Evolutionary Trainer-based Deep Q-Network for Dynamic Flexible Job Shop Scheduling

Yun Liu, Fangfang Zhang, *Member, IEEE*, Yanan Sun, *Senior Member, IEEE* and Mengjie Zhang *Fellow, IEEE*

***Abstract*—Dynamic flexible job shop scheduling (DFJSS) aims to achieve the optimal efficiency for production planning in the face of dynamic events. In practice, deep Q-network (DQN) algorithms have been intensively studied for solving various DFJSS problems. However, these algorithms often cause moving targets for the given job-shop state. This will inevitably lead to unstable training and severe deterioration of the performance. In this paper, we propose a training algorithm based on genetic algorithm to efficiently and effectively address this critical issue. Specifically, a state feature extraction method is first developed, which can effectively represent different job shop scenarios. Furthermore, a genetic encoding strategy is designed, which can reduce the encoding length to enhance search ability. In addition, an evaluation strategy is proposed to calculate a fixed target for each job-shop state, which can avoid the parameter update of target networks. With the designs, the DQNs could be stably trained, thus their performance is greatly improved. Extensive experiments demonstrate that the proposed algorithm outperforms the state-of-the-art peer competitors in terms of both effectiveness and generalizability to multiple scheduling scenarios with different scales. In addition, the ablation study also reveals that the proposed algorithm can outperform the DQN algorithms with different updating frequencies of target networks.**

***Index Terms*—Dynamic flexible job shop scheduling, deep Q-network, evolutionary algorithm, dispatching rules**

I. Introduction

Job shop scheduling (JSS) involves finding effective schedules for processing a number of jobs on a set of machines, aiming to minimize lead time and total cost [1]. In JSS, each job consists of multiple operations with strict sequencing constraints, and each operation can be processed by a specific machine. Flexible JSS (FJSS) is an extension of JSS [2]. In FJSS, each operation can be processed by several candidate machines, and the processing time for each operation varies among different machines with dissimilar configurations. In principle, two important tasks need to be processed simultaneously in FJSS [3]. One is machine assignment, aiming to allocate operations to available machines when an operation can proceed [4]. The other is the operation sequence to determine the next operation to be processed by an idle machine [5]. Although FJSS allows for more flexibility and adaptability than JSS, it does not take into account the dynamic events frequently occurring in real-world applications, such as new job arrivals [6] and machine breakdown [7]. Dynamic FJSS (DFJSS) is developed to simultaneously handle both machine assignments and operation sequencing under a dynamic environment, such as job arrivals over time and machine breakdown. DFJSS is more practical in the actual manufacturing environment, such as aircraft assembly [8] and automotive industry [9]. Furthermore, DFJSS is more challenging than JSS and FJSS due to the uncertainty brought by dynamic events [10].

Existing methods to solve DFJSS problems can be generally divided into three different categories. They are exact optimization algorithms [11], approximate solution optimization algorithms [12], and dispatching rules [13]. In practice, the exact optimization algorithms can often guarantee the global optimal solution yet are not efficient for dynamic scheduling environments of DFJSS [14]. Approximate optimization algorithms can get a good solution for both FJSS and DFJSS in a reasonable time. Unfortunately, such methods cannot timely respond to frequent reschedules in DFJSS [15]. Dispatching rules are currently the most popular methods for handling DFJSS owing to their simplicity of implementation and ability to react in real-time. These rules can efficiently allocate resources based on empirically designed criteria by prioritizing operations and machines at decision points (i.e., the moment when dynamic events occur) [16]. Many dispatching rules have been widely used in DFJSS, such as LPT (Longest processing time) [17], WSPT (Weighted shortest processing time) [18], and some combination rules [19].

Typically, the dispatching rules are manually designed for a particular DFJSS problem, which highly relies on expertise [20]. Moreover, most dispatching rules follow the same criteria at decision points without incorporating feedback from the environment. For DFJSS, environmental change may cause the rule inapplicable, which limits its adaptability and performance [21]. Therefore, it is necessary to select the appropriate dispatching rules adaptively according to the environment. Based on this, the selection methods of dispatching rules have been studied, such as analytic hierarchy [22], semantics [23], and the Deep Q-network (DQN) based deep reinforcement learning [24]. Among those, DQN is well-known for feedback exploration [25]. Specifically, during the training process, DQN receives immediate feedback in the form of rewards calculated by the outcomes of the decisions [26]. Meanwhile, DQN learns to approximate the targets representing the expected accumulation of these rewards, to achieve optimal decisions. The targets are evaluated via target network, which

This work was supported by National Natural Science Foundation of China under Grant 62276175. *(corresponding author: Yanan Sun)*

Yun Liu and Yanan Sun are with the College of Computer Science, Sichuan University, Chengdu 610065, PR China (e-mail: yliu@stu.scu.edu.cn; ysun@scu.edu.cn).

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

has become a common practice in solving DFJSS.

However, the use of target networks has the limitation of unstable training [27]. This will affect the generalization and effectiveness of DQN in solving different DFJSS instances. To be specific, the target networks need to be updated during the training. If the target network updates frequently, the Q-network will try to follow the rapidly moving target. Consequently, the oscillation will appear [28] and the selection of the optimal dispatching rule would be difficult in DFJSS. In contrast, if the target network is not updated frequently enough, the Q-network may fail to adapt to the changes, and poor dispatching rules would be selected [29]. To address this issue, research has been devoted to determining the optimal updating frequency. For example, Zhao et al. [30] conducted experiments to simultaneously optimize the production objectives. Li et al. [31] weighted the parameters of two networks in DQN at each training step and then updated the frequency. In essential, the primary cause of training instability in DQN lies in the update of target networks, which introduces moving targets. Although these methods alleviate the instability to some extent, the moving targets still exist in their training processes, which will further bring the following issues for DFJSS. First, the moving targets may introduce fluctuations in the training signal, further bringing noise [32]. Existing research has reported that the noise may lead to overfitting [33]. Second, the update frequency still needs to be properly predefined, which requires expertise and multiple experimental validations. In practice, the setting process is prohibitively time-consuming [34].

The goal of this paper is to design an effective algorithm for target evaluation based on genetic algorithms (GAs). To the best of our knowledge, this study is the first to use a GA in DQN training for DFJSS. GA is a stochastic global search algorithm, that adapts the meta-heuristic pattern motivated by the theory of natural evolution [35]. GA has potential in generating high-quality targets for achieving stable training of DQNs. On one hand, DQN requires weight updating based on the gradient information during the process of training, which is the primary cause of training instability, as discussed above. Whereas GA does not rely on gradient information, and several critical parameters affecting solutions are predetermined and remain constant during the optimization process [36]. On the other hand, GA is based on a population of candidate solutions rather than a network [37]. By maintaining a diverse population and implementing efficient fitness evaluation, GAs can potentially discover high-quality targets [38]. Note that other evolutionary algorithms can be also used to calculate the scheduling solutions for training the Q-network, while in this work, GA is used just with the consideration of its popularity in the community.

The benefit of the proposed algorithm lies in the direct calculation of the best targets, and not requiring any discussion on the update frequency of the target network during the training process of DQN. The contributions of the proposed method are summarized as follows.

- A new genetic encoding strategy is designed, which can flexibly encode the schedule at each decision point and greatly reduce the length of chromosomes, accordingly successfully avoiding the disadvantages of the existing strategy. The existing encoding strategies usually encode a complete solution of DFJSS, which are lacking of flexibility and cannot adapt to dynamic events. Moreover, the encoding of a complete solution will also inevitably include some invalid information, which unnecessarily increases the search space and lower the search efficiency of GAs.
- A new fitness evaluation strategy is developed, which can facilitate obtaining targets for particular states with the new mutation operations. The existing methods are based on the target networks, which often cause moving targets and make the training unstable. With the developed evaluation strategy, the calculated fixed targets can benefit the Q-network to achieve stable training for the application across different scheduling tasks.
- A new state feature extraction method is developed, which can effectively extract valid information from the state expression. Raw data often contains a large number of features, which can lead to the curse of dimensionality. With the developed method, the state features are efficiently extracted, and the resulting representations can be more domain-independent, allowing the model to generalize well across different DFJSS instances.
- Extensive experiments are conducted on multiple DFJSS instances with different scales. The results reveal the excellent feature extraction capability of the proposed algorithm, facilitating efficient learning of the underlying patterns and relationships of the training data. In addition, a comparative analysis is also performed on the proposed algorithm against DQN algorithms with different update ways of the target network. The results show that employing GA instead of the target network to train the Q-network can effectively improve the stability of the training.

The remainder of this paper is summarized as follows. The background and related work are described in Section II, followed by the details of the proposed algorithm in Section III. Sections IV and V present the experiment designs and experimental results, respectively. Finally, the conclusions and future work are drawn in Section VI.

II. Background and Related Work

This section introduces the fundamentals of DFJSS, learning dispatching rules for DFJSS, and DQN algorithms for DFJSS, and the related work on the encoding strategy of GAs in DFJSS.

A. Dynamic Flexible Job Shop Scheduling (DFJSS)

In a traditional DFJSS problem [9], there are m machines $M = \{M_1, M_2, ..., M_m\}$ and n jobs $J = \{J_1, J_2, ..., J_n\}$ arrive at the shop floor over time. Each job J_i has a sequence of k operations $O_i = \{O_{i1}, O_{i2}, ..., O_{ik}\}$ that must be processed in order. Each operation can be processed by more than a single candidate machine, with different machines having different processing times for an operation. The processing time of O_{ij} depends on the machine M_f on which O_{ij}

would be ultimately processed. DFJSS aims to make decisions on routing and sequencing simultaneously under dynamic environments. In practice, there are various dynamic environments. In this paper, we follow the convention of the DFJSS community [39] that focuses on the dynamic environments of new job arrival and machine failures. Specifically, in the phase of new job arrival, the information about the new job is unknown until it arrives at the job shop. The information about the failed machine is unknown when the event of machine failure occurs. In a traditional DFJSS, the following constraints are also needed to be considered:

1) Each machine handles at most one operation at a time;
2) Each operation can only be performed by one of its candidate machines;
3) Each sequence of operations belonging to the same job is predefined, and the current operation cannot be started until the previous operation has been processed;
4) Scheduling is non-preemptive, i.e., once the scheduling of an operation starts, the process cannot be aborted until it is completed.

In summary, DFJSS assigns the appropriate machines for the operations and determines the sequence of operations in the machine queue. The goal of DFJSS is to minimize the makespan (i.e., the completion time of the last process in all machine queues) while satisfying constraints.

B. *Learning Dispatching Rules for DFJSS*

Dispatching rules are critical in solving scheduling problems because they can react in real-time based on the latest information. In recent years, some methods that can automatically design the dispatching rules have been studied [40]. These methods are commonly known as hyper-heuristics, which can be classified into the categories of *heuristic generation* and *heuristic selection*.

The methods falling into the first category refer to creating new heuristics based on existing low-level heuristics to handle all situations, such as least work in the queue and first come first serve [4]. In the literature, genetic programming (GP) stands out owing to its flexibility and has been successfully utilized to generate fresh high-level scheduling heuristics [41]. Different from the first category, the methods of the second category do not generate completely new scheduling heuristics. Instead, these methods select the most appropriate ones from the existing heuristics for execution based on different job-shop states. These methods combine the strengths of several heuristics and have been extensively validated for their effectiveness in solving DFJSS in different scenarios [42]. At present, there are some traditional methods for the selection, such as genetic algorithms [43], reinforcement learning (RL) [44], and Bayesian learning [45]. According to recent literature [46], RL, in particular the DQN, has emerged as a growing trend to select scheduling heuristics for DFJSS. The work in this paper is also based on RL by following the heuristic selection.

In the last few years, researchers have shown significant interest in exploring heuristic selection with DQN methods. These methods employ offline training to train models and subsequently utilize the trained models for online applications. The overall process is shown in Fig. 1. For example, Luo et al. [6] trained a double DQN model offline and applied it online to select the most proper rules from six designed composite dispatching rule pools to solve the DFJSS, with the objective of minimizing total tardiness. Lei et al. [47] obtained an end-to-end hierarchical DQN model through offline training and employed it online for solving various DFJSS scenarios with different scales. Zeng et al. [48] achieved offline training by randomly generating DFJSS scenarios for a DQN agent, and the trained agent sequentially constructs a complete scheduling solution in an online application.

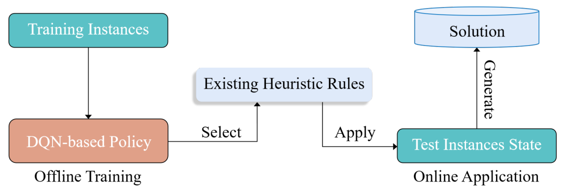


Fig. 1. Overall process of heuristic selection via DQN in DFJSS.

In general, offline training can make use of pre-existing data to train the DQN before the deployment. In the real online application, the trained policy is tested on unseen instances to construct a complete scheduling solution. According to the scheduling solution, the processing information of each job, such as the start and finish time of each operation, can be determined. This paradigm can learn patterns and features from pre-existing data and improve data utilization. Owing to this, the proposed algorithm also follows this paradigm.

C. *Deep Q-network for DFJSS*

Markov Decision Process (MDP) [49] is a mathematical framework that describes the stochasticity and uncertainty in decision making, which consists of state space S, action space A, state transition probability P, and reward function R. MDP can provide effective guidance for making optimal decisions in unknown environments. The Q-learning can be modeled as a MDP. In Q-learning, an agent observes the current state $s_t \in S$ at a decision point t and takes an action $a_t \in A$. After that, the agent gets into a new state $s_{t+1} \in S$ with the transition probability $p \in P$ and receives an immediate reward $r_t \in R$. Furthermore, the Q-value, denoted as $Q_\pi(s, a)$, is the maximum expected reward upon taking an action a in state s and following a specific policy π (also known as the Q-function or action-value function).

The objective of Q-learning is to learn the optimal $Q_\pi(s, a)$, which enables the agent to take the good action in a given state. Bellman [50] has proved that the Q-value $Q_\pi(s_t, a_t)$ at a decision point t can be described as Eq. (1),

$$Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}}[r + \gamma max_{a_{t+1}} Q_\pi(s_{t+1}, a_{t+1})|s_t, a_t], \quad (1)$$

where $\mathbb{E}$ represents the mathematical expectation and $\gamma \in [0, 1]$ is the discount factor to balance the relative importance of immediate reward r and cumulative reward $Q_\pi(s_{t+1}, a_{t+1})$. However, as real-world problems become increasingly complicated, the action space becomes large and the state space is vast, giving rise to the issue of dimensionality explosion.

This results in an exponential growth in the search space of the Q-function, requiring more resources for learning the optimal π. To address the issue, Mnih et al. [51] proposed the deep Q-network (DQN), which uses neural networks as the approximator of the Q-function. In practice, neural networks can efficiently represent high-dimensional space, allowing for more effective learning in problems where the number of possible states and actions is vast. Moreover, the Q-function can be approximated as a continuous function by using neural networks, making it feasible to estimate Q-values for unseen state-action pairs.

There are two promising properties that make DQN suitable for solving DFJSS problems. First, DQN has the ability to handle stochasticity by approximating the Q-function that adapts to dynamic changes in the manufacturing process, which is crucial for DFJSS with dynamic events. Second, DQN can adapt to dynamic environment because of the interaction between environment and agent. This nature enables the adaptive selection of the scheduling decisions by learning the experience from the interaction.

Over the past years, an increasing number of researchers have attempted to apply DQN in solving the DFJSS problem. For example, Li et al. [31] applied DQN to learn to choose the appropriate rule based on the production state at each decision point, which can effectively deal with disturbance events and unseen situations through learning. Johnson et al. [52] designed some cooperative DQN-based agents for scheduling dynamically arriving assembly jobs in a robot assembly cell, which demonstrated improved performance against heuristic methods. Wang et al. [53] combined DQN with a real-time processing framework to process each dynamic event and generate a complete scheduling solution, which allows for stable optimization of multi-objective in job-shop scenarios with varying scales. In summary, existing works provide valuable insights into the use of DQNs to improve efficiency and optimization in DFJSS.

D. Graph Convolutional Network for DFJSS

In DFJSS, the job-shop states have been effectively represented as graph data based on the constraints (elaborated in Section III-B). Graph embedding has garnered significant attention due to the ability to accurately preserve attributes and graphical structures in vector space, facilitating efficient knowledge querying. Among various graph embedding technologies, graph convolutional network (GCN) stands out for its exceptional capability in comprehensively learning graph information [54]. GCN is a deep learning model designed for graph data, enabling effective information propagation and feature learning by capturing relationships and characteristics among nodes. Recent studies have leveraged GCN for graph embedding in DFJSS. For example, Zhang et al. [55] introduced an industrial knowledge graph to express the job-shop state and proposed a GCN-based embedding algorithm, achieving the effective decomposition of scheduling tasks. Xiao et al. [56] employed graph generation to represent the process of scheduling and compressed the topology of the graph by performing graph embedding with GCN. The method can be swiftly adapted to realize flexible production. Su et al. [57] utilized GCN to extract the embeddings of the job-shop state expressed by a graph, facilitating the construction of effective scheduling solutions through optimal operations dispatch to machines. These studies collectively provide valuable insights into the application of GCN for graph embedding in DFJSS, highlighting the potential for improving scheduling processes.

E. Encoding Strategy of GAs in DFJSS

GAs typically work on a population of individuals that evolves over generations using bioinspired operators, including selection, crossover, and mutation, to generate high-quality solutions for a wide range of problems. In GAs, the individuals, representing candidate solutions, are called chromosomes. Commonly, chromosome encoding serves as the foundation of GA for solving problems, exerting a direct impact on the quality of the generated optimal solution. Therefore, the valid encoding strategy needs to be carefully designed.

In the job shop scheduling community, encoding operations of all jobs is a straightforward way to represent a feasible solution. However, these encoding strategies usually lead to high-dimensional search space and lack flexibility for environmental changes. For example, Li et al. [58] encoded all jobs and machines into a chromosome with two parts. The first represents the operation sequencing, and the second represents the machine selection. Ning et al. [59] divided the encoding into three parts, including the operation vector, machine vector, and time vector. He et al. [60] utilized an operation-based encoding strategy, where the value representing an operation in the chromosome is set by the serial number of the job.

It is evident that these strategies encode all the information in the job shop at once, in other words, a chromosome can be considered as a complete scheduling solution. Although these strategies have demonstrated favorable outcomes for FJSS, it is still necessary to study a new encoding strategy to minimize the makespan. Specifically, the work uses GA to calculate the target in DQN at the decision point. The calculation of the target relies on the encoding of the schedule at a decision point (i.e., a portion of the entire scheduling solution). However, the existing encoding strategies make encoding of some redundant information, deteriorating the search ability. Moreover, to our knowledge, no technique was designed in GAs for DFJSS to generate scheduling solutions directly at decision points. Therefore, it is necessary to design a new encoding strategy to allow GA to flexibly calculate the target at each decision point.

III. Proposed Algorithm

In this section, the proposed evolutionary trainer-based deep Q-network (ETDQN) for dispatching rule selection is detailed.

A. Algorithm Overview

Algorithm 1 outlines the framework of the proposed ETDQN algorithm, where our contributions in this paper are highlighted in bold. First, the DFJSS problem is modeled by the Markov Decision Process (MDP) (Line 1). Then, a Q-network is initialized with the random weight for DFJSS (Line

2). Next, the Q-network is trained with the specified maximal epochs (Lines 3-14). Finally, the optimal Q-network is applied to solve the DFJSS problem (Line 15).

Algorithm 1: Framework of the proposed ETDQN algorithm

```
1  Model DFJSS with the Markov Decision Process formulation;
2  $Q(\theta) \leftarrow$ Initialize a Q-network with random weight $\theta$ for DFJSS;
3  for $i \leftarrow 1$ to the predefined training epoch number do
4      $q \leftarrow$ Predict the Q-value based on the designed state feature extraction method;
5      $P_0 \leftarrow$ Initialize the population with the proposed encoding strategy;
6      $t \leftarrow 0$;
7      while $t <$ the predefined maximal generation number do
8          Evaluate the fitness of individuals in $P_t$ with the proposed fitness evaluation startegy;
9          $P_{t+1} \leftarrow$ Generate a new population with the proposed offspring generation method;
10         $t \leftarrow t + 1$;
11     end
12     $q' \leftarrow$ Obtain the fitness of the best individual in $P_t$;
13     $\theta \leftarrow$ Update $\theta$ by performing gradient descent on minimizing $(q' - q)^2$;
14 end
15 Apply $Q(\theta)$ to solve DFJSS problem.
```

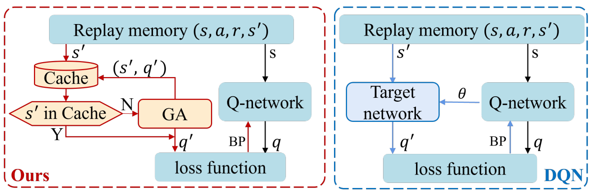
During the training, the Q-values are predicted first based on the designed state feature extraction method (Line 4). Then, the population is initialized based on the proposed encoding strategy (Line 5). After that, the evolution begins to take effect until a predefined maximal generation number is reached (Lines 7-11). Next, the fitness of the best individual is obtained to be the target for calculating loss (Line 12). Finally, the weights of the Q-network are updated by performing gradient descent to minimize the loss (Line 13). During the evolution, all individuals are evaluated first based on the proposed fitness evaluation strategy (Line 8), and a new population is generated with the proposed offspring generation method to participate in subsequent evolution (Line 9).

In summary, this paper aims to address the issue of training instability caused by the target network updates, which typically deteriorates the performance of the Q-network for DFJSS. Fig. 2 shows one key step in the training of ETDQN and DQN, where ETDQN utilizes GA (instead of the target network) to evolve a population of schedules with different targets for finding an 'optimal' target. With this design, ETDQN can work on different targets, improving the performance of Q-network for DFJSS. In the following sub-sections, the key steps in Algorithm 1 are detailed.

B. Markov Decision Process (MDP) Formulation

The MDP formulation aims to provide an effective way to model DFJSS as a discrete-time sequential decision process. Typically, MDP can be formalized by a tuple (S, A, P, R), where each element of the tuple represents specific meanings related to the DFJSS problem. The details of each element are as follows.

1) State space S: S encompasses the possible states of the DFJSS at each decision point, and these states can encode factors that influence the scheduling decisions, such as job attributes and machine availability. In this paper, the state is represented by the disjunctive graph [48] because of the intuitive presentation of the scheduling process. In theory, the disjunctive graph can generally be defined as $G = (N, V \cup E)$, where N represents the operation set of all jobs in the job shop, V is the set of conjunctive arcs, and E is the set of disjunctive arcs. Specifically, the conjunctive arcs join all the operations of the same job in order and the disjunctive arcs connect the operation that can be executed on the same machine. Note that the weights of the disjunctive arcs represent the processing time of operations. Moreover, the vertexes also include critical information about the operations, such as the operation name, start time, end time, assigned machine, and job ID.



(a) ETDQN (b) DQN

Fig. 2. One key step in the training of (a) the proposed ETDQN, and (b) basic DQN. To obtain the loss for updating the Q-network, DQN calculates the target q' with a target network that can update by copying the parameters of the Q-network and repeats the computation for repeatedly sampled data. On the contrary, GA does not require any parameter updates and computes for all samples only once.

2) Action set A: A refers to the set of possible actions that can be selected to take in a specific state. The actions typically consist of two types of selections: job selection and machine selection. Specifically, the job selection contains seven common sequencing rules, which are listed in Table I. The machine selection only contains the SPTM rule [48], which consistently prioritizes the machine with the shortest processing time.

TABLE I
SEVEN COMMON SEQUENCE RULES

Rules	Description
SR	Select the job with the shortest remaining time
LR	Select the job with the longest remaining time
SPT	Select the operation with the shortest processing time
LPT	Select the operation with the longest processing time
FOPNR	Select the job with the fewest operations remaining
MORPNR	Select the job with the most operations remaining
Random	Select operations in completely random order

3) State transition probability P: P refers to the probability distribution that governs the transitions from one state to another based on the actions taken by the model. Specifically, P, precisely modeled as $P(s'|s,a)$, quantifies the likelihood of reaching state s' from state s when the action a is executed. It is essential to use P to capture the stochastic nature of DFJSS because various factors can influence state transitions, such as new job arrival and machine failure. In this paper, the single-step state transition is chosen because of the immediate feedback on the outcomes of action.

4) Reward function R: R uses scalar feedback to guide training by performing action a in state s, which is based on scheduling objectives. This study focuses on the totall time to solve the DFJSS problem, where makespan is a best metric for determining the reward. A smaller makespan means a more optimal scheduling solution. Minimizing makespan maximizes

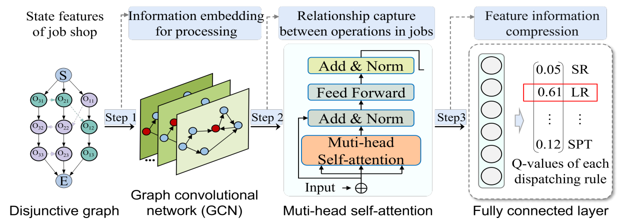

State features of job shop
Information embedding for processing
Relationship capture between operations in jobs
Feature information compression
S
E
Step 1
Step 2
Step3
Add & Norm
Feed Forward
Add & Norm
Muti-head Self-attention
Input
0.05 SR
0.61 LR
0.12 SPT
Q-values of each dispatching rule
Disjunctive graph
Graph convolutional network (GCN)
Muti-head self-attention
Fully connected layer

Fig. 3. Process of the proposed state feature extraction method.

machine utilization between decision points. Therefore, the immediate reward can be computed as Eq. (2):

$$R = \sum_{k=1}^{K} r_k = \frac{1}{M} \sum_{m=1}^{M} \frac{\sum_{i=1}^{J} \sum_{j=1}^{O} P_{ijm}}{C_{max}(k)} = \frac{P}{M \times C_{max}(k)}, \tag{2}$$

where k can be interpreted as a discrete decision point in RL. O is the number of operations belonging to job i, and J is the number of jobs. P is the total processing time and M is the number of machines. $C_{max}(k)$ is the makespan at decision point k. Because M are both constant, the cumulative reward can be further calculated as $P/C_{max}(k)$.

C. State Feature Extraction

The Q-network is the key component of the proposed ET-DQN algorithm, which extracts state information to evaluate the optimal action and is composed of three parts. The first is the graph convolutional neural network (GCN) [61], the second is the multi-headed self-attention mechanism [62], and the third is a fully connected layer. Consequently, the process of state feature extraction can be summarized in three key steps. First, GCN is responsible for embedding valid information from the disjunctive graph representing the state of the job shop. Second, the multi-head self-attention mechanism is used to focus on the vital state features by capturing the relationship between different operations in all jobs. Third, the fully connected layer takes effect to compress these features to a fixed-length vector, which corresponds to the Q-value of different dispatching rules.

Fig. 3 shows an example of this process. Specifically, in this example, there are two machines to process three jobs, and each job contains the same number of operations. In the disjunctive graph, S and E represent two virtual nodes, which represent the source and sink. The other nodes represent operations, such as O_{31} means the first operation belonging to the third job. Moreover, each node contains information about the operations, such as start time, end time, and the assigned machine. The solid lines represent the dependencies between all operations belonging to the same job, while the dotted lines represent the processing order between operations that are assigned to the same machines. With this information, the state of the job shop is represented by the disjunctive graph. In the first step, the disjunctive graph is directly input into the GCN to form a vector, i.e., the embedding. In the second step, this vector is input into the multi-head self-attention network, which captures the relationship between the operations of the jobs, such as the dependencies between operations in the same job and the constraints on machine availability. Moreover, This step outputs a vector containing contextually enriched representations of the input vectors. In the third step, the output in the second step is input into a fully connected layer, and the best dispatching rule is selected, which is similar to a traditional classification task in machine learning.

The motivation of the proposed state feature extraction method is detailed in three parts. For the first part, the scheduling process can be represented as a graph structure, as discussed in Section III-B. The literature [54] shows the reliability of the GCN for end-to-end learning on graph data in various domains. These studies guide us to embed vertex and structural information in graphs using GCN. Specifically, the graph data is first fed into the GCN. GCN utilizes the connections in the graph structure to combine information from neighboring nodes, iteratively updating the feature representations of the nodes. This process enables efficient extraction and representation of features in graph data. Moreover, GCN can usually handle any size of the graph. This is particularly essential for addressing the dynamic variations in the disjunctive graph representing the job-shop state, arising from the dynamic events in DFJSS. For the second part, DFJSS has constraints on the operations of the same job and the operations processed by the same machine. Therefore, we argue that DFJSS has a sequence of information. There is a lot of work that uses the multi-head self-attention mechanism for sequence processing [63]. These methods can calculate the attention between each word in the sentence so as to learn the word dependence relationship and capture the internal structure of the sentence [64]. Inspired by these methods, we employ the multi-head self-attention mechanism to learn the relationships between different operations, enhancing the ability to focus on the most critical elements during decision-making. After these two parts, considering the fixed number of actions in our model, the feature information must be compressed into a fixed-length vector. Therefore, a fully connected layer is used to enable flexible accommodation of output size. The fully connected layer can aggregate the extracted features from previous layers into a fixed-length vector. This enables variable-size features to be mapped to the number of actions needed in this work.

D. Population Initialization

As introduced in Section II-A, a scheduling solution usually consists of information on machines and operations. The performance of a scheduling solution in DFJSS highly relies on its flexibility, which can be achieved using sequential scheduling. In the proposed encoding strategy, we sequentially schedule the candidate operations and idle machines to replace the convention of encoding all information at once in GA (the advantages will be discussed later in this section). The candidate operation is either the first operation of a job or an operation that follows a completed predecessor. In addition, such machines are considered idle that have processed the last operation in the waiting queue at the decision point.

An example of the proposed encoding strategy is shown in Fig. 4, which is composed of three main steps. The first

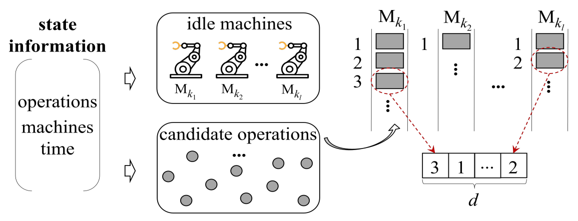


Fig. 4. Process of the proposed genetic encoding strategy.

is to extract the information of the candidate operations and idle machines based on the state. The second is to assign all candidate machines to the waiting queue of each idle machine. The third is to set the length of encoding as the number of idle machines and determine the value of each dimension. In the example, the first dimension in the encoding represents the machine M_{k_1}, and the dimension value of 3 indicates that the third operation in the waiting queue of M_{k_1} is selected. The advantage of this design is the lower dimensionality of the individual representation. Specifically, compared with the existing encoding strategy based on all jobs in the job shop, the proposed encoding strategy is based on machines. There is a mass of jobs and a small number of machines in DFJSS, especially in large-scale DFJSS scenarios where the number of jobs far exceeds the number of machines. Therefore, the limited number of machines results in a lower dimensionality of the individual representation.

Algorithm 2: Population Initialization

```
Input: state disjunctive graph $G$; the population size $N$; the waiting
       queue of machines $Q$.
Output: the initialized population $P_0$.
1  $M, O \leftarrow$ Obtain the idle machines and candidate operations in the
     current job shop according to $G$;
2  $D \leftarrow$ Calculate the number of machines in $M$;
3  $Q \leftarrow$ Update $Q$ by assigning $Q$ to $M$;
4  $P_0 \leftarrow \emptyset$ ;
5  $t \leftarrow 0$;
6  while $t < N$ do
7      $X \leftarrow \emptyset$;
8      for $i \leftarrow 1$ to $D - 1$ do
9          $m \leftarrow$ Obtain the $i$-th machine from $M$;
10         $N_q \leftarrow$ Calculate the number of operations in $Q[m.name]$;
11         $j \leftarrow$ Uniformly generated an integer between $[1, N_q]$;
12         $X \leftarrow X \cup j$;
13         Remove the $j$-th operations from $Q[m.name]$;
14     end
15     $P_0 \leftarrow P_0 \cup X$;
16     $t \leftarrow t + 1$;
17 end
18 Return $P_0$.
```

Based on the aforementioned encoding strategy, Algorithm 2 shows the details of the population initialization. Briefly, N individuals are uniformly initialized in the same way and then stored in P_0. During the initialization of one individual, the candidate operations and idle machines are obtained based on the disjunctive graph representing the state (Line 1). Next, the number of idle machines is calculated as the length of chromosomes (Line 2), and the waiting queues of machines are updated by assigning the candidate operations to idle machines (Line 3). After that, each element is configured (Lines 8-14), and then a complete individual is stored in P_0 (Line 15). During the configuration of each element, the information of the corresponding idle machine is obtained first (Line 9). Then, the number of candidate operations in the waiting queue of the machine is calculated and then assigned to N_q (Line 10). Next, an integer, assigned to j, is generated randomly between $[1, N_q]$ (Line 11) and stored into X (Line 12). After that, the corresponding operations (i.e. the j-th operation in the waiting queue) are removed from the queue (Line 13). The motivations behind this design are: 1) multiple dynamic events in DFJSS require adaptive adjustment with the current scheduling, encoding all the information at once is inflexible to confirm its optimal scheduling solution; and 2) evaluating individuals with the big length encoding can be time-consuming and inefficient, because the occurrence of dynamic events in DFJSS may invalidate part of the information in the individuals.

Algorithm 3: Fitness Evaluation

```
Input: The population $P_t$ of the individual to be evaluated; the state
       $S$; the reward $r$; the current generation number $t$; the predicted
       maximal generation number $T$.
Output: the population $P_t$ of the individuals with their fitness
        values.
1  if $epoch == 0$ then
2      $Cache \leftarrow \emptyset$;
3      Set $Cache$ to a global variable;
4  end
5  if $the\ identifier\ of\ S\ in\ Cache$ then
6      $v \leftarrow$ Query the fitness by $identifier$ from $Cache$;
7      Set $v$ to all individuals in $P_t$;
8  else
9      $v_{best} \leftarrow 0$;
10     $\gamma \leftarrow$ Uniformly generate a number from $(0, 1)$;
11     for $individual\ in\ P_t$ do
12         Decode a scheduling solution based on the information
             encoded in $individual$;
13         $C_{max} \leftarrow$ Calculate makespan with the scheduling solution;
14         $v \leftarrow r + \gamma \times \frac{1}{C_{max}}$;
15         if $v > v_{best}$ then
16             $v_{best} \leftarrow v$;
17         end
18     end
19     if $t == T - 1$ then
20         Put the identifier of $S$ and $v_{best}$ into $Cache$;
21     end
22 end
23 Return $P_t$.
```

E. Fitness Evaluation

Algorithm 3 manifests the fitness evaluation in the proposed ETDQN algorithm. Briefly, given the population P_t containing all the individuals for evaluating the fitness, Algorithm 3 evaluates each individual of P_t in the same manner, and finally returns P_t containing the individuals whose fitness has been evaluated. Specifically, if the fitness evaluation is for the first training step, a global cache (denoted as $Cache$) is created, storing the best fitness of the individuals with the unseen state of the job shop (Lines 1-4). The fitness is evaluated in two ways according to the state. If the state is found in $Cache$, the fitness is directly obtained from $Cache$ (Lines 5-8). Otherwise, the individual is asynchronously placed in the same job-shop environment for fitness evaluation (Lines 9-21). For each individual (denoted by $individual$) in P_t, first, a scheduling solution is decoded from the individual (Line

12) and the makespan is computed based on the scheduling solution (Line 13). Then, Line 14 shows the fitness evaluation of the individual. After that, the best fitness is assigned to v_{best} (Lines 15-17). Finally, if the fitness evaluation is for the maximal generation, the identifier and the best fitness are associated to put into $Cache$ for ease of query (Lines 19-21). Note that the identifiers are used to identify job-shop states in the $Cache$.

Because the proposed algorithm is concerned with evaluating fixed targets to achieve stable training in solving DFJSS problems, the process is modeled as an optimization problem and optimized by GA instead of the target network in DQN. Specifically, we define the calculation of targets as fitness, shown in Eq. (3). The definition is based on the Bellman equation, as discussed in Section II-C:

$$f(x) = r + \gamma \times \frac{P}{M \times C_{max}(x)}, \tag{3}$$

where x is the scheduling solution in decision points, encoded as detailed in Section III-B, and the other variables in the equation follow the explanations given in the preceding paragraphs. The process of optimizing targets by GA instead of target network can be summarized as maximizing $f(x)$ by searching for the optimal scheduling solution. This is because the target network converges to the optimal target by updating the weights, which is similar to that GAs iteratively search for the best fitness. Moreover, the fitness function is devised to quantify the reward that may result from implementing a scheduling solution represented by an individual in a particular state, which is effectively equivalent to the principle of the target in DQN. Furthermore, the cache component is used to speed up the target evaluation, which is based on the following considerations: 1) it is common for the state to be repeatedly sampled for training based on conventions of RL; 2) due to the convergence of the GA towards an optimal solution as it progresses, and considering the complexity of fitness evaluation, there is no need to reevaluate the target of the resampled state again.

F. Offspring Generation

The details of offspring generation are shown in Algorithm 4, which consists of three parts, i.e., crossover (Lines 1-10), mutation (Lines 12-20), and environmental selection (Lines 21-27). During the process of crossover, the single-point crossover is used and $|P_t|$ offsprings will be generated, where $|\cdot|$ measures the number of elements in the collection. Specifically, two different parents are randomly selected first (Line 3). After that, a number is randomly generated between 0 and 1 (Line 4), which is used to determine whether to perform the crossover operation or not. If the generated number is lower than the predefined crossover probability, two offspring are generated from the two-parent individuals by performing a single-point crossover operation (Line 6). Otherwise, the two parent individuals are considered to be the offspring and put into X_t (Line 9). During the process of mutation, a random number is generated first (Line 13), and the mutation is applied to the current individual if the generated number is less than the predefined mutation probability p_m (Lines 14-21). When performing mutation on an individual, a random position (denoted as i) is selected first (Line 15), and then obtain the idle machine corresponding to the position (Line 16). Next, a mutated value is randomly selected according to the waiting queue of the machine and performed on the position i (Lines 17-19). In the proposed algorithm, the available mutated values are defined as the operations in the waiting queue. During the process of selection, $|P_t|$ individuals are selected from the current population $(Q_t \cup X_t)$ by using the roulette wheel selection strategy and put into the next population (denoted as P_{t+1}) (Lines 22-23). Next, the best individual is found from $(Q_t \cup X_t)$ and determined whether it has been put into P_{t+1} or not. If not, it will replace the individual with the worst fitness in P_{t+1} (Lines 24-27).

Algorithm 4: Offspring Generation

```
Input: The population $P_t$ containing individuals with fitness; the crossover probability $p_c$; the mutation probability $p_m$; the length of individuals $L$; the waiting queue $Q$; the idle machines $M$.
Output: The offspring population $P_{t+1}$.
1  $X_t \leftarrow \emptyset$;
2  while $|X_t| < |P_t|$ do
3      $p_1, p_2 \leftarrow$ Randomly select two different individuals from $P_t$;
4      $r \leftarrow$ Randomly generate a number between (0,1);
5      if $r < p_c$ then
6          $o_1, o_2 \leftarrow$ Generate two offspring from $p_1$ and $p_2$ by performing a single-point crossover operation;
7          $X_t \leftarrow X_t \cup o_1 \cup o_2$;
8      else
9          $X_t \leftarrow X_t \cup p_1 \cup p_2$;
10     end
11 end
12 for $individual$ in $X_t$ do
13     $r \leftarrow$ Randomly generate a number between (0,1);
14     if $r < p_m$ then
15         $i \leftarrow$ Uniformly generate an integer between $[1, L]$;
16         $m \leftarrow$ Obtain the $i$-th idle machines from $M$;
17         $q \leftarrow$ Obtain the waiting queue of $m$ from $Q$;
18         $v \leftarrow$ Uniformly generate an integer between $[1, |q|]$;
19         Perform mutation $v$ at the point $i$ of $individual$;
20     end
21 end
22 $G_t \leftarrow P_t \cup X_t$;
23 $P_{t+1} \leftarrow$ Select the next population from $G_t$ by using the roulette wheel selection strategy;
24 $p_{best} \leftarrow$ Find the individual with the best fitness from $G_t$;
25 if $p_{best}$ is not in $P_{t+1}$ then
26     Replace the one who has the worst fitness in $P_{t+1}$ by $P_{best}$;
27 end
28 Return $P_{t+1}$.
```

Next, the motivation for designing such an offspring generation strategy is given. First, the proposed algorithm encodes the individuals based on a portion of the entire scheduling solution for calculating the targets at decision points. While the traditional offspring generation strategies for DFJSS are for the individuals encoding entire scheduling solutions. Second, existing algorithms perform the mutation operator by swapping or perturbing job sequences, which limits the search space to some extent [65]. The proposed algorithm selects the mutation values from the waiting queues of machines, which increases the diversity of the solution while ensuring feasibility.

Please note that the generated offspring are still feasible. Specifically, individuals are encoded based on machines, and the values for each dimension are selected from the respective waiting queue. The proposed crossover operator exchanges the values of the same dimension, which only means that the order of operations performed on the same machine is exchanged. Moreover, the designed mutation operation ensures that the mutated values are always appropriate for the corresponding machine because the mutated values are selected from the corresponding waiting queue.

IV. Experiment Design

A. Simulation Model

It is a convention in the DFJSS community to use simulation model for experimental verification. By following the conventions [66], in this paper, we use the simulation model with different random seeds to simulate 123 DFJSS instances. They can be divided into eight different scales, including 6×6, 10×6, 10×15, 15×6, 15×8, 20×6, 100×10, and $2,000\times10$. Note that 10×6 means that the DFJSS has ten jobs to be processed by six machines.

In addition, two dynamic events are separately considered, i.e., new job arrival and machine failure. Specifically, new jobs will arrive over time according to a Poisson process. Each job has a different operation number randomly generated from $[1,\cdots,10]$. The number of candidate machines for each operation is uniformly generated from $[1,\cdots,10]$. The processing time for each candidate machine to execute the operation is uniformly generated from $[1,\cdots,99]$. The new job will join other unfinished jobs, waiting to be assigned. In the event of machine failure, there are some principles to be followed [31]. 1) If a machine fails, it will stop running before the repair is completed. 2) During the failure, the faulty machine cannot be selected. 3) If the disrupted operations have other candidate machines, they can be immediately put back into the job shop to wait for machine allocation.

These simulations described above will be employed to train and test ETDQN. In the training phase, we employ an instance with the scale of 10×6 for the purpose based on the suggestion [6]. In the test phase, we divide these instances into four categories to demonstrate ETDQN. The first is the instances with the same scales as the training data to verify the effectiveness of ETDQN, regarding dataset I. The second is the instances having different scales from the training instances (denoted as dataset II), including 6×6, 10×15, 15×6, 15×8, and 20×6, to evaluate the generalization of ETDQN across DFJSS instances of different sizes. The third is two instances with the scale of 100×10 (denoted as dataset III), to demonstrate the improved feature extraction capability of ETDQN with fixed targets. The fourth is an instance with the scale of $2,000\times10$ (denoted as dataset IV), to verify the effectiveness and efficiency of ETDQN in dealing with the larger scale DFJSS instance.

B. Peer Competitors

The chosen peer competitors are divided into three categories based on the intention of experiments.

The first includes seven common dispatching rules manually designed by experts, which are selected as the representatives of non-learning-based approaches. The details of these rules have been introduced in Section III-B. Note that we label each dispatching rule with the sequencing rule for simplification, e.g., 'SPT+SPTM' is labeled as 'SPT'. Specifically, these rules are limited to the same criterion. For example, SPT is to prioritize the execution of jobs with the shortest processing time first, and the machine with the shortest processing time at any decision point.

The second contains two high-quality dispatching rules automatically generated by genetic programming (GP), named GPRule_1 and GPRule_2. Both are selected because they are often claimed as high-quality GP rules demonstrating superiority across multiple scheduling scenarios [3]. According to the study [3], GPRule_1 consists of the best routing rule and the corresponding sequencing rule achieved by CCGP^{2a} (mimic), while GPRule_2 contains the corresponding best rules obtained by CCGP^2.

The third refers to three heuristic selection methods, including a random policy (RP) [48] serving as baseline, and two state-of-the-art DQN-based algorithms, i.e., DDQN [67] and DLDQN [68], to verify the decision-making ability of ETDQN in dispatching rule selection. These rules may select different criteria at decision points. Specifically, RP means that the seven common dispatching rules will be selected with equal probability. DDQN and DQLDQN are learning-based algorithms, which can automatically decide dispatching rules based on the state of the job shop.

C. Parameter Settings

All parameter settings are set based on the conventions [31], [59], [69], which are summarized in Table II.

TABLE II
Parameters setting

Parameters name	Parameters value
Elitism	Top 1
Batch size	64
Learning rate	$1e-4$
Mutation rate	0.1
Crossover rate	0.8
Population size	20
Training epoch	$4,000$
Size of $Cache$	$100,000$
Size of replay memory	$100,000$
Maximum number of iterations of GA	20

During the test phase, due to the stochastic nature of the algorithms falling into the third category, 30 independent runs are performed on each instance. While the algorithms of the first and second categories only need to be executed once because they are deterministic rules. Furthermore, the *p-value* is calculated by performing the Wilcoxon signed-rank test with a significance level of 0.05. The proposed ETDQN algorithm is implemented by Pytorch, and the code runs on a computer equipped with a CPU of Intel(R) Core(TM) i5-12500H.

V. Experimental Results

In this section, the experimental results and analysis of ETDQN against peer competitors are shown in Section V-A. Then, the performance of fixed targets in ETDQN algorithm is specifically investigated in Section V-B. Finally, the selected dispatching rules are further analyzed in Section V-C

TABLE III
COMPARISON RESULTS OF MAKESPAN BETWEEN ETDQN AGAINST WELL-KNOWN ALGORITHMS ON THE INSTANCES IN DATASET II AND DATASET IV.

Dynamic event	algorithm	6 × 6		10 × 15		15 × 6		15 × 8		20 × 6		2000 × 10	
		mean	*p-value*	*mean*	*p-value*	*mean*	*p-value*	*mean*	*p-value*	*mean*	*p-value*	*mean*	*p-value*
new jobs arrival	SR	81.15	9.7e-05	74.10	1.2e-05	110.60	9.0e-08	88.95	1.2e-06	126.85	1.6e-05	45599.00	8.7e-05
	LR	78.60	3.6e-04	75.95	2.0e-06	107.90	1.2e-05	88.50	2.4e-06	132.20	2.1e-06	45790.00	3.9e-05
	SPT	87.80	1.4e-08	71.80	4.6e-04	107.75	2.6e-06	85.05	8.7e-06	113.85	4.3e-06	24191.00	1.2e-05
	LPT	82.90	6.7e-06	74.70	4.8e-06	108.45	5.2e-08	85.95	3.6e-08	116.70	4.7e-08	67592.00	3.3e-06
	FOPNR	81.75	1.5e-05	75.40	2.7e-06	106.15	2.8e-08	86.65	3.2e-06	125.25	1.9e-08	45454.00	2.9e-05
	MORPNR	77.90	6.7e-05	74.85	4.4e-06	100.05	4.9e-08	86.90	3.6e-05	117.25	9.6e-06	45685.00	6.7e-05
	Random	80.09	6.0e-05	77.87	5.5e-07	110.26	1.2e-08	91.56	1.8e-09	119.03	1.9e-09	68296.50	9.0e-06
	GPRule$_1$	72.30	8.7e-03	74.80	2.1e-04	105.30	1.6e-04	85.40	6.8e-05	118.50	1.2e-02	22163.15	9.5e-03
	GPRule$_2$	73.90	3.7e-04	78.15	7.9e-06	104.60	1.8e-03	86.10	9.7e-04	120.85	1.8e-02	22670.40	1.8e-03
	DDQN	73.94	5.2e-04	74.89	9.7e-04	105.04	1.6e-04	85.83	7.3e-05	122.29	7.3e-05	22177.83	8.3e-03
	DLDQN	73.82	3.4e-04	75.01	6.2e-03	105.06	1.4e-03	85.82	2.3e-04	122.35	1.8e-02	22172.67	8.1e-03
	RP	75.44	3.8e-06	76.10	3.8e-06	106.90	3.8e-06	87.01	3.8e-06	123.98	3.8e-06	23571.00	6.7e-04
	ETDQN	**60.45**	-	**54.51**	-	**81.36**	-	**74.04**	-	**101.31**	-	**22108.17**	-
machine failure	SR	46.25	5.7e-11	57.65	1.2e-09	84.15	2.8e-09	76.95	8.8e-05	109.10	2.9e-05	48049.00	10.0e-05
	LR	45.75	7.2e-11	61.25	6.5e-10	79.65	5.3e-08	75.20	1.4e-06	111.10	1.4e-06	48625.00	1.9e-06
	SPT	44.95	2.4e-08	59.25	7.1e-08	84.35	8.6e-07	78.25	4.6e-06	108.10	6.5e-05	26614.00	1.5e-05
	LPT	46.55	2.2e-09	61.30	2.5e-09	85.85	2.1e-09	78.65	1.6e-06	109.10	8.7e-06	26614.00	1.5e-06
	FOPNR	46.05	5.0e-11	56.80	5.1e-08	82.15	1.9e-10	76.45	6.5e-07	110.00	8.7e-06	48015.00	2.3e-05
	MORPNR	46.40	1.3e-09	56.55	7.2e-08	82.10	1.6e-07	76.20	7.1e-06	107.55	4.7e-04	48234.00	4.5e-05
	Random	47.15	5.7e-11	67.35	6.5e-10	91.25	2.6e-09	83.65	5.6e-07	112.25	8.8e-06	48234.00	3.8e-06
	GPRule$_1$	36.65	1.8e-03	43.90	2.9e-06	85.65	6.0e-03	69.30	2.4e-03	86.75	9.5e-05	24166.85	1.9e-03
	GPRule$_2$	39.95	4.8e-04	47.35	1.9e-06	86.75	1.7e-03	71.20	2.7e-05	87.40	1.9e-06	24497.80	8.9e-03
	DDQN	36.77	3.8e-06	41.72	1.9e-05	85.74	3.8e-06	69.08	1.3e-04	84.63	2.1e-04	24431.50	9.1e-03
	DLDQN	37.05	3.8e-06	41.89	3.8e-06	85.24	7.1e-03	69.18	5.3e-05	84.55	2.4e-03	24339.20	9.1e-03
	RP	37.47	3.8e-06	43.08	3.8e-06	87.32	3.8e-06	70.64	3.8e-06	86.73	3.8e-06	26482.25	1.6e-04
	ETDQN	**36.04**	-	**40.41**	-	**63.23**	-	**67.88**	-	**82.93**	-	**23901.05**	-

*: *mean* indicates the average of best objectives found across the remaining problem instances.
*: *p-value* represents the probability of observed differences in paired data assuming no true difference between the paired samples.

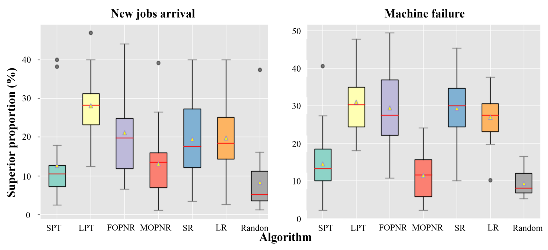


Fig. 5. Box plots of superior proportion of ETDQN to other single dispatching rules on test instances in dataset I under two dynamic events.

A. Overall Results

To validate the effectiveness and generalizability, we conduct a comparative analysis between the proposed ETDQN algorithm and two categories of peer competitors on DFJSS instances with six different scales, considering the presence of new job arrivals and machine failures. The mean makespan (denoted as $mean$) and *p-values* are recorded in Table III. Furthermore, Table IV presents the average test time between ETDQN against heuristic selection methods in the second category, to demonstrate the decision efficiency of ETDQN. In addition, to validate that ETDQN can make effective decisions based on states of the job shop, we compute the superior proportion of learned dispatching rules by ETDQN to the dispatching rules in the first category of peer competitors and presented these results in Fig. 5. Note that superior proportion represents the degree of advantage of one number over another. Specifically, the superior proportion of x to y can be calculated by $(y - x)/max(x, y)$, as defined in [31], where x and y are the objective values obtained by ETDQN and the single dispatching rules under comparison respectively.

It is clearly shown in Table III that by comparing the average performance, the proposed ETDQN algorithm outperforms all the chosen peer competitors. Compared to the peer competitors in the first category, ETDQN achieves performance over the baseline in the experimental evaluations. Specifically, these common dispatching rules can process DFJSS instances within a reasonable timeframe. Moreover, there are significant differences in performance when solving DFJSS instances of different sizes. While ETDQN can stand out in solving DFJSS instances of any size. These results demonstrate that ETDQN can successfully extract pertinent features from states and make informed scheduling decisions, leading to a significant reduction in the makespan. Comparison with GPRule$_1$ and GPRule$_2$ reveals that ETDQN still achieves the best results among all scales investigated in the experiments. Furthermore, these two rules were automatically evolved and generated from different scenarios and simulations, and the two rules also belong to a kind of *heuristic generation*. This is different from the main contribution of this paper focusing on *heuristic selection*. So such a comparison just provides a rough idea on how well GP-generated rules on other scheduling problems perform on the scheduling problems presented in this paper. Compared to the peer competitors in the third category, the makespan obtained by ETDQN is shorter than those of DDQN and DLDQN. Specifically, in terms of makespan, ETDQN obtains $0.31\% \sim 22.54\%$ and $0.29\% \sim 22.56\%$ improvements over DDQN and DLDQN under the dynamic events of new jobs arrival, as well $1.97\% \sim 26.24\%$ and $1.80\% \sim 25.81\%$ improvements over DDQN and DLDQN under the dynamic events of machine failure. As the size of DFJSS instances increases, the performance of ETDQN still exhibits a noticeable gap compared to DDQN and DLDQN. The results illustrate that ETDQN can be more effective in selecting the appropriate dispatching rules for different decision points in most problem instances. In addition, it can also be observed in Table III that all *p-values* calculated by ETDQN with competitors are much less than 0.05, which means that the objectives obtained

TABLE IV
AVERAGE TEST TIME FOR ETDQN AGAINST RP, DDQN, AND DLDQN ON DIFFERENT DFJSS INSTANCES. THE TIME UNIT IS IN SECONDS (s).

Dynamic event	size	RP	DDQN	DLDQN	ETDQN
new jobs arrival	6×6	0.1920	0.7913	0.7269	0.6593
	10×15	0.1547	1.1076	1.1515	1.0146
	15×6	0.2813	2.2059	2.2365	2.2037
	15×8	0.2814	1.9925	1.9204	1.1078
	20×6	0.5554	2.9635	2.9434	2.8144
	2000×10	407.82	682.99	671.79	623.40
machine failure	6×6	0.1510	0.3526	0.3740	0.4188
	10×15	0.1640	0.6139	0.6288	0.5779
	15×6	0.2020	1.5935	1.5818	1.5464
	15×8	0.1519	1.2225	1.3265	1.1313
	20×6	0.2554	2.3635	2.2434	2.2144
	2000×10	413.78	692.22	684.13	679.81

by ETDQN are obviously different from the competitors. In summary, ETDQN achieves the best makespan on all DFJSS instances under both new job arrival and machine failure.

According to Fig. 5, the lower whisker of all boxes is above the zero level, which indicates that ETDQN achieves the best performance compared to these selected dispatching rules. The yellow triangles represent the average of the corresponding superior proportions. It is obvious that the average superior proportions of ETDQN to these single dispatching rules are between $5\% \sim 30\%$ and $7\% \sim 31\%$, respectively. Meanwhile, it can be seen from the upper quartile of the box plots that ETDQN can achieve superiority of more than 10% for each dispatching rule on 75% of the test instances. One possible reason is that single dispatching rules do not take into account the state information and stubbornly use one rule at any rescheduling moment. In contrast, the trained ETDQN can effectively extract the state features of all operations and machines by means of the disjunctive graph, and determine the appropriate rule for the current rescheduling state accordingly.

According to Table IV, it is evident that ETDQN consistently records the shortest testing time compared to DDQN and DLDQN. By capturing and leveraging features in the state space, ETDQN excels in selecting dispatching rules that align optimally with the current job-shop state. The attribute highlights the superior capability of feature extraction by ETDQN. This capability allows ETDQN to make more effective selections of dispatching rules based on the current state, ultimately leading to quicker completion of processing FJSS instances, as discussed in Section V-C. Furthermore, it is noteworthy that RP exhibits the shortest test time, with ETDQN following as the second-ranked method in terms of test time when addressing the same DFJSS instance. Specifically, compared to ETDQN, DDQN, and DLDQN, RP incurred less time due to the utilization of a random policy for selecting dispatching rules, without the need to extract the job-shop states. However, as evident from the analysis above, it is clear that ETDQN, DDQN, and DLDQN can select more appropriate dispatching rules by extracting state features, resulting in a shorter makespan compared with RP. Shorter makespan can provide a competitive advantage in practical production, allowing companies to respond quickly to changing demands and market dynamics. Therefore, we believe that investing a certain amount of time in extracting state information holds value in effectively selecting dispatching rules for DFJSS.

TABLE V
COMPARISON RESULTS OF MAKESPAN BETWEEN ETDQN AGAINST THE DQN AND DQN^2 ON PROBLEM INSTANCES IN DATASETS I AND II.

Size	algorithm	new jobs arrival		machine failure	
		mean	*p-value*	*mean*	*p-value*
6×6	DQN	72.2249	9.68e-03	44.1010	1.24e-04
	DQN^2	73.1337	4.80e-03	44.0770	3.88e-04
	ETDQN	**60.4541**	-	**42.1470**	-
10×6	DQN	71.6210	3.70e-06	54.0690	1.45e-05
	DQN^2	70.4480	3.09e-06	54.5860	1.35e-05
	ETDQN	**62.7240**	-	**47.0350**	-
10×15	DQN	68.2851	1.73e-03	54.1445	1.17e-04
	DQN^2	68.3109	1.57e-03	53.5848	4.39e-04
	ETDQN	**54.5123**	-	**48.8300**	-
15×6	DQN	95.7413	4.52e-04	74.1700	1.81e-05
	DQN^2	95.3410	7.30e-04	73.7445	1.22e-05
	ETDQN	**81.3595**	-	**63.2345**	-
15×8	DQN	83.3310	4.57e-04	70.5850	9.51e-03
	DQN^2	82.8196	7.29e-04	70.8600	9.50e-03
	ETDQN	**74.0420**	-	**65.4000**	-
20×6	DQN	110.9985	2.24e-03	104.5780	1.91e-03
	DQN^2	109.7182	7.14e-03	104.5040	2.63e-03
	ETDQN	**101.3080**	-	**97.0065**	-

During the experiments, we also noticed that the training time of ETDQN is longer than that of the DQN-based algorithms. This is because of the utilization of GA which performs iterative evolutionary operations upon the population. However, ETDQN will not be disadvantaged since it can automatically select good rules for testing.

B. Ablation Study

Further experiments are performed to examine the effectiveness of the proposed ETDQN algorithm with fixed targets. By comparing different training way, we would investigate whether the targets will affect the performance of solving DFJSS instances or not. To achieve this, we compare ETDQN with two DQN algorithms with different update frequencies of the target networks, which are selected to verify whether the fixed targets will affect the performance of training. Specifically, one version of DQN employs a soft update strategy for updating the target network, where the target network is updated at a fixed interval [70]. The other version of DQN uses a soft update strategy for updating the target network, where the weights of the target network are updated at each training step by weighting strategy [6]. For convenience, they are called DQN and DQN^2, respectively.

Table V presents the average makespan values (denoted as $mean$) calculated by each algorithm to optimally solve DFJSS instances of six different sizes and records the *p-values* calculated by ETDQN with DQN and DQN^2. Furthermore, to substantiate the significant improvements in scalability and feature extraction of ETDQN with fixed targets, we compared ETDQN against two versions of DQN on two DFJSS instances with the size of 100×10 (denoted as Scenario_1 and Scenario_2), considering the same dynamic events so those in the above experiments. Fig. 6 presents the results of each algorithm with 30 independent runs on each DFJSS instance. In addition, to verify the effectiveness and efficiency of ETDQN in addressing the DFJSS instance with scale of 2000×10, the mean and standard deviation of makespan (denoted as *mean* and *std*, respectively), as well as the average test time (denoted as *time*), are recorded in Table VI.

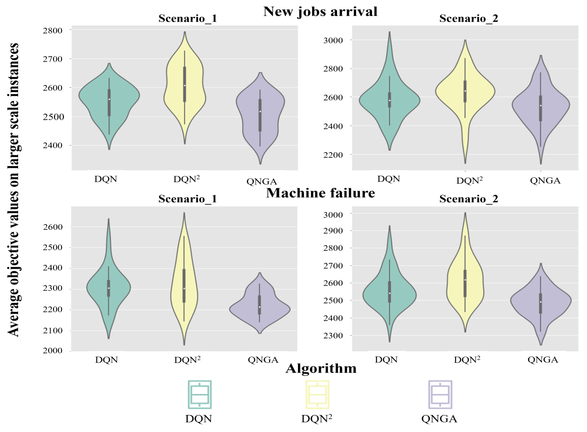


Fig. 6. Violin plot of the average objective of DQN, DQN^2, and ETDQN on the instances in dataset III over 30 independent runs in both dynamic events.

In Table V, it is clearly shown that the $mean$ obtained by ETDQN is more satisfactory than those obtained by DQN and DQN^2 in both dynamic environments. Specifically, in the presence of new job arrivals, the average makespan obtained by ETDQN can reach at least 12.42% and 10.96% improvement over DQN and DQN^2, respectively. Additionally, the average makespan of ETDQN is respectively improved by at least 4.43% and 4.38% compared to DQN and DQN^2 under the dynamic events of machine failure. Moreover, all *p-values* calculated by ETDQN with DQN and DQN^2 are much less than 0.05, which indicates the objective values obtained by ETDQN are obviously different from those calculated by DQN and DQN^2. The result shows that the decision-making ability of the proposed ETDQN algorithm is significantly improved in solving DFJSS instances with six different sizes.

Fig. 6 shows the violin plot of the makespan of ETDQN, DQN, and DQN^2, where the horizon axis denotes the comparison algorithms, and the vertical axis denotes the calculated makespan. According to Fig. 6, ETDQN achieves better performance (i.e., smaller makespan) than DQN and DQN^2 under the two dynamic events. Specifically, in the case of new job arrival, ETDQN achieves a majority of makespan distributed at lower positions compared to DQN and DQN^2. Moreover, the maximum makespan and minimum makespan obtained by ETDQN are better than those obtained by DQN and DQN^2. These results indicate that ETDQN exhibits the best overall performance compared with DQN and DQN^2. In the case of machine failure, ETDQN cannot guarantee to find the minimum makespan in any scenario, like Scenario_1. However, most obtained makespan values by ETDQN distribute at a lower position than that of DQN and DQN^2. This demonstrates ETDQN achieves the best average performance. Moreover, the majority of makespan values obtained by ETDQN in the two scenarios are relatively concentrated in distribution, which indicates the model has reliable stability. These results demonstrate that ETDQN with fixed targets obtains much better performance in solving DFJSS instances with scale of 100×10 under two dynamic events.

Table VI clearly shows that ETDQN consistently has the best average performance in spite of being confronted with DFJSS instances with scale of 2000×10. In terms of *mean*, the average performance of ETDQN compared to DQN and DQN^2 improves by 1.89% and 1.13% in the presence of new job arrival, while improves by 6.51% and 5.52% facing with machine failure, respectively. Moreover, ETDQN can obtain the minimum *std* in both dynamic events. This means that the results produced by ETDQN in different dynamic environments show more reliable stability compared to DQN and DQN^2, which is crucial in practical applications to ensure stable scheduling decisions. In addition, test time is an important criterion to measure the efficiency of algorithms. Table VI records the CPU time spent by DQN, DQN^2 and ETDQN in processing the same DFJSS instance in dataset IV. It is obvious that the computation time of ETDQN has significant differences compared with DQN and DQN^2, and ETDQN consistently achieves the shortest time in finding the highest-quality solution in both dynamic events. It is worth noting that ETDQN, trained on the small-scale DFJSS instance, demonstrates enhanced performance and efficiency compared to both DQN and DQN^2. This indicates a notable enhancement in the feature extraction capability of ETDQN with fixed targets.

TABLE VI
COMPARISON RESULTS OF MAKESPAN AND TEST TIME BETWEEN ETDQN AGAINST THE DQN AND DQN^2 ON THE DFJSS INSTANCES WITH THE SCALE OF 2000×10.

Dynamic event	criteria	DQN	DQN^2	ETDQN
new jobs arrival	*mean*	22534.78	22360.67	**22108.17**
	std	309.9249	256.1432	**166.7492**
	time	728.40	682.80	**623.40**
machine failure	*mean*	25565.10	25298.35	**23901.05**
	std	241.4742	238.4616	**222.5991**
	time	790.80	782.40	**679.81**

*: *mean* and *std* indicate mean and standard deviation of makespan.
*: *time* represents the average test time of three algorithms (unit: s).

One reason that ETDQN outperforms DQN and DQN^2 is that the training of DQN and DQN^2 is unstable due to moving targets. Specifically, the training of DQN and DQN^2 can result in moving targets due to the updates of target network. The moving target destabilizes the gradient descent direction because the targets are used to calculate the loss for training. This will inevitably deteriorate the prediction network to capture feature information. However, ETDQN uses GA to replace the target network in calculating the targets. There are two advantages. On the one hand, GA often exhibits good global search ability in practice, and can explore solution spaces through genetic operations, which can search for an approximately optimal target for each state. The target acts as the only objective for the job-shop state to guide Q-network learning more effectively. On the other hand, GA does not require parameter updates like the target network during training, This can avoid moving targets in the root and ensures that the Q-network updates towards fixed targets during the training process, thereby alleviating poor performance caused by unstable training.

Therefore, ETDQN can effectively extract state information at rescheduling points and make more appropriate decisions when facing different scheduling scenarios. In addition, a possible reason for the improved computational efficiency of ETDQN in testing is further discussed in Section V-C.

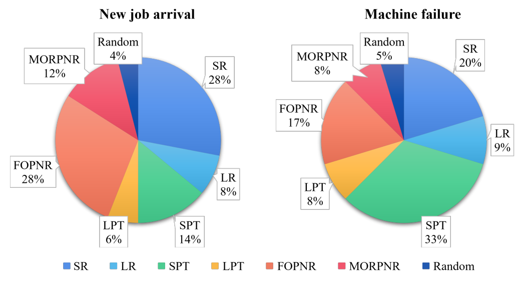


Fig. 7. Frequency distribution of each dispatching rule.

C. Further Analyses on the Selected Dispatching Rules

In this section, we first outline how the proposed ETDQN selects dispatching rules and then analyse the effects of different dynamic events and selection strategies on the sequence of dispatching rules. In the process of dispatching rules selection, the proposed ETDQN method computes the Q-values of dispatching rules (described in Section III-B) by extracting the state features of the job shop, and the one with the maximum Q-value is selected. Then, the next operations are assigned to the idle machines by executing the selected dispatching rule. This process is continued until all operations in the job shop have been assigned. To facilitate the discussion, we record the selected dispatching rules. Table VII documents the number of the selected dispatching rules, and Fig. 7 shows the distribution of the selection frequency for each dispatching rule by DTDQN in both dynamic events.

In Fig. 7, it can be seen that SR and FOPNR are selected more frequently than other dispatching rules when new jobs arrive. One possible reason is that the trained Q-network, based on the analysis of workshop state features, identifies the effectiveness of these rules. Specifically, the Q-network gradually learns to evaluate the performance of the dispatching rules in different job-shop states during the training process. The process is achieved through interaction with the workshop environment, observing decision outcomes in various states, and receiving rewards. When new jobs arrive, the Q-network tends to select rules that minimize the time required to complete a job. The decision is based on the learned knowledge and evaluation of the job-shop state. SR and FOPNR are designed to minimize the processing time of a job, reducing the potential impact of the new job on the progress of other ongoing jobs and ensuring fast completion.

In the case of machine failure, it can be seen from Fig. 7 that SPT is selected with the highest frequency compared with other dispatching rules. It is not hard to find that the processing time is given more attention in this case. This might be because shorter processing times allow operations to be completed before a failure disrupts the entire production process, mitigating the impact on workflow. Moreover, the short processing time means that the machine can be released quickly and put back into the following application. This allows resources to be used more frequently and efficiently, mitigating the resource competition due to machine damage.

Another insightful observation from Fig. 7 is that *Random* is always selected with the smallest frequency in both dynamic events. This is because *Random* lacks insight into the dependencies between operation priorities and dynamic environments. The feature prevents it from making decisions that mitigate the intricate impact of dynamic events, rendering it support for the nuanced demands of dynamic job-shop scheduling. This also underscores the significance of the well-designed dispatching rules in efficiently addressing scheduling challenges from other perspectives.

TABLE VII
TIMES OF SELECTING DISPATCHING RULES BETWEEN ETDQN AGAINST THE COMPETITORS IN NEW JOB ARRIVAL OR MACHINE FAILURE.

Algorithms	new job arrival	machine failure
RP	63	79
DQN	59	72
DQN^2	57	69
DDQN	55	66
DLDQN	55	64
ETDQN	**52**	**61**

Table VII records the times of the selecting dispatching rules using different algorithms. For example, in the case of new job arrivals, ETDQN requires 52 scheduling rule selections to complete a scheduling instance. According to Table VII, the times of the selecting dispatching rules by ETDQN are always the fewest compared to other heuristic selection methods in both dynamic events. The result indicates that each dispatching rule selected by ETDQN is better suited for the current state, enabling the scheduling task to be completed more efficiently. Specifically, ETDQN can learn the state features of the job shop more efficiently, allowing ETDQN to have a powerful ability to select dispatching rules. These selected dispatching rules can effectively increase the machine utilization and thus the efficiency of processing DFJSS instances, which has been demonstrated in Section V-A and V-B. Conversely, if the selected dispatching rules are not suitable for the current state, it may not be able to allocate operations to a machine, causing the machine to remain idle. As a result, additional complexity can introduced to the scheduling process, resulting in more dispatching rules needing to be selected to handle all the demanded jobs. This also explains why the times of the dispatching rules selected by ETDQN, DDQN, DQLDQN, DQN, and DQN^2 based on the learning method are less than that of RP based on the random strategy. In addition, it can also be noticed that this difference is more pronounced in the case of machine failure, which is because effective dispatching rule selection results in a shorter makespan, reducing the scheduling complexity caused by the machine failure.

VI. Conclusions and Future Work

The goal of this paper is to train an effective Q-network by GA for heuristic selection in DFJSS. This goal has been achieved by proposing an evolutionary trainer based on GA. The proposed encoding strategy can flexibly encode the schedule at decision points concisely and effectively. The proposed fitness evaluation strategy can guide the population to search for the ‘optimal’ targets used for training the Q-network. The results show that the evolutionary trainer-based deep Q-network has learned to correctly decide on appropriate dispatching rules in the different rescheduling states. Specifically, the proposed ETDQN achieves the minimum makespan on most test instances compared with peer competitors under the dynamic events with new jobs arrival or machine failure. The comparison of the performance of ETDQN, DQN, and DQN^2 shows that the scheduling solution obtained by ETDQN can take less processing time in the randomly generated DFJSS scenario with a larger scale than the training data. Moreover, the superiority of ETDQN is verified by comparing it with other well-known heuristic selection policies in multiple unseen problem instances.

Some interesting directions can be further investigated in the future. Using the disjunction graph to represent states results in a complicated graph, which adversely impacts computational efficiency when solving scheduling tasks of large scales, such as thousands or more. It is worth studying to design an appropriate graph representation approach, which can present sufficient information while maintaining computational efficiency. Moreover, although this work has shown that the performance of existing scheduling heuristics based on simple single rules is limited, this does not mean that a single constructed rule with careful design can not work well. Therefore, it is necessary to further investigate the single-constructed rules by fully considering the trade-offs in scheduling.

REFERENCES

[1] S. C. Liu, Z. G. Chen, Z. H. Zhan, S. W. Jeon, S. Kwong, and J. Zhang, "Many-objective job-shop scheduling: a multiple populations for multiple objectives-based genetic algorithm approach," *IEEE Transactions on Cybernetics*, vol. 53, no. 3, pp. 1460–1474, 2023.

[2] 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*, vol. 19, no. 2, pp. 1600–1610, 2023.

[3] 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, 2021.

[4] 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, 2024.

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

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

[7] J. Park, Y. Mei, S. Nguyen, G. Chen, and M. Zhang, "Investigating a machine breakdown genetic programming approach for dynamic job shop scheduling," in *Proceedings of Genetic Programming: 21st European Conference*, 2018, pp. 253–270.

[8] Z. Bao, L. Chen, and K. Qiu, "An aircraft final assembly line balancing problem considering resource constraints and parallel task scheduling," *Computers & Industrial Engineering*, vol. 182, p. 109436, 2023.

[9] 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*, early access 2023, doi: 10.1109/TEVC.2023.3244607.

[10] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Collaborative multifidelity-based surrogate models for genetic programming in dynamic flexible job shop scheduling," *IEEE Transactions on Cybernetics*, vol. 52, no. 8, pp. 8142–8156, 2022.

[11] H. Chen, C. Chu, and J.-M. Proth, "An improvement of the lagrangean relaxation approach for job shop scheduling: a dynamic programming method," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 5, pp. 786–795, 1998.

[12] I. Chaouch, O. B. Driss, and K. Ghédira, "A novel dynamic assignment rule for the distributed job shop scheduling problem using a hybrid ant-based algorithm," *Applied Intelligence*, vol. 49, pp. 1903–1924, 2018.

[13] T. Ning, M. Huang, X. Liang, and H. Jin, "A novel dynamic scheduling strategy for solving flexible job-shop problems," *Journal of Ambient Intelligence and Humanized Computing*, vol. 7, pp. 721–729, 2016.

[14] P. Brucker, B. Jurisch, and B. Sievers, "A branch and bound algorithm for the job-shop scheduling problem," *Discrete Applied Mathematics*, vol. 49, no. 1-3, pp. 107–127, 1994.

[15] M. Nouiri, A. Bekrar, A. Jemai, S. Niar, and A. C. Ammari, "An effective and distributed particle swarm optimization algorithm for flexible job-shop scheduling problem," *Journal of Intelligent Manufacturing*, vol. 29, pp. 603–615, 2018.

[16] I. Pergher and A. T. de Almeida, "A multi-attribute, rank-dependent utility model for selecting dispatching rules," *Journal of Manufacturing Systems*, vol. 46, pp. 264–271, 2018.

[17] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems.* JSTOR, 1994.

[18] Y. H. Lee, K. Bhaskaran, and M. Pinedo, "A heuristic to minimize the total weighted tardiness with sequence-dependent setups," *IIE Transactions*, vol. 29, no. 1, pp. 45–52, 1997.

[19] Y.-R. Shiue, K.-C. Lee, and C.-T. Su, "Real-time scheduling for a smart factory using a reinforcement learning approach," *Computers & Industrial Engineering*, vol. 125, pp. 604–614, 2018.

[20] Y. H. 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.

[21] Z. Zhuang, Y. Li, Y. Sun, W. Qin, and Z.-H. Sun, "Network-based dynamic dispatching rule generation mechanism for real-time production scheduling problems with dynamic job arrivals," *Robotics and Computer-Integrated Manufacturing*, vol. 73, p. 102261, 2022.

[22] V. Subramaniam, G. Lee, G. Hong, Y. Wong, and T. Ramesh, "Dynamic selection of dispatching rules for job shop scheduling," *Production planning & control*, vol. 11, no. 1, pp. 73–81, 2000.

[23] H. Zhang and U. Roy, "A semantics-based dispatching rule selection approach for job shop scheduling," *Journal of Intelligent Manufacturing*, vol. 30, pp. 2759–2779, 2019.

[24] Y. He, L. Xing, Y. Chen, W. Pedrycz, L. Wang, and G. Wu, "A generic markov decision process model and reinforcement learning method for scheduling agile earth observation satellites," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 52, no. 3, pp. 1463–1474, 2022.

[25] S. S. Mousavi, M. Schukat, and E. Howley, "Deep reinforcement learning: an overview," in *Proceedings of SAI Intelligent Systems Conference*, 2018, pp. 426–440.

[26] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[27] A. Pich'e, V. Thomas, J. Marino, G. M. Marconi, C. J. Pal, and M. E. Khan, "Beyond target networks: improving deep q-learning with functional regularization," *ArXiv: 2106.02613*, 2021.

[28] J. F. Hernandez-Garcia and R. S. Sutton, "Understanding multi-step deep reinforcement learning: A systematic study of the dqn target," *ArXiv: 1901.07510*, 2019.

[29] L. Graesser and W. Keng, "Foundations of deep reinforcement learning." Pearson Education, 2019.

[30] J.-D. Zhang, Z. He, W.-H. Chan, and C.-Y. Chow, "Deepmag: Deep reinforcement learning with multi-agent graphs for flexible job shop scheduling," *Knowledge-Based Systems*, vol. 259, p. 110083, 2023.

[31] 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.

[32] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016, p. 2094–2100.

[33] H. Hasselt, "Double Q-learning," in *Advances in Neural Information Processing Systems*, vol. 23, 2010.

[34] S. Kim, K. Asadi, M. L. Littman, and G. D. Konidaris, "Deepmellow: removing the need for a target network in deep Q-learning," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, 2019, pp. 2733–2739.

[35] D. Heiss-Czedik, "An introduction to genetic algorithms," *Artificial Life*, vol. 3, no. 1, pp. 63–65, 1997.

[36] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.

[37] A. Slowik and H. Kwasnicka, "Evolutionary algorithms and their applications to engineering problems," *Neural Computing and Applications*, vol. 32, pp. 12 363–12 379, 2020.

[38] I. Lee, R. Sikora, and M. Shaw, "A genetic algorithm-based approach to flexible flow-line scheduling with variable lot sizes," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 27, no. 1, pp. 36–54, 1997.

[39] 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.

[40] E. K. Burke, M. Gendreau, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, "Hyper-heuristics: a survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, pp. 1695–1724, 2013.

[41] F. Zhang, Y. Mei, S. Nguyen, K. C. Tan, and M. Zhang, "Instance-rotation-based surrogate in genetic programming with brood recombination for dynamic job-shop scheduling," *IEEE Transactions on Evolutionary Computation*, vol. 27, no. 5, pp. 1192–1206, 2023.

[42] N. R. Sabar, M. Ayob, G. Kendall, and R. Qu, "A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems," *IEEE Transactions on Cybernetics*, vol. 45, no. 2, pp. 217–228, 2015.

[43] J. A. V. Rodríguez and A. Salhi, "A robust meta-hyper-heuristic approach to hybrid flow-shop scheduling," in *Proceedings of Evolutionary Scheduling*, 2007, pp. 125–142.

[44] Y. Du, J.-q. Li, X.-l. Chen, P.-y. Duan, and Q.-k. Pan, "Knowledge-based reinforcement learning and estimation of distribution algorithm for flexible job shop scheduling problem," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 7, no. 4, pp. 1036–1050, 2023.

[45] D. Oliva and M. S. R. Martins, "A bayesian based hyper-heuristic approach for global optimization," in *Proceedings of 2019 IEEE Congress on Evolutionary Computation*, 2019, pp. 1766–1773.

[46] S. Luo, L. Zhang, and Y. Fan, "Real-time scheduling for dynamic partial-no-wait multiobjective flexible job shop by deep reinforcement learning," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 4, pp. 3020–3038, 2022.

[47] 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, 2024.

[48] Y. Zeng, Z. Liao, X. Li, and B. Yuan, "You only train once: A highly generalizable reinforcement learning method for dynamic job shop scheduling problem," early access 2022, doi: 10.36227/techrxiv.20324070.

[49] M. L. Puterman, "Markov decision processes," in *Handbooks in Operations Research and Management Science*. Amsterdam, North-Holland: Elsevier, 1990, pp. 331–434.

[50] R. Bellman, R. E. Kalaba *et al.*, *Dynamic programming and modern control theory*. Citeseer, 1965, vol. 81.

[51] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv:1312.5602*, 2013.

[52] D. Johnson, G. Chen, and Y. Lu, "Multi-agent reinforcement learning for real-time dynamic production scheduling in a robot assembly cell," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 7684–7691, 2022.

[53] 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.

[54] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2021.

[55] P. Zheng, L. Xia, C. Li, X. Li, and B. Liu, "Towards self-x cognitive manufacturing network: An industrial knowledge graph-based multi-agent reinforcement learning approach," *Journal of Manufacturing Systems*, vol. 61, pp. 16–26, 2021.

[56] Q. Xiao, B. Niu, B. Xue, and L. Hu, "Graph convolutional reinforcement learning for advanced energy-aware process planning," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 53, no. 5, pp. 2802–2814, 2022.

[57] C. Su, C. Zhang, D. Xia, B. Han, C. Wang, G. Chen, and L. Xie, "Evolution strategies-based optimized graph reinforcement learning for solving dynamic job shop scheduling problem," *Applied Soft Computing*, vol. 145, p. 110596, 2023.

[58] X. Li and X. Liang, "Research on flexible job shop scheduling problem," *Journal of Physics: Conference Series*, vol. 1549, no. 3, p. 032126, 2020.

[59] G. Ning and D. Cao, "Multi-step genetic algorithm for solving dynamic flexible job shop scheduling problem," in *Proceedings of 2021 IEEE International Conference on Advances in Electrical Engineering and Computer Applications*, 2021, pp. 23–28.

[60] L. He, R. Chiong, W. Li, S. Dhakal, Y. Cao, and Y. Zhang, "Multiobjective optimization of energy-efficient job-shop scheduling with dynamic reference point-based fuzzy relative entropy," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 1, pp. 600–610, 2021.

[61] S. Zhang, H. Tong, J. Xu, and R. Maciejewski, "Graph convolutional networks: a comprehensive review," *Computational Social Networks*, vol. 6, no. 1, pp. 1–23, 2019.

[62] A. Vaswani, N. M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *ArXiv: 1706.03762*, 2017.

[63] J. Li, C. Wang, X. Fang, K. Yu, J. Zhao, X. Wu, and J. Gong, "Multi-label text classification via hierarchical transformer-cnn," in *Proceedings of 2022 14th International Conference on Machine Learning and Computing*, 2022, pp. 120–125.

[64] D. Jurafsky and J. H. Martin, "Speech and language processing - an introduction to natural language processing, computational linguistics, and speech recognition," in *Prentice Hall series in artificial intelligence*, 2000.

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

[66] L. Zhao, J. Fan, C. Zhang, W. Shen, and J. Zhuang, "A drl-based reactive scheduling policy for flexible job shops with random job arrivals," *IEEE Transactions on Automation Science and Engineering*, early access 2023, doi: 10.1109/TASE.2023.3271666.

[67] 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.

[68] S. Yang, J. Wang, and Z. Xu, "Real-time scheduling for distributed permutation flowshops with dynamic job arrivals using deep reinforcement learning," *Advanced Engineering Informatics*, vol. 54, p. 101776, 2022.

[69] N. Chen, N. Xie, and Y. Wang, "An elite genetic algorithm for flexible job shop scheduling problem with extracted grey processing time," *Applied Soft Computing*, vol. 131, p. 109783, 2022.

Yun Liu received her M.E. degree in Computer Science from Wenzhou University, Wenzhou, China, in 2022. She is currently pursuing a Ph.D. degree in Computer Science from Sichuan University, Chengdu, China. Her current research interests include evolutionary computation, job shop scheduling, and deep reinforcement learning.

Fangfang Zhang (Member, IEEE) Fangfang Zhang is a postdoctoral research fellow in the Center for Data Science and Artificial Intelligence & School of Engineering and Computer Science, Victoria University of Wellington, New Zealand. Her research interests include evolutionary computation and job shop scheduling, surrogate. She is an Associate Editor of Expert Systems With Applications, and Swarm and Evolutionary Computation. She is the Secretary of IEEE New Zealand Central Section.

Yanan Sun (Senior Member, IEEE) is a Professor at the College of Computer Science at Sichuan University, China. His research focuses on evolutionary computation, neural networks, and their applications. He is a recipient of national yong talent plan. He is the founding chair of the IEEE CIS Task Force on evolutionary deep learning and applications. He is an Associate Editor of IEEE Transactions on Evolutionary Computation, and IEEE Transactions on Neural Networks and Learning Systems.

Mengjie Zhang (Fellow, IEEE) received the B.E. and M.E. degrees from 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 the Center for Data Science and Artificial Intelligence & School of Engineering and Computer Science, Victoria University of Wellington, New Zealand. He is a Fellow of the Royal Society and a Fellow of Engineering New Zealand, a Fellow of IEEE, and an IEEE Distinguished Lecturer.