Genetic Programming and Reinforcement Learning on Learning Heuristics for Dynamic Scheduling: A Preliminary Comparison

Meng Xu, Yi Mei, Fangfang Zhang, and Mengjie Zhang,
Victoria University of Wellington, NEW ZEALAND

***Abstract*—Scheduling heuristics are commonly used to solve dynamic scheduling problems in real-world applications. However, designing effective heuristics can be time-consuming and often leads to suboptimal performance. Genetic programming has been widely used to automatically learn scheduling heuristics. In recent years, reinforcement learning has also gained attention in this field. Understanding their strengths and weaknesses is crucial for developing effective scheduling heuristics. This paper takes a typical genetic programming method and a typical reinforcement learning method in dynamic flexible job shop scheduling for investigation. The results show that the investigated genetic programming algorithm outperforms the studied reinforcement learning method in the examined scenarios. Also, the study reveals that the compared reinforcement learning method is more stable as the amount of training data changes, and the investigated genetic programming method can learn more effective scheduling heuristics as training data increases. Additionally, the study highlights the potential and value of genetic programming in real-world applications due to its good generalization ability and interpretability. Based on the results, this paper suggests using the investigated reinforcement learning method when training data is limited and stable results are required, and using the investigated genetic programming method when training data is sufficient and high interpretability is required.**

***Index Terms*—Machine learning (artificial intelligence), heuristic learning, genetic programming, reinforcement learning, dynamic scheduling.**

I. Introduction

DYNAMIC scheduling [1] is a typical combinatorial optimization problem with the goal of optimizing resource allocation [2]. Dynamic scheduling has been widely applied to model practical applications, such as order picking in the warehouse [3], the manufacturing industries [4], and grid/cloud computing [5]. Compared to static scheduling problems that assume all jobs are known in advance, dynamic scheduling is closer to reality. Dynamic flexible job shop scheduling (DFJSS) is a prominent problem in the field of dynamic scheduling, where machine allocation (routing) and operation sequencing need to be considered simultaneously in the presence of uncertain events. Such events may include dynamic job arrivals [6] or machine breakdowns [7]. Due to the uncertainty and NP-hardness of DFJSS, achieving satisfactory performance is a challenging task [8].

Classical methods for solving scheduling problems, such as exact methods [9], [10] and (meta-) heuristic solution optimization methods [11], [12], are not suitable for DFJSS, since they cannot react to dynamic environments in real-time and it takes time for them to obtain promising solutions. Scheduling heuristics are the most commonly used methods for dynamic scheduling in industries due to their advantages of simplicity to implement and ability to quickly react to the changing environments [13]. The first-in-first-out and the shortest-processing-time-first are common scheduling heuristics in practical applications [14]. However, the performance of scheduling heuristics depends on various factors and scheduling scenarios. Furthermore, manually designing effective scheduling heuristics can be time-consuming and heavily rely on domain knowledge [15].

Genetic programming (GP) [16], [17] and reinforcement learning (RL) [18] have demonstrated their effectiveness in automatically learning scheduling heuristics for dynamic scheduling problems. GP has been a successful approach for learning effective scheduling heuristics in DFJSS for a long time, and it remains dominant in this research field [19]. GP uses the evolutionary principles for learning scheduling heuristics through an iterative, population-based, and stochastic evolutionary process [20]. Recently, RL has gained increasing attention in learning scheduling heuristics for dynamic scheduling problems. RL learns scheduling heuristics by interacting with the dynamic scheduling environment. Specifically, at each step, the RL agent chooses an action from the action space based on the current state, using the policy (scheduling heuristic) that maps states to actions. The agent then receives a reward and moves to the next state according to the state transition probability. For an episodic problem, the process repeats until the agent reaches a terminal state. After each episode, a discounted accumulated reward is obtained according to a discount factor. The agent's goal is to maximize the expectation of such long-term accumulated reward starting from any initial state [21]. GP and RL are mechanically dissimilar, although they both learn policies to make decisions at decision points. However, no existing work has compared these two learning paradigms in solving the DFJSS problem. Figuring out their strengths and weaknesses is valuable for learning effective scheduling heuristics as well as for practical applications. Typically, test performance (generalization ability), the influence of training data, and interpretability are key indicators for judging how promising a method is, especially with regard to real-world applications. Generalization ability determines how well the trained heuristic performs on unseen

Corresponding author: Fangfang Zhang (fangfang.zhang@ecs.vuw.ac.nz).

instances, which can directly reflect the quality of the learned heuristic. In practical applications, collecting data can be challenging. Thus, the amount of training data becomes a critical factor that influences the performance of the learned heuristics. Interpretability refers to the ability to understand and explain the learned heuristics, providing users with trust and confidence. This paper aims to provide a preliminary study by taking typical GP and RL algorithms in DFJSS as a case study for investigation and analyzing the above metrics. A typical GP (MTGP) [22] method and a typical RL (DRL) [23] method designed for the DFJSS problem are investigated. To be specific, this paper aims to explore the following questions:

1) Which of the MTGP or the DRL is better for solving the DFJSS problem?
2) How does the amount of training data affect the performance of the MTGP and the DRL?
3) How well can the MTGP and the DRL be generalized to solve more complex instances (i.e., considering more jobs and more workcenters)?
4) Which of the MTGP and the DRL can obtain more interpretable scheduling heuristics?

Overall, this work provides an empirical comparison of various aspects of the investigated MTGP and DRL methods, thus guiding future research in the research domain of the DFJSS problem.

II. Background

A. Problem Description

In the DFJSS problem, the shop floor has a set of machines denoted by $\Gamma = \{\Omega_1, ..., \Omega_k\}$, which belongs to certain workcenters $\mathcal{W} = \{W_1, ..., W_w\}$. Jobs $\mathcal{J} = \{J_1, J_2, ..., J_n\}$ arrive dynamically at the shop floor, and each job J_j has an arrival time t_j^{arr}, a due date t_j^{due}, and must visit each workcenter to be processed following a predetermined sequence of operations $O_{J_j} = [O_{j,1}, O_{j,2}, ..., O_{j,q_j}]$. Each operation $O_{j,i}$ can be processed only by a subset ($k_{j,i}$) of machines $\Gamma_{j,i} \subseteq \Gamma$. The processing time $t_{j,i,m}^{pro}$ of the operation $O_{j,i}$ depends on the selected machine $\Omega_m \in \Gamma_{j,i}$ to process it.

The scheduling difficulty and production performance are affected by three main factors, which are listed as follows:

1) **Job arrival rate**: a higher job arrival rate can lead to a higher machine utilization level and also cause heavier system congestion;
2) **Job and machine heterogeneity**: the processing times for operations on different machines can be different;
3) **Tightness of due dates**: a tighter due date gives less flexibility in operation and less job slack, thus tending to lead to higher tardiness;

For DFJSS, a solution or schedule ρ refers to a set of allocation of the processing start time $t_{j,i}^{start}$ of each operation $O_{j,i}$ on its assigned machine Ω_m for all jobs. Total tardiness $Ttotal$ is an important objective in practical industries, which will be optimized in this paper. The definition of $Ttotal$ is listed as follows.

$$Ttotal = \sum_{j=1}^{n} tard(J_j),\ tard(J_j) = \max\{t_j^{com} - t_j^{due}, 0\}$$

where $tard(J_j)$ denotes the tardiness of the job J_j. The t_j^{com} represents the completion time of the job J_j.

The mathematical model of the DFJSS problem is shown as follows:

$$\min\ Ttotal \tag{1}$$

s.t.:

$$\begin{gathered} t_{j_1,i_1}^{start} \le t_{j_2,i_2}^{start} - t_{j_1,i_1,m}^{pro} + L \cdot (1 - x_{j_1,i_1,j_2,i_2,m}), \\ \forall j_1, j_2 = 1, \ldots, n;\ \forall i_1 = 1, \ldots, q_{j_1}; \\ \forall i_2 = 1, \ldots, q_{j_2};\ \forall m = 1, \ldots, k \end{gathered} \tag{2}$$

$$t_{j,1}^{start} \ge t_j^{arr},\ \forall j = 1, \ldots, n \tag{3}$$

$$\begin{gathered} t_{j,i}^{com} \ge t_{j,i}^{start} + t_{j,i,m}^{pro} \cdot y_{j,i,m},\ \forall j = 1, \ldots, n; \\ \forall i = 1, \ldots, q_j;\ \forall m = 1, \ldots, k \end{gathered} \tag{4}$$

$$t_{j,i+1}^{start} \ge t_{j,i}^{com},\ \forall j = 1, \ldots, n;\ \forall i = 1, \ldots, q_j - 1 \tag{5}$$

$$\textstyle\sum_{m=1}^{k_{j,i}} y_{j,i,m} = 1,\ \forall j = 1, \ldots, n;\ \forall i = 1, \ldots, q_j \tag{6}$$

$$x_{j_1,i_1,j_2,i_2,m} = \begin{cases} 1, \text{ if } O_{j_1,i_1} \text{ is processed by } \Omega_m \text{ before } O_{j_2,i_2}, \\ 0, \text{ otherwise.} \end{cases} \tag{7}$$

$$y_{j,i,m} = \begin{cases} 1, \text{ if } O_{j,i} \text{ is assigned to } \Omega_m, \\ 0, \text{ otherwise.} \end{cases} \tag{8}$$

Constraint (2) specifies that a machine can only process one operation at a time, with L representing a sufficiently large constant. Constraint (3) indicates that the processing of each job's first operation can only begin after the job is released. Constraint (4) defines the dependence of the processing start time $t_{j,i}^{start}$ and the processing completion time $t_{j,i}^{com}$ for the operation $O_{j,i}$. Constraint (5) states that the operation $O_{j,i+1}$ must wait for the completion of its preceding operation $O_{j,i}$ before it can be processed on the assigned machine. Constraint (6) signifies that an operation can only be processed on one of its candidate machines.

B. Learning Scheduling Heuristics

The scheduling heuristic is an effective technique for dealing with uncertainty because it can make a quick decision based on the system state at the moment [24]. For DFJSS, which involves both machine assignment (routing) and operation sequencing, a scheduling heuristic contains two rules: a routing rule and a sequencing rule. During the scheduling process, the routing rule tells each workcenter which machine a ready operation is assigned to be processed [25], and the sequencing rule tells each machine which operation to process next [25]. The design of effective scheduling heuristics needs extensive trial and error, and their performance highly depends on the intuition and experience of human experts [26]. To address this issue, automatic heuristic learning has been a growing trend. The fundamentals of automatically learning heuristics are derived from the following considerations. A class of problem instances may have similar attributes, differing only in the data that follow different distributions [26]. Through learning, the underlying patterns of a particular problem class can be discovered, which can be used to learn

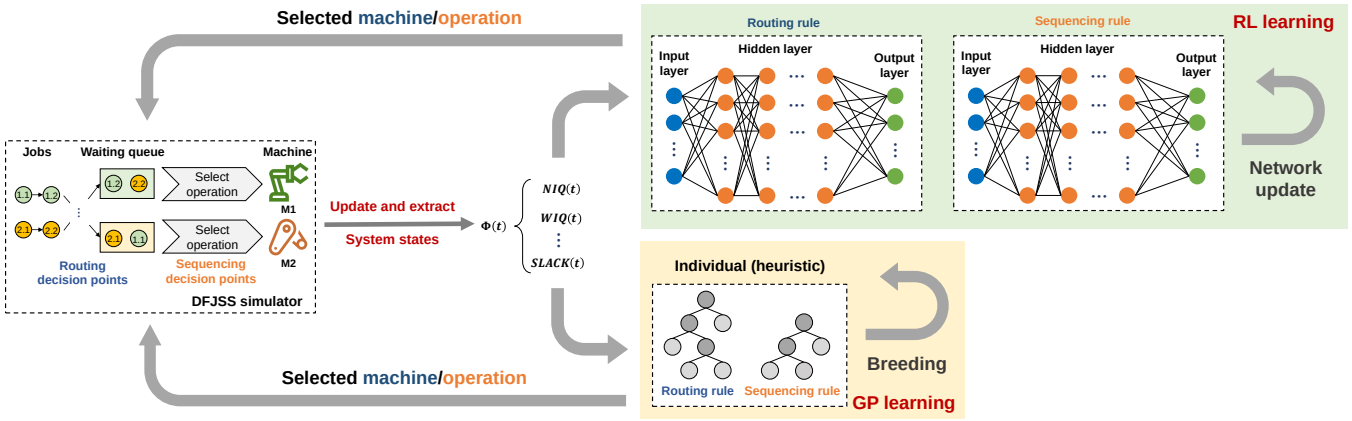


Fig. 1. The workflow about how scheduling heuristics learned by GP and RL work for the DFJSS problem.

alternative heuristics superior to those designed by humans [26].

In the field of automatic learning scheduling heuristics, RL [18] and GP [16] are two typical approaches to automatically learn heuristics [23], [27]–[30]. Fig. 1 illustrates the general framework for how GP and RL learn scheduling heuristics for the DFJSS problem. It can be seen that GP and RL have some similarities.

- Firstly, GP and RL make decisions (selecting machine/operation) based on the input state features related to jobs, machines, and environments, which can be static (e.g., the number of machines on the shop floor) or dynamic (e.g., the system time).
- Secondly, different from traditional solution approaches directly searching the solution space, both GP and RL approaches explore the heuristic space [31].
- Thirdly, they will finally output a scheduling heuristic by interacting with the dynamic environment.

There are also some differences between GP and RL.

- Firstly, GP and RL have different structures of the learned scheduling heuristics, which can be seen in Fig. 1. It shows that the learned scheduling heuristic by RL is typically represented by the neural network [28]. Different from RL, the learned scheduling heuristic by GP has the tree-based structure [32]. Regardless of the structure of the scheduling heuristics, they are utilized to choose a machine whenever an operation is ready and determine an operation whenever a machine becomes idle.
- Secondly, they learn the scheduling heuristics in different ways. GP maintains a population, and the learning process is carried out through the crossover and mutation of scheduling heuristics (individuals) in the population, while the learning process of RL is based on the iterative adjustment on a single scheduling heuristic (neural network).

A simulation model is usually used to measure how good a learned scheduling heuristic is. Algorithm 1 illustrates the simulation process of applying a scheduling heuristic for a DFJSS instance [33]. To be specific, the simulation starts when the first job(s) arrive at the shop floor and the scheduling process continues by triggering events sequentially, include the job arrival event (`JobArrivalEvent`), the operation visit event (`OperationVisitEvent`), the process start event (`ProcessStartEvent`), and the process end event (`ProcessEndEvent`). The job arrival event signifies the arrival of the new job(s) to the shop floor, while the operation visit event denotes that an operation becomes ready and is assigned to a machine. The process start event and the process end event represent the start and end of an operation processed by a machine, respectively. The simulation stops when there is no event in the event queue and outputs the resultant schedule.

C. Related Work

GP learns heuristics by first initializing a group of heuristics randomly, then follows the natural biological principle of survival of the fittest through heuristic evaluation, selection, and breeding (e.g., by crossover, mutation) to generate offspring. This process continues until termination conditions are met, and then the best-performing heuristic(s) are output as the learned heuristic(s). There has been a lot of research using GP-based methods to solve the DFJSS problem. Regarding individual representation, GP comprises various types, including tree-based GP [22], linear GP [34], gene expression programming [35], and others. Among these variants, tree-based GP stands out as a popular type, showcasing its effectiveness through successful applications within the DFJSS domain [22], [25], [30], [33], [36], [37]. At the early stage, a GP with a multi-tree representation was presented to learn both the sequencing rule and the routing rule simultaneously for DFJSS [22]. Then, a number of studies regarding the improvement of the genetic operators of GP emerged. For example, some studies [33], [36] focused on the selection process, improving the diversity of selected parents to generate offspring to avoid premature convergence. In addition, some researchers integrated various techniques to enhance the performance of GP, such as feature selection, multi-task learning, and surrogate models. In [30], a GP with surrogate was proposed to automatically learn scheduling heuristics more efficiently. More recently, some research use GP to address multi-objective DFJSS to meet a number of objectives simultaneously [25], [37]. For GP, parameter settings are crucial and can significantly impact its performance. Different from RL, where the network structure

Algorithm 1: The DFJSS simulation

Input: The DFJSS instance with dynamic arrival jobs set $\mathcal{J}$, the sequencing rule $h_s(\cdot)$, the routing rule $h_r(\cdot)$
Output: The DFJSS schedule: ρ

$\rho \leftarrow \{\}$ and the event queue $\Delta \leftarrow \{\}$;
foreach $J_j \in \mathcal{J}$ **do**
  Create a `JobArrivalEvent` $\mathcal{E}_j$ and $\Delta \leftarrow \Delta \cup \mathcal{E}_j$;
end
while $\Delta \neq \emptyset$ **do**
  Get the next event $\mathcal{E}_e = \{O_{j,i}, W_w, \Omega_m\}$ from Δ;
  if $\mathcal{E}_e$ *is the* `JobArrivalEvent` **then**
    `// Trigger the job arrival event`
    Calculate the priority $h_r(\Omega^*, O_{j,i})$ of each machine $\Omega^* \in W_w$ for $O_{j,i}$;
    Assign $O_{j,i}$ to Ω^* with the highest priority, create an `OperationVisitEvent` $\mathcal{E}^*$ and $\Delta \leftarrow \Delta \cup \mathcal{E}^*$;
  else if $\mathcal{E}_e$ *is the* `OperationVisitEvent` **then**
    `// Trigger the operation visit event`
    if Ω_m *is idle* **then**
      Create a `ProcessStartEvent` $\mathcal{E}^*$ and $\Delta \leftarrow \Delta \cup \mathcal{E}^*$;
    else
      Add $O_{j,i}$ to the waiting queue of Ω_m;
    end
  else if $\mathcal{E}_e$ *is the* `ProcessStartEvent` **then**
    `// Trigger the process start event`
    Calculate the processing time $t^{pro}_{j,i,m}$ of $O_{j,i}$ on Ω_m;
    Create a `ProcessEndEvent` $\mathcal{E}^*$ and $\Delta \leftarrow \Delta \cup \mathcal{E}^*$;
  else if $\mathcal{E}_e$ *is the* `ProcessEndEvent` **then**
    `// Trigger the process end event`
    Record the processing start time $t^{start}_{j,i}$ of $O_{j,i}$ on Ω_m into the schedule: $\rho \leftarrow \rho \cup \{O_{j,i}, \Omega_m, t^{start}_{j,i}\}$;
    if *The job* J_j *is not completed* **then**
      Calculate the priority $h_r(\Omega^*, O_{j,i+1})$ of each candidate machine Ω^* for its next operation $O_{j,i+1}$;
      Assign $O_{j,i+1}$ to Ω^* with the highest priority, create an `OperationVisitEvent` $\mathcal{E}^*$ and $\Delta \leftarrow \Delta \cup \mathcal{E}^*$;
    end
    if *The waiting queue of the machine* Ω_m *is not empty* **then**
      Calculate the priority $h_s(O^*, \Omega_m)$ of each operation O^* in its waiting queue;
      Choose O^* with the highest priority, create a `ProcessStartEvent` $\mathcal{E}^*$ and $\Delta \leftarrow \Delta \cup \mathcal{E}^*$;
    end
  end
end
return The obtained DFJSS schedule ρ;

is defined in advance and only its parameters need to be tuned during the learning process, GP requires exploration of both the parameters and the structures during the learning process. Therefore, careful consideration and experimentation with parameter settings and structural variations are necessary to achieve good performance with GP.

RL learns scheduling heuristics by first formulating the problem as a Markov decision process [38], then at each decision point, RL interacts with the environment, from which the experience is collected and eventually the policy is obtained after many episodes. In recent years, there has been an increasing trend towards using RL to solve scheduling problems. For example, in [27], a deep RL (DRL) method was developed to solve the static job shop scheduling problem and demonstrated its better performance than the existing manually designed scheduling heuristics. It represented the scheduling problem as a disjunctive graph, then a graph neural network scheme was developed to embed the states that are met during the scheduling process. In [28], a DRL method was designed for solving the static flexible job shop scheduling problem, achieving better performance than the existing manually designed scheduling heuristics. To be specific, the operation selection and the machine assignment were considered as one integrated decision. Then, it uses a new heterogeneous graph to represent the scheduling state and proposes a heterogeneous graph-neural-network-based framework for capturing the complex relationships that exist within operations and machines. In [39], a DRL method with a double deep Q-network (DQN) model was proposed for minimizing the total penalty of earliness and tardiness in DFJSS. Also, a soft ϵ-greedy policy was designed to balance the exploration and exploitation in the search space and improve learning efficiency. In [40], a DQN was proposed to solve DFJSS. Six composite scheduling heuristics were developed to choose an operation and assign the operation to an available machine whenever an operation is completed or a new job arrives. The proposed DQN was trained to select among these six scheduling heuristics. A recent work [23] used a DRL method for learning scheduling heuristics to solve the DFJSS problem with constant job arrivals. It trained the sequencing rule and the routing rule using the double DQN, separately. In addition, a surrogate reward-shaping function was developed to enhance both learning efficiency and scheduling performance. There are also some studies using RL for solving multi-objective DFJSS problems [41]–[44]. For RL, the reward function is a very important factor in guiding the search direction. A well-designed reward function heavily relies on domain knowledge and also demands a great deal of effort in its design and refinement. Also, actions might have a widespread and lasting impact that can be difficult to measure by a scalar reward [23]. In addition, RL is a highly parametrized algorithm, which requires human expertise and time to find good hyper-parameters.

GP and RL share similar mechanisms for solving the DFJSS problems. However, so far they have been investigated separately, and there is no analysis of or comparison between them. While it is difficult to reach conclusions only by comparing the principles between them, we can conduct empirical comparisons to show the advantages and disadvantages of GP and RL, from which some novel ideas through combining GP and RL might emerge.

Following the above investigation, this paper chooses two representative techniques for comparison: the multi-tree GP (MTGP) [22] and the deep RL (DRL) [23]. The choice to use the MTGP for the investigation is because it is specially designed for the DFJSS problem with a two-tree representation and can represent the classical GP with a classical evolution process. The DRL is chosen as it is a typical and recently designed RL algorithm specifically for the DFJSS problem to learn both routing and sequencing rules. It is representative of RL with a classical DQN architecture and training process.

III. METHODS

This section first introduces the used system state features for learning. Then, the investigated MTGP [22] and DRL [23] methods are described.

A. State Features

For scheduling heuristic learning, some features need to be extracted to represent the scheduling system state, which serves as the input of the MTGP and DRL. These features are used to form the scheduling heuristics and are updated as the scheduling process goes on. In this paper, a number of features for learning scheduling heuristics are extracted to solve the DFJSS problem [23], which are described as follows.

1) $PT_{j,i,m}(t)$: The processing time of the operation $O_{j,i}$ on the machine Ω_m at time t.
2) $WKR_j(t)$: The work remaining, representing the total processing time of the job J_j for the remaining operations at time t.
3) $CR_j(t)$: The completion rate, denoting the percentages of completed operations among all the operations of the job J_j at time t.
4) $TTD_j(t)$: The time until due, meaning the remaining time of the job J_j until the due date at time t.
5) $SLACK_j(t)$: The slack of the job J_j at time t, $SLACK_j(t) = t_j^{due} - t - WKR_j(t)$.
6) $WIQ_m(t)$: The remaining work (total processing time of all the operations) in the waiting queue of machine Ω_m at time t.
7) $NIQ_m(t)$: The number of operations in the waiting queue of machine Ω_m at time t.
8) $MRT_m(t)$: The ready time of machine Ω_m at time t, i.e., when machine becomes idle.
9) $MWT_m(t)$: The waiting time of machine Ω_m at time t, $MWT_m(t) = t - MRT_m(t)$.
10) $MBT_m(t)$: The busy time, denoting the total working time of machine M_m at time t.
11) $NPT_{j,i+1}(t)$: The median of the processing time for the next operation $O_{j,i+1}$ at time t.
12) $NOR_j(t)$: The number of the remaining operations of the job J_j at time t.
13) $OWT_{j,i}(t)$: The waiting time of the operation $O_{j,i}$ at time t, $OWT_{j,i}(t) = t - ORT_{j,i}(t)$, where $ORT_{j,i}(t)$ denotes the ready time of the operation $O_{j,i}$, which denotes the time the operation arrived at the queue of the machine.
14) t: The current time of the scheduling system.
15) $TIS_j(t)$: The time that the job J_j has been in the scheduling system at time t, $TIS_j(t) = t - t_j^{arr}$.

It can be seen that the value of the features might change over time, and the learned heuristics can take advantage of the latest information. The MTGP and DRL are expected to learn effective scheduling heuristics by combining these features with functions.

B. The Multi-Tree Genetic Programming Method

This study uses the MTGP method developed for the DFJSS problem in [22] to do the investigation. The MTGP starts training by initializing a number of individuals as a population. Each individual has two trees: one representing the routing rule, and the other representing the sequencing rule. The fitness evaluation process gives each individual a fitness by applying the individual to the DFJSS training set, which includes a number of DFJSS instances. The MTGP selects individual(s) as parent(s) for reproduction/mutation/crossover to generate offspring. Tournament selection is adopted to select parents. For reproduction, each time a parent is selected, it is copied into the next generation directly. For mutation, each time a parent is selected, it randomly chooses a subtree from one tree (either the routing or sequencing rule, sampled at random) of the parent and replaces the subtree with a newly generated subtree. During crossover, each time two parents are selected. For one tree (either the routing or sequencing rule, sampled at random) of the two parents, a subtree is randomly selected from each parent and swapped. For the other tree, the entire tree is swapped directly. The crossover and mutation used in the MTGP method are shown in Figs. 2 and 3.

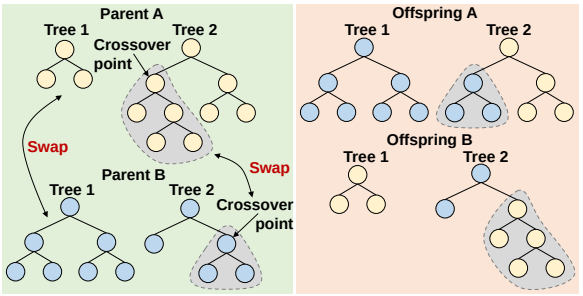


Fig. 2. The process of the tree swapping crossover in the MTGP

.

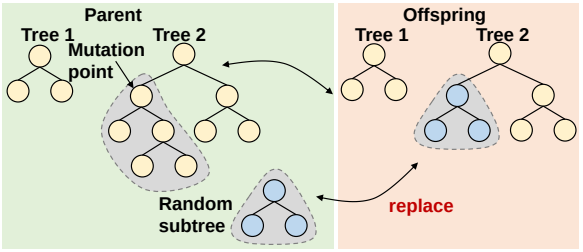


Fig. 3. The process of the subtree mutation in the MTGP

.

Based on the above process, the MTGP improves the scheduling heuristics until the stopping criterion is reached. Then, the best scheduling heuristic from the last generation is output as the learned result. In addition, to improve the generalization ability of the learned scheduling heuristics, this paper uses a seed rotation strategy. The seed rotation strategy rotates the random seed of the DFJSS simulation at different generations, leading to different realized instances at each generation of the MTGP. This changes the instance(s) for each new generation, which is a widely used strategy to improve genralization of using GP for DFJSS [45], [46].

C. The Deep Reinforcement Learning Method

This paper uses the DRL method proposed for the DFJSS problem in [23] for investigation. The DRL models the DFJSS problem as a Markov decision process (MDP) with a 5-elements tuple representation: (S, A, P, γ, R). For each decision point in the MDP, the DRL interacts with the scheduling system based on a policy $\pi(S, A)$. At each decision point t, the DRL observes the system state $s_t \in S$ and takes an action

$a_t \in A$ according to the policy $\pi(S, A)$. Then the system state is updated to the next decision point s_{t+1}, and a reward $r_t \in R$ is received.

In the DRL method, double DQN is used as the learner of the routing rule and the sequencing rule. The learning processes of these two rules are carried out separately and independently. When learning the routing rule, the first-in-first-out is used as the sequencing rule for sequencing decisions. When learning the sequencing rule, the earliest available machine is used as the routing rule for routing decisions. During the learning process, the DRL uses an experience replay mechanism to store historical information. The Q-network learns based on a minibatch of experience e_t, where $e_t = (s_t, a_t, r_t, s_{t+1})$ represents a transition record in the MDP. The Q-network has an action network and a target network. During the learning process, the parameter of the action network (θ^A) is updated for each training iteration i_{th}. At the same time, the update of the parameters (θ^T) of the target network is synchronized with θ^A based on a low frequency to ensure stability throughout the learning process. The parametrized target value y_i used for learning at the training iteration i_{th} is calculated by Eq. (9).

$$y_i = r_t + \gamma Q(s_{t+1}, argmax_a Q(s_{t+1}, a|\theta_i^A)|\theta^T) \quad (9)$$

Then, the parameters θ_i^A are tuned to minimize the loss ($L(\theta_i^A)$) at iteration i_{th}, which is calculated as Eq. (10).

$$L(\theta_i^A) = \frac{2}{N}\sum(y_i - Q(s_t, a_t|\theta_i^A))^2 \quad (10)$$

During the learning process, the action for the routing rule corresponds to the options of machines within the workcenter. That is, the size of the output layer (actions) of the network for learning a routing rule equals the number of machines each workcenter contains. However, the action space for the sequencing rule is not so straightforward in DFJSS. The number of operations in the waiting queue of machines can vary in different states, which makes direct operation selection infeasible. Hence, four manually designed sequencing rules are used as the actions for the sequencing rule, which are listed as follows.

1) Shortest processing time (SPT): gives the job with the shortest processing time the highest priority;
2) Least work remaining (LWR): gives the job with the smallest remaining processing time the highest priority;
3) Least critical ratio (LCR): gives the job with the smallest ratio of its time until due versus its remaining processing time the highest priority;
4) Minimum slack (MS): gives the job with the smallest slack the highest priority.

In addition, the DRL uses two slack-driven surrogate reward functions for learning the routing and sequencing rules, respectively. To learn the routing rule, the reward is calculated based on the difference between the actual slack and the estimated slack. To learn the sequencing rule, the reward is calculated as the difference between the slack gain or loss of the selected job and the average slack gain or loss of the jobs that are not selected. More details regarding the calculation of reward functions can be seen in [23].

IV. Experiment Design

A. Dataset

This paper considers a dynamic production system with new dynamic job arrivals. To measure the performance of the MTGP and DRL, four scenarios are considered based on three important factors.

1) **Expected job arrival rate/system utilization level**: the job arrival rate is related to the utilization level of the system $E(u)$, which can be calculated as follows.

$$E(u) = \frac{\mathbb{E}(t) \div \frac{k}{w}}{\mathbb{E}(in)} \times 100\% = \frac{\mathbb{E}(t) \times w}{\mathbb{E}(in) \times k} \times 100\%,$$

where k and w mean the number of machines and workcenters. $\mathbb{E}(t)$ denotes the expected processing time of all operations on all machines. $\mathbb{E}(in)$ represents the expected time interval between job arrivals. In this paper, the expected utilization level of the system $E(u)$ is assumed to be 90% to simulate a busy production factory. Also, this paper assumes that the time interval X between adjacent job arrivals follows the exponential distribution: $X \sim Exp(\beta), \beta = \mathbb{E}(in)$.

2) **Heterogeneity of the processing time**: for each operation $O_{j,i}$, its processing time $t_{j,i,m}^{pro}$ on the machine Ω_m is randomly sampled from a uniform distribution $U[L_p, H_p]$, where L_p and H_p denote the low and high limits of the processing time, respectively. For different scenarios, different average processing times are considered: (1) High heterogeneity processing time with $t_{j,i,m}^{pro} \sim U[5, 25]$, and (2) Low heterogeneity processing time with $t_{j,i,m}^{pro} \sim U[10, 20]$.

3) **Due date tightness**: for each job J_j, the due date t_j^{due} is assigned based on its expected total processing time and the due date factor α_j. This paper considers two types of tightness range $U[L_d, H_d]$: (1) High tension of due date with $\alpha_j \sim U[1, 2]$; (2) Low tension of due date with $\alpha_j \sim U[1, 3]$. The due date is calculated as follows.

$$t_j^{due} = t_j^{arr} + \alpha_j \sum_{i=1}^{q_j} \left(\frac{\sum_{m=1}^{k_{j,i}} t_{j,i,m}^{pro}}{k_{j,i}} \right), \alpha_j \sim U[L_d, H_d]$$

Based on the above descriptions, the four scenarios used in this paper are defined as follows:

- **HH**: High heterogeneity of processing time $[5, 25]$ and high tension of due date $[1, 2]$;
- **HL**: High heterogeneity of processing time $[5, 25]$ and low tension of due date $[1, 3]$;
- **LH**: Low heterogeneity of processing time $[10, 20]$ and high tension of due date $[1, 2]$;
- **LL**: Low heterogeneity of processing time $[10, 20]$ and low tension of due date $[1, 3]$.

There are three workcenters on the shop floor, each of which has two machines. Each instance simulates the production for 1000 time units, during which about 124 jobs arrive on the shop floor. The settings for the datasets used in our paper are the same as [23].

B. Parameter Setting

The features described in Section III-A are used by both the MTGP and DRL. For the DRL method, the original source code is downloaded and directly run for the experiments. The DRL method is implemented in Python based on the package PyTorch [47]. The parameters for the DRL and the network structures are listed in Table I [23]. The function set used by the MTGP is $\{+, -, max, min, \times, \div\}$, where $\div$ is protected and returns 1 if divided by 0. The other parameters of the MTGP are shown in Table II. The MTGP method is implemented in Python based on the package DEAP [48]. As the MTGP holds a population and has the advantage of using multi-processing to evaluate individuals in parallel, the multiprocessing package [49] of Python is used to speed up the training process of the MTGP.

For both the MTGP and DRL, the same 100 training instances are used to obtain good scheduling heuristics. The DRL returns the learned scheduling heuristic after performing the 100 training instances. For the MTGP, two instances are applied for each generation, and finally, the best scheduling heuristic learned after 50 generations (total 100 instances) is returned. After training, 100 unseen instances are used to measure the test performance, i.e., generalization, of the learned scheduling heuristics. Besides the MTGP and DRL, this paper considers four manually designed routing rules and four manually designed sequencing rules, and combines them into 16 scheduling heuristics to compare with the learned scheduling heuristics by the MTGP and DRL methods. The comparison with the manually designed rules provides a more intuitive view of the effectiveness of the MTGP and DRL methods and reflects their good generalization ability. The manually designed routing and sequencing rules are listed as follows.

Routing rules:

1) Earliest completion time (ECT): gives the machine that has the smallest sum of available time and remaining processing time the highest priority;
2) Minimum execution time (MET): gives the machine that has the minimum execution time the highest priority;
3) Earliest available (EA): gives the machine that has the earliest available time the highest priority;
4) Least work in the queue (LWIQ): gives the machine that has the least work remaining (total processing time) in its waiting queue the highest priority.

Sequencing rules:

1) Shortest processing time (SPT): gives the operation that has the shortest processing time the highest priority;
2) Earliest due date (EDD): gives the operation whose job has the earliest due date the highest priority;
3) Least work remaining (LWR): gives the operation whose job has the least work remaining (processing time) the highest priority;
4) First-in-first-out (FIFO): gives the operation that arrives the first the highest priority.

TABLE I
THE PARAMETER CONFIGURATION OF THE DRL METHOD.

Parameter	Routing	Sequencing
Exploration rate (ϵ)	0.3 decays to 0.1	0.3 decays to 0.1
Discount factor (γ)	0.8	0.8
Learning rate	0.01 decays to 0.001	0.01 decays to 0.001
Minibatch size	128	64
Replay memory size	512	256
Input layer size	9	25
Output layer size	2	4
Hidden layer size	16×16×16×8×8	48×36×36×24×24×12
Channels	1	6

TABLE II
THE PARAMETER CONFIGURATION OF THE MTGP METHOD.

Parameter	Value
Population size	50
Maximal generations	50
Population initialisation method	The ramped-half-and-half
Initial minimum|maximum tree depth	2 | 6
Number of elitism	10
Maximal tree depth	8
Reproduction probability	0.05
Crossover probability	0.80
Mutation probability	0.15
Selection method	Tournament selection

C. Comparison Design

To answer the four questions proposed in Section I, the experiments are divided into four parts:

1) Test performance comparison: to compare the performance of the DRL [23], the MTGP [22], and the manually designed scheduling heuristics in the four scenarios.
2) Training data influence analysis: to study the relationship between the number of training instances and the performance of the MTGP and DRL on test instances.
3) Performance on more complex scenarios analysis: to study the generalization ability of the learned scheduling heuristics by the MTGP and DRL on more complex unseen instances than the training instances.
4) Interpretability analysis: to analyze the sizes and structures of the learned scheduling heuristics by the MTGP and DRL to understand the inner mechanism of the learned scheduling heuristics.

V. EXPERIMENTAL RESULTS

A. Test Performance

This section analyzes the experimental results of the manually designed scheduling heuristics, the MTGP [22], and the DRL [23] methods on the DFJSS problem in different scenarios. The objective values obtained by learned scheduling heuristics from each generation of the 30 independent runs of the MTGP on the 100 unseen instances in the four scenarios are shown in Fig. 4. Meanwhile, the convergence curves of the sequencing training loss and routing training loss of one run of the DRL in the scenario HH are shown in Figs. 5 and 6, respectively. The convergence curves from other runs of the DRL show the same pattern. As can be seen from the convergence curves, for the MTGP algorithm, the convergence rate is fast in the early stage, which gradually slows down

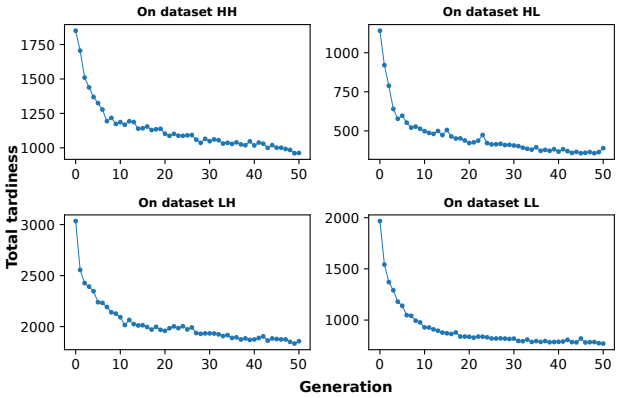


Fig. 4. The objective values obtained by learned scheduling heuristics from each generation of the MTGP on the 100 unseen instances of 30 independent runs in the four scenarios.

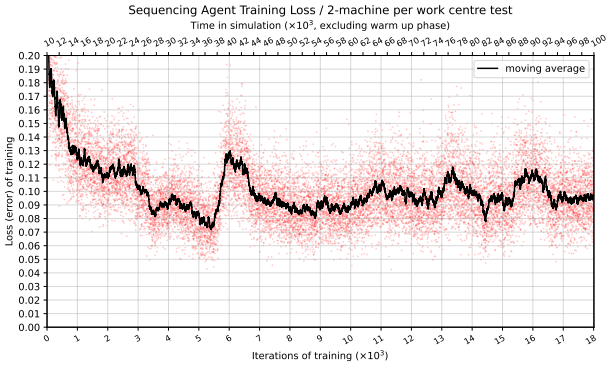


Fig. 5. The convergence curves of the **sequencing training loss** of the DRL of one run in the scenario HH.

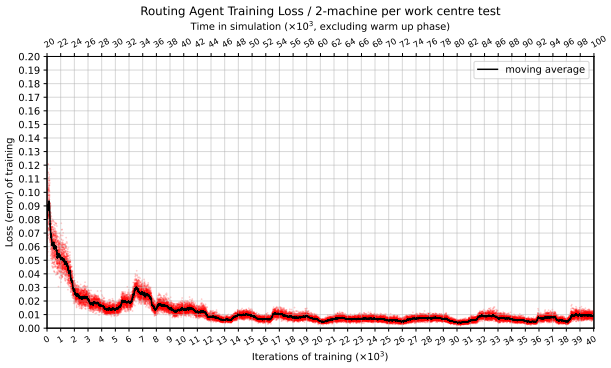


Fig. 6. The convergence curves of the **routing training loss** of the DRL of one run in the scenario HH.

and eventually reaches a relatively stable performance. For the DRL, both the sequencing training loss and the routing training loss converge quickly in the early stage, and the loss curve fluctuates obviously after convergence. In summary, both the MTGP and DRL methods ultimately converge, allowing for a comparison of their converged test performance.

The mean (standard deviation) of the test performance on 100 unseen instances of 30 independent runs of manually designed scheduling heuristics, the DRL, and the MTGP in the four scenarios are shown in Table III, and the box plots of the test performance on 100 unseen instances of 30 independent runs of the MTGP and DRL are shown in Fig. 7.

TABLE III
THE TEST PERFORMANCE OF MANUALLY DESIGNED SCHEDULING HEURISTICS, THE DRL, AND THE MTGP IN THE FOUR SCENARIOS.

Algorithm	HH	HL	LH	LL
ECT+SPT	1210.57(0.00)	777.73(0.00)	2299.63(0.00)	1559.03(0.00)
ECT+EDD	1270.07(0.00)	585.07(0.00)	2230.43(0.00)	1101.67(0.00)
ECT+LWR	1506.56(0.00)	960.29(0.00)	2381.54(0.00)	1525.45(0.00)
ECT+FIFO	1511.46(0.00)	920.72(0.00)	2506.36(0.00)	1538.48(0.00)
MET+SPT	1559.90(0.00)	1046.00(0.00)	4188.50(0.00)	3192.06(0.00)
MET+EDD	1773.54(0.00)	822.71(0.00)	4409.94(0.00)	2776.26(0.00)
MET+LWR	2149.38(0.00)	1426.26(0.00)	4542.56(0.00)	3328.43(0.00)
MET+FIFO	2140.90(0.00)	1339.91(0.00)	4814.27(0.00)	3373.93(0.00)
EA+SPT	3780.44(0.00)	2770.86(0.00)	3268.13(0.00)	2333.47(0.00)
EA+EDD	4211.34(0.00)	2588.23(0.00)	3188.26(0.00)	1785.30(0.00)
EA+LWR	4321.01(0.00)	3084.76(0.00)	3343.75(0.00)	2232.70(0.00)
EA+FIFO	4557.56(0.00)	3175.03(0.00)	3590.99(0.00)	2344.19(0.00)
LWIQ+SPT	3891.32(0.00)	2836.76(0.00)	3372.82(0.00)	2373.74(0.00)
LWIQ+EDD	4367.13(0.00)	2730.62(0.00)	3387.47(0.00)	1877.37(0.00)
LWIQ+LWR	4483.96(0.00)	3218.77(0.00)	3496.32(0.00)	2338.09(0.00)
LWIQ+FIFO	4762.90(0.00)	3351.49(0.00)	3793.20(0.00)	2497.48(0.00)
DRL	1170.68(37.13)	617.22(66.95)	2134.25(54.61)	1089.42(62.75)
MTGP(↑)	**962.94(178.56)**	**389.28(130.55)**	**1856.63(216.37)**	**770.37(102.55)**

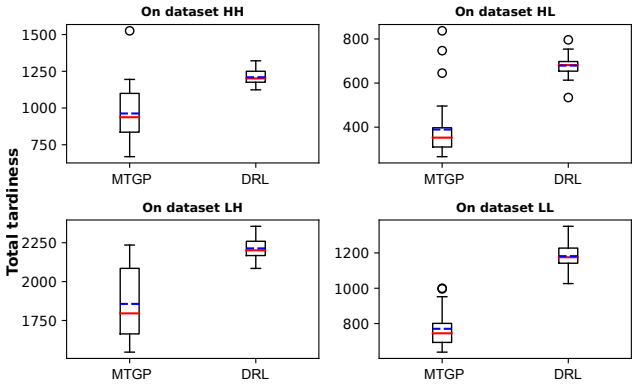


Fig. 7. The box plots of the **test performance** on unseen instances of the MTGP and DRL in the four scenarios.

Comparisons are conducted using the Wilcoxon rank sum test with a significance level of 0.05. The "↑" next to the MTGP in Table III denotes that the corresponding results of MTGP are significantly better than the results of all the compared algorithms.

It can be seen from Table III and Fig. 7 that the DRL outperforms, or is competitive with, all manually designed scheduling heuristics in the four scenarios. The MTGP performs much better than all the manually designed scheduling heuristics and the DRL in all the four scenarios. These results verify the effectiveness of the MTGP and DRL algorithms. The advantages of the MTGP and DRL methods over manual scheduling heuristics are consistent with the existing studies, since they both search the heuristic space to automatically learn better scheduling heuristics. The reasons that the MTGP outperforms the DRL might be due to the following reasons.

Firstly, during the learning process, the MTGP considers not only the effectiveness of the routing rule and the sequencing rule itself, but also their cooperation. In the MTGP, each individual has a routing rule as well as a sequencing rule. These two rules are evolved and evaluated simultaneously. However, the DRL learns these two rules independently, which might be stuck in poor local optima.

TABLE IV
THE COMPARISON RESULTS BETWEEN THE DRL AND THE MTGP IN THE FOUR SCENARIOS WITH **DIFFERENT NUMBER OF TRAINING INSTANCES**.

Scenario	Training instances	DRL	MTGP
HH	50	1188.48(71.32)	969.09(228.44)
	100	1170.68(37.13)(=)	962.94(178.56)(=)
	200	1187.48(54.26)(=)(=)	**897.81(123.01)**(=)(↑)
HL	50	620.42(57.36)	408.88(133.03)
	100	617.22(66.95)(=)	389.28(130.55)(=)
	200	**598.51(55.48)**(=)(↑)	378.06(111.49)(=)(=)
LH	50	2188.08(71.64)	1887.95(310.79)
	100	2134.25(54.61)(↑)	1856.63(216.37)(=)
	200	**2150.21(67.97)**(↑)(=)	1794.28(186.70)(=)(=)
LL	50	1115.14(100.1)	854.17(246.05)
	100	1089.42(62.75)(=)	**770.37(102.55)**(↑)
	200	1100.31(80.74)(=)(=)	**768.27(110.50)**(↑)(=)

TABLE V
THE IMPROVEMENT PERCENTAGES OF THE TEST PERFORMANCE IN THE FOUR SCENARIOS OF THE MTGP AND DRL **AS THE NUMBER OF TRAINING INSTANCES INCREASES**.

Scenario	(a,b)	DRL	MTGP
HH	(50,100)	-1.52%	0.63%
	(50,200)	0.08%	7.36%
	(100,200)	1.41%	6.76%
HL	(50,100)	-0.52%	4.79%
	(50,200)	3.51%	7.54%
	(100,200)	3.13%	2.88%
LH	(50,100)	-2.52%	0.02%
	(50,200)	-1.76%	4.96%
	(100,200)	0.74%	3.36%
LL	(50,100)	-2.36%	9.81%
	(50,200)	-1.35%	10.06%
	(100,200)	0.99%	0.27%

Secondly, the fitness (the total tardiness in this paper) of candidate scheduling heuristics in the MTGP is the same as the final objective (the total tardiness), while the reward function in the DRL is different from the final objective. The DRL learns and searches for candidates by giving feedback (reward) immediately after each decision point, and the accumulated rewards are used to evaluate the policy (scheduling heuristic). In principle, the DRL would provide more detailed information than the MTGP which ignores the intermediate information during the simulation. The detailed information will be reflected as rewards or penalties depending on the designed reward function. The reward function is critical to the DRL and affects the quality of the final scheduling heuristic learned. However, the design of the reward function is quite difficult and requires a lot of domain knowledge. The reward obtained from each decision point can give a positive effect when the reward function is well designed, but might have a negative influence when the reward function gives misleading information. In other words, a poorly designed reward function might mislead the DRL search process away from the final goal. In this case, it can be seen that although the DRL is able to provide more detailed information than the MTGP, it is hard to design the reward function. With a well-designed reward function, RL is expected to learn better rules that are more targeted and effective.

Thirdly, compared to the MTGP, the search space of the sequencing rule is limited in the DRL. As the number of actions in the DRL needs to be decided in advance, while the number of candidate operations at different sequencing decision points varies, a fixed number of manually designed sequencing rules are used to form the action space for the sequencing rule. However, in the MTGP, the sequencing rule allows for more possibilities, which can be achieved by randomly combining functions and terminals.

B. Influence of the Number of Training Instances

This section analyzes the impact of the number of training samples on the effectiveness of the algorithms. An empirical comparison is performed by setting three different numbers of training samples: 50, 100, and 200. In this case, the number of training instances for the DRL is adjusted to 50 or 200 to facilitate the learning of the scheduling heuristic. As for the MTGP, 50 generations remain consistent, with one instance used per generation when the training set comprises 50 instances and four instances assigned per generation when the training set contains 200 instances. The same as the setting in Section V-A, 100 unseen instances are used to measure the performance of the learned scheduling heuristics. The experimental results are shown in Table IV. For each scenario and each algorithm, the results are compared with different numbers of training instances. For example, for the HH scenario, the (=)(↑) for the MTGP with 200 training instances indicates that the MTGP with 200 training instances has no statistical difference from the MTGP with 50 training instances and performs significantly better than the MTGP with 100 training instances.

From Table IV, it can be seen that, in terms of horizontal comparison, the MTGP always performs better than the DRL. In the vertical comparison, using a relatively smaller or larger number of training instances does not affect the test performance of the MTGP and DRL dramatically. For the DRL, in the HH and LL scenarios, training with 50 instances and 200 instances obtain similar test performance with training on 100 instances. In the HL and LH scenarios, with more training instances (200 instances), the DRL performs significantly better than that with fewer training instances. For the MTGP, in the HL and LH scenarios, training with 50, 100, and 200 instances can obtain similar test performance. In the HH and LL scenarios, with more training instances (200 instances), the MTGP performs significantly better than that with fewer training instances.

In addition, Eq. (11) is used to calculate the improvement percentage ξ of the test performance in the four scenarios of the MTGP and DRL as the number of training instances increases.

$$\begin{gathered}\xi_{a,b} = (1 - \frac{Ttotal_a}{Ttotal_b}) \times 100\%, \\ (a,b) \in \{(50,100),(50,200),(100,200)\}\end{gathered} \tag{11}$$

The results are shown in Table V. It can be seen that as the number of training instances increases, the DRL sometimes gives negative improvement percentages (less than 0), while the MTGP always gives positive improvement percentages (larger than 0). This suggests that MTGP outperforms DRL by producing higher improvement percentages across all sce-

TABLE VI
THE TEST PERFORMANCE OF THE DRL, THE MTGP, AND COMPARISON METHODS ON EIGHT SCENARIOS **WITH A LARGE NUMBER OF JOBS ARRIVAL**.

Algorithm	2000 unit time (248 jobs)				5000 unit time (620 jobs)			
	HH	HL	LH	LL	HH	HL	LH	LL
ECT+SPT	2464.57(0.00)	1558.93(0.00)	4942.07(0.00)	3404.59(0.00)	5969.10(0.00)	3812.28(0.00)	14422.81(0.00)	10306.29(0.00)
ECT+EDD	2878.21(0.00)	1276.25(0.00)	4949.30(0.00)	2531.80(0.00)	6827.39(0.00)	3132.33(0.00)	14960.80(0.00)	8253.09(0.00)
ECT+LWR	3303.35(0.00)	2093.25(0.00)	5112.70(0.00)	3305.92(0.00)	8060.41(0.00)	5138.47(0.00)	15466.49(0.00)	10421.32(0.00)
ECT+FIFO	3160.73(0.00)	1878.98(0.00)	5597.03(0.00)	3539.85(0.00)	7869.59(0.00)	4710.19(0.00)	16516.99(0.00)	10852.17(0.00)
MET+SPT	3578.21(0.00)	2464.59(0.00)	9833.27(0.00)	7769.08(0.00)	9053.58(0.00)	6306.01(0.00)	31785.62(0.00)	26223.25(0.00)
MET+EDD	4306.56(0.00)	2115.43(0.00)	10695.20(0.00)	7072.82(0.00)	11294.89(0.00)	5777.62(0.00)	36363.18(0.00)	26181.51(0.00)
MET+LWR	5146.58(0.00)	3530.01(0.00)	10920.32(0.00)	8303.70(0.00)	13080.94(0.00)	9020.55(0.00)	36642.67(0.00)	29298.13(0.00)
MET+FIFO	5052.42(0.00)	3258.08(0.00)	11479.69(0.00)	8405.46(0.00)	12779.05(0.00)	8278.97(0.00)	38504.44(0.00)	29800.63(0.00)
EA+SPT	9392.86(0.00)	7094.31(0.00)	7337.65(0.00)	5382.54(0.00)	27894.03(0.00)	21983.59(0.00)	22581.28(0.00)	17167.73(0.00)
EA+EDD	11266.56(0.00)	7981.20(0.00)	7387.13(0.00)	4494.71(0.00)	34779.28(0.00)	24906.38(0.00)	23012.79(0.00)	15024.19(0.00)
EA+LWR	12052.81(0.00)	9070.10(0.00)	7729.31(0.00)	5357.62(0.00)	36889.03(0.00)	28995.71(0.00)	23935.82(0.00)	17209.25(0.00)
EA+FIFO	12406.51(0.00)	9076.90(0.00)	8511.21(0.00)	5807.89(0.00)	36628.47(0.00)	27975.50(0.00)	25817.25(0.00)	18274.63(0.00)
LWIQ+SPT	9873.21(0.00)	7433.28(0.00)	7719.68(0.00)	5601.55(0.00)	28959.28(0.00)	22673.19(0.00)	23732.51(0.00)	17938.37(0.00)
LWIQ+EDD	11688.17(0.00)	7834.65(0.00)	7886.95(0.00)	4816.37(0.00)	35462.49(0.00)	25863.40(0.00)	24400.87(0.00)	15709.88(0.00)
LWIQ+LWR	12190.28(0.00)	9170.81(0.00)	7993.81(0.00)	5488.46(0.00)	37225.87(0.00)	29091.99(0.00)	24730.12(0.00)	17785.42(0.00)
LWIQ+FIFO	13266.50(0.00)	9801.66(0.00)	8827.58(0.00)	6035.38(0.00)	38590.33(0.00)	29681.75(0.00)	26933.27(0.00)	19196.87(0.00)
DRL	2553.03(90.83)	1309.18(138.31)	4666.21(187.39)	2416.71(154.72)	6126.96(241.12)	3136.38(321.79)	13745.5(609.96)	7540.53(442.77)
MTGP(↑)	**2043.05(425.56)**	**822.33(350.60)**	**3826.65(508.94)**	**1590.13(231.43)**	**4927.87(1100.08)**	**1988.61(928.22)**	**10911.87(1458.69)**	**4843.17(688.88)**

narios. The negative improvement percentages of DRL can be attributed to the fact that it makes decisions based on a minibatch of experiences (a number of decisions), which may not encompass all possible decisions of the training instances. As a result, DRL may not respond effectively to changes in the number of training instances. Nonetheless, this property highlights the advantage of DRL in producing relatively stable results with limited training data, particularly in real-world applications. However, improving performance with adequate training data can be challenging. MTGP exhibits significantly higher improvement percentages than DRL, indicating that it can learn more effective scheduling heuristics with enough training data. However, the MTGP is significantly impacted by training data and is less stable than DRL.

In summary, it is evident that as the number of training instances increases, the MTGP consistently outperforms DRL and derives significantly greater benefits from the additional training data. However, it is worth noting that the DRL is more stable (i.e., smaller standard deviation) and less sensitive to changes in training data. This insight could assist industries in selecting an appropriate method based on the specific problem and available resources.

C. Generalization to More Complex Scenarios

Generalization ability is a key criterion for measuring the quality of an algorithm. On one hand, the generalization ability of an algorithm can be verified by the performance of its trained heuristics on unseen test instances at the same scales. In this case, according to the comparison of the test performance in Section V-A, it is evident that the MTGP has better generalization ability than the DRL. On the other hand, the generalization ability can be further proven by extending the use of the trained heuristics on unseen test instances at more complex scales. For this purpose, the scheduling heuristics learned by the MTGP and DRL are tested on more complex instances with three workcenters, and each containing two machines, or with more jobs. To be specific, the test instances are with 248 jobs arrival (2000 unit time), 620 jobs arrival (5000 unit time), six workcenters, and nine workcenters. The same as Section V-A, 100 unseen instances are used for testing. Additionally, the test performance of 16 manually designed scheduling heuristics is also shown as the baseline for comparison for verifying the effectiveness of the MTGP and DRL.

Tables VI and VII give the mean (standard deviation) of the test performance on 100 unseen instances of the 30 independent runs of manually designed scheduling heuristics and the learned scheduling heuristics by the MTGP and DRL on the 16 complex scenarios. It can be seen that the learned scheduling heuristics by the MTGP and DRL can outperform the manually designed ones, even when applied directly to more complex instances without retraining. Specifically, the DRL outperforms most of the manually designed scheduling heuristics across all 16 scenarios, except for ECT+SPT and ECT+EDD. However, the DRL performs worse than ECT+SPT and ECT+EDD on a few scenarios, such as scenarios HH with 248 jobs and 620 jobs, scenario HL with six workcenters, and scenario LH with nine workcenters. On the other hand, the MTGP significantly outperforms all manually designed scheduling heuristics and the DRL, with a statistically superior performance across all 16 scenarios. These results illustrate that both the MTGP and DRL can be applied to more complex scenarios without the need for retraining. Furthermore, the MTGP demonstrates an even better generalization ability than the DRL.

These results demonstrate the strong generalization ability of the scheduling heuristics learned by both the MTGP and DRL, especially the MTGP. Additionally, these findings suggest that both the MTGP and DRL have the potential to efficiently train on small-scale instances and obtain effective heuristics to be used on larger-scale instances. This can be very helpful in practical applications as it enhances efficiency and provides satisfactory performance.

D. Interpretability Analysis

The interpretability of a model affects people's trust in its real-world applications, especially for problems with serious

TABLE VII
THE TEST PERFORMANCE OF THE DRL, THE MTGP, AND COMPARISON METHODS ON EIGHT SCENARIOS **WITH MORE WORKCENTERS**.

Algorithm	6W2M				9W2M			
	HH	HL	LH	LL	HH	HL	LH	LL
ECT+SPT	2064.13(0.00)	1188.56(0.00)	4306.75(0.00)	2641.58(0.00)	3277.57(0.00)	1809.18(0.00)	7037.32(0.00)	4213.86(0.00)
ECT+EDD	2072.81(0.00)	677.98(0.00)	3871.73(0.00)	1535.96(0.00)	3040.24(0.00)	745.99(0.00)	5951.53(0.00)	1957.94(0.00)
ECT+LWR	2546.12(0.00)	1442.97(0.00)	4191.03(0.00)	2424.05(0.00)	3865.55(0.00)	2117.95(0.00)	6666.17(0.00)	3779.77(0.00)
ECT+FIFO	2721.87(0.00)	1477.21(0.00)	4740.88(0.00)	2667.71(0.00)	4385.92(0.00)	2321.10(0.00)	8000.07(0.00)	4436.86(0.00)
MET+SPT	2869.70(0.00)	1770.18(0.00)	7644.24(0.00)	5347.11(0.00)	4136.47(0.00)	2424.18(0.00)	11181.16(0.00)	7409.28(0.00)
MET+EDD	3139.20(0.00)	1030.69(0.00)	7382.28(0.00)	3676.64(0.00)	4083.20(0.00)	1071.75(0.00)	10175.79(0.00)	4323.78(0.00)
MET+LWR	3879.36(0.00)	2350.19(0.00)	7869.49(0.00)	5237.75(0.00)	5278.78(0.00)	3039.67(0.00)	11270.29(0.00)	7098.79(0.00)
MET+FIFO	4029.84(0.00)	2325.42(0.00)	8739.26(0.00)	5601.48(0.00)	5866.33(0.00)	3197.48(0.00)	12894.30(0.00)	6166.80(0.00)
EA+SPT	6502.85(0.00)	4262.83(0.00)	5867.10(0.00)	3741.67(0.00)	9584.38(0.00)	5997.11(0.00)	9473.10(0.00)	5881.28(0.00)
EA+EDD	7059.43(0.00)	3499.17(0.00)	5417.55(0.00)	2432.45(0.00)	9469.86(0.00)	4171.22(0.00)	8171.37(0.00)	3174.08(0.00)
EA+LWR	7375.26(0.00)	4807.66(0.00)	5731.48(0.00)	3446.52(0.00)	10132.08(0.00)	6133.37(0.00)	8826.54(0.00)	5163.08(0.00)
EA+FIFO	8421.70(0.00)	5340.45(0.00)	6600.26(0.00)	3901.09(0.00)	12133.88(0.00)	7222.11(0.00)	10668.17(0.00)	7907.79(0.00)
LWIQ+SPT	6920.89(0.00)	4569.00(0.00)	6165.40(0.00)	3903.02(0.00)	9898.59(0.00)	6143.23(0.00)	9654.39(0.00)	5944.05(0.00)
LWIQ+EDD	7375.49(0.00)	3745.04(0.00)	5786.08(0.00)	2594.14(0.00)	9854.07(0.00)	4394.14(0.00)	8483.86(0.00)	3427.61(0.00)
LWIQ+LWR	7584.30(0.00)	4922.64(0.00)	6009.81(0.00)	3639.31(0.00)	10495.00(0.00)	6390.71(0.00)	9140.49(0.00)	5346.25(0.00)
LWIQ+FIFO	8848.62(0.00)	5657.50(0.00)	6869.03(0.00)	4060.71(0.00)	12318.73(0.00)	7370.78(0.00)	10888.21(0.00)	6300.68(0.00)
DRL	1904.02(109.91)	674.67(156.46)	3694.60(133.67)	1356.30(176.97)	2822.25(199.02)	823.76(281.27)	5765.97(189.94)	1846.93(346.94)
MTGP(↑)	**1590.04(421.11)**	**366.24(181.32)**	**3339.09(449.99)**	**937.45(137.56)**	**2441.90(996.12)**	**388.37(203.67)**	**5314.27(771.25)**	**1219.37(191.55)**

TABLE VIII
THE LEARNED **ROUTING RULE AND SEQUENCING RULE SIZES** BY THE MTGP AND DRL IN THE FOUR SCENARIOS.

Scenario	Routing		Sequencing	
	DRL	MTGP	DRL	MTGP
HH	64(0.0)	**47.53(53.07)**(↑)	180(0.0)	**42.87(83.38)**(↑)
HL	64(0.0)	**44.67(59.27)**(↑)	180(0.0)	**65.80(93.95)**(↑)
LH	64(0.0)	**40.73(47.02)**(↑)	180(0.0)	**58.73(59.61)**(↑)
LL	64(0.0)	**52.67(57.83)**(↑)	180(0.0)	**58.67(70.84)**(↑)

ethical concerns [50]. In [51], interpretability is defined as the capability to offer explanations in a comprehensible manner to humans. Additionally, it has been observed that smaller rules (with fewer nodes within the tree) generally exhibit better interpretability [52]. This section analyzes the interpretability of the scheduling heuristics learned by the MTGP and DRL.

Firstly, a comparison is conducted regarding the sizes of the routing rule and sequencing rule learned by MTGP and DRL. Table VIII presents the mean and standard deviation of the routing rule size and sequencing rule size obtained from 30 independent runs of MTGP and DRL across four different scenarios. For the DRL, the rule size is determined by counting the number of hidden nodes in the neural network associated with it. For the MTGP, the rule size is determined by counting the number of nodes in the corresponding tree. To compare the two approaches, a Wilcoxon rank-sum test is conducted with a significance level of 0.05. In Table VIII, the symbol "↑" following the results of the MTGP method indicates that the corresponding rule sizes of the MTGP are significantly smaller than those of the DRL algorithm. The results demonstrate that the MTGP method consistently yields significantly smaller routing and sequencing rule sizes compared to the DRL method in all four scenarios. In conclusion, the MTGP can learn significantly smaller scheduling heuristics than the DRL. However, it should be noted that the network structure of the DRL, directly related to the rule size, is usually designed in advance and fixed during the learning process, and that the DRL only learns the parameters. So the rule size is known and fixed during the learning process. It is possible that a small network (with a few nodes) can be used as a basic structure. For the MTGP, it allows different scheduling heuristics to have different sizes depending on the tree depth.

Apart from the rule size, this paper also analyzes the structures of the learned scheduling heuristics by the MTGP and DRL. The network structures of the routing rule and sequencing rule learned by the DRL of a single run on the scenario HH are shown in Figs. 8 and 9. Note that there are more than four nodes in the input layers and hidden layers of the network structures of both the routing rule and sequencing rule. However, to save space and make it easier to demonstrate and analyze, the maximum number of nodes on each layer is restricted to four. The value along each edge represents the learned weights. The selected scheduling heuristics have promising test performance. It can be seen that the structures of both the routing rule and sequencing rule exhibit a significant level of complexity, with numerous nodes and weights comprising their structures. The routing rule comprises an input layer, an output layer, and five hidden layers, with each hidden layer containing a substantial number of nodes. Similarly, the sequencing rule comprises an input layer, an output layer, and six hidden layers, with even more nodes than the routing rule. These structures are predetermined. Understanding the rationale behind the initial design of these structures, as well as the reasoning behind decision-making based on the complex networks and learned weights, can be challenging.

Different from the DRL, the MTGP needs not only to tune the parameters but also to build the structures of the heuristics [53]. Taking a learned scheduling heuristic by the MTGP for scenario HH as an example, Fig. 10 illustrates the tree structures of a promising routing rule and a promising sequencing rule derived from this scheduling heuristic.

As observed, the routing rule is determined by three terminals: TIS, PT, and WIQ, with PT being the most frequently utilized terminal, appearing five times in the rule. The WIQ is used three times, and the TIS is only used once. Since TIS is a feature associated with the operation, it remains consistent

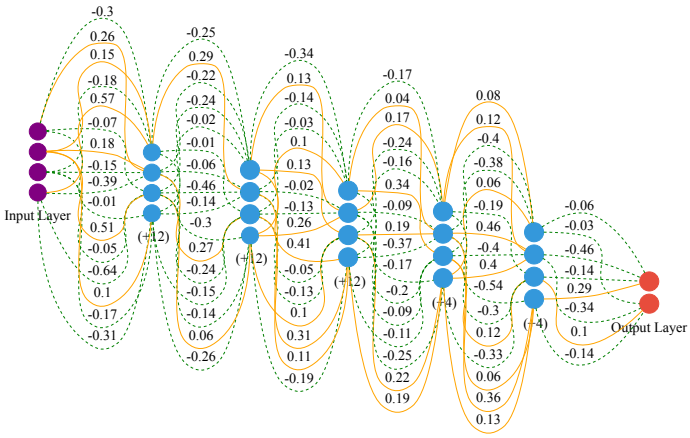


Fig. 8. The network structure of routing rule from one scheduling heuristic learned by the DRL.

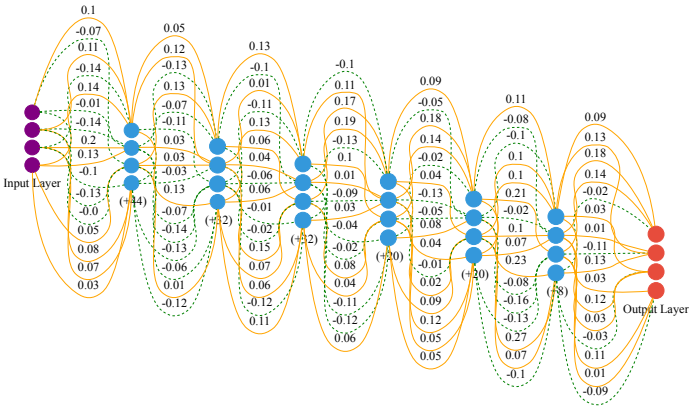


Fig. 9. The network structure of sequencing rule from one scheduling heuristic learned by the DRL.

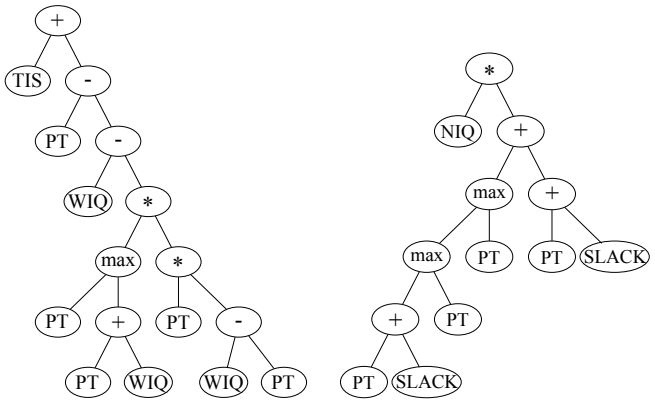
across all candidate machines. In this case, the routing rule can be expressed as a simplified form, denoted as R_0 and shown in Eq. (12).

$$R_0 = (PT - WIQ) \times (1 - PT \times (PT + WIQ)) + TIS \tag{12}$$

Based on the range of processing time (PT) and remaining work in the waiting queue (WIQ), $1 - PT \times (PT + WIQ)$ must be smaller than 0. Thus, this rule suggests that if the remaining work in the waiting queue (WIQ) is larger than the processing time (PT) of the ready operation, then this routing rule prefers machines with faster processing rates (i.e., smaller PT) and larger work remaining in the queue (i.e., larger WIQ). On the other hand, if the WIQ of the machine is smaller than the PT of the current ready operation, then this routing rule favors machines that have a slower process rate (larger PT) and less work remaining in the queue (smaller WIQ). This ensures that the selected machines are best suited to handle the current workload.

The sequencing rule consists of three terminals: NIQ, PT, and SLACK, with PT being the most commonly used terminal, appearing four times in the rule. The SLACK is used twice, while the NIQ is used only once. This sequencing rule can be simplified as S_0, as depicted in Eq. (13).

$$S_0 = 2 \times NIQ \times (PT + SLACK) \tag{13}$$



(a) The routing tree. (b) The sequencing tree.

Fig. 10. The tree structures of routing rule and sequencing rule from one scheduling heuristic learned by the MTGP on scenario HH.

Since NIQ represents the number of operations in the waiting queue, it is constant for all candidate operations. Therefore, this sequencing rule primarily focuses on the operation's processing time (PT) and slack (SLACK). It tends to prioritize operations with smaller processing time (PT) and tighter slack.

On the basis of the above analysis of the scheduling heuristic learned through the MTGP, it is evident that attributes of the machine, such as processing time and work remaining in the waiting queue play crucial roles in routing decisions. With respect to the sequencing decision, information about the operation, such as processing time and time remaining before the due, play decisive roles. This observation is consistent with our intuition that machines with shorter processing times are better for processing ready operations, while operations that are about to be due should be completed first. Also, the objective in this paper is tardiness, and SLACK is a very important terminal related to tardiness. Based on the visual presentation of the tree structure of the routing rule and the sequencing rule and the feature analysis, it becomes possible to distinguish the importance of different terminals for each rule and thus have a more comprehensive and intuitive understanding of the scheduling heuristics learned through the MTGP method. This shows the good interpretability of the MTGP.

In conclusion, it is observed that the MTGP is able to provide a more intuitive scheduling heuristic compared to the DRL. The scheduling heuristic learned by the MTGP is easier for humans to comprehend, providing greater confidence for real-world applications.

E. Further Discussions

According to the results and analyses presented in the previous sections, for the present stage, the following conclusions can be drawn for the research questions investigated in this paper.

RQ1: Which of the MTGP and the DRL is better for solving the DFJSS problem?

The experimental results demonstrate that the learned scheduling heuristics by the MTGP outperform those learned

by the DRL method and manually designed scheduling heuristics. The superior performance of the MTGP over the DRL approach may be attributed to several factors. Firstly, the MTGP considers the cooperation between routing rules and sequencing rules, which is not taken into account in the DRL approach. Secondly, designing an appropriate reward function in DRL is challenging, while the MTGP does not require the design of a reward function. Finally, the sequencing action space in DRL is limited, whereas the MTGP can explore a more extensive search space, potentially leading to better scheduling heuristics.

RQ2: How does the amount of training data affect the performance of the MTGP and the DRL?

The DRL approach is less reliant on training data than the MTGP for learning scheduling heuristics, which demonstrates the stability advantage of the DRL. However, the MTGP can learn better scheduling heuristics with an increasing amount of training data. This is likely due to the fact that the DRL makes decisions based on a minibatch of experience, focusing on local decisions, while the MTGP considers its performance on the entire instances, focusing on global decisions. Therefore, the choice between the MTGP and DRL for learning scheduling heuristics should be based on the specific problem at hand and available resources. This finding provides valuable insights for selecting the appropriate algorithm for real-world applications.

RQ3: How well can the MTGP and the DRL be generalized to solve more complex instances (i.e., considering more jobs and more workcenters)?

Both the MTGP and DRL have the ability to generalize to solve complex instances without retraining. This indicates that effective scheduling heuristics learned in small-scale instances can be applied to large-scale instances. This feature makes both the MTGP and DRL valuable tools in real-world applications. Furthermore, the MTGP has shown better generalization ability than the DRL, highlighting its potential advantage in solving complex scheduling problems.

RQ4: Which of the MTGP and the DRL can achieve better interpretability for learning scheduling heuristics?

The scheduling heuristics learned by the MTGP demonstrate better interpretability, which is attributed to the smaller rule size and the ability to visualize its individual representations. The superior interpretability of learned scheduling heuristics by the MTGP provides users with greater confidence in using them for practical applications. Additionally, the MTGP is capable of performing parameter and structure learning simultaneously. In contrast, the DRL requires experts to design the structure in advance, only tuning the parameters during the learning process. However, determining an optimal structure for complex scheduling problems in advance can be challenging, requiring different structures for different problems or scenarios. Consequently, the MTGP, at the present stage, not only offers superior interpretability but is also easier to use than the DRL.

To summarize, both the MTGP and DRL have their strengths and weaknesses, and the choice between them for learning scheduling heuristics depends on the specific problem and available resources. If the problem is complex, the reward function is difficult to define, or high interpretability is required, then the MTGP is a better choice. On the other hand, if the amount of training data is limited and stable results are expected, then the DRL is preferable. Additionally, the MTGP is better suited for optimizing global performance, while the DRL excels at local decision-making. Combining the two approaches could be a promising solution, allowing for effective coordination between global and local decision-making.

VI. Conclusions

The aim of this paper is to fill the research gap in comparing GP and RL methods that learn scheduling heuristics for dynamic scheduling problems. In order to achieve this goal, this paper investigated and conducted a comparison of a recently proposed and representative GP (i.e., MTGP) method and a recently proposed and representative RL (i.e., DRL) method to automatically learn scheduling heuristics for the DFJSS problem in terms of performance, the influence of training data, generalization ability, and interpretability. Extensive experiments are conducted on different scenarios to compare the effectiveness of the MTGP and DRL methods. The results and analyses demonstrate that both the MTGP and DRL methods have their strengths and weaknesses. In terms of test performance on unseen instances, both the MTGP and DRL outperform the widely used manually designed scheduling heuristics. The MTGP method performs better than the DRL method in the investigated scenarios. The DRL method offers good stability and is less sensitive to the training data, whereas the MTGP method has the potential to learn better scheduling heuristics with an increase in training data. Additionally, both the MTGP and DRL methods show good generalization ability. Finally, the MTGP method exhibits significant advantages over the DRL method in terms of interpretability.

Although this paper shows that the DRL method performs worse than the MTGP method on the DFJSS problem at the present stage, RL-based methods have their unique advantages and are expected to be potentially improved by, for example, using well-designed reward functions. For future work, the plan includes conducting further investigations on GP and RL and exploring different combinations of these two algorithms. This could be achieved by incorporating different strategies and mechanisms, such as feature selection, surrogate, and multi-task, to improve the performance of both GP and RL. Moreover, it would be interesting to explore the potential of using RL to train a policy that selects suitable scheduling heuristics at different decision points, where the candidate scheduling heuristics are learned by GP. Another promising direction would be to design a new evaluation strategy by combining the immediate feedback of RL and the long-term feedback of GP, which could be helpful in learning good scheduling heuristics. Furthermore, it is interesting to incorporate RL to perform smarter mutation or crossover for GP or adaptively decide the functions for GP. The investigation will focus on whether exploring these combinations could

potentially enhance the efficiency and effectiveness of learning scheduling heuristics for dynamic scheduling problems.

REFERENCES

[1] X. Shen and X. Yao, "Mathematical modeling and multi-objective evolutionary algorithms applied to dynamic flexible job shop scheduling problems," *Information Sciences*, vol. 298, pp. 198–224, 2015.

[2] Y. Fu and P. W. Anderson, "Application of statistical mechanics to np-complete problems in combinatorial optimisation," *Journal of Physics A: Mathematical and General*, vol. 19, no. 9, p. 1605, 1986.

[3] Y. Iwasaki, I. Suzuki, M. Yamamoto, and M. Furukawa, "Job-shop scheduling approach to order-picking problem," *Transactions of the Institute of Systems, Control and Information Engineers*, vol. 26, no. 3, pp. 103–109, 2013.

[4] M. Li and G.-G. Wang, "A review of green shop scheduling problem," *Information Sciences*, vol. 589, pp. 478–496, 2022.

[5] S. Singh and I. Chana, "A survey on resource scheduling in cloud computing: Issues and challenges," *Journal of Grid Computing*, vol. 14, no. 2, pp. 217–264, 2016.

[6] D. Ouelhadj and S. Petrovic, "A survey of dynamic scheduling in manufacturing systems," *Journal of Scheduling*, vol. 12, no. 4, pp. 417–431, 2009.

[7] F. Zhang, S. Nguyen, Y. Mei, and M. Zhang, *Genetic Programming for Production Scheduling: An Evolutionary Learning Approach*. Springer, 2021.

[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] K. Kofler, I. u. Haq, and E. Schikuta, "A parallel branch and bound algorithm for workflow qos optimization," in *Proceedings of the International Conference on Parallel Processing*, 2009, pp. 478–485.

[10] S. Mohammadi, L. PourKarimi, and H. Pedram, "Integer linear programming-based multi-objective scheduling for scientific workflows in multi-cloud environments," *Journal of Supercomputing*, vol. 75, no. 10, pp. 6683–6709, 2019.

[11] J.-q. Li and Q.-k. Pan, "Solving the large-scale hybrid flow shop scheduling problem with limited buffers by a hybrid artificial bee colony algorithm," *Information Sciences*, vol. 316, pp. 487–502, 2015.

[12] R. Zhang and C. Wu, "Bottleneck machine identification method based on constraint transformation for job shop scheduling with genetic algorithm," *Information Sciences*, vol. 188, pp. 236–252, 2012.

[13] 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.

[14] 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.

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

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

[17] Y. Bi, B. Xue, and M. Zhang, "An effective feature learning approach using genetic programming with image descriptors for image classification [research frontier]," *IEEE Computational Intelligence Magazine*, vol. 15, no. 2, pp. 65–77, 2020.

[18] Z. Liu, Y. Wang, X. Liang, Y. Ma, Y. Feng, G. Cheng, and Z. Liu, "A graph neural networks-based deep q-learning approach for job shop scheduling problems in traffic management," *Information Sciences*, vol. 607, pp. 1211–1223, 2022.

[19] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Survey on genetic programming and machine learning techniques for heuristic design in job shop scheduling," *IEEE Transactions on Evolutionary Computation*, pp. 1–21, mar 2023.

[20] S. Chand, Q. Huynh, H. Singh, T. Ray, and M. Wagner, "On the use of genetic programming to evolve priority rules for resource constrained project scheduling problems," *Information Sciences*, vol. 432, pp. 146–163, 2018.

[21] Y. Li, "Deep reinforcement learning: An overview," *ArXiv Preprint ArXiv:1701.07274*, 2017.

[22] 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.

[23] R. Liu, R. Piplani, and C. Toro, "Deep reinforcement learning for dynamic scheduling of a flexible job shop," *International Journal of Production Research*, pp. 1–21, 2022.

[24] F. Zhang, Y. Mei, S. Nguyen, K. C. Tan, and M. Zhang, "Multitask genetic programming based generative hyper-heuristics: A case study in dynamic scheduling," *IEEE Transactions on Cybernetics*, vol. 52, no. 10, pp. 10 515–10 528, 2022.

[25] 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.

[26] Y. Wu, W. Song, Z. Cao, J. Zhang, and A. Lim, "Learning improvement heuristics for solving routing problems," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 9, pp. 5057–5069, 2021.

[27] C. Zhang, W. Song, Z. Cao, J. Zhang, P. S. Tan, and X. Chi, "Learning to dispatch for job shop scheduling via deep reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1621–1632, 2020.

[28] W. Song, X. Chen, Q. Li, and Z. Cao, "Flexible job shop scheduling via graph neural network and deep reinforcement learning," *IEEE Transactions on Industrial Informatics*, 2022.

[29] 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.

[30] Y. Zhou, 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.

[31] Y. Jia, Y. Mei, and M. Zhang, "Learning heuristics with different representations for stochastic routing," *IEEE Transactions on Cybernetics*, vol. 53, no. 5, pp. 3205–3219, 2023.

[32] 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.

[33] M. Xu, Y. Mei, F. Zhang, and M. Zhang, "Genetic programming with diverse partner selection for dynamic flexible job shop scheduling," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2022, pp. 615–618.

[34] M. Brameier, W. Banzhaf, and W. Banzhaf, *Linear genetic programming*. Springer, 2007, vol. 1.

[35] C. Ferreira, "Gene expression programming in problem solving," in *Soft computing and industry: recent applications*. Springer, 2002, pp. 635–653.

[36] M. Xu, Y. Mei, F. Zhang, and M. Zhang, "Genetic programming with cluster selection for dynamic flexible job shop scheduling," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2022, pp. 1–8.

[37] 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.

[38] M. L. Puterman, "Markov decision processes," *Handbooks in Operations Research and Management Science*, vol. 2, pp. 331–434, 1990.

[39] J. Chang, D. Yu, Y. Hu, W. He, and H. Yu, "Deep reinforcement learning for dynamic flexible job shop scheduling with random job arrival," *Processes*, vol. 10, no. 4, p. 760, 2022.

[40] S. Luo, "Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning," *Applied Soft Computing*, vol. 91, p. 106208, 2020.

[41] H. Wang, J. Cheng, C. Liu, Y. Zhang, S. Hu, and L. Chen, "Multi-objective reinforcement learning framework for dynamic flexible job shop scheduling problem with uncertain events," *Applied Soft Computing*, vol. 131, p. 109717, 2022.

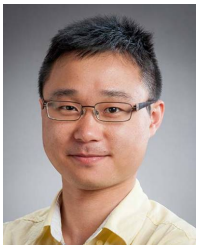
[42] S. Luo, L. Zhang, and Y. Fan, "Dynamic multi-objective scheduling for flexible job shop by deep reinforcement learning," *Computers & Industrial Engineering*, vol. 159, p. 107489, 2021.

[43] ——, "Real-time scheduling for dynamic partial-no-wait multiobjective flexible job shop by deep reinforcement learning," *IEEE Transactions on Automation Science and Engineering*, 2021.

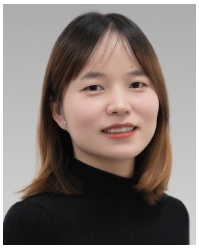
[44] Y. Li, W. Gu, M. Yuan, and Y. Tang, "Real-time data-driven dynamic scheduling for flexible job shop with insufficient transportation resources

using hybrid deep q network," *Robotics and Computer-Integrated Manufacturing*, vol. 74, p. 102283, 2022.
[45] 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.
[46] C. W. Pickardt, T. Hildebrandt, J. Branke, J. Heger, and B. Scholz-Reiter, "Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems," *International Journal of Production Economics*, vol. 145, no. 1, pp. 67–77, 2013.
[47] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, 2019.
[48] F. M. De Rainville, F. A. Fortin, M. A. Gardner, M. Parizeau, and C. Gagné, "Deap: A python framework for evolutionary algorithms," in *Proceedings of the Conference Companion on Genetic and Evolutionary Computation*, 2012, pp. 85–92.
[49] Z. A. Aziz, D. N. Abdulqader, A. B. Sallow, and H. K. Omer, "Python parallel processing and multiprocessing: A review," *Academic Journal of Nawroz University*, vol. 10, no. 3, pp. 345–354, 2021.
[50] D. V. Carvalho, E. M. Pereira, and J. S. Cardoso, "Machine learning interpretability: A survey on methods and metrics," *Electronics*, vol. 8, no. 8, p. 832, 2019.
[51] Y. Zhang, P. Tiňo, A. Leonardis, and K. Tang, "A survey on neural network interpretability," *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2021.
[52] Y. Mei, S. Nguyen, and M. Zhang, "Constrained dimensionally aware genetic programming for evolving interpretable dispatching rules in dynamic job shop scheduling," in *Proceedings of the Asia-Pacific Conference on Simulated Evolution and Learning*, 2017, pp. 435–447.
[53] Y. Mei, Q. Chen, A. Lensen, B. Xue, and M. Zhang, "Explainable artificial intelligence by genetic programming: A survey," *IEEE Transactions on Evolutionary Computation*, pp. 1–21, 2022.

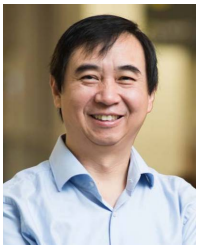
Meng Xu received the B.Sc. and M.Sc. degrees from the Bejing Institute of Technology, Beijing, China, in 2017 and 2020, respectively. She is currently pursuing a Ph.D. degree in computer science with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. Her current research interests include evolutionary computation, hyper-heuristic learning/optimization, 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 and Associate Dean (Research) at the Faculty of Engineering, Victoria University of Wellington, Wellington, New Zealand. His research interests include evolutionary computation for combinatorial optimisation, genetic programming, automatic algorithm design, explainable AI, multi-objective optimization, transfer/multitask learning, and optimization. He has published on top journals in EC and Operations Research (OR) such as IEEE TEVC, IEEE TCYB, EJOR, IEEE Transactions on Services Computing, and ACM Transactions on Mathematical Software. He won an IEEE Transactions on Evolutionary Computation Outstanding Paper Award 2017. He is an Associate Editor of IEEE Transactions on Evolutionary Computation, and an Editorial Board Member/Associate Editor of four other international journals. He is the Chair of the 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 three IEEE CIS Task Forces and two IEEE CIS Technical Committees. 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 postdoctoral research fellow in computer science with the School of Engineering and Computer Science, Victoria University of Wellington, New Zealand. She has over 55 papers in refereed international journals and conferences. Her research interests include evolutionary computation, hyper-heuristic learning/optimization, 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 received the B.E. and M.E. degrees from the Artificial Intelligence Research Center, Agricultural University of Hebei, Hebei, China, and the Ph.D. degree in computer science from RMIT University, Melbourne, VIC, Australia, in 1989, 1992, and 2000, respectively. He is a Professor of Computer Science, Head of the Evolutionary Computation Research Group. His research interests include evolutionary computation, genetic programming, multi-objective optimization, job shop scheduling. He has published over 700 papers in refereed international journals and conferences. Prof. Zhang is a Fellow of Royal Society of New Zealand, a Fellow of Engineering New Zealand, a Fellow of IEEE, an IEEE CIS Distinguished Lecturer. He was the chair of the IEEE CIS Intelligent Systems and Applications Technical Committee, the chair for the IEEE CIS Emergent Technologies Technical Committee and the Evolutionary Computation Technical Committee, and a member of the IEEE CIS Award Committee. He is a vice-chair of the IEEE CIS Task Force on Evolutionary Feature Selection and Construction, a vice-chair of the Task Force on Evolutionary Computer Vision and Image Processing, and the founding chair of the IEEE Computational Intelligence Chapter in New Zealand.