on an instance with 2 jobs (each has 2 operations) and 2 machines. We assume that job J_1 arrives at the shop floor at time 0, the processing time of operation $O_{1,1}$ on machines M_1 and M_2 are 10 and 10, and the processing time of operation $O_{1,2}$ on machines M_1 and M_2 are 20 and 20. Job J_2 arrives at the shop floor at time 5, the processing time of operation $O_{2,1}$ on machines M_1 and M_2 are 20 and 20, and the processing time of operation $O_{2,2}$ on machines M_1 and M_2 are also 20 and 20. The transportation times between the entry/exit/machines are 0. The terminals MI, MR, and PT represent the index of the machine, the machine ready time, and the processing time, respectively. As we can see, the scheduling heuristic is encoded as two trees. When J_1 arrives at the shop floor at time 0, the routing rule $(PT + MR)/MI$ calculates the priorities of M_1 and M_2 for $O_{1,1}$ as $(10 + 0)/1 = 10$ and $(10 + 0)/2 = 5$. Then, M_2 with a smaller value is selected to process $O_{1,1}$ starting at time 0, and the ready time of M_2 changes to 10. When J_2 arrives at the shop floor at time 5, the routing rule $(PT + MR)/MI$ calculates the priorities of M_1 and M_2 for $O_{2,1}$ as $(20+0)/1 = 20$ and $(20+10)/2 = 15$. Then, M_2 with a smaller value is selected to process $O_{2,1}$. However, as M_2 is still processing $O_{1,1}$ at time 5, $O_{2,1}$ needs to wait for M_2 to become ready at time 10. After M_2 completes $O_{1,1}$ at time 10, it starts to process $O_{2,1}$, and its ready time changes to 30. Meanwhile, $O_{1,2}$ becomes ready at time 10 and is assigned to M_1 to be processed immediately. Finally, $O_{2,2}$ is assigned to M_2 after $O_{2,1}$ is completed at time 30, and is completed at time 50. The sequencing rule is not needed in this example as there is never more than one operation in a machine's queue. However, the principle to use the sequencing rule is the same as the routing rule.

C. Multi-case Fitness Evaluation

In DFJSS, one is typically interested in the "steady-state" performance of a scheduling heuristic, i.e., when the job shop state such as utilisation level becomes stable in the simulation. To this end, the DFJSS simulation needs to have a substantially large number (e.g., 5000) of job arrivals to be able to exclude the warm-up stage (e.g., the first 1000 job completions) and measure the "steady-state" performance of scheduling heuristics. As a result, even a single simulation is time-consuming. To balance training efficiency and generalisation, most existing GP approaches [34], [68]–[70] use a single simulation for fitness evaluation in each generation, but change to a new simulation (e.g., by changing the random seed) in each new generation. Specifically, the evaluated scheduling heuristic is applied to the simulation to generate a schedule, and its fitness is set to the objective value (e.g., mean flowtime, maximal tardiness) of its generated schedule. If considering each simulation as a fitness case (the case-fitness is the objective value of the generated schedule), then the traditional fitness evaluation with a single simulation is not applicable for LS, since there are not enough number of cases. To address this issue, we propose a new multi-case fitness evaluation strategy that can extract a large number of cases from a single simulation.

The multi-case fitness evaluation strategy is described in Algorithm 1. Specifically, we divide the jobs to be processed in the simulation into a number of groups. First, we index the jobs in the order of their arrivals in the simulation, i.e., job 1 is the first arrived job. This way, different evaluated individuals have the same jobs in each case, and their case-fitnesses are comparable. Then, we create each case based on a subset of jobs. If there are totally n jobs to be divided into c groups (assuming n can be divided by c), then each group contains $g = n/c$ jobs. Specifically, group i consists of the jobs with

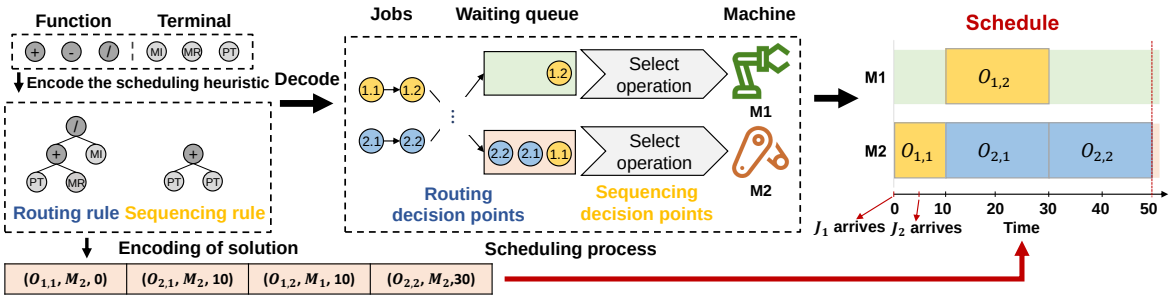


Fig. 4. An example of the encoding by a given scheduling heuristic and the decoding process for DFJSS.

Algorithm 1: Multi-case fitness evaluation for DFJSS.

Input: The individual to be evaluated: x; DFJSS simulation: sim; Number of jobs in the simulation: n; Number of cases: c.
Output: Standard fitness $sf(x)$; Case-fitness vector $\mathbf{cf}(x)$.

1 Calculate the number of jobs in each case $g = n/c$;
2 Run the simulation sim with the scheduling heuristic x to obtain the corresponding schedule $S = sim(x)$;
3 **for** $j = 1 \rightarrow n$ **do**
4 | Obtain the job completion time C_j from S;
5 **end**
6 **for** $i = 1 \rightarrow c$ **do**
7 | Set the ith group of jobs $\mathcal{J}_i \leftarrow \{g \times (i-1)+1, \ldots, g \times i\}$;
8 | Calculate the case-fitness $\mathbf{cf}_i(x) \leftarrow \texttt{obj}_{j \in \mathcal{J}_i}(C_j)$;
9 **end**
10 Calculate $sf(x)$ based on Eq. (16) or Eq. (17).;
11 **return** $sf(x)$, $\mathbf{cf}(x)$;

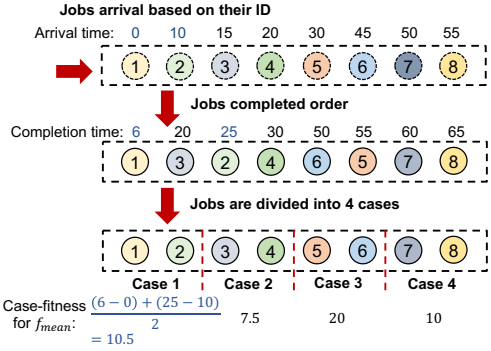


Fig. 5. An example of dividing 8 jobs in a simulation into 4 groups.

index from $g \times (i-1)+1$ to $g \times i$. Fig. 5 gives an example of dividing 8 jobs in a simulation into 4 cases. Each job has an arrival time and a completion time. Note that the jobs might not be completed in the same order they arrive (e.g., job 3 arrives after job 2, but is completed before it). The 8 jobs are divided into 4 cases/groups based on their arrival order. Then, we can calculate the objective function based on the jobs in each case. For example, if the objective is the mean flowtime, then the case-fitness of case i is calculated by Eq. (14).

$$\mathbf{cf}_i(x) = \frac{1}{g} \sum_{j=g\times(i-1)+1}^{g\times i} (C_j - r_j). \tag{14}$$

If the objective is the maximal tardiness, then the case-fitness of case i is calculated by Eq. (15).

$$\mathbf{cf}_i(x) = \max_{j \in \{g\times(i-1)+1,\ldots,g\times i\}} T_j. \tag{15}$$

The standard fitness can be either calculated directly from the job completion time, or from the case-fitnesses. For max-objectives, the standard fitness can be calculated by Eq. (16). For mean-objectives, the standard fitness can be calculated by Eq. (17).

$$sf_{max} = \max\{\mathbf{cf}_1, \mathbf{cf}_2, \cdots, \mathbf{cf}_c\}, \tag{16}$$

$$sf_{mean} = \frac{1}{c} \sum_{i=1}^{c} \mathbf{cf}_i. \tag{17}$$

D. New Parent Selection

The new parent selection selects parents by tournament selection in the first h generations, and then switches to LS afterwards. The traditional tournament selection is adopted, which first samples k individuals randomly, and then selects the one with the best standard fitness. The LS will be described in more details below.

The LS used in this paper is extended from the ϵ-LS [63]. First, a *candidate pool* of individuals is created by randomly sampled individuals from the population. In other words, the candidate pool is a random subset of the population. Then, the traditional ϵ-LS is applied to the candidate pool to select the parent. The use of the candidate pool borrows the advantages of limiting the selection pressure [54] of LS.

Algorithm 2 shows the LS for DFJSS. After shuffling the order of the cases (line 1) and creating the candidate pool Γ with p individuals (line 2), the individuals in the candidate pool are removed case by case. For each case a, the best case-fitness $\mathbf{cf}_a^*$ among all the individuals in the candidate pool is first identified. Then, the ϵ parameter is calculated for the current case by the function $\texttt{CalculateEpsilon}(\Gamma, a)$, which will be described in Algorithm 3. Then, all the individuals in the candidate pool whose case-fitness on a is worse than $\mathbf{cf}_a^* + \epsilon$ are removed from Γ. This individual removal is repeated for all the cases in the shuffled order ϕ, or until there is only a single individual left in Γ (line 11). If there are still multiple individuals left in Γ after enumerating all the cases, then the parent is randomly selected from Γ (line 14).

In Algorithm 2, the ϵ parameter is the threshold to accept slightly worse individuals than the best in a case. It is important to set a proper ϵ value to be neither too greedy nor too random. Here, we set the ϵ value based on the distribution

Algorithm 2: Lexicase selection for DFJSS.

Input: The population: pop; Number of cases: c; Candidate pool size: p.
Output: The selected individual: y.

Get a random permutation $\phi \leftarrow \texttt{shuffle}(1, ..., c)$;
Create the candidate pool Γ by randomly selecting p individuals with no replacement from pop;
for $j = 1 \rightarrow c$ **do**
  $a \leftarrow \phi[j]$;
  Get the best case-fitness $\mathbf{cf}_a^*$:

$$\mathbf{cf}_a^* \leftarrow \min\{\mathbf{cf}_a(x) \mid x \in \Gamma\};$$

  $\epsilon \leftarrow \texttt{CalculateEpsilon}(\Gamma, a)$;
  for $x \in \Gamma$ **do**
    if $\mathbf{cf}_a(x) > \mathbf{cf}_a^* + \epsilon$ **then** Remove x from Γ;
  end
  if *there is only one individual in* Γ **then**
    return $y \leftarrow \Gamma[1]$;
  end
end
Random select an individual y from candidates Γ;
return y;

Algorithm 3: $\texttt{CalculateEpsilon}(\Gamma, a)$.

Input: The candidate pool: Γ; index of current case: a.
Output: The epsilon: ϵ

Calculate the median of the the current case fitness of all the individuals in the candidate pool:

$$\mu \leftarrow \texttt{median}\{\mathbf{cf}_a(x) \mid x \in \Gamma\};$$

// Calculate the difference between each case fitness and the median
for $x \in \Gamma$ **do**
  $\delta(x) \leftarrow |\mathbf{cf}_a(x) - \mu|$;
end
Set ϵ to the median of all the differences

$$\epsilon \leftarrow \texttt{median}\{\delta(x) \mid x \in \Gamma\}$$

return ϵ;

of the case fitness of the individuals in the candidate pool [65]. It is described in Algorithm 3. First, we calculate the median of the case fitness of all the individuals in Γ on case a. Then, we calculate the difference between each case-fitness and the median. Finally, we set ϵ to the median of all such differences. By using the median of all the differences, we can have a more robust estimation on the variance of the case fitness in the population than by using a fixed ϵ value. Such advantage is more obvious in the GP for DFJSS, where the training instance is changed at each new generation, which can result in large noise in the fitness evaluation [69].

E. Breeding

The proposed algorithm uses reproduction, subtree mutation, and tree swapping crossover operators [33] to generate offspring. For reproduction, the parents are directly inherited to the next generation. For subtree mutation, a new subtree is randomly generated by sampling from the terminals and functions. We then randomly select a subtree from the parent and replace it with the newly generated subtree. For tree swapping crossover, for one tree, we randomly select a subtree from each parent and swap them. For another tree, we simply swap the entire tree. The process of the subtree mutation and tree swapping crossover are shown in Figs. 6 and 7.

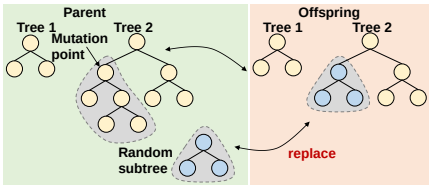


Fig. 6. The process of the subtree mutation.

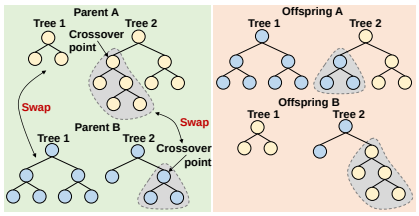


Fig. 7. The process of the tree swapping crossover.

IV. Experiment Design

A. Simulation Model

In the experiment, we use the simulation model [69] that assumes 5000 jobs need to be processed by 10 heterogeneous machines. The processing rate of each machine is randomly generated within the range $[10, 15]$. The travel time between machines and the travel time between the entry/exit point and each machine are sampled from a uniform discrete distribution between 7 and 100.

For DFJSS simulation, new jobs will arrive over time according to a Poisson process with a rate λ. Each job has a different number of operations that are randomly generated by a uniform discrete distribution between 2 and 10. In addition, the importance of jobs might be different, which is indicated by weights. The weights of 20%, 60%, and 20% of jobs are set as one, two, and four, respectively. The workload of each operation is assigned by uniform discrete distribution within the range $[100, 1000]$. The due date factor is set to 1.5, which means that the due date of a job is set to 1.5 times of its total processing time after its arrival time.

The utilisation level is a key factor in simulating different DFJSS scenarios. A higher utilisation level indicates a generally busier job shop scenario. In this paper, we consider eight scenarios based on four objectives and two utilisation levels (0.85 and 0.95). To ensure the accuracy of the collected data, warm-up jobs (the first 1000 jobs) were used to obtain typical scenarios that occur in the long-term simulation of the DFJSS system. Then, data were collected for the next 4000 jobs. The simulation stops after completing 6000 jobs. This way, we can almost make sure that the first arrived 4000 jobs have been completed. We generate a single replication of simulation, but change the random seed at each generation of GP to improve the generalisability of the evolved scheduling heuristics [68].

B. Parameter Setting

In the experiments, the terminal and function sets of GP are shown in Table I. The terminal set consists of the features

TABLE I
THE GP TERMINAL AND FUNCTION SET FOR DFJSS.

Notation	Description
NIQ	the number of operations in the queue
WIQ	the work in the queue
MWT	the waiting time of the machine = t - MRT
PT	the processing time of the operation
NPT	the median processing time for next operation
OWT	the waiting time of the operation = t - ORT
WKR	the work remaining
NOR	the number of operations remaining
W	the job weight
TIS	time in system = t - releaseTime
TRANT	the transportation time
Function	$+, -, \times, /, max, min$

TABLE II
THE PARAMETER SETTINGS OF GP.

Parameter	Value
Population size	1024
Number of generations	51
Method for initialising population	ramped-half-and-half
Initial minimum/maximum depth	2 / 6
Elitism	10
Maximal depth	8
Crossover rate	0.80
Mutation rate	0.15
Reproduction rate	0.05
Terminal/non-terminal selection rate	10% / 90%
Parent selection	Lexicase/Tournament selection
Switching criterion h	5
The number of cases c	10, 25, 40, 50, 80, 100
Candidate pool size p	10, 20, 30, 40, 50, 60, 70, 80 100, 200, 400, 600, 800, 1024

related to machines (e.g., NIQ, WIQ, and MWT), operations (e.g., PT, NPT, and OWT), jobs (e.g., WKR, NOR, W, and TIS), and transport (e.g., TRANT). In the function set, the arithmetic operators take two arguments. The "/" operator is protected and returns 1 if divided by zero. The max and min functions take two arguments and return the maximum and minimum of their arguments, respectively. The proposed algorithm is implemented based on an open-source platform called ECJ [71]. The GP parameter settings are shown in Table II. In the experiments, we set the minimal number of cases to 10 so that there are enough cases for the LS to play a reasonable role. The maximal number of cases is set to 100 (so that each case contains at least 40 jobs) to control the uncertainty of the case-fitness.

C. Comparison Design

The proposed GPLS algorithm has two important parameters, which are the number of cases (c) and candidate pool size (p). At the beginning, a sensitivity analysis is done to select the best pair of parameters (c, p). Then, to verify the effectiveness of the proposed GPLS algorithm, we compare it with the following algorithms:

1) **GP7**: The traditional GP approach that uses tournament selection with size 7. This is the standard GP hyper-heuristic for DFJSS.
2) **GP4**: The traditional GP approach that uses tournament selection with size 4. It is known that tournament size can control the selection pressure of tournament selection [55]. By reducing the tournament size from 7 to 4, the traditional GP can have a better population diversity.
3) **GPLS**^{i}: The GPLS with multiple **i**nstances [65]. In the fitness evaluation, it uses c different instances, each of which is a short simulation with $1000/c$ warm-up jobs, and stops after completing the next $4000/c$ jobs.
4) **GPm**: GP with **m**ulti-case fitness evaluation but without LS. It simply sets the fitness to the mean of all the case-fitnesses. Note that GPm is also a proposed algorithm in this paper that only uses the multi-case fitness. The comparison with GPm can help find whether LS works.
5) **STS**^{r}: The GP with Semantic Tournament Selection [60]. It uses the case-fitness vector, and selects the individual that is significantly better than the other based on the Wilcoxon rank sum test. The tie is broken **r**andomly.
6) **STS**^{s}: The GP with Semantic Tournament Selection [60]. It uses the case-fitness vector, and selects the individual that is significantly better than the other based on the Wilcoxon rank sum test. The tie is broken by selecting the individual with **s**maller program size.

To verify different hypotheses, we form the following comparisons:

1) Comparison with GP7, GP4, and GPLS^i: This can verify the effectiveness of the proposed multi-case fitness evaluation strategy.
2) Comparison with GPm, STS^r, and STS^s: This can verify the effectiveness of the LS under the proposed multi-case fitness evaluation.

Each compared algorithm is run 30 times independently to train the scheduling heuristics for each scenario. To evaluate the test performance of the trained scheduling heuristics, we use 30 unseen simulations in the test set, each with 1000 warm-up jobs and 5000 jobs afterwards. The *test performance* is defined as the average objective value of the schedules generated by the scheduling heuristics over the 30 test simulations.

V. RESULTS AND DISCUSSIONS

A. Sensitivity Analysis

To have a comprehensive sensitivity analysis of the c and p parameters, we consider a wide range of parameter values, i.e., $c \in \{10, 25, 40, 50, 80, 100\}$ and $p \in \{10, 20, 30, 40, 50, 60, 70, 80, 100, 200, 400, 600, 800, 1024\}$. Note that when $p = 1024$ (the entire population), the LS is equivalent to the traditional ϵ-LS.

Fig. 8 shows the heat map of the test performance of GPLS with different (c, p) parameter values on the 8 test scenarios. The darker (blue) colour indicates smaller values (better test performance), while the lighter (yellow) colour indicates larger values (worse test performance). From Fig. 8, we can see that a large number of cases and small candidate pool size (bottom right region) tend to lead to poor performance, while a relatively small number of cases (e.g, 25∼50) and a relatively large candidate pool size (e.g., 400∼800) tend to result in promising test performance. Different scenarios show some different distributions of the test performance. For example,

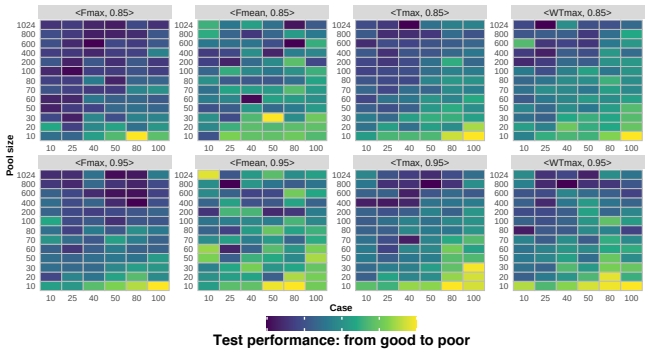


Fig. 8. Heat map of test performance of GPLS with different numbers of case and pool size on eight scenarios.

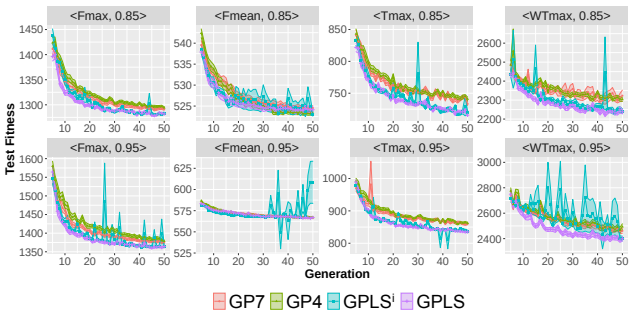


Fig. 9. Convergence curves of the test performance of the compared methods on the 8 scenarios.

the colors of the <Fmax, 0.85> and <Fmax, 0.95> scenarios are much darker than the <Fmean, 0.85> and <Fmean, 0.95> scenarios. However, the best regions for all the scenarios are consistently located around the top left.

In addition, we compare each parameter setting with the baseline GP7, using the Wilcoxon rank sum test with a significance level of 0.05. We found that 9 parameter settings, i.e., (10, 200), (25, 80), (25, 200), (25, 400), (25, 600), (25, 800), (40, 200), (40, 800) and (80, 600), can obtain significantly better than GP7 on 7 out of the 8 test scenarios. Among these 9 settings, (25, 800) performed slightly better than others. Therefore, we select $c = 25$ and $p = 800$ for GPLS in the subsequent experiments.

B. Test Performance

In this section, we conduct the two comparisons in terms of test performance to verify the effectiveness of the newly proposed multi-case fitness evaluation and LS, respectively.

1) Effectiveness of Multi-Case Fitness Evaluation: To verify the effectiveness of the multi-case fitness evaluation strategy, we compare GPLS with GP7, GP4, and GPLS^i. GP7 is the baseline GP with the standard tournament selection. GP4 uses the tournament selection but has a better population diversity by reducing the tournament size. GPLS^i uses a case-fitness list, where each case-fitness is based on a set of short instances/simulations.

Table III shows the mean (standard deviation) of the test performance of the 30 independent runs of GP7, GP4, GPLS^i, and GPLS on the 8 scenarios. We have also conducted the Wilcoxon rank sum test with a significance level of 0.05 to make pairwise comparisons (each algorithm is compared with all the algorithms to its left in the table). The "↑/↓/=" after each entry in the table indicates that the corresponding results are significantly better/worse than or similar to the results of the compared algorithm. For example, for the <Fmax, 0.85> scenario, the (↑)(↑)(=) for GPLS indicates that GPLS performed significantly better than GP7 and GP4, and there is no statistical difference between GPLS and GPLS^i.

From the table, we can see that GPLS obtains significantly better performance than GP7 and GP4 on 7 out of the 8 test scenarios. GPLS performs similarly to GPLS^i on all the scenarios. In addition, GPLS^i performed significantly better than GP7 and GP4 on 6 out of the 8 test scenarios, which is slightly worse than GPLS. The advantages of GPLS and GPLS^i over GP7 and GP4 verify the effectiveness of the use of multi-case fitness instead of a single aggregated fitness in parent selection for evolving DFJSS scheduling heuristics.

Fig. 9 shows the convergence curves of the test performance of GP7, GP4, GPLS^i, and GPLS on the 8 scenarios. From the figure, we can see that GPLS has a faster convergence than GP7 and GP4, as its convergence curves are almost always below that of GP7 and GP4 for all the scenarios. This verifies the effectiveness of the proposed multi-case fitness selection strategy. On the other hand, although GPLS^i showed competitive final test performance in Table III, its convergence curves in Fig. 9 are shown to be quite fluctuating and unstable, especially in the <Fmean, 0.95> scenario, where it showed worse test performance and large variance. A possible reason is that GPLS^i uses several small-scale instances (160 jobs and 40 warm-up jobs) for evaluation, and these instances are all from the beginning stage of the scheduling system and the small number of warm-up jobs can not simulate a stable scheduling system very well. This leads to overfitting and worse generalisation of the evolved scheduling heuristics.

Overall, the advantages of GPLS over GP7, GP4, and GPLS^i verify the effectiveness of the proposed multi-case fitness evaluation.

2) Effectiveness of LS: To verify the effectiveness of the use of LS, we compare GPLS with GPm, STS^r, and STS^s. All the compared algorithms employ the multi-case fitness, but have different parent selection schemes. GPm simply averages the case-fitnesses into a single fitness, and uses the tournament selection. For the mean-objectives, it becomes equivalent to the baseline GP. However, for the max-objectives, its fitness is different from the standard fitness. STS^r and STS^s select parents by conducting statistical significance test on the case-fitness lists. They are also state-of-the-art GP parent selection methods that improve population diversity.

Table IV shows the mean (standard deviation) of the test performance of the 30 independent runs of GP7, GPm, STS^r, STS^s, and GPLS on the 8 scenarios. In addition, the results of GP7 are also given as a reference. From the table, it can be seen that GPLS achieved significantly better performance than GPm on 2 scenarios, and is comparable with GPm on all the remaining scenarios. It significantly outperformed STS^r and STS^s for all the 8 scenarios. This verifies the effectiveness of the proposed LS.

TABLE III
THE MEAN (STANDARD DEVIATION) OF THE TEST PERFORMANCE OF THE 30 INDEPENDENT RUNS OF **GP7, GP4, GPLS^i, GPLS** ON 8 SCENARIOS.

Scenario	GP7	GP4	GPLS^i	GPLS
<Fmax, 0.85>	1291.40(12.62)	1294.29(10.08)(=)	1280.10(10.95)(↑)(↑)	1282.77(13.77)(↑)(↑)(=)
<Fmax, 0.95>	1375.33(16.77)	1378.48(16.63)(=)	1362.07(18.37)(↑)(↑)	1359.44(13.49)(↑)(↑)(=)
<Fmean, 0.85>	524.54(2.93)	523.28(2.36)(↑)	523.07(2.07)(↑)(=)	523.46(3.56)(↑)(=)(=)
<Fmean, 0.95>	566.69(3.23)	567.02(3.67)(=)	596.42(175.53)(=)(=)	565.84(2.62)(=)(=)(=)
<Tmax, 0.85>	733.27(14.69)	740.84(16.44)(↓)	722.33(14.49)(↑)(↑)	717.05(13.34)(↑)(↑)(=)
<Tmax, 0.95>	860.42(19.09)	861.83(17.92)(=)	835.51(13.01)(↑)(↑)	831.09(11.74)(↑)(↑)(=)
<WTmax, 0.85>	2323.13(161.67)	2239.42(45.33)(=)	2224.77(56.65)(↑)(↑)	2240.34(65.35)(↑)(↑)(=)
<WTmax, 0.95>	2460.96(100.40)	2494.17(123.86)(=)	2443.69(117.17)(=)(↑)	2398.00(110.64)(↑)(↑)(=)

TABLE IV
THE MEAN (STANDARD DEVIATION) OF THE TEST PERFORMANCE OF THE 30 INDEPENDENT RUNS OF **GP7, GPM, STS^r, STS^s, GPLS** ON 8 SCENARIOS.

Scenario	GP7	GPm	STS^r	STS^s	GPLS
<Fmax, 0.85>	1291.40(12.62)	1282.42(12.36)(↑)	1317.59(19.20)(↓)(↓)	1372.22(65.59)(↓)(↓)(↓)	1282.77(13.77)(↑)(=)(↑)(↑)
<Fmax, 0.95>	1375.33(16.77)	1367.34(19.42)(↑)	1406.85(22.18)(↓)(↓)	1488.40(81.23)(↓)(↓)(↓)	1359.44(13.49)(↑)(↑)(↑)(↑)
<Fmean, 0.85>	524.54(2.93)	523.87(3.38)(=)	526.20(4.51)(=)(↓)	532.30(6.40)(↓)(↓)(↓)	523.46(3.56)(↑)(=)(↑)(↑)
<Fmean, 0.95>	566.69(3.23)	567.73(3.27)(=)	570.20(5.02)(↓)(↓)	578.63(3.84)(↓)(↓)(↓)	565.84(2.62)(=)(↑)(↑)(↑)
<Tmax, 0.85>	733.27(14.69)	720.74(15.91)(↑)	763.96(19.89)(↓)(↓)	835.54(46.86)(↓)(↓)(↓)	717.05(13.34)(↑)(=)(↑)(↑)
<Tmax, 0.95>	860.42(19.09)	830.88(11.50)(↑)	883.68(23.94)(↓)(↓)	971.03(59.00)(↓)(↓)(↓)	831.09(11.74)(↑)(=)(↑)(↑)
<WTmax, 0.85>	2323.13(161.67)	2250.66(64.28)(↑)	2358.00(64.15)(↓)(↓)	2464.33(98.52)(↓)(↓)(↓)	2240.34(65.35)(↑)(=)(↑)(↑)
<WTmax, 0.95>	2460.96(100.40)	2428.43(79.14)(=)	2586.20(117.66)(↓)(↓)	3434.35(412.91)(↓)(↓)(↓)	2398.00(110.64)(↑)(=)(↑)(↑)

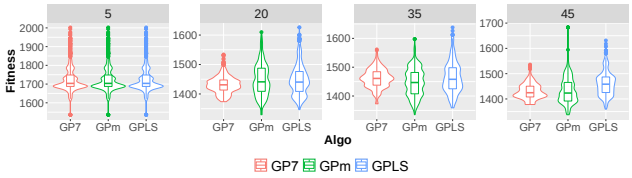


Fig. 10. The violin plots of the standard fitness of the selected parents of GP7, GPm, and GPLS at generation 5, 20, 35, and 45 in scenarios <Fmax, 0.85> of a single run.

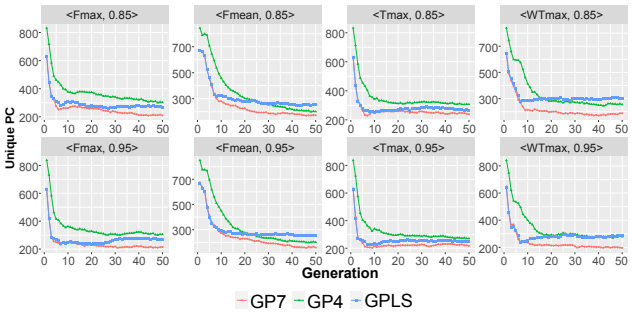


Fig. 11. Convergence curves of unique PCs of GPLS and comparison methods on eight scenarios.

GPm obtained significantly better performance than GP7 on 5 scenarios, which shows that even averaging the case-fitness can improve the performance, especially on the max-objectives. This is because the multi-case fitness evaluation uses more information, e.g., the completion time of c jobs (each for a case) rather than a single job with the maximal objective value. This can improve the generalisation. This might also be one of the reasons that GPLS performs well. However, STS^r and STS^s perform significantly worse than GP7 on almost all the 8 scenarios. This is contradictory to the results in [60], where STS^r and STS^s performed better than tournament selection and semantic in selection [59] for solving regression problems. A possible reason is that the DFJSS problem is more complex than regression problems and the strategy to select parents based on the statistical test might give poor individuals high probabilities to be selected in DFJSS.

In summary, the above two comparisons with different sets of other existing GP approaches verify the effectiveness of the proposed GPLS, and the newly proposed multi-case fitness evaluation and LS.

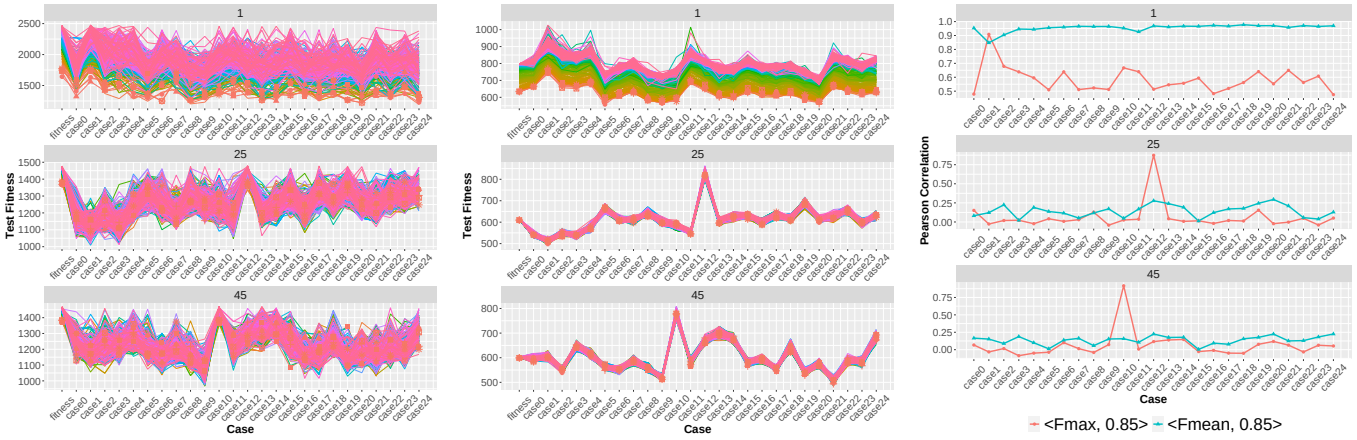
C. Distribution of Parent Fitness

Using the multi-case fitness evaluation instead of the standard fitness is expected to reduce the selection pressure and allow more specialist individuals with worse standard fitness to be selected as parents.

To analyse such behaviour of the multi-case fitness evaluation, we plot the distribution of the standard fitness of the selected parents in a run of GP7, GPm, and GPLS. Fig. 10 shows the distributions at generation 5, 20, 35, and 45 in the <Fmax, 0.85> scenario. Note that all the three algorithms use the tournament selection on the standard fitness until generation 5. Thus, They have exactly the same process in the first generation, and the same distribution of parent fitness at generation 5. After that, we can see that both GPm and GPLS achieved a wider-spread distribution than GP7. An interesting pattern is that GPm and GPLS were able to select parents with better standard fitness than GP7. This indicates that the multi-case fitness evaluation can help the algorithms generate better individuals, which can then be used as parents in subsequent generations.

D. Phenotypic Diversity

Besides the fitness, the phenotypic characterisation (PC) [72] is a more comprehensive representation of the behaviour of a GP rule. The PC of a GP rule is represented as a numeric



(a) <Fmax, 0.85> (b) <Fmean, 0.85> (c) Pearson correlation

Fig. 12. The standard fitness and case-fitness of the top 400 individuals of GPLS at generation 1, 25 and 45 in the <Fmax, 0.85> and <Fmean, 0.85> scenarios, as well as their Pearson correlation between the standard fitness.

vector, where each dimension is the decision made by the rule under a decision situation (e.g., the index of the selected candidate by an idle machine from its queue).

Fig. 11 shows the convergence curves of number of unique PCs in the population in GP7, GP4 and GPLS on the 8 scenarios. From the figure, we can see that GP4 always contains more unique PCs in the population than GP7 for all the scenarios, which is expected. The convergence curves of GPLS and GP7 overlap with each other in the first 5 generations due to the switching criterion. After generation 5, GPLS starts to have more unique PCs than GP7. Compared to GP4, GPLS has more unique PCs on three scenarios (<Fmean, 0.85>, <Fmean, 0.95>, <WTmax, 0.85>) in the later stages of the evolutionary process, while has fewer unique PCs on four scenarios (<Fmax, 0.85>, <Fmax, 0.95>, <Tmax, 0.85>, <Tmax, 0.95>). Considering Table III, where GPLS significantly outperformed GP4 although it has slightly lower phenotypic diversity, we can see that increasing phenotypic diversity, although important, is not the only reason for the performance improvement in GPLS.

VI. Further Analysis

A. Correlation between Case-Fitnesses

For LS based on the case-fitness list, an important factor is the correlation between the fitnesses of different cases. Ideally, the population should contain individuals that specialise at different cases to increase the selection diversity. If the case-fitnesses are highly correlated, the selected parent will be the same regardless of the order of the case. That is, if an individual A is better than B on one case, it is highly likely to be better on other cases. As a result, LS has no advantage over the tournament selection.

To investigate the correlation between case-fitnesses, we plot the case-fitnesses of the top 400 individuals in the population of GPLS (that are most likely to be selected by LS) in the early stage (generation 1), middle stage (generation 25) and late stage (generation 45) for the <Fmax, 0.85> and <Fmean, 0.85> scenarios. Figs. 12(a) and 12(b) show the results, where each individual corresponds to a line and the standard fitness of the individual is also given on the most left as a reference.

From the figures, we can see that in both scenarios, initially there are a few individuals that perform well on almost all the cases (their lines are below the others). However, as the evolution proceeds, the lines of different individuals become more mixed together, and it is difficult to see any individual's line always below that of the others for all the cases. This suggests that we can select different individuals under different orders of cases. In addition, we see that the distribution of the case-fitness has a higher variance in the <Fmax, 0.85> scenario than that in the <Fmean, 0.85> scenario. This is expected since the Fmax objective is more sensitive to outliers (the job with the maximal flowtime).

We also calculate the Pearson correlation between the standard fitness and each case-fitness of the top 400 individuals at generations 1, 25, and 45, as shown in Fig. 12(c). From the figure, we can see that in generation 1, all the case-fitnesses are very highly correlated with the standard fitness. This is consistent with our observations in Figs. 12(a) and 12(b). However, generations 25 and 45 show different patterns. Specifically, in the <Fmax, 0.85> scenario, all the cases except case 11 in generation 25 and case 9 in generation 45 have low correlations with the standard fitness. This implies that the standard fitness is mostly determined by a single case, which contains the job with the largest flowtime. In the <Fmean, 0.85> scenario, all the cases have low correlations with the standard fitness. This suggests that we can select a wide range of specialist individuals by LS.

B. Frequency of Case Usage

Different cases might have different abilities to distinguish promising individuals. For example, if a case usually leads to ties between individuals, then other subsequent cases are needed to further identify the best individual. In this sense, different cases might be used at different frequencies.

To investigate this issue, we plot the frequency that each case is used by GPLS at generations 5, 20, 35, and 45 in the <Fmax, 0.85> scenario, as shown in Fig. 13. It can be

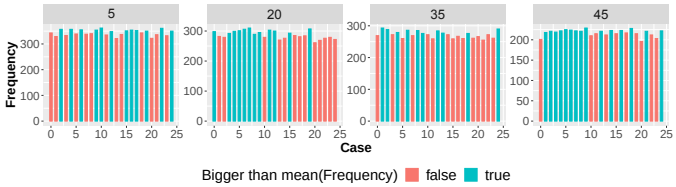


Fig. 13. The frequency of the cases used by GPLS at generations 5, 20, 35, and 45 in the <Fmax, 0.85> scenario.

TABLE V
THE MEAN (STANDARD DEVIATION) OF THE SIZE OF THE EVOLVED BEST **ROUTING RULES** OF 30 INDEPENDENT RUNS OF GP7 AND GPL FOR THE 8 SCENARIOS.

Scenario	GP7	GPLS
<Fmax, 0.85>	36.53(11.65)	52.60(17.51)(+)
<Fmax, 0.95>	37.87(14.51)	53.33(16.67)(+)
<Fmean, 0.85>	52.27(15.39)	55.73(18.48)(=)
<Fmean, 0.95>	50.47(16.58)	48.33(12.14)(=)
<Tmax, 0.85>	37.27(13.31)	46.60(13.78)(+)
<Tmax, 0.95>	33.07(17.33)	44.40(14.88)(+)
<WTmax, 0.85>	40.80(14.03)	46.00(12.37)(=)
<WTmax, 0.95>	39.60(20.47)	47.33(10.35)(+)

TABLE VI
THE MEAN (STANDARD DEVIATION) OF THE SIZE OF THE EVOLVED BEST **SEQUENCING RULES** OF 30 INDEPENDENT RUNS OF GP7 AND GPLS FOR THE 8 SCENARIOS.

Scenario	GP7	GPLS
<Fmax, 0.85>	42.60(15.86)	44.73(14.48)(=)
<Fmax, 0.95>	44.00(14.48)	48.47(11.86)(=)
<Fmean, 0.85>	33.67(12.14)	32.27(17.88)(=)
<Fmean, 0.95>	39.73(17.82)	29.33(13.78)(=)
<Tmax, 0.85>	48.40(17.89)	50.80(16.96)(=)
<Tmax, 0.95>	41.53(14.75)	50.67(15.08)(=)
<WTmax, 0.85>	26.27(22.64)	40.87(19.90)(+)
<WTmax, 0.95>	28.40(12.53)	45.87(19.73)(+)

seen that different cases have very similar frequencies to each other. This indicates that all the cases have similar abilities to distinguish the individuals, which is a good sign.

C. Rule Size

In this section, we investigate how the proposed multi-case fitness evaluation strategy and LS affect the size of the evolved rules by comparing GPLS with GP7 (baseline GP). Tables V and VI show the mean and standard deviation of the sizes of the evolved best routing rules and sequencing rules of GPLS and GP7 (baseline GP) on the 8 scenarios. In the tables, the "+/−/=" after each entry indicates that the corresponding rule size obtained by GPLS is significantly larger/smaller than or similar to the size obtained by GP7.

From Table V, we can see that GPLS obtained significantly larger routing rule size than GP7 on 5 scenarios (<Fmax, 0.85>, <Fmax, 0.95>, <Tmax, 0.85>, <Tmax, 0.95>, <WTmax, 0.95>) and there is no statistical significant difference on the other scenarios. Table VI shows that GPLS obtained significantly larger sequencing rule size than GP on 2 scenarios (<WTmax, 0.85>, <WTmax, 0.95>) and similar on the other scenarios.

The above results suggest that GPLS tends to allow larger GP rules (especially routing rules) to survive, which might be a reason of the superior performance of GPLS.

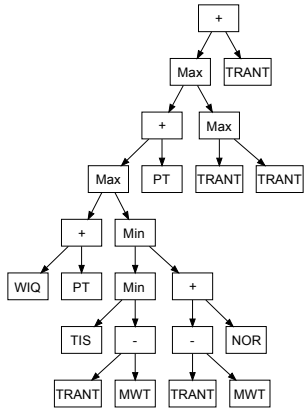


Fig. 14. An example of the evolved **routing rule** by GPLS on scenario <Fmax, 0.85>.

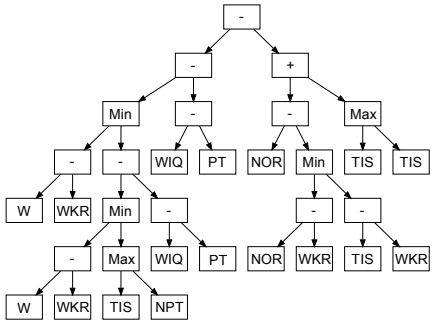


Fig. 15. An example of the evolved **sequencing rule** by GPLS on scenario <Fmax, 0.85>.

D. Insight on the Evolved Scheduling Heuristics

Figs. 14 and 15 give the tree structures of the routing rule and the sequencing rule from a scheduling heuristic evolved by GPLS on the <Fmax, 0.85> scenario.

The routing rule is a combination of six terminals (TRANT, MWT, PT, NOR, WIQ, and TIS), where TRANT is the most frequently used terminal (used 5 times). It is followed by MWT and PT, which are used 2 times. NOR, WIQ, and TIS, on the other hand, are used only once. The routing rule (Fig. 14) can be simplified to R_0 as shown in Eq. (18).

$$\begin{aligned} R_0 = \max\{\max\{WIQ + PT, \min\{\min\{TIS, \\ TRANT - MWT\}, TRANT - MWT \\ + NOR\}\} + PT, TRANT\} + TRANT \end{aligned} \tag{18}$$

If the operation has been waiting for a long time (a large TIS), this rule can be further simplified as $R_1 \approx \max\{TRANT + WIQ + 2PT, 2TRANT - MWT + PT, 2TRANT\}$. For a candidate machine with an empty queue (WIQ=MWT=0), R_1 can be further simplified as $R_2 \approx \max\{TRANT + 2PT, 2TRANT + PT\}$, which means that an idle machine with a smaller transportation time (TRANT) and a smaller processing time (PT) is preferred. For a candidate machine with a busy queue (WIQ is large) of the machine (large WIQ), R_1 can be further simplified as $R_3 \approx TRANT + WIQ + 2PT$, which prefers small transportation time, small work in the queue, and small processing time. In other words, this routing rule tends to select machines with

smaller transportation time (TRANT), smaller processing time (PT), and smaller work remaining in the waiting queue (WIQ).

The sequencing rule is a combination of seven terminals (WKR, TIS, WIQ, PT, NOR, W, and NPT). WKR and TIS are the most frequently used terminals, which are both used 4 times. It is followed by WIQ, PT, NOR, and W, which are used 2 times. NPT is used only once. The sequencing rule (Fig. 15) can be simplified to S_0 as shown in Eq. (19).

$$\begin{aligned} S_0 = &\min\{W - WKR, \min\{W - WKR, \max\{TIS, \\ &NPT\}\} - (WIQ - PT)\} - (WIQ - PT) - \\ &NOR + \min\{NOR - WKR, TIS - WKR\} \\ &- \max\{TIS, TIS\} \end{aligned} \tag{19}$$

This rule shows that the selection strategy is mainly based on the weight of the job (W), the processing time of the operation (PT), the work remaining of the operation (WKR), and the time that the operation has been waiting (TIS). Note that in most practical decision situations, the subtree $W - WKR$ outputs a smaller value than the subtree $\max\{TIS, NPT\}$, the number of operations remaining (NOR) is always smaller than TIS, and the work remaining in the waiting queue (WIQ) is always the same. Therefore, this sequencing rule can be further simplified as $S_1 \approx W + 2PT - 2WKR - TIS$. In other words, this rule tends to select operations with a smaller weight (W), a smaller processing time (PT), a larger work remaining (WKR), and a larger time in system (TIS).

Overall, we can see that for the routing decisions, the attributes of the machines, such as the transportation time and processing time (related to the processing rate of machines) play decisive roles. For the sequencing decisions, the information of operations, such as the processing time (related to the workload of operations), time in the system, and work remaining, play decisive roles. This phenomenon is consistent with our intuition that for the max-flowtime objective, the machines that are not busy and have short transportation time are good choices to process ready operations, and the operations that stay in the system for a long time and have much remaining work should be completed as soon as possible.

E. Further Discussions

In [73], it is argued that the LS is good for solving modal problems, which is defined as follows: "*a problem is modal if a solution has to do something qualitatively different in different circumstances, that is, on inputs from different classes.*" The symbolic regression problems are proven to be modal problems [73].

DFJSS is more complex than symbolic regression, with dynamic job arrival events and complex system states. In order to improve training efficiency, we divide one instance of DFJSS into multiple cases to obtain a list of case-fitnesses, which means that different cases might not be independent. If this occurs, then the scheduling heuristics will not be qualitatively different in different circumstances (cases). Our analysis on the parents' fitness in Section V-C showed that the scheduling heuristics do give different scheduling solutions in different circumstances (cases), and the proposed GPLS works on the DFJSS problem. Therefore, we conservatively hypothesise that DFJSS can be considered as a modal problem, but more experiments and analyses should be performed to verify it in the future. In addition, considering the analysis in Section VI-A, GPLS performs significantly better, especially on max-objectives is due to the scheduling heuristics performing more differently on max-objectives than those on mean-objectives, which is consistent with the description in [73].

VII. Conclusions

The goal of this paper was to propose a GP with LS for evolving better scheduling heuristics of the large-scale DFJSS problem with thousands of jobs. This goal has been achieved by proposing a GPLS method with a new multi-case fitness evaluation strategy. The multi-case fitness evaluation strategy divides a large DFJSS simulation into multiple cases to extract a list of case-fitnesses. This is more efficient than directly evaluating individuals on multiple instances. The case-fitness list can be used by LS for selecting parents. The effectiveness of the proposed GPLS is verified by comparing it with a range of existing GP approaches on 8 large-scale DFJSS scenarios. Further analysis shows that the effectiveness of the GPLS is attributed to the utilisation of the information of more jobs, the increased phenotypic diversity of the population by using parents with a wider fitness distribution to generate offspring, and the increased rule size.

Parent selection is an important part of GP. In the future, there are still some interesting directions that can be further investigated in studying LS for solving DFJSS problems. We plan to design more diverse cases for DFJSS to take full advantage of LS. In addition, we would like to use surrogate models to estimate a list of fitnesses for LS to select parents, which can be more time-efficient. Furthermore, we plan to investigate industry practices, enhance our simulation to bring it closer to real-world applications, and then examine our proposed algorithm on more realistic simulations.

References

[1] J. Zhang, G. Ding, Y. Zou, S. Qin, and J. Fu, "Review of job shop scheduling research and its new perspectives under industry 4.0," *Journal of Intelligent Manufacturing*, vol. 30, no. 4, pp. 1809–1830, 2019.

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

[3] R. Li, W. Gong, C. Lu, and L. Wang, "A learning-based memetic algorithm for energy-efficient flexible job shop scheduling with type-2 fuzzy processing time," *IEEE Transactions on Evolutionary Computation*, 2022.

[4] R. H. Caldeira, A. Gnanavelbabu, and T. Vaidyanathan, "An effective backtracking search algorithm for multi-objective flexible job shop scheduling considering new job arrivals and energy consumption," *Computers & Industrial Engineering*, vol. 149, p. 106863, 2020.

[5] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Automatic programming via iterated local search for dynamic job shop scheduling," *IEEE Transactions on Cybernetics*, vol. 45, no. 1, pp. 1–14, 2014.

[6] J. Xiong, L.-n. Xing, and Y.-w. Chen, "Robust scheduling for multi-objective flexible job-shop problems with random machine breakdowns," *International Journal of Production Economics*, vol. 141, no. 1, pp. 112–126, 2013.

[7] 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.

[8] M. Shahgholi Zadeh, Y. Katebi, and A. Doniavi, "A heuristic model for dynamic flexible job shop scheduling problem considering variable processing times," *International Journal of Production Research*, vol. 57, no. 10, pp. 3020–3035, 2019.

[9] 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.

[10] ——, "Collaborative multifidelity-based surrogate models for genetic programming in dynamic flexible job shop scheduling," *IEEE Transactions on Cybernetics*, pp. 1–15, 2021.

[11] K. E. Stecke, "Formulation and solution of nonlinear integer production planning problems for flexible manufacturing systems," *Management Science*, vol. 29, no. 3, pp. 273–288, 1983.

[12] M. Zhou, H.-S. Chiu, and H. H. Xiong, "Petri net scheduling of fms using branch and bound method," in *Proceedings of the Conference on IEEE Industrial Electronics*, vol. 1, 1995, pp. 211–216.

[13] C. Soto, B. Dorronsoro, H. Fraire, L. Cruz-Reyes, C. Gomez-Santillan, and N. Rangel, "Solving the multi-objective flexible job shop scheduling problem with a novel parallel branch and bound algorithm," *Swarm and Evolutionary Computation*, vol. 53, p. 100632, 2020.

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

[15] M. T. Jensen, "Generating robust and flexible job shop schedules using genetic algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 3, pp. 275–288, 2003.

[16] D. C. Mattfeld and C. Bierwirth, "An efficient genetic algorithm for job shop scheduling with tardiness objectives," *European Journal of Operational Research*, vol. 155, no. 3, pp. 616–630, 2004.

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

[18] M. Saidi Mehrabad and P. Fattahi, "Flexible job shop scheduling with tabu search algorithms," *International Journal of Advanced Manufacturing Technology*, vol. 32, no. 5-6, pp. 563–570, 2007.

[19] G. Vilcot and J. C. Billaut, "A tabu search algorithm for solving a multicriteria flexible job shop scheduling problem," *International Journal of Production Research*, vol. 49, no. 23, pp. 6963–6980, 2011.

[20] H. Fan, H. Xiong, and M. Goh, "Genetic programming-based hyper-heuristic approach for solving dynamic job shop scheduling problem with extended technical precedence constraints," *Computers & Operations Research*, vol. 134, p. 105401, 2021.

[21] K. Shanker and Y. J. J. Tzen, "A loading and dispatching problem in a random flexible manufacturing system," *International Journal of Production Research*, vol. 23, no. 3, pp. 579–595, 1985.

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

[23] J. R. Koza and R. Poli, "Genetic programming," in *Search Methodologies*, 2005, pp. 127–164.

[24] 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*, 2019, pp. 347–355.

[25] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 5, pp. 621–639, 2012.

[26] ——, "Hybrid evolutionary computation methods for quay crane scheduling problems," *Computers & Operations Research*, vol. 40, no. 8, pp. 2083–2093, 2013.

[27] E. K. Burke, S. Gustafson, and G. Kendall, "Diversity in genetic programming: An analysis of measures and correlation with fitness," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 1, pp. 47–62, 2004.

[28] Q. Chen, B. Xue, and M. Zhang, "Preserving population diversity based on transformed semantics in genetic programming for symbolic regression," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 3, pp. 433–447, 2020.

[29] T. Helmuth, L. Spector, and J. Matheson, "Solving uncompromising problems with lexicase selection," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 5, pp. 630–643, 2014.

[30] T. Helmuth and A. Abdelhady, "Benchmarking parent selection for program synthesis by genetic programming," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2020, pp. 237–238.

[31] S. A. Troise and T. Helmuth, "Lexicase selection with weighted shuffle," in *Genetic Programming Theory and Practice XV*, 2018, pp. 89–104.

[32] L. Meng, C. Zhang, Y. Ren, B. Zhang, and C. Lv, "Mixed-integer linear programming and constraint programming formulations for solving distributed flexible job shop scheduling problem," *Computers & Industrial Engineering*, vol. 142, p. 106347, 2020.

[33] 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*, 2018, pp. 472–484.

[34] Y. Zhou, J.-j. Yang, and Z. Huang, "Automatic design of scheduling policies for dynamic flexible job shop scheduling via surrogate-assisted cooperative co-evolution genetic programming," *International Journal of Production Research*, vol. 58, no. 9, pp. 2561–2580, 2020.

[35] Y. Zeiträg, J. R. Figueira, N. Horta, and R. Neves, "Surrogate-assisted automatic evolving of dispatching rules for multi-objective dynamic job shop scheduling using genetic programming," *Expert Systems with Applications*, vol. 209, p. 118194, 2022.

[36] Y. Zakaria, Y. Zakaria, A. BahaaElDin, and M. Hadhoud, "Niching-based feature selection with multi-tree genetic programming for dynamic flexible job shop scheduling," in *Proceedings of the International Joint Conference on Computational Intelligence*, 2019, pp. 3–27.

[37] F. Zhang, Y. Mei, S. Nguyen, K. C. Tan, and M. Zhang, "Task relatedness based multitask genetic programming for dynamic flexible job shop scheduling," *IEEE Transactions on Evolutionary Computation*, aug 2022.

[38] ——, "Multitask genetic programming based generative hyper-heuristics: A case study in dynamic scheduling," *IEEE Transactions on Cybernetics*, vol. 52, no. 10, pp. 10 515–10 528, oct 2022.

[39] Y. Zhou and J. Yang, "Automatic design of scheduling policies for dynamic flexible job shop scheduling by multi-objective genetic programming based hyper-heuristic," *Procedia CIRP*, vol. 79, pp. 439–444, 2019.

[40] F. Zhang, Y. Mei, and M. Zhang, "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*, 2019, pp. 1366–1373.

[41] Y. Zhou, J.-J. Yang, and L.-Y. Zheng, "Hyper-heuristic coevolution of machine assignment and job sequencing rules for multi-objective dynamic flexible job shop scheduling," *IEEE Access*, vol. 7, pp. 68–88, 2018.

[42] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Multitask multi-objective genetic programming for automated scheduling heuristic learning in dynamic flexible job shop scheduling," *IEEE Transactions on Cybernetics*, aug 2022.

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

[44] E. Kieffer, G. Danoy, M. R. Brust, P. Bouvry, and A. Nagih, "Tackling large-scale and combinatorial bi-level problems with a genetic programming hyper-heuristic," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 1, pp. 44–56, 2019.

[45] B. Tan, H. Ma, Y. Mei, and M. Zhang, "A cooperative coevolution genetic programming hyper-heuristic approach for on-line resource allocation in container-based clouds," *IEEE Transactions on Cloud Computing*, 2020. [Online]. Available: https://doi.org/10.1109/TCC.2020.3026338

[46] M. Gulić and D. Jakobović, "Evolution of vehicle routing problem heuristics with genetic programming," in *Proceedings of the International Convention on Information and Communication Technology, Electronics and Microelectronics*, 2013, pp. 988–992.

[47] J. Jacobsen-Grocott, Y. Mei, G. Chen, and M. Zhang, "Evolving heuristics for dynamic vehicle routing with time windows using genetic programming," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2017, pp. 1948–1955.

[48] T. Blickle, "Tournament selection," *Evolutionary Computation*, vol. 1, pp. 181–186, 2000.

[49] B. L. Miller, D. E. Goldberg *et al.*, "Genetic algorithms, tournament selection, and the effects of noise," *Complex Systems*, vol. 9, no. 3, pp. 193–212, 1995.

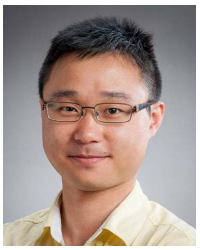
[50] T. Blickle and L. Thiele, "A mathematical analysis of tournament selection." in *Proceedings of the ICGA*, vol. 95, 1995, pp. 9–15.

[51] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," in *Foundations of Genetic Algorithms*, 1991, vol. 1, pp. 69–93.

[52] A. Sokolov and D. Whitley, "Unbiased tournament selection," in *Proceedings of the Conference on Genetic and Evolutionary Computation*, 2005, pp. 1131–1138.

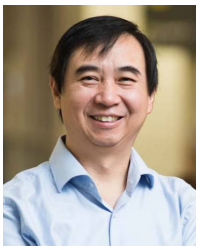
[53] H. Xie, M. Zhang, and P. Andreae, "Automatic selection pressure control in genetic programming," in *Proceedings of the International Conference on Intelligent Systems Design and Applications*, vol. 1, 2006, pp. 435–440.
[54] ——, "An analysis of constructive crossover and selection pressure in genetic programming," in *Proceedings of the Conference on Genetic and Evolutionary Computation*, 2007, pp. 1739–1748.
[55] H. Xie, M. Zhang, P. Andreae, and M. Johnson, "An analysis of multi-sampled issue and no-replacement tournament selection," in *Proceedings of the International Conference on Genetic and Evolutionary Computation*, 2008, pp. 1323–1330.
[56] H. Xie and M. Zhang, "Parent selection pressure auto-tuning for tournament selection in genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 1, pp. 1–19, 2012.
[57] J. L. Jiménez Laredo, S. S. Nielsen, G. Danoy, P. Bouvry, and C. M. Fernandes, "Cooperative selection: improving tournament selection via altruism," in *Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimization*, 2014, pp. 85–96.
[58] A. Sokolov, D. Whitley, and A. da Motta Salles Barreto, "A note on the variance of rank-based selection strategies for genetic algorithms and genetic programming," *Genetic Programming and Evolvable Machines*, vol. 8, no. 3, pp. 221–237, 2007.
[59] E. Galvan-Lopez, B. Cody-Kenny, L. Trujillo, and A. Kattan, "Using semantics in the selection mechanism in genetic programming: a simple method for promoting semantic diversity," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2013, pp. 2972–2979.
[60] T. H. Chu, Q. U. Nguyen, and M. O'Neill, "Semantic tournament selection for genetic programming based on statistical analysis of error vectors," *Information Sciences*, vol. 436, pp. 352–366, 2018.
[61] P. Day and A. K. Nandi, "Binary string fitness characterization and comparative partner selection in genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 6, pp. 724–735, 2008.
[62] M. W. Aslam, Z. Zhu, and A. K. Nandi, "Diverse partner selection with brood recombination in genetic programming," *Applied Soft Computing*, vol. 67, pp. 558–566, 2018.
[63] W. La Cava, L. Spector, and K. Danai, "Epsilon-lexicase selection for regression," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2016, pp. 741–748.
[64] L. Spector, W. La Cava, S. Shanabrook, T. Helmuth, and E. Pantridge, "Relaxations of lexicase parent selection," in *Genetic Programming Theory and Practice XV*. Springer, 2018, pp. 105–120.
[65] L. Planinić, M. Durasević, and D. Jakobović, "On the application of ϵ-lexicase selection in the generation of dispatching rules," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2021, pp. 2125–2132.
[66] T. Helmuth, N. F. McPhee, and L. Spector, "The impact of hyper-selection on lexicase selection," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2016, pp. 717–724.
[67] G. Gao, Y. Mei, B. Xin, Y.-H. Jia, and W. N. Browne, "Automated coordination strategy design using genetic programming for dynamic multipoint dynamic aggregation," *IEEE Transactions on Cybernetics*, 2021.
[68] 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 Conference on Genetic and Evolutionary Computation*, 2010, pp. 257–264.
[69] F. Zhang, Y. Mei, S. Nguyen, M. Zhang, and K. C. Tan, "Surrogate-assisted evolutionary multitasking genetic programming for dynamic flexible job shop scheduling," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 4, pp. 651–665, aug 2021.
[70] M. Xu, F. Zhang, Y. Mei, and M. Zhang, "Genetic programming with archive for dynamic flexible job shop scheduling," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2021, pp. 2117–2124.
[71] S. Luke, "Ecj then and now," in *Proceedings of the genetic and evolutionary computation conference companion*, 2017, pp. 1223–1230.
[72] T. Hildebrandt and J. Branke, "On using surrogates with genetic programming," *Evolutionary Computation*, vol. 23, no. 3, pp. 343–367, 2015.
[73] L. Spector, "Assessment of problem modality by differential performance of lexicase selection in genetic programming: a preliminary report," in *Proceedings of the Conference Companion on Genetic and Evolutionary Computation*, 2012, pp. 401–408.

Meng Xu received the B.Sc. and M.Sc. degrees from the Bejing Institute of Technology, Beijing, China, in 2017 and 2020, respectively. She is currently pursuing a Ph.D. degree in computer science with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. Her current research interests include evolutionary computation, hyper-heuristic learning/optimisation, job shop scheduling, and workflow scheduling.



Yi Mei received the B.Sc. and Ph.D. degrees from the University of Science and Technology of China, Hefei, China, in 2005 and 2010, respectively. He is an Associate Professor at the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. His research interests include evolutionary computation and machine learning for combinatorial optimisation, genetic programming, hyper-heuristics, and explainable AI. He has over 180 fully referred publications, including the top journals in EC and Operations Research such as IEEE TEVC, IEEE TCYB, Evolutionary Computation Journal, European Journal of Operational Research, ACM Transactions on Mathematical Software. He is an Associate Editor of IEEE Transactions on Evolutionary Computation, and a guest editor of the Genetic Programming Evolvable Machine journal. He is the Founding Chair of IEEE Taskforce on Evolutionary Scheduling and Combinatorial Optimisation. He serves as a Vice-Chair of the IEEE CIS Emergent Technologies Technical Committee, and a member of Intelligent Systems Applications Technical Committee. He is a reviewer of over 50 international journals. He is a Fellow of Engineering New Zealand and an IEEE Senior Member.

Fangfang Zhang received the B.Sc. and M.Sc. degrees from Shenzhen University, China, and the Ph.D. degree in Computer Science from Victoria University of Wellington, New Zealand, in 2014, 2017, and 2021, respectively. She is currently a post-doctoral research fellow in computer science with the School of Engineering and Computer Science, Victoria University of Wellington, New Zealand. She has over 45 papers in refereed international journals and conferences. Her research interests include evolutionary computation, hyper-heuristic learning/optimisation, job shop scheduling, surrogate, and multitask learning. Dr. Fangfang is an Associate Editor of Expert Systems With Applications. She is a member of the IEEE Computational Intelligence Society and Association for Computing Machinery, and has been serving as a reviewer for top international journals. She is the Secretary of IEEE New Zealand Central Section, and was the Chair of the IEEE Student Branch at Victoria University of New Zealand, and the chair of Professional Activities Coordinator.



Mengjie Zhang is currently a Professor of Computer Science, the Head of the Evolutionary Computation and Machine Learning Research Group, and the Associate Dean (Research and Innovation) with the Faculty of Engineering, Victoria University of Wellington, New Zealand. His current research interests include genetic programming, image analysis, feature selection and reduction, job-shop scheduling, and evolutionary deep learning and transfer learning. He has published over 800 research papers in refereed international journals and conferences. He is a Fellow of the Royal Society of New Zealand, a Fellow of Engineering New Zealand, a Fellow of IEEE, and an IEEE Distinguished Lecturer.