Niching Genetic Programming to Learn Actions for Deep Reinforcement Learning in Dynamic Flexible Scheduling

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

***Abstract*—Dynamic Flexible Job Shop Scheduling (DFJSS) is a critical combinatorial optimisation problem known for its dynamic nature and flexibility of machines. Traditional scheduling methods face limitations in adapting to such dynamic and flexible environments. Recently, there has been a trend in employing reinforcement learning (RL) to train scheduling agents for selecting manual scheduling heuristics at various decision points for DFJSS. However, the effectiveness of RL is constrained by the limited efficacy of the manually designed scheduling heuristics. Additionally, the process of manually designing diverse scheduling heuristics as the actions demands significant expert knowledge. In response, this paper proposes a Niching genetic programming (GP)-assisted RL method that leverages the evolutionary capabilities of GP to help RL solve the DFJSS problem effectively. Specifically, instead of using those manual scheduling heuristics, the RL actions are replaced with scheduling heuristics evolved by the Niching GP to optimise and adapt these heuristics based on real-time feedback from the environment. Experimental results demonstrate the effectiveness of the proposed method in comparison to the widely used manual scheduling heuristics and the baseline deep RL method. Further analyses reveal that the effectiveness of the proposed method is due to the behavioral differences among heuristics learned by the Niching GP, serving as actions for the RL. In addition, the effectiveness of the proposed algorithm benefits from the comparable percentages of contributions made by these learned heuristics throughout the long-term scheduling process.**

***Index Terms*—heuristic learning, genetic programming, reinforcement learning, dynamic scheduling.**

I. Introduction

Dynamic flexible job shop scheduling (DFJSS) is a complex optimisation challenge that arises in manufacturing and production environments [1]. It extends the traditional job shop scheduling (JSS) [2] by incorporating dynamic event(s) and flexibility of machines, introducing variability in both job characteristics and machine capabilities [3]. In a typical manufacturing setting, a job shop consists of a set of machines, each with different processing capabilities, and a set of jobs with operations, each with specific processing requirements [4]. The goal is to determine the operation sequence (sequencing) and machine assignment (routing), taking into account various constraints and objectives [5], [6]. The dynamic aspect of DFJSS introduces uncertainties and changes over time, such as varying job arrival times, processing times, and machine breakdowns [7]. The flexibility of DFJSS refers to the ability to assign jobs to different machines. Each job may have multiple feasible routes through different machines, and decisions need to be made regarding the optimal routing and sequencing of jobs to maximise efficiency and meet various performance criteria [8].

The dynamic nature of DFJSS makes it necessary to handle it with an adaptive scheduling strategy that can adapt to real-time changes in manufacturing [9]. Scheduling heuristics, such as the shortest processing time as the sequencing rule and the work in the waiting queue as the routing rule, have been widely used for solving the DFJSS problem, which can react in real-time [10]. However, manually designing scheduling heuristics is a time-consuming process that demands expertise in the domain [11]. In this case, reinforcement learning (RL) [12] has gained popularity for automatically learning high-quality scheduling heuristics/agents in DFJSS.

Currently, some RL-based approaches presume a fixed and constant number of machines/operations as actions at decision points and learn an end-to-end strategy to solve the JSS problems [13]. Consequently, these approaches face challenges when applied to DFJSS with varying numbers of machines/operations [13]. In response to this challenge, researchers have attempted to use indirect ways to overcome the difficulty of having different numbers of candidate machines/operations at different decision points. In particular, the utilisation of manually designed scheduling heuristics as actions in RL has been adopted for handling the varying number of machines/operations [14], [15]. In this approach, when RL selects a specific action (scheduling heuristic), the chosen scheduling heuristic is subsequently employed to make decisions regarding the candidate machines and operations. However, this kind of methods still have limitations. Firstly, the manually designed scheduling heuristics often exhibit an average level of quality, thereby constraining the overall quality of high-level scheduling heuristics learned by RL. Secondly, the manual design process of a set of diverse scheduling heuristics is time-consuming and requires a lot of domain knowledge from experts. Genetic programming (GP) [16] has been applied with notable success to address JSS problems [17]. Instead of relying on manually designed heuristics, GP employs an evolutionary process to iteratively learn and refine candidate scheduling heuristics over multiple

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

Manuscript received April 19, 2021; revised August 16, 2021.

generations. The key advantage of using GP for JSS lies in its ability to automatically explore a wide range of potential scheduling heuristics and adapt to the specific characteristics of the problem at hand with little domain knowledge [18]. Traditional GP methods typically focus on obtaining the best scheduling heuristic without emphasising the diversity within the population. Niching [19] has proven to be an effective strategy employed in GP to enhance its effectiveness [20]. This is achieved by increasing population diversity, fostering the presence of multiple diverse scheduling heuristics within the population.

Drawing on the strengths of GP and the niching strategy, this paper investigates the enhancement of RL by employing GP with a niching strategy to autonomously learn effective and diverse scheduling heuristics as actions to address the DFJSS problems. The integration of GP aims to empower the RL method in learning more effective scheduling agents, enabling intelligent decision-making using specific heuristics across diverse scheduling scenarios. To be specific, the contributions of this paper are as follows.

1) This paper proposes a novel two-stage learning method that integrates the benefits of GP into RL for addressing the DFJSS problems. To the best of our knowledge, this is the first attempt to combine these two methods in this domain. The proposed method enhances RL's capability in learning intelligent scheduling agents and provides valuable insights for future research on the integration of GP and RL.
2) This paper proposes a Niching GP for the first stage. The proposed Niching GP can automatically learn a set of diverse and high-quality scheduling heuristics as actions for RL. This approach offers several advantages: (1) it reduces time requirement and dependency on domain-specific knowledge compared to manual heuristic design; (2) it yields actions with superior performance compared to manual heuristics; (3) it offers a variety of heuristics as actions, capable of addressing diverse decision-making scenarios.
3) This paper employs RL to utilise the learned high-quality and diverse scheduling heuristics from Niching GP as actions in the second stage. This contributes to intelligently selecting appropriate scheduling heuristics when encountering different decision points.
4) The proposed method outperforms the baseline RL method and widely used manually designed scheduling heuristics for DFJSS. Additionally, the effectiveness of Niching GP is validated by comparing the proposed method with a classical GP-assisted RL method. Further analyses confirm that the effectiveness of the proposed method is attributed to the contributions made by the learned heuristics throughout the long-term scheduling process.

The subsequent sections of this paper are organised as follows. Section II provides the problem formulation of DFJSS and an overview of related works, encompassing RL and GP methods applied to scheduling problems. Section III describes the proposed method. The experimental design and results are detailed in Sections IV and V, respectively. Additional analyses are presented in Section VI. Lastly, Section VII offers conclusions for this paper.

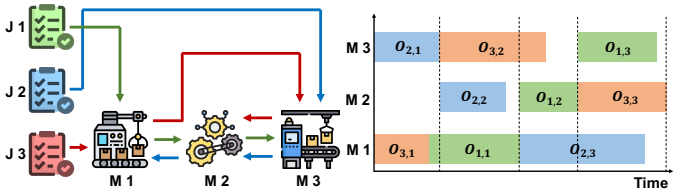


Fig. 1. An example diagram illustrating the representation of the solution (schedule) in DFJSS.

II. Background

A. Dynamic Flexible Job Shop Scheduling

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

The scheduling complexity and production performance are influenced by three important factors, which are described as follows:

1) **Job arrival rate**: a higher job arrival rate can result in busier machine utilisation level and increased system congestion;
2) **Job and machine heterogeneity**: variability in the processing times for operations on different machines introduces complexity;
3) **Tightness of due dates**: a tighter due date reduces operational flexibility and job slack, often leading to higher tardiness.

In the context of DFJSS, a solution or schedule denotes a set of allocations for the processing start time $t_{j,i}^{start}$ of each operation $O_{j,i}$ on its assigned machine Ω_m for all jobs. Fig. 1 gives an example diagram illustrating the representation of the solution (schedule) in DFJSS considering 3 jobs and 3 machines.

Total tardiness T_{total} is a crucial objective in practical industries, and it is the focus of optimisation in this paper. The definition of T_{total} is outlined as follows.

$$\begin{aligned} Ttotal &= \textstyle\sum_{j=1}^{n} tard(J_j), \\ tard(J_j) &= \max\{t_j^{com} - t_j^{due}, 0\} \end{aligned} \tag{1}$$

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

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

$$\min\ Ttotal \tag{2}$$

s.t.:

$$t^{start}_{j_1,i_1} \leq t^{start}_{j_2,i_2} - t^{pro}_{j_1,i_1,m} + L \cdot (1 - x_{j_1,i_1,j_2,i_2,m}), \quad \forall j_1, j_2 = 1, \ldots, n;\ \forall i_1 = 1, \ldots, q_{j_1};\ \forall i_2 = 1, \ldots, q_{j_2};\ \forall m = 1, \ldots, k \tag{3}$$

$$t^{start}_{j,1} \geq t^{arr}_{j},\ \forall j = 1, \ldots, n \tag{4}$$

$$t^{com}_{j,i} \geq t^{start}_{j,i} + t^{pro}_{j,i,m} \cdot y_{j,i,m},\ \forall j = 1, \ldots, n;\ \forall i = 1, \ldots, q_j;\ \forall m = 1, \ldots, k \tag{5}$$

$$t^{start}_{j,i+1} \geq t^{com}_{j,i},\ \forall j = 1, \ldots, n;\ \forall i = 1, \ldots, q_j - 1 \tag{6}$$

$$\textstyle\sum_{m=1}^{k_{j,i}} y_{j,i,m} = 1,\ \forall j = 1, \ldots, n;\ \forall i = 1, \ldots, q_j \tag{7}$$

$$x_{j_1,i_1,j_2,i_2,m} = \begin{cases} 1, \text{ if } O_{j_1,i_1} \text{ is processed by } \Omega_m, \\ \quad \text{before } O_{j_2,i_2}, \\ 0, \text{ otherwise.} \end{cases} \tag{8}$$

$$y_{j,i,m} = \begin{cases} 1, \text{ if } O_{j,i} \text{ is assigned to } \Omega_m, \\ 0, \text{ otherwise.} \end{cases} \tag{9}$$

Constraint (3) specifies that a machine can only process one operation at a time, with L representing a sufficiently large constant. Constraint (4) denotes that the processing of each job's first operation can commence only after the job is released. Constraint (5) establishes the relationship between the processing start time $t^{start}_{j,i}$ and the processing completion time $t^{com}_{j,i}$ for the operation $O_{j,i}$. Constraint (6) 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 (7) indicates that an operation can be processed only on one of its candidate machines.

B. Related Work

1) Reinforcement Learning for JSS: RL-based approaches have recently attracted much attention for addressing diverse scheduling problems, either through direct or indirect approaches [21]. In an indirect approach, RL agents are combined with manually designed scheduling heuristics. Conversely, the direct approach involves extracting state features by observing the environment and generating a scheduling scheme directly with the agent, often referred to as "end-to-end". End-to-end RL approaches are primarily employed in static scheduling problems or scenarios where the number of jobs/machines is predetermined. The action space remains constant, set to the same number as the jobs/machines, across different decision points. For instance, in [22], the actions directly represent candidate machines, while in [23], the actions correspond to candidate jobs. In [24], an innovative end-to-end deep RL (DRL) framework is designed to tackle the scheduling problem. It demonstrates promising results in this context, but it still encounters challenges when applied to more complex dynamic scenarios. Following [24], the same authors further propose a new end-to-end DRL method for solving the DFJSS problem [25]. The main principle is to divide the DFJSS problem into multiple static scheduling problems. However, the inherent essence of such an approach is still to view the dynamic problem as a static problem to be solved. This does not allow for timely response to changes in the scheduling environment and real-time decision-making, which ultimately affects the quality of the solution. Moreover, such an approach is not efficient. The advantage of end-to-end methods lies in minimising the need for human-designed heuristics. This approach allows the model to learn decision-making directly from raw observations. However, a limitation of these methods is their inability to handle problems with dynamically changing action spaces, such as DFJSS. In DFJSS, the number of jobs can vary in response to a dynamic environment. Consequently, the number of available actions of DRL cannot remain constant throughout, as it needs to adapt to the requirements of changing jobs in the scheduling system.

In the indirect application of DRL to scheduling problems, manually designed scheduling heuristics are commonly employed as the agents' actions. These DRL methods address the challenges faced by end-to-end DRL approaches, which encounter difficulties with dynamically changing action spaces. Instead of directly utilising machines/operations as actions, these methods employ manually designed rules as actions. In [26], a DRL method with a Deep Q-Network (DQN) network is proposed for the DJSS problem. This paper specifically focuses on determining the sequencing rule, and the actions employed consist of widely used manually designed sequencing rules. In [27], a DQN model is proposed to solve a multi-objective flexible JSS problem with crane transportation and setup times. To tackle the two subproblems within flexible JSS, operation sequencing and machine routing, this paper integrates manually designed composite scheduling heuristics, combining these subproblems into a unified framework. Then an agent is learned to make decisions among these composite scheduling heuristics. Several other studies also adopt a similar approach by manually designing composite scheduling heuristics as actions for RL to solve the DFJSS problem [14], [15], [28], [29]. However, it is crucial to acknowledge that, for these studies, the strategy of combining the two subproblems is typically not optimal. It cannot fully capture the interaction between the routing and sequencing rules [30], which could substantially reduce the search space, potentially constraining the exploration of superior solutions.

In addition to the aforementioned methods that directly utilise manually designed composite scheduling heuristics as actions, some approaches leverage RL to learn the weights associated with these manually designed composite scheduling heuristics, resulting in a weighted scheduling heuristic [31]. This method can be effective in adapting to dynamic and changeable environments, as it does not necessitate consideration of the impact of variations in the number of candidate machines/operations at different decision points. However, a notable limitation of this method is the pre-determination of the aggregation function, which solely relies on a weighted sum function. This predetermined choice still narrows down the search space by limiting the range of functions available. Additionally, no sufficient evidence is provided on explaining why this particular predetermined aggregation function is able to yield good results.

In [32], a DRL method with a double DQN model is proposed to solve the DFJSS problems. Different from the

aforementioned studies utilising composite scheduling heuristics as actions to jointly address sequencing and routing decisions, they employ a distinct training strategy. Specifically, the sequencing rule and the routing rule are trained separately. For training the routing rule, the actions directly correspond to the machines, while for the sequencing rule, manual rules are used as actions, which can address the challenge posed by the varying number of jobs in different decision-making scenarios. This method would not reduce the search space as much as in the above studies. In such cases, DRL is utilised to enhance the performance of traditional methods but cannot fully address their inherent limitations. For instance, if the manually designed scheduling heuristics lack high quality, their effectiveness as actions can be constrained, and the incorporation of DRL might not yield substantial improvements.

Following the work [32], to overcome its limitation that needs a lot of domain knowledge of experts to design scheduling heuristics, a possible way is to use heuristic generation methods to automatically learn some high-quality and diverse scheduling heuristics as actions. Therefore, it is reasonable and necessary to explore the combination of heuristic generation methods and DRL in the DFJSS problem.

2) Genetic Programming for JSS: GP hyper-heuristic is a powerful heuristic generation method that belongs to the broader family of evolutionary computation domain [33], [34]. GP draws inspiration from biological evolution to automatically evolve populations of heuristics to solve complex problems. Through a process of fitness evaluation, selection, and evolution, GP iteratively refines the population of heuristics over multiple generations to improve their performance on a given problem [35]. In DFJSS, GP has been used to automatically evolve scheduling heuristics that can adapt to the dynamic and flexible nature of the problem. Several studies have explored the effectiveness of GP in evolving heuristics for DFJSS [10], [36], [37]. GP allows the evolution of scheduling heuristics in the form of computer programs, typically represented as trees, where nodes signify functions and terminals that are abstracted from the scheduling system [38], [39]. Its ability to discover complex and non-linear relationships makes it a valuable tool in handling the complexities of the DFJSS problem where the structure of the solution is not known in advance [40]. The key advantages of using GP for DFJSS include its ability to explore a vast search space of scheduling heuristics, automatically adapt to changing conditions, and evolve solutions that may be difficult to manually design [41].

Researchers continue to refine GP-based approaches, exploring hybrid methods that combine GP with other optimisation or machine learning techniques to enhance solution quality and computational efficiency. Noteworthy examples include the integration of GP with surrogate models [38], multi-task optimisation [42], feature selection [43], and ensemble [3], [44] techniques for addressing the DFJSS problem. Moreover, GP has the potential to learn a diverse set of scheduling heuristics by considering phenotypic diversity [45] during the evolutionary process [46].

3) Summary: Based on the above description, it is evident that GP and RL share similarities and differences. As for the similarities, both GP and RL are hyper-heuristic methods, focusing on exploring the heuristic space and learning scheduling heuristics/agents through iterative interactions with the scheduling environment. On the differences, RL typically serves as a heuristic selection method. It selects from a pool of manually designed low-level scheduling heuristics (actions) at decision points. RL primarily concentrates on refining and optimising a single scheduling heuristic, drawing immediate feedback from the scheduling environment following each action, thereby emphasising local information. In contrast, GP prioritises global information, which is acquired upon completing an entire instance rather than after processing each single decision point. Instead of focusing on optimising a single scheduling heuristic, GP maintains a population of such heuristics and collectively optimises them through an evolutionary process involving selection, crossover, mutation, and reproduction. As a result, GP explores a broader and more diverse search space, thus reducing the possibility of falling into local optima. Moreover, GP is a commonly used heuristic generation method in the DFJSS domain, and can generate/construct new and sophisticated heuristics. Given the distinct advantages of GP and RL, investigating their integration to tackle the DFJSS problem presents a novel and intriguing approach, previously less explored within this domain [47], [48]. In this paper, we explore this integration by harnessing the capabilities of GP to automatically learn a diverse set of effective scheduling heuristics. These learned heuristics are then employed as actions within an RL framework to solve the DFJSS problem.

III. The Niching GP-assisted DRL Method

A. Overall Framework

In this paper, a two-stage framework is proposed, leveraging the strengths of GP to help RL develop effective scheduling agents for solving the DFJSS problems. The framework is denoted as Niching GP-assisted DRL (NichGPDRL). The overall framework is shown in Fig. 2. In the first stage, a Niching GP (NichGP) method is presented to autonomously learn a diverse set of high-quality scheduling heuristics. Subsequently, in the second stage, the learned sequencing rules by the NichGP are utilised as actions for the DRL method to adaptively select the most appropriate learned heuristic at different decision points. The proposed NichGPDRL method leverages the strengths of evolutionary capabilities and diversity maintenance capabilities of the NichGP to learn a set of high-quality and diverse scheduling heuristics simultaneously and automatically. By employing these learned heuristics as actions for DRL, the proposed NichGPDRL overcomes the limitations of the baseline DRL method, which typically requires significant time and domain knowledge to manually design scheduling heuristics as actions. Additionally, these manually designed scheduling heuristics might not exhibit high-quality. The subsequent sections provide detailed insights into the state features utilised by the proposed NichGPDRL, how the proposed NichGP method learns a diverse set of high-quality scheduling heuristics simultaneously, and how the DRL method leverages these learned scheduling heuristics to further enhance its effectiveness on DFJSS.

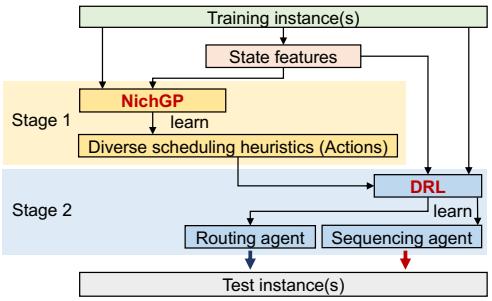


Fig. 2. The overall framework of the proposed NichGPDRL method.

B. State Features

The state features used for both the NichGP and DRL are related to the following characterisations. For more details, please refer to [32].

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 of an operation arrives 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, $TIS_j(t) = t - t_j^{arr}$.

C. Stage 1: Niching GP training

Fig. 3 gives the flowchart of NichGP. Different from the traditional GP method, NichGP uses a Niching strategy [20] after fitness evaluation to remove duplicated and poor individuals from the population. In this case, NichGP aims to achieve the coexistence of multiple high-quality and diverse scheduling heuristics in the population. In the DFJSS domain, phenotypic diversity is more meaningful than genotypic diversity [1]. In this paper, phenotypic diversity is considered to manage a niche. We adopt the phenotypic characterisation (PC) [20], [49] to measure the phenotypic diversity. PC is defined as a vector of values. Each value within the vector signifies the rank by the reference rule assigned to the candidate machine/operation where the calculated rule designates the highest priority at a given decision point. This paper uses the sequencing rule and the routing rule from the best individual evolved at each generation as the reference sequencing rule and reference routing rule, respectively. To process a DFJSS instance, typically comprising thousands of decision points, we adopt a computational efficiency strategy by focusing on 20 sequencing decision points and an equivalent number of routing decision points. Therefore, a PC of a scheduling heuristic contains 40 values. To be specific, an unseen instance is used to get all the decisions which involves 7 candidate machines/operations. Then we shuffle the decisions and randomly get 20 sequencing decision points and 20 routing decision points. Table I provides an illustrative example of calculating the PC of an individual (scheduling heuristic), considering three points for each type of decision and three candidate operations/machines at each decision point. In Table I, characters hold different meanings. For instance, $1(O_1)$ signifies the candidate operation O_1 at the first sequencing decision point, while $2(M_1)$ denotes the candidate machine M_1 at the second routing decision point. The encircled numbers $\textcircled{i}, i \in [1, 2, 3]$ indicate that the candidate operation/machine in the same row holds the first or third rank according to the reference sequencing/routing rule, while it holds the first rank according to the sequencing/routing rule. For example, at the first sequencing decision point, with three candidate operations $[O_1, O_2, O_3]$ under consideration, the reference rule assigns rankings as $[1, 3, 2]$. Conversely, the sequencing rule assigns rankings as $[3, 1, 2]$, placing operation O_2 in the first position, whereas the reference rule assigns it the third rank. Therefore, the first value of the PC for this example is 3. Ultimately, the PC for this instance involves a combination of sequencing and routing decisions, denoted as $[3, 1, 2, 2, 1, 3]$.

The pseudo-code of NichGP is detailed in Algorithm 1. NichGP computes the PC, denoted as pc_i, for each individual ind_i within the population at every generation (see line 5). Subsequently, a clearing strategy [19] is applied, penalising individuals within a niche that show poor performance in fitness by assigning them an infinite fitness value (see line 6) [36]. The fitness function utilised in NichGP to compute the fitness of individuals is the average objective value of the schedules generated by the GP individual across the training instance(s). In this paper, it is represented as $f(x) = \frac{1}{c}\sum_{i=1}^{c} Ttotal(x)$, where c represents the number of training instance(s). The clearing strategy takes into account two crucial parameters: the *radius* δ of each niche, signifying the PC distance between niches, and the *capacity* κ of each niche, representing the number of high-quality individuals in fitness to be retained in

TABLE I
AN EXAMPLE OF CALCULATING THE PC OF AN INDIVIDUAL.

Sequencing				Routing			
Decision points	Reference rule	Sequencing rule	Decision	Decision points	Reference rule	Routing rule	Decision
$1(O_1)$	1	3		$1(M_1)$	②	1	
$1(O_2)$	③	1	3	$1(M_2)$	1	2	2
$1(O_3)$	2	2		$1(M_3)$	3	3	
$2(O_1)$	3	2		$2(M_1)$	2	2	
$2(O_2)$	①	1	1	$2(M_2)$	3	3	1
$2(O_3)$	2	3		$2(M_3)$	①	1	
$3(O_1)$	1	2		$3(M_1)$	1	2	
$3(O_2)$	②	1	2	$3(M_2)$	③	1	3
$3(O_3)$	3	3		$3(M_3)$	2	3	

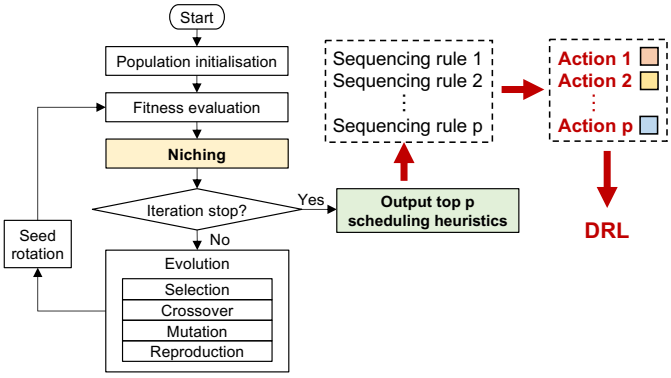


Fig. 3. The flowchart of the GP method of stage 1.

Algorithm 1: The pseudo-code of NichGP.

Input: Population size: N; Generations: G.
Output: A list of top p individuals: $\Delta = [ind_1, \ldots, ind_p]$.
Initialise population pop with N individuals;
$g \leftarrow 0$;
while $g < G$ **do**
  Fitness evaluation for each individual ind_i in pop;
  `// Calculate PC of each individual`
  $pop \leftarrow calculatePC(pop)$;
  `// Use clearing strategy to penalize individuals within a niche that show poor performance`
  $Clearing(pop)$;
  Selection;
  Crossover/Mutation/Reproduction;
  $g \leftarrow g + 1$;
end
$\Delta \leftarrow \emptyset$;
$i \leftarrow 0$;
while $i < p$ **do**
  $\Delta \leftarrow \Delta \cup ind_i$;
end
return Δ;

the niche. In this paper, we use Euclidean distance to calculate the distance between PCs. Notably, in contrast to traditional GP, NichGP employs the clearing strategy each time when the fitness evaluation process is conducted. Moreover, top p individuals are output as the actions for DRL. It is important to highlight that, while DRL only requires sequencing rules since it can learn an end-to-end routing agent for the DFJSS problem considered [32], our method involves evolving both routing and sequencing rules simultaneously using NichGP. The reason for this lies in our findings: when we experimented by keeping the routing rule fixed as a manual rule and evolving only the sequencing rule, we observed that the performance of the sequencing rules is constrained by the fixed routing rule, which will affect the effectiveness of the proposed method.

D. Stage 2: DRL training

We adopt the DRL framework presented in [32]. However, our method differs in that we replace the actions used to train the sequencing agent with those learned through the proposed NichGP. The specifics of the DRL framework are presented as follows. Markov decision process (MDP) [50] is used to model the process that the DRL for solving the DFJSS problem. The MDP is a fundamental framework in RL that mathematically formalises decision-making in an environment where an agent interacts to achieve a goal. The MDP has a 5-element tuple representation: $< S, A, P, \gamma, R >$. S represents the state space, which is the set of all possible situations or configurations that the environment can be in. A denotes the action space, which is the set of all possible actions that the agent can take. P is the transition probabilities, the probabilities associated with transitioning from one state (s_t) to another (s_{t+1}) after taking a specific action (a_t). It characterises the dynamics of the environment. R means the reward function, that specifies the immediate reward (r_t) the agent receives after transitioning. It is related to the goal or objective of the agent. γ represents the discount factor which is a parameter between 0 and 1 that discounts future rewards. It helps in balancing the trade-off between immediate and future rewards.

Fig. 4 illustrates the flowchart of the DRL method and showcases an example of the representation of the routing agent and sequencing agent. In this example, both the routing agent and the sequencing agent consist of one input layer, one output layer, and two hidden layers, each consisting of three nodes represented in circular shapes. The outputs of the routing agent directly correspond to the candidate machines, while the outputs of the sequencing agent represent the sequencing rules learned by the proposed NichGP represented in square shapes. The routing agent and sequencing agent serve as priority functions, guiding routing and sequencing decisions in the scheduling process. Specifically, the routing agent is employed to select a machine when an operation becomes ready for processing, while the sequencing agent is used to determine which operation will be processed next when a machine becomes available. This approach enables the assign-

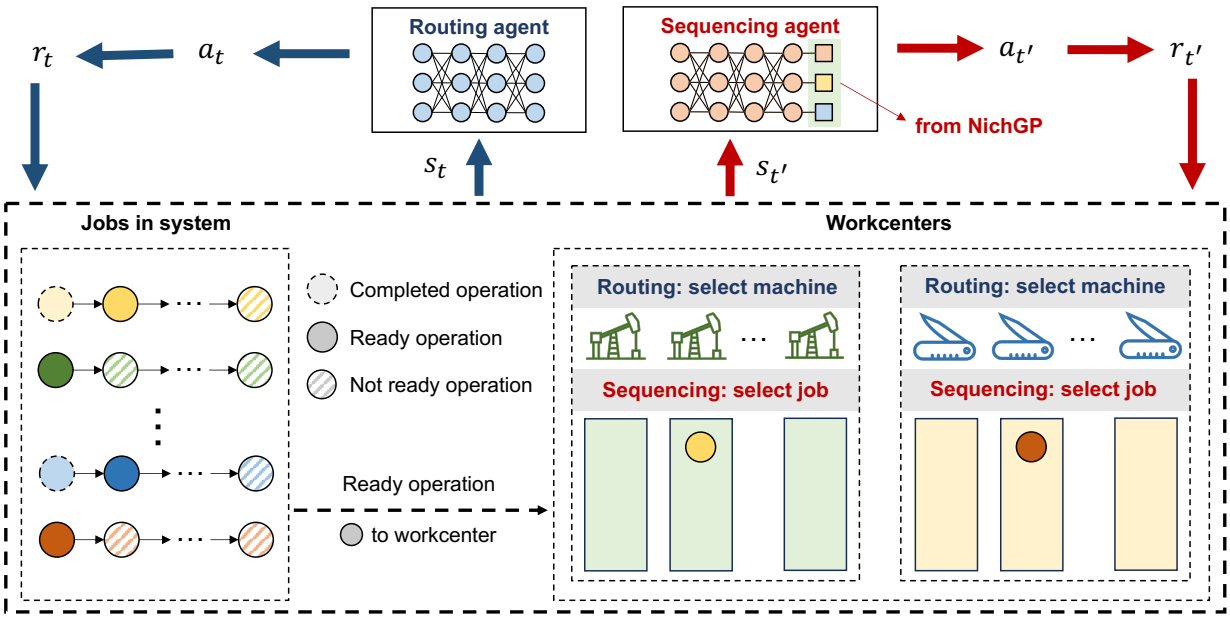


Fig. 4. The flowchart of the RL method of stage 2.

ment of operations to machines and the determination of their processing start times, ultimately leading to the generation of a scheduling solution. To mitigate the exponential growth in the state-action space during multi-agent DRL training [51], the sequencing agent and the routing agent are trained separately. This separation is employed to manage the coordination of exploration efforts, striking a balance between local and global exploration trade-offs [51]. Additionally, during the training of the sequencing agent, the routing rule is kept fixed as the earliest available machine, and the sequencing rules derived from the learned scheduling heuristics are utilised as actions. When training the routing agent, the sequencing rule is set as the First-in-First-out operation, and the routing agent takes end-to-end training, enabling it to make direct decisions among candidate machines. During the training process, the routing/sequencing agent observes the current state (s_t) of the environment and then selects an action (a_t) based on its policy (π). The environment transitions to a new state (s_{t+1}) according to the transition probabilities (P). The agent receives a reward (r_t) from the reward function. The agent updates its knowledge (Q function and policy) based on the observed state, action, reward, and the resulting state. The process repeats over multiple time steps as the agent learns to maximise cumulative rewards.

In this paper, a double DQN architecture is used to train the agent. Double DQN is an enhancement of the traditional DQN, designed to address a common issue known as overestimation bias in Q-learning [52]. It is developed to improve the stability and performance of DQN algorithms. Double DQN uses two separate neural networks: one for action selection (referred to as the action network) and another for Q-value evaluation (referred to as the target network). The action-selection network is responsible for determining the best action, while the target network is used to estimate the Q-value of that action.

The ultimate goal of the agent is to find an optimal policy that maximises the expected cumulative reward over time in the given environment. This is typically achieved through the DRL method that iteratively improves the agent's policy based on its experiences. These experiences play a central role in the learning process. By exploring different actions in various states, the agent can gather information about the environment, which is essential for making informed decisions. The proposed DRL uses experience replay, where past experiences are stored in memory and randomly replayed during learning. This helps the agent learn from a diverse set of experiences and mitigates the impact of correlated sequential data. About the action representation, the routing agent's actions directly correspond to the selected machine within the workcenter, given the fixed number of machines in this study. The sequencing agent's actions entail the selection of a NichGP-learned sequencing rule. This approach addresses the challenge posed by a changing queue, making the direct selection of operations impractical. More details about the reward function and other process can be find in [32].

IV. Experimental Design

A. Dataset

This paper explores a dynamic production system with continuous job arrivals. To assess the performance of the proposed method and comparison methods, we examine four scenarios based on the following three critical factors:

1) **Job Arrival Rate / System Utilisation level**: the job arrival rate is closely linked to the system's utilisation level, denoted as $E(u)$. This metric 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 represent the number of machines and workcenters, respectively. $\mathbb{E}(t)$ stands for the expected

processing time of all operations across all machines, and $\mathbb{E}(in)$ represents the expected time interval between job arrivals. In this study, we assume a 90% utilisation level ($E(u)$) to simulate a busy production environment [32]. Additionally, we assume that the time interval (X) between successive job arrivals follows an exponential distribution [53]: $X \sim Exp(\mathbb{E}(in))$.

2) **Heterogeneity of Processing Time**: the processing time ($t^{pro}_{j,i,m}$) for each operation ($O_{j,i}$) on machine Ω_m is randomly sampled from a uniform distribution ($U[L_p, H_p]$), with L_p and H_p denoting the lower and upper limits of the processing time, respectively. For different scenarios, we consider different average processing times:
   - High heterogeneity: $t^{pro}_{j,i,m} \sim U[5, 25]$
   - Low heterogeneity: $t^{pro}_{j,i,m} \sim U[10, 20]$
3) **Due Date Tightness**: the due date (t^{due}_j) for each job (Jj) is assigned based on its expected total processing time and the due date factor (α_j). This paper considers two types of due date tightness ranges ($U[L_d, H_d]$):
   - High tension: $\alpha_j \sim U[1, 2]$
   - Low tension: $\alpha_j \sim U[1, 3]$

   The due date is calculated as follows:

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

Based on the above descriptions, the four scenarios used to evaluate the performance of the comparison methods are consistent with those presented in [32], as follows:

- **HH**: the processing time exhibits high heterogeneity ($[5, 25]$), and high tension in due dates ($[1, 2]$);
- **HL**: the processing time exhibits high heterogeneity ($[5, 25]$), and low tension in due dates ($[1, 3]$);
- **LH**: the processing time has low heterogeneity ($[10, 20]$), and high tension in due dates ($[1, 2]$);
- **LL**: the processing time has low heterogeneity ($[10, 20]$), and low tension in due dates ($[1, 3]$).

Furthermore, the shop floor is equipped with 3 workcenters, each of which is equipped with 2 machines. Each instance of the simulation covers a production period of 1000 time units, during which approximately 124 jobs arrive on the shop floor. The dataset settings utilised in our paper are consistent with those outlined in [32].

B. Parameter Setting

The features described in Section III-B are used by both the NichGP and DRL. NichGP employs a function set $\{+, -, *, /, \max, \min\}$, where / is protected, returning 1 in case of division by 0. Additional parameters for NichGP are presented in Table II. The NichGP method is implemented in Python using the DEAP package [54]. With the advantage of parallel evaluation, the multiprocessing package [55] in Python is employed to speed up NichGP's training process. The DRL implementation, utilising PyTorch [56] in Python, is detailed in Table III [32], encompassing parameters and network structures.

TABLE II
THE PARAMETER SETTINGS OF THE PROPOSED NICHGP METHOD.

Parameter	Value
Population size	200
Number of generations	50
Number of instances per generation	2
Method for initialising population	Ramped-half-and-half
Initial minimum/maximum depth	2 / 6
Elitism	10
Maximal depth	8
Crossover rate	0.80
Mutation rate	0.15
Reproduction rate	0.05
Terminal/non-terminal selection rate	10% / 90%
Radius δ/capacity κ	$[0, 1, 2, 3, 4, 5]$/1
Parent selection	Size-4 tournament selection
Output as actions for DRL	Top 4 individuals

TABLE III
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

C. Comparison Design

To verify the effectiveness of our proposed NichGPDRL method, we compare our method with the following methods. This paper incorporates four manually designed routing rules and four manually designed sequencing rules which are widely used in industry [15], [25]. These are combined to create 16 scheduling heuristics, forming the basis for comparison with the learned scheduling heuristics from NichGP and DRL methods. This comparative analysis with manually designed rules offers an intuitive perspective on the effectiveness of the proposed NichGPDRL method, highlighting their strong generalisation capabilities. The manually designed routing and sequencing rules taken into comparison 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 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.

In addition, the GP-assisted DRL (GPDRL) is used as a baseline to verify the effectiveness of the proposed NichGP method. Moreover, the method that NichGPDRL without DRL training is employed (NichGP#). For NichGP#, the sequencing rule remains fixed as one of the candidate rules (actions), serving to validate the efficacy of the DRL learning process. For each method, 30 independent runs are conducted to train 30 scheduling heuristics. In both NichGP (stage 1) and DRL (stage 2), the training process involves 100 instances. In the case of NichGP, two instances are applied per generation, and the top 4 scheduling heuristics, learned over 50 generations (totaling 100 instances), are returned. For DRL, the learned sequencing and routing agents are obtained after completing the 100 training instances. After training, we evaluate the objective values generated by the learned scheduling agents in 30 independent runs across 100 unseen instances for each of the four scenarios.

1) 16 widely used manual scheduling heuristics;
2) DRL [32]: the routing agent is trained to directly select from machines/operations while the sequencing agent is trained to select from manual sequencing rules;
3) GPDRL: the routing agent is trained to directly select from machines/operations while the sequencing agent is trained to select from the learned sequencing rules by the original GP;
4) NichGP# ($\# \in [1, 2, 3, 4]$): the routing agent is trained through DRL [32] using an end-to-end way, directly selecting from machines/operations. Meanwhile, the sequencing agent remains fixed as one of the rules learned by the proposed NichGP;
5) NichGPDRL (ours): the routing agent is trained to directly select from machines/operations while the sequencing agent is trained to select from the learned sequencing rules by the proposed NichGP.

By comparing with these methods, we can validate the effectiveness of the proposed NichGPDRL method, assess the performance of learned sequencing rules by GP methods in comparison to manually designed ones, and evaluate the efficacy of the proposed niching strategy in learning diverse and high-quality sequencing rules, as opposed to the traditional GP method.

V. Results and Discussions

A. Influence of the Radius Parameter in NichGP

The parameter radius, denoted as δ, plays a critical role in NichGP, representing the degree of dissimilarity between individuals within the NichGP. This dissimilarity level, in turn, can have an impact on the performance of the learned agent through RL. To explore this effect, we conduct experiments using six δ values: 0, 1, 2, 3, 4, and 5. Specifically, when $\delta = 0$, it signifies that a distance larger than 0 between scheduling heuristics is considered acceptable, while other scheduling heuristics are penalised. The respective training performance of the proposed NichGPDRL method for each δ value is shown in Table IV. The results of the Friedman test [57], revealing a p-value of 0.002 (falling below the significance threshold of 0.05), indicate significant differences among different δ values. Considering the average rank of these methods across all four scenarios, a radius of 0 attains the highest rank with a value of 1.25. Additionally, we observe that as the radius value increases, the rank decreases. This phenomenon suggests that increasing the radius does not lead to performance improvement but rather degrades it. A radius of 0 yields the best performance. Consequently, for subsequent experiments and analyses, we use NichGPDRL to represent the proposed method with a radius of 0.

TABLE IV
The mean and standard deviation training performance of 30 independent runs of the proposed NichGPDRL method with **different δ values**.

δ	HH	HL	LH	LL	Rank
0	537.40(333.24)	173.40(194.79)	1492.33(900.92)	636.03(627.94)	1.25
1	551.82(353.12)	172.60(184.53)	1519.75(885.75)	651.72(648.31)	1.75
2	561.35(344.87)	187.45(194.74)	1540.70(859.74)	664.25(656.11)	3
3	588.73(386.7)	194.70(202.84)	1594.92(937.7)	689.90(676.96)	4
4	616.37(367.32)	205.52(183.95)	1599.77(903.5)	695.33(693.09)	5
5	648.45(384.99)	225.05(257.65)	1667.58(933.76)	730.05(713.3)	6

B. Test Performance

Table V gives the mean and standard deviation test performance of 30 independent runs of the proposed NichGPDRL methods and comparison methods. The Friedman test results, yielding a p-value of 4.28×10^{-8} (smaller than 0.05), indicate significant differences among the methods. Notably, all scheduling heuristics learned by hyper-heuristic methods (DRL, NichGP#, GPDRL, and the proposed NichGPDRL) outperform all the manually designed ones. Among the hyper-heuristic methods, DRL exhibits the worst performance. The proposed NichGPDRL obtains the best performance, with NichGP1 following closely as the second best. NichGP3 secures the third position, while NichGP2 takes the fourth position. NichGP4 and GPDRL closely follow in the fifth position.

To be more specific, we employ the Wilcoxon rank sum test [58] to compare the proposed NichGPDRL method with each comparison method across the four scenarios. The results, denoted by significantly better ($\uparrow$), worse ($\downarrow$), or statistically similar (=) compared to other methods, are presented alongside the results of the comparison method in Table V. Upon analysing these results, we observe that the proposed NichGPDRL significantly outperforms DRL and all manually designed scheduling heuristics across all four scenarios. When compared with NichGP#, the proposed NichGPDRL performs better than NichGP1 and NichGP4 on the scenario HL. When compared with GPDRL, NichGPDRL performs significantly better than GPDRL on two scenarios (LH and LL) while performing similarly with GPDRL on the other scenarios (HH and HL). Importantly, it never performs worse than the other methods in any other scenarios. These results validate

TABLE V
THE MEAN AND STANDARD DEVIATION **TEST PERFORMANCE** OF 30 INDEPENDENT RUNS OF THE PROPOSED NICHGPDRL METHODS AND COMPARISON METHODS.

Algorithm	HH	HL	LH	LL	Rank
ECT+SPT	1014.29(0.00)(↑)	611.06(0.00)(↑)	2457.59(0.00)(↑)	1699.70(0.00)(↑)	9
ECT+EDD	1082.88(0.00)(↑)	441.37(0.00)(↑)	2424.66(0.00)(↑)	1265.08(0.00)(↑)	8
ECT+FIFO	1245.73(0.00)(↑)	705.08(0.00)(↑)	2748.60(0.00)(↑)	1727.35(0.00)(↑)	10.5
ECT+LWR	1305.73(0.00)(↑)	791.42(0.00)(↑)	2505.60(0.00)(↑)	1607.48(0.00)(↑)	10.5
MET+SPT	1437.83(0.00)(↑)	960.87(0.00)(↑)	4534.33(0.00)(↑)	3509.15(0.00)(↑)	16.5
MET+EDD	1719.43(0.00)(↑)	782.12(0.00)(↑)	4755.69(0.00)(↑)	3087.73(0.00)(↑)	16.25
MET+FIFO	1965.50(0.00)(↑)	1223.73(0.00)(↑)	5274.61(0.00)(↑)	3800.25(0.00)(↑)	18.5
MET+LWR	2028.35(0.00)(↑)	1345.42(0.00)(↑)	4966.10(0.00)(↑)	3693.70(0.00)(↑)	18.5
EA+SPT	3261.69(0.00)(↑)	2297.7(0.00)(↑)	3409.09(0.00)(↑)	2430.36(0.00)(↑)	15.25
EA+EDD	3824.05(0.00)(↑)	2235.12(0.00)(↑)	3403.57(0.00)(↑)	1901.17(0.00)(↑)	14.5
EA+FIFO	4254.86(0.00)(↑)	2871.10(0.00)(↑)	3930.47(0.00)(↑)	2617.09(0.00)(↑)	19.75
EA+LWR	3937.37(0.00)(↑)	2740.29(0.00)(↑)	3491.85(0.00)(↑)	2352.15(0.00)(↑)	16.75
LWIQ+SPT	3496.10(0.00)(↑)	2461.64(0.00)(↑)	3601.45(0.00)(↑)	2568.60(0.00)(↑)	17.25
LWIQ+EDD	4003.76(0.00)(↑)	2305.48(0.00)(↑)	3524.99(0.00)(↑)	2045.50(0.00)(↑)	16.5
LWIQ+FIFO	4468.28(0.00)(↑)	3022.68(0.00)(↑)	4064.07(0.00)(↑)	2720.61(0.00)(↑)	21
LWIQ+LWR	4219.96(0.00)(↑)	2945.80(0.00)(↑)	3688.77(0.00)(↑)	2505.47(0.00)(↑)	19
DRL	1003.30(43.95)(↑)	463.28(43.48)(↑)	2383.47(70.08)(↑)	1219.51(70.33)(↑)	7.25
NichGP1	929.08(54.08)(=)	385.49(34.95)(↑)	2255.22(83.52)(=)	1125.61(77.46)(=)	3
NichGP2	943.44(77.26)(=)	388.38(46.60)(=)	2285.84(95.21)(=)	1115.44(47.15)(=)	4
NichGP3	932.07(63.95)(=)	381.97(43.54)(=)	2273.90(91.91)(=)	1122.20(79.08)(=)	3.75
NichGP4	944.85(63.73)(=)	390.78(43.78)(↑)	2269.25(100.79)(=)	1119.19(53.11)(=)	4.25
GPDRL	929.84(43.46)(=)	375.08(32.04)(=)	2299.29(88.82)(↑)	1211.13(64.51)(↑)	4.25
NichGPDRL	927.50(42.44)	372.18(19.13)	2263.42(70.21)	1121.14(75.55)	1.75

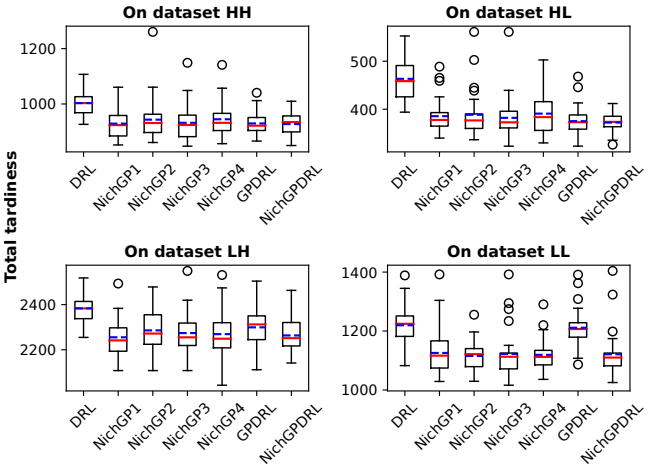


Fig. 5. The box plots of the test performance of 30 independent runs of the proposed NichGPDRL, GPDRL, and DRL methods.

the effectiveness of the proposed NichGPDRL method and highlight the effectiveness of the proposed niching strategy in aiding GP to generate high-quality and diverse actions for DRL compared to the original GP.

Fig. 5 visualises the box plots of the test performance of 30 independent runs of the proposed NichGPDRL and other hyper-heuristic methods. The box plots highlight the significant advantages of the proposed algorithm compared to the DRL algorithm. It illustrates the benefits of combining GP and DRL over using DRL independently. The demonstrated advantages emphasise the effectiveness of the proposed NichGPDRL of integrating GP into DRL. Furthermore, the effectiveness of the proposed NichGP# is verified through these results.

VI. FURTHER ANALYSES

A. Action Contribution

In the learning process of RL, the selection process involves choosing from four scheduling heuristics (actions) at each decision point. This section focuses on the analysis of *action contribution*, defined as the percentage of times each action is utilised relative to the total number of decision points. The mean of action contributions is calculated across 30 independent runs for the DRL, GPDRL, and the proposed NichGPDRL methods across four scenarios. The results are presented in Fig. 6 using pie plots.

The pie plots reveal that, in the case of the DRL method, action 1 (A1, SPT) assumes a key role across all four scenarios, securing the highest percentage in each scenario, exceeding 50% in three of them. Subsequently, action 3 (A3, SLACK) secures the second-highest percentage. Action 4 (A4, CR) attains the third rank, while action 2 (A2, WIQ) ranks the lowest, constituting about only 1% on all four scenarios. In contrast to the DRL method, GPDRL, and NichGPDRL methods exhibit similar action contributions across the four actions. This shows that the scheduling heuristics evolved by GP and NichGP can generate key rules that contribute to the overall performance.

B. Behaviour Difference between Scheduling Heuristics

This paper proposes a NichGP method to iteratively evolve a set of diverse scheduling heuristics for use as actions in RL. This section investigates the *behaviour difference* among scheduling heuristics evolved through the proposed NichGP. To ensure a fair comparison, we employ an unseen instance to generate 100 sequencing decisions. Subsequently, each scheduling heuristic evolved by NichGP and GP is evaluated on these 100 decisions, with the Hamming distance utilised to calculate the behavioral difference between each pair of

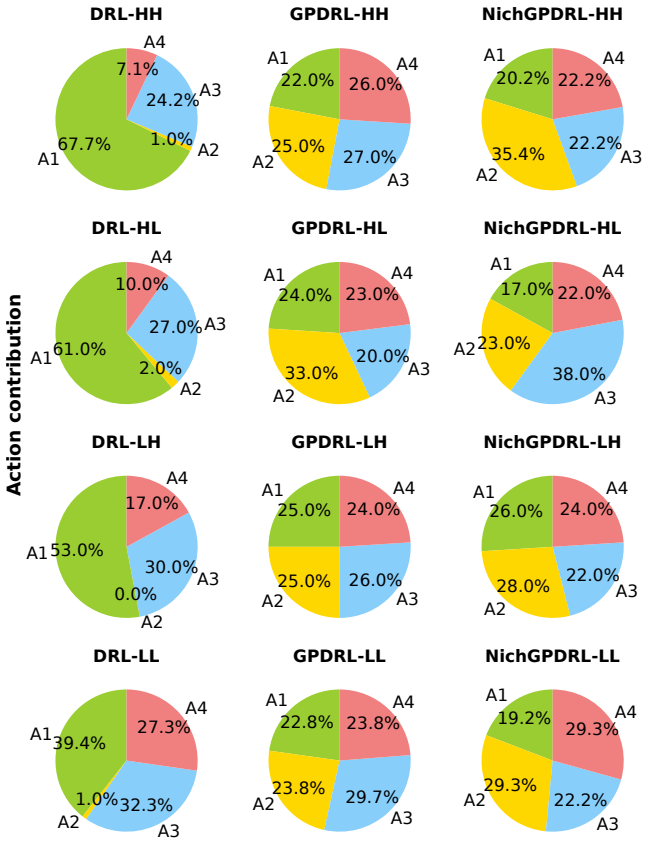


Fig. 6. The pie plots of the **action contribution** of 30 independent runs of the DRL, GPDRL, and the proposed NichGPDRL methods on four scenarios.

TABLE VI
THE MEAN AND STANDARD DEVIATION **BEHAVIOUR DIFFERENCE** OF 30 INDEPENDENT RUNS OF THE PROPOSED NICHGPDRL AND GPDRL METHODS.

Scenario	GPDRL	NichGPDRL
HH	0.051(0.097)	0.161(0.066)(↑)
HL	0.054(0.058)	0.126(0.089)(↑)
LH	0.045(0.080)	0.205(0.074)(↑)
LL	0.045(0.058)	0.119(0.044)(↑)

sequencing rules (a total of 6 pairs for 4 sequencing rules). The average percentage of behavioral difference across all pairs is then computed as the behaviour difference ρ. The function to calculate the behaviour difference is as Eq. (10).

$$\rho = \frac{\sum_{i=1}^{6} Hamming(seq_a, seq_b)}{6} \tag{10}$$

where, seq_a and seq_b represent different sequencing rules. Table VI gives the mean and standard deviation behaviour difference of 30 independent runs of the proposed NichGP-DRL and GPDRL methods. It can be seen that the proposed NichGPDRL shows significantly higher behaviour difference than the GPDRL on all the four scenarios. The NichGPDRL outperforms the GPDRL about more than 3 times on behaviour difference. This verifies the effectiveness of the proposed NichGP in learning diverse scheduling heuristics for DRL.

C. Generalisation to More Complex Scenarios

On one hand, an algorithm's generalisation ability can be assessed by examining the performance of its trained heuristics on unseen test instances at similar scales. In this context, as demonstrated in the comparison of test performances in Table V, it is evident that the proposed NichGPDRL exhibits superior generalisation ability compared to DRL. On the other hand, generalisation ability can be further validated by extending the application of trained heuristics to unseen test instances at more complex scales. To achieve this, scheduling heuristics learned by the proposed NichGPDRL and comparison methods are tested on instances featuring a higher number of jobs and a large number of workcenters. Specifically, the test instances involve 248 jobs arriving over 2000 units of time and 620 jobs arriving over 5000 units of time. Similar to Table V, 100 unseen instances are employed for testing. Furthermore, the test performance of 16 manually designed scheduling heuristics is presented as a baseline for comparison, to verify the effectiveness of the proposed method.

Tables VII and VIII present the mean and standard deviation of test performances for the proposed NichGPDRL and comparison methods across two sets of scenarios: one characterised by a substantial number of job arrivals and the other featuring different numbers of workcenters. The "↑" next to NichGPDRL in Tables VII and VIII signifies that the results of NichGPDRL are significantly better than those of all the compared algorithms. These findings demonstrate that NichGPDRL outperforms DRL and all manually designed scheduling heuristics, even when directly applied to more complex instances without retraining. While DRL outperforms most manually designed scheduling heuristics in 16 scenarios, exceptions include ECT+EDD on scenario HL with 248 jobs, ECT+SPT on scenario HH with 620 jobs, ECT+EDD on scenario LL with six workcenters, each with two machines, and ECT+EDD on scenarios HL and LH with nine workcenters, each with two machines. These findings demonstrate that both NichGPDRL and DRL methods can be applied to more complex scenarios without retraining. Moreover, NichGPDRL exhibits superior generalisation ability compared to DRL as it provides even better performance.

D. Structure Analysis of Sequencing Rules

In this section, we analyse the structure of sequencing rules evolved by NichGP and used as actions for DRL. We randomly select a run and the structures of the 4 sequencing rules are shown in Fig. 7.

For the sequencing rule 1, it comprises four terminals: PT, SLACK, NIQ, and WKR. Notably, SLACK emerges as the predominant terminal within this rule, having been utilised four times. Following SLACK, both PT and WKR are employed twice, while NIQ is employed only once. The simplified representation of the sequencing rule 1 is denoted as S_1 and is expressed in Eq. (11). Given that NIQ, representing the number of operations in the waiting queue for a machine, remains constant for all candidate operations, it can be disregarded. This rule prioritises jobs that have a shorter processing time for the current operation, a smaller slack, and

TABLE VII
THE MEAN AND STANDARD DEVIATION TEST PERFORMANCE OF THE DRL, THE NICHGPDRL, 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	2175.22(0.00)	1353.44(0.00)	5257.93(0.00)	3698.47(0.00)	5670.90(0.00)	3540.29(0.00)	14346.88(0.00)	10342.80(0.00)
ECT+EDD	2441.94(0.00)	1026.63(0.00)	5246.53(0.00)	2806.27(0.00)	6479.19(0.00)	2924.44(0.00)	14474.19(0.00)	8065.42(0.00)
ECT+LWR	2830.33(0.00)	1763.24(0.00)	5510.60(0.00)	3653.18(0.00)	7733.09(0.00)	4849.61(0.00)	15235.86(0.00)	10315.41(0.00)
ECT+FIFO	2730.64(0.00)	1588.24(0.00)	5938.43(0.00)	3806.03(0.00)	7528.22(0.00)	4450.15(0.00)	16426.80(0.00)	10918.11(0.00)
MET+SPT	3342.30(0.00)	2278.09(0.00)	10838.54(0.00)	8710.86(0.00)	8741.83(0.00)	6000.12(0.00)	30705.00(0.00)	25227.22(0.00)
MET+EDD	4088.80(0.00)	2065.01(0.00)	12083.85(0.00)	8248.97(0.00)	10906.19(0.00)	5394.65(0.00)	34920.13(0.00)	25013.67(0.00)
MET+LWR	4830.47(0.00)	3282.75(0.00)	12079.90(0.00)	9342.84(0.00)	12775.60(0.00)	8787.98(0.00)	35324.91(0.00)	28118.48(0.00)
MET+FIFO	4726.93(0.00)	3020.21(0.00)	12819.83(0.00)	9601.56(0.00)	12569.91(0.00)	8111.35(0.00)	37149.28(0.00)	28687.94(0.00)
EA+SPT	8602.50(0.00)	6419.72(0.00)	7761.20(0.00)	5715.40(0.00)	25364.02(0.00)	19582.27(0.00)	22400.26(0.00)	17089.66(0.00)
EA+EDD	10251.10(0.00)	6596.72(0.00)	8042.18(0.00)	4788.28(0.00)	31222.59(0.00)	22163.05(0.00)	23034.26(0.00)	14708.75(0.00)
EA+LWR	10526.01(0.00)	7767.38(0.00)	8125.91(0.00)	5670.46(0.00)	33417.17(0.00)	25770.86(0.00)	23429.87(0.00)	16962.71(0.00)
EA+FIFO	11028.04(0.00)	7884.14(0.00)	8895.89(0.00)	6108.21(0.00)	33899.75(0.00)	25385.47(0.00)	25530.17(0.00)	18210.69(0.00)
LWIQ+SPT	8955.59(0.00)	6648.55(0.00)	8133.55(0.00)	5960.71(0.00)	26428.54(0.00)	20302.60(0.00)	23554.17(0.00)	17867.04(0.00)
LWIQ+EDD	10569.78(0.00)	6836.88(0.00)	8335.29(0.00)	5063.08(0.00)	31893.24(0.00)	22543.42(0.00)	24278.45(0.00)	15397.05(0.00)
LWIQ+LWR	10975.34(0.00)	8123.35(0.00)	8442.37(0.00)	5864.28(0.00)	33286.78(0.00)	25493.15(0.00)	24435.33(0.00)	17631.24(0.00)
LWIQ+FIFO	11560.02(0.00)	8316.44(0.00)	9231.12(0.00)	6336.31(0.00)	34631.34(0.00)	25898.14(0.00)	26808.94(0.00)	19191.36(0.00)
DRL	2154.31(96.78)	1041.14(86.75)	4996.29(157.03)	2599.83(153.23)	5781.12(253.85)	2834.20(214.81)	14065.40(448.60)	7773.84(431.70)
NichGPDRL(↑)	2027.05(100.62)	882.36(45.47)	4813.97(234.03)	2415.20(147.26)	5386.88(231.38)	2397.40(144.05)	13534.79(609.15)	7333.81(578.69)

TABLE VIII
THE MEAN AND STANDARD DEVIATION TEST PERFORMANCE OF THE DRL, THE NICHGPDRL, AND COMPARISON METHODS ON EIGHT SCENARIOS WITH A LARGE NUMBER OF WORKCENTERS.

Algorithm	6W2M				9W2M			
	HH	HL	LH	LL	HH	HL	LH	LL
ECT+SPT	1771.04(0.00)	988.88(0.00)	4323.62(0.00)	2646.83(0.00)	3171.25(0.00)	1742.07(0.00)	7411.42(0.00)	4459.58(0.00)
ECT+EDD	1780.71(0.00)	522.47(0.00)	3983.73(0.00)	1503.14(0.00)	2918.00(0.00)	712.32(0.00)	6182.24(0.00)	2045.61(0.00)
ECT+LWR	2205.35(0.00)	1229.67(0.00)	4187.14(0.00)	2440.70(0.00)	3603.28(0.00)	1942.15(0.00)	6829.04(0.00)	3862.74(0.00)
ECT+FIFO	2244.21(0.00)	1183.49(0.00)	4860.78(0.00)	2749.25(0.00)	4209.50(0.00)	2199.16(0.00)	8202.33(0.00)	4522.05(0.00)
MET+SPT	2674.45(0.00)	1637.45(0.00)	7225.88(0.00)	5022.53(0.00)	4191.48(0.00)	2422.35(0.00)	11246.05(0.00)	7509.85(0.00)
MET+EDD	2803.78(0.00)	853.85(0.00)	7005.63(0.00)	3453.68(0.00)	4084.71(0.00)	935.13(0.00)	10013.13(0.00)	4300.95(0.00)
MET+LWR	3499.82(0.00)	2085.59(0.00)	7596.67(0.00)	5030.36(0.00)	5312.86(0.00)	3000.78(0.00)	11339.11(0.00)	7204.76(0.00)
MET+FIFO	3668.17(0.00)	2067.90(0.00)	8573.12(0.00)	5477.07(0.00)	5872.18(0.00)	3139.97(0.00)	12896.23(0.00)	7883.16(0.00)
EA+SPT	6080.39(0.00)	3915.84(0.00)	6100.94(0.00)	3922.60(0.00)	9681.76(0.00)	6014.37(0.00)	9606.73(0.00)	5984.38(0.00)
EA+EDD	6625.08(0.00)	3117.37(0.00)	5667.42(0.00)	2458.70(0.00)	9566.29(0.00)	4112.08(0.00)	8376.61(0.00)	3331.55(0.00)
EA+LWR	6573.10(0.00)	4097.93(0.00)	5811.79(0.00)	3534.29(0.00)	10398.22(0.00)	6304.28(0.00)	9149.84(0.00)	5372.83(0.00)
EA+FIFO	7772.72(0.00)	4799.38(0.00)	6756.52(0.00)	4019.94(0.00)	12197.22(0.00)	7209.15(0.00)	10862.16(0.00)	6261.37(0.00)
LWIQ+SPT	6253.02(0.00)	3993.08(0.00)	6205.48(0.00)	3972.28(0.00)	10156.43(0.00)	6274.87(0.00)	10104.80(0.00)	6250.91(0.00)
LWIQ+EDD	6862.33(0.00)	3397.43(0.00)	5896.02(0.00)	2732.58(0.00)	10142.49(0.00)	4272.82(0.00)	8779.17(0.00)	3587.16(0.00)
LWIQ+LWR	7100.61(0.00)	4486.30(0.00)	5999.24(0.00)	3637.71(0.00)	10680.63(0.00)	6479.56(0.00)	9455.17(0.00)	5539.43(0.00)
LWIQ+FIFO	8034.78(0.00)	4975.42(0.00)	6958.58(0.00)	4174.30(0.00)	12520.61(0.00)	7477.10(0.00)	11321.88(0.00)	6575.32(0.00)
DRL	1597.57(97.59)	570.07(103.14)	3841.77(142.62)	1401.16(165.89)	2695.19(199.71)	863.03(223.10)	6249.33(271.75)	1970.43(352.45)
NichGPDRL(↑)	1460.45(72.37)	375.57(23.01)	3709.01(137.02)	1244.96(64.20)	2446.60(149.44)	462.48(34.11)	5976.72(244.07)	1613.10(89.89)

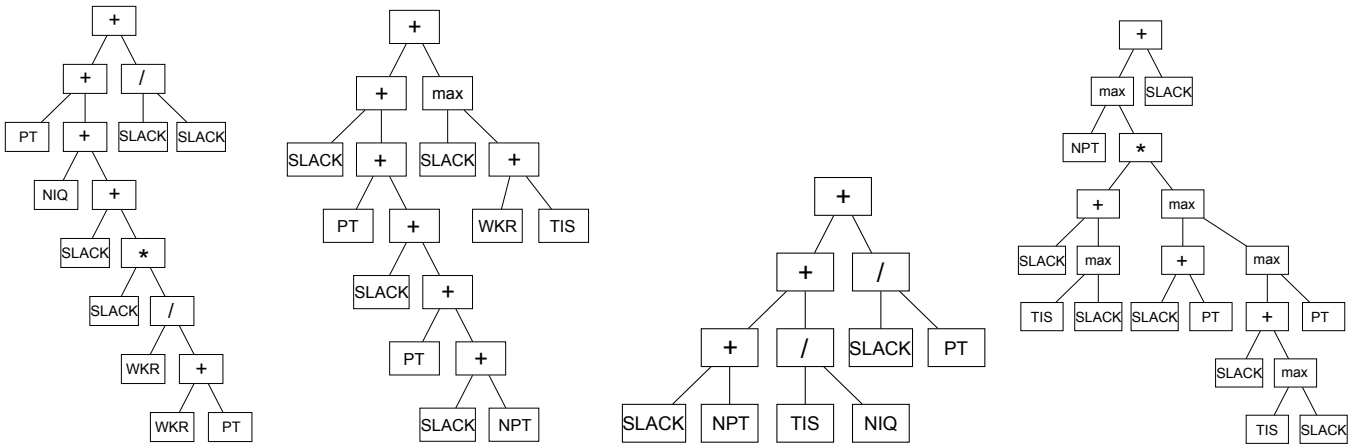
a higher percentage of the current processing time relative to the remaining processing time of the job.

$$S_1 = PT + NIQ + SLACK + \frac{SLACK}{1 + \frac{PT}{WKR}} + 1 \tag{11}$$

Regarding the sequencing rule 2, it comprises five terminals: PT, SLACK, NPT, WKR, and TIS. Notably, SLACK emerges as the most frequently used terminal in this rule, having been employed four times. Subsequently, PT is used twice, while NPT, WKR, and TIS are each used only once. The simplified representation of the sequencing rule 2 is denoted as S_2 and is expressed in Eq. (12). The interpretation of this rule varies depending on specific scenarios. When the SLACK time of a job surpasses the sum of the remaining processing time and the time the job has spent in the system, the rule gives preference to jobs with shorter processing times, shorter slack time, and smaller processing times for their subsequent operations. Conversely, if the SLACK time of a job is less than the sum of its remaining processing time and the time it has stayed in the system, the rule favors jobs with shorter processing times, shorter slack time, smaller processing times for their next operations, shorter remaining processing time, and shorter time spent in the system.

$$\begin{aligned} S_2 =& 2 \times PT + 3 \times SLACK + NPT \\ &+ \max\{SLACK, WKR + TIS\} \end{aligned} \tag{12}$$

Concerning the sequencing rule 3, it comprises five terminals (PT, SLACK, NPT, NIQ, and TIS), with SLACK being the predominant terminal, utilised twice. PT, NPT, NIQ, and TIS are each used only once. The simplified representation of the sequencing rule 3 is denoted as S_3 and is illustrated in Eq. (13). Since NIQ represents the number of operations in the waiting queue of the machine and remains constant for all candidate operations, it can be disregarded. This rule prioritises jobs with shorter slack time, smaller processing time



(a) Sequencing rule 1. (b) Sequencing rule 2. (c) Sequencing rule 3. (d) Sequencing rule 4.

Fig. 7. The tree structures of four sequencing rules evolved by a NichGP run.

for their next operation, shorter time spent in the system, and longer processing time.

$$S_3 = SLACK + NPT + \frac{TIS}{NIQ} + \frac{SLACK}{PT} \tag{13}$$

Regarding the sequencing rule 4, it consists of four terminals (PT, SLACK, NPT, and TIS), with SLACK being the most frequently utilised terminal, employed six times. PT follows with two occurrences, while NPT and TIS are each used only once. The simplified representation of the sequencing rule 4 is denoted as S_4 and is expressed in Eq. (14). This rule has different meanings in distinct situations. When the processing time of the next operation significantly exceeds the slack time, the current processing time, and the duration the job has spent in the system, it favors jobs with shorter processing times for the next operation and shorter slack times. Alternatively, under different circumstances, the rule prioritises jobs with shorter processing times, shorter slack times, and shorter durations spent in the system.

$$\begin{aligned} S_4 = &\max\{NPT, (SLACK + \max\{TIS, SLACK\}) \\ &\times \max\{SLACK + PT, \max\{SLACK, \\ &\max\{TIS, SLACK\}, PT\}\}\} + SLACK \end{aligned} \tag{14}$$

Based on the above analyses of various sequencing rules, it is evident that both slack time and current processing time are crucial features that hold significant importance in their decision-making criteria. Additionally, the processing time of the next operation and the time spent in the system are also important factors in selecting a candidate job for processing as the next task. Despite these similarities, there are notable differences. For instance, the sequencing rules 1 and 2 consider the remaining processing time of the job, while sequencing rules 3 and 4 do not take this into account. Furthermore, the sequencing rule 3 exhibits a preference for jobs with longer processing times, while all other rules favor jobs with shorter processing times. This observation indicates that these rules focus on distinct situations during the long-term scheduling process and exhibit different behaviours. The variations among these rules empower the proposed NichGPDRL algorithm to make informed decisions in selecting an expert rule at specific decision points, thereby facilitating intelligent scheduling.

VII. Conclusions

This paper presents a Niching GP-assisted DRL method for learning sequencing and routing agents to address the DFJSS problem. Specifically, the Niching GP method is employed to learn a diverse set of high-quality scheduling heuristics. Subsequently, the sequencing rules derived from these learned scheduling heuristics serve as actions for the DRL method. This method tackles the challenge of adapting to altered operations at various decision points, enabling the learning of intelligent agents capable of making optimal selections among these actions to generate effective schedules. Comparative results against baseline DRL and widely used manually designed scheduling heuristics validate the effectiveness of the proposed method. Additionally, comparisons against traditional GP-assisted DRL confirm the effectiveness of the proposed Niching GP method. Furthermore, contrasting results against the proposed method without the DRL training process, where the sequencing rule is fixed as one of the candidate actions, verify the efficacy of the DRL learning process. Further analysis of action contribution demonstrates that the scheduling heuristics learned by the Niching GP method contribute similar percentages to the overall performance. The behavioral analysis reveals that the proposed Niching GP method can learn diverse scheduling heuristics compared to traditional GP methods. Structural analysis indicates that the learned scheduling heuristics by the Niching GP exhibit similarities but also show distinct performances at different decision points. In summary, this paper shows that the Niching GP-assisted DRL method contributes to the development of effective agents for solving the DFJSS problem.

Future work could focus on proposing more effective strategies to integrate GP into RL to take full advantage of both for solving the DFJSS problem. Additionally, it would be interesting to explore the adaptation of the proposed method for solving other complex problems.

REFERENCES

[1] M. Xu, Y. Mei, F. Zhang, and M. Zhang, "Genetic programming with lexicase selection for large-scale dynamic flexible job shop scheduling," *IEEE Transactions on Evolutionary Computation*, 2023, DOI:10.1109/TEVC.2023.3244607.

[2] H. Xiong, S. Shi, D. Ren, and J. Hu, "A survey of job shop scheduling problem: The types and models," *Computers & Operations Research*, vol. 142, p. 105731, 2022.

[3] M. Xu, Y. Mei, F. Zhang, and M. Zhang, "Genetic programming for dynamic flexible job shop scheduling: Evolution with single individuals and ensembles," *IEEE Transactions on Evolutionary Computation*, 2023, DOI:10.1109/TEVC.2023.3334626.

[4] ——, "Genetic programming with diverse partner selection for dynamic flexible job shop scheduling," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, jul 2022, pp. 615–618.

[5] W. Ren, Y. Yan, Y. Hu, and Y. Guan, "Joint optimisation for dynamic flexible job-shop scheduling problem with transportation time and resource constraints," *International Journal of Production Research*, vol. 60, no. 18, pp. 5675–5696, 2022.

[6] J. Xie, L. Gao, K. Peng, X. Li, and H. Li, "Review on flexible job shop scheduling," *IET Collaborative Intelligent Manufacturing*, vol. 1, no. 3, pp. 67–77, 2019.

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

[8] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Survey on genetic programming and machine learning techniques for heuristic design in job shop scheduling," *IEEE Transactions on Evolutionary Computation*, vol. 28, no. 1, pp. 147–167, 2023.

[9] F. Zhang, Y. Mei, S. Nguyen, K. C. Tan, and M. Zhang, "Task relatedness based multitask genetic programming for dynamic flexible job shop scheduling," *IEEE Transactions on Evolutionary Computation*, vol. 27, no. 6, pp. 1705–1719, dec 2023.

[10] 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.

[11] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Multitask multiobjective genetic programming for automated scheduling heuristic learning in dynamic flexible job shop scheduling," *IEEE Transactions on Cybernetics*, vol. 53, no. 7, pp. 4473–4486, 2023.

[12] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[13] V. Huang, C. Wang, H. Ma, G. Chen, and K. Christopher, "Cost-aware dynamic multi-workflow scheduling in cloud data center using evolutionary reinforcement learning," in *International Conference on Service-Oriented Computing*. Springer, 2022, pp. 449–464.

[14] 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.

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

[16] J. R. Koza *et al.*, *Genetic programming II*. MIT press Cambridge, 1994, vol. 17.

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

[18] J. C. Tay and N. B. Ho, "Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems," *Computers & Industrial Engineering*, vol. 54, no. 3, pp. 453–473, 2008.

[19] Y. Zakaria, Y. Zakaria, A. BahaaElDin, and M. Hadhoud, "Niching-based feature selection with multi-tree genetic programming for dynamic flexible job shop scheduling," in *Proceedings of the International Joint Conference on Computational Intelligence*. Springer, 2021, pp. 3–27.

[20] Y. Mei, S. Nguyen, B. Xue, and M. Zhang, "An efficient feature selection algorithm for evolving job shop scheduling rules with genetic programming," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 1, no. 5, pp. 339–353, 2017.

[21] J.-P. Huang, L. Gao, and X.-Y. Li, "An end-to-end deep reinforcement learning method based on graph neural network for distributed job-shop scheduling problem," *Expert Systems with Applications*, vol. 238, p. 121756, 2024.

[22] Y. Zhang, H. Zhu, D. Tang, T. Zhou, and Y. Gui, "Dynamic job shop scheduling based on deep reinforcement learning for multi-agent manufacturing systems," *Robotics and Computer-Integrated Manufacturing*, vol. 78, p. 102412, 2022.

[23] 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," *Proceedings of the Advances in Neural Information Processing Systems*, vol. 33, pp. 1621–1632, 2020.

[24] K. Lei, P. Guo, W. Zhao, Y. Wang, L. Qian, X. Meng, and L. Tang, "A multi-action deep reinforcement learning framework for flexible job-shop scheduling problem," *Expert Systems with Applications*, vol. 205, p. 117796, 2022.

[25] K. Lei, P. Guo, Y. Wang, J. Zhang, X. Meng, and L. Qian, "Large-scale dynamic scheduling for flexible job-shop with random arrivals of new jobs by hierarchical reinforcement learning," *IEEE Transactions on Industrial Informatics*, vol. 20, no. 1, pp. 1007–1018, 2023.

[26] Y. Zeng, Z. Liao, Y. Dai, R. Wang, X. Li, and B. Yuan, "Hybrid intelligence for dynamic job-shop scheduling with deep reinforcement learning and attention mechanism," *arXiv preprint arXiv:2201.00548*, 2022, DOI:10.1109/10.48550/arXiv.2201.00548.

[27] Y. Du, J. Li, C. Li, and P. Duan, "A reinforcement learning approach for flexible job shop scheduling problem with crane transportation and setup times," *IEEE Transactions on Neural Networks and Learning Systems*, 2022, DOI:10.1109/TNNLS.2022.3208942.

[28] L. Zhang, Y. Feng, Q. Xiao, Y. Xu, D. Li, D. Yang, and Z. Yang, "Deep reinforcement learning for dynamic flexible job shop scheduling problem considering variable processing times," *Journal of Manufacturing Systems*, vol. 71, pp. 257–273, 2023.

[29] Z. Wu, H. Fan, Y. Sun, and M. Peng, "Efficient multi-objective optimization on dynamic flexible job shop scheduling using deep reinforcement learning approach," *Processes*, vol. 11, no. 7, p. 2018, 2023.

[30] 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.

[31] Y. Gui, D. Tang, H. Zhu, Y. Zhang, and Z. Zhang, "Dynamic scheduling for flexible job shop using a deep reinforcement learning approach," *Computers & Industrial Engineering*, vol. 180, p. 109255, 2023.

[32] R. Liu, R. Piplani, and C. Toro, "Deep reinforcement learning for dynamic scheduling of a flexible job shop," *International Journal of Production Research*, vol. 60, no. 13, pp. 4049–4069, 2022.

[33] W. B. Langdon and M. Harman, "Optimizing existing software with genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 1, pp. 118–135, 2014.

[34] M. Đurasević, D. Jakobović, and K. Knežević, "Adaptive scheduling on unrelated machines with genetic programming," *Applied Soft Computing*, vol. 48, pp. 419–430, 2016.

[35] W. B. Langdon and R. Poli, *Foundations of genetic programming*. Springer Science & Business Media, 2013.

[36] M. Xu, F. Zhang, Y. Mei, and M. Zhang, "Genetic programming with archive for dynamic flexible job shop scheduling," in *Proceedings of the IEEE Congress on Evolutionary Computation*, jul 2021, pp. 2117–2124.

[37] 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.

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

[39] K. Jaklinović, M. Đurasević, and D. Jakobović, "Designing dispatching rules with genetic programming for the unrelated machines environment with constraints," *Expert Systems with Applications*, vol. 172, p. 114548, 2021.

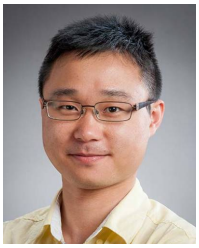
[40] M. Đurasević and D. Jakobović, "Comparison of schedule generation schemes for designing dispatching rules with genetic programming in the unrelated machines environment," *Applied Soft Computing*, vol. 96, p. 106637, 2020.

[41] S. Shady, T. Kaihara, N. Fujii, and D. Kokuryo, "A novel feature selection for evolving compact dispatching rules using genetic programming for dynamic job shop scheduling," *International Journal of Production Research*, vol. 60, no. 13, pp. 4025–4048, 2022.

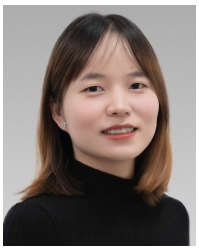
[42] F. Zhang, Y. Mei, S. Nguyen, K. C. Tan, and M. Zhang, "Multitask genetic programming based generative hyper-heuristics: A case study in

dynamic scheduling," *IEEE Transactions on Cybernetics*, vol. 52, no. 10, pp. 10 515–10 528, oct 2022.
[43] 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.
[44] M. Đurasević and D. Jakobović, "Comparison of ensemble learning methods for creating ensembles of dispatching rules for the unrelated machines environment," *Genetic Programming and Evolvable Machines*, vol. 19, pp. 53–92, 2018.
[45] D. Jackson, "Promoting phenotypic diversity in genetic programming," in *Proceedings of International Conference on Parallel Problem Solving from Nature*. Springer, 2010, pp. 472–481.
[46] E. K. Burke, S. Gustafson, and G. Kendall, "Diversity in genetic programming: An analysis of measures and correlation with fitness," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 1, pp. 47–62, 2004.
[47] M. Xu, Y. Mei, F. Zhang, and M. Zhang, "Genetic programming and reinforcement learning on learning heuristics for dynamic scheduling: A preliminary comparison," *IEEE Computational Intelligence Magazine*, vol. 19, no. 2, pp. 18–33, 2024.
[48] X. Chen, R. Bai, R. Qu, J. Dong, and Y. Jin, "Deep reinforcement learning assisted genetic programming ensemble hyper-heuristics for dynamic scheduling of container port trucks," *IEEE Transactions on Evolutionary Computation*, 2024, DOI:10.1109/TEVC.2024.3381042.
[49] T. Hildebrandt and J. Branke, "On using surrogates with genetic programming," *Evolutionary Computation*, vol. 23, no. 3, pp. 343–367, 2015.
[50] J. Filar and K. Vrieze, *Competitive Markov decision processes*. Springer Science & Business Media, 2012.
[51] H. Bai, R. Cheng, and Y. Jin, "Evolutionary reinforcement learning: A survey," *Intelligent Computing*, vol. 2, p. 0025, 2023.
[52] Z. Ren, G. Zhu, H. Hu, B. Han, J. Chen, and C. Zhang, "On the estimation bias in double q-learning," *Proceedings of the Advances in Neural Information Processing Systems*, vol. 34, pp. 10 246–10 259, 2021.
[53] H. Xiong, H. Fan, G. Jiang, and G. Li, "A simulation-based study of dispatching rules in a dynamic job shop scheduling problem with batch release and extended technical precedence constraints," *European Journal of Operational Research*, vol. 257, no. 1, pp. 13–24, 2017.
[54] 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 Genetic and Evolutionary Computation Conference Companion*, 2012, pp. 85–92.
[55] 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.
[56] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Proceedings of the Advances in Neural Information Processing Systems*, vol. 32, 2019.
[57] D. W. Zimmerman and B. D. Zumbo, "Relative power of the wilcoxon test, the friedman test, and repeated-measures anova on ranks," *Journal of Experimental Educational*, pp. 75–86, 1993.
[58] F. Wilcoxon, S. Katti, R. A. Wilcox *et al.*, "Critical values and probability levels for the wilcoxon rank sum test and the wilcoxon signed rank test," *Selected Tables in Mathematical Statistics*, vol. 1, pp. 171–259, 1970.

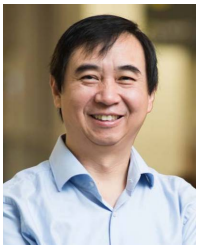
Meng Xu received the B.Sc. and M.Sc. degrees from the Beijing Institute of Technology, Beijing, China, in 2017 and 2020, respectively. She is currently pursuing a Ph.D. degree in computer science with the Centre for Data Science and Artificial Intelligence & School of Engineering and Computer Science,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



Yi Mei received the B.Sc. and Ph.D. degrees from the University of Science and Technology of China, Hefei, China, in 2005 and 2010, respectively. He is an Associate Professor at the Centre for Data Science and Artificial Intelligence & School of Engineering and Computer Science, and an 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 optimisation, transfer/multitask learning, and optimisation. He has published in top journals in evolutionary computation and operations research 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 lecturer with the Centre for Data Science and Artificial Intelligence & School of Engineering and Computer Science, Victoria University of Wellington, New Zealand. She has over 65 papers in refereed international journals and conferences. Her research interests include evolutionary computation, hyper-heuristic learning/optimisation, job shop scheduling, surrogate, and multitask learning. Dr. Fangfang is an Associate Editor of Expert Systems With Applications, and Swarm and Evolutionary Computation. 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 the IEEE New Zealand Central Section and was the Chair of the IEEE Student Branch at Victoria University of New Zealand, and the chair of the Professional Activities Coordinator. She is a Vice-Chair of the IEEE Taskforce on Evolutionary Scheduling and Combinatorial Optimisation.



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 and head of the group of the Centre for Data Science and Artificial Intelligence & School of Engineering and Computer Science. His research interests include evolutionary computation, genetic programming, multi-objective optimisation, and job shop scheduling. He has published over 700 papers in refereed international journals and conferences. Prof. Zhang is a Fellow of the Royal Society of New Zealand, a Fellow of Engineering New Zealand, a Fellow of IEEE, and an IEEE CIS Distinguished Lecturer. He was the chair of the IEEE CIS Intelligent Systems and Applications Technical Committee, the chair of 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.