Int j simul model 20 (2021) 2, 375-386
Original scientific paper

A DEEP REINFORCEMENT LEARNING BASED SOLUTION FOR FLEXIBLE JOB SHOP SCHEDULING PROBLEM

Han, B. A. & Yang, J. J.
Department of Industrial and Manufacturing Systems Engineering, Beihang University, Beijing 100191, China
E-Mail: hanbaoan@buaa.edu.cn, jjyang@buaa.edu.cn

Abstract
Flexible job shop Scheduling problem (FJSP) is a classic problem in combinatorial optimization and a very common form of organization in a real production environment. Traditional approaches for FJSP are ill-suited to deal with complex and changeable production environments. Based on 3D disjunctive graph dispatching, this work proposes an end-to-end deep reinforcement learning (DRL) framework. In this framework, a modified pointer network, which consists of an encoder and a decoder, is adopted to encode the operations to be scheduled according to the selected scheduling features. Then with the attention mechanism, an input is pointed as an action in each decoding step, and a recurrent neural network (RNN) is used to model the decoder network. To train the network to minimize the makespan, a policy gradient algorithm is applied to optimize its parameters. The trained model generates the scheduling solution as a sequence of consecutive actions in real-time without retraining for every new problem instance. Experimental results show that this method can obtain better performance than the classic heuristic rules when only one model is trained on all the test instances.
(Received in December 2020, accepted in March 2021. This paper was with the authors 1 month for 2 revisions.)

Key Words: Flexible Job Shop Scheduling Problem (FJSP), Deep Reinforcement Learning (DRL), End-to-End, Pointer Network, Attention Mechanism, 3D Disjunctive Graph

1. INTRODUCTION

FJSP is an extension of the traditional JSP, it is more intractable since each operation can be assigned on one or more available machines [1]. At present, most studies assume a static manufacturing environment, where all the workshop information is determined in advance, hence outputting a deterministic schedule without any modification during its execution. However, in today's changeable manufacturing systems, dynamic events such as insertions or modifications of orders, machine failures, variations in processing times and so on, will lead to a deviation far from the original plan, thus seriously affecting the production efficiency.

FJSP has been widely studied over the past decades and various methods have been proposed. Heuristic rules [2], can respond to dynamic events instantly, but because of their short-sightedness, it is difficult to guarantee optimal solutions and scheduling performance varies in different scheduling environments. Meta-heuristic algorithms usually search for scheduling solutions iteratively by evolutionary operators or particle movements, such as genetic algorithm (GA) [3] and particle swarm optimization (PSO) [4]. Although such methods can obtain high-quality solutions, they cannot meet the real-time requirements due to a long time of iterative optimization. Moreover, once the problem structure changes, such methods need to be redesigned with poor universality. Because FJSP is a NP-hard problem [5], for which it is very difficult to find the optimal solution, in the actual environment, the algorithm gives up finding the optimal solution and instead tries to find an approximate feasible solution within a reasonable time. If a scheduling solution can be constructed automatically according to the current scheduling state at different decision steps, the efficiency and quality can be guaranteed at the same time, which is the main motivation of this study.

Reinforcement learning (RL) [6] does not require a complete mathematical model of the learning environment, and adjusts the learning strategy through the evaluative reward obtained

through interaction with the environment. Traditional tabular RL is often used to solving small-scale problems with discrete state space. However, realistic RL tasks always face a continuous state space in which there are infinite states. At this time, value function approximation shall be considered. Deep Reinforcement Learning (DRL) [7, 8] is a brand new algorithm that combines deep learning (DL) and RL to realize end-to-end learning from perception to action. Since the 90s, (D)RL has been used to solve the scheduling problem [9-14]. Most of these studies selected the appropriate heuristic rules according to the scheduling states. Different RL algorithms were used for training to adaptively select scheduling strategies. The scheduling results can usually be obtained in a short time and were superior to ordinary heuristic rules. However, this type of RL essentially runs in the heuristic search space rather than the solution search space, so the solution quality still depends on the selected heuristic, which has been proved by our previous research [6].

In recent years, with more and more extensive applications in natural language, the sequence-to-sequence model combines with DRL to give rise to a new method to solve combinatorial optimization problems [15, 16]. These end-to-end methods take a given problem instance as the input and use the trained deep neural network (DNN) to directly output the solution, instead of indirectly constructing the solution by selecting heuristic rules. They have the advantages of fast solving speed and strong generalization ability. Since it is difficult to obtain the optimal solution for most combinatorial optimization problems, the supervised learning method that requires a large number of labelled samples for training is intractable to apply in practice [17]. Therefore, most of the current studies use DRL to train the model [18, 19].

With the motivations above, we utilize an end-to-end DRL approach to solve FJSP to minimize makespan. The contributions of this paper can be listed as follows: (1) An end-to-end model-based DRL scheduling framework was presented, in which a modified pointer network and attention mechanism were adopted. To our knowledge, this is the first attempt to solve FJSP using this framework. (2) The static and dynamic representation features were constructed for scheduling from the perspective of the overall, task and machine respectively, and were input into the model by a specific combination so that the model was able to determine job and machine with the highest priority simultaneously. (3) A single model can be trained to find near-optimal solutions for problem instances sampled from a given distribution, only by observing the reward signals.

2. END-TO-END DRL SCHEDULING FRAMEWORK

Our contribution is to propose an adaptive scheduling framework that combines an end-to-end DRL and a 3D disjunctive graph. The proposed framework includes three parts: scheduling environment, offline learning, and online application (Fig. 1).

The scheduling environment is modelled with the 3D disjunctive graph. The disjunctive graph-based scheduling solution is to first initialize the ready task set, from which the job with the highest priority is dispatched to the machine with the highest priority based on the agent's action. Then the job is removed from the constraint network and its subsequent task is added to the ready task set. This process is repeated until the ready task set is empty and the scheduling result is obtained. Such a complete process is called an episode.

In the offline learning phase, the state x_t is input into a critic network and actor network to output a baseline and action respectively. The policy gradient algorithm is used to learn a stochastic strategy π with the parameter θ.

Although it takes a long time to train during the learning phase, once the optimal strategy is learned, in the online application phase it can be applied to new scheduling problems, and optimal results can be obtained in a short time.

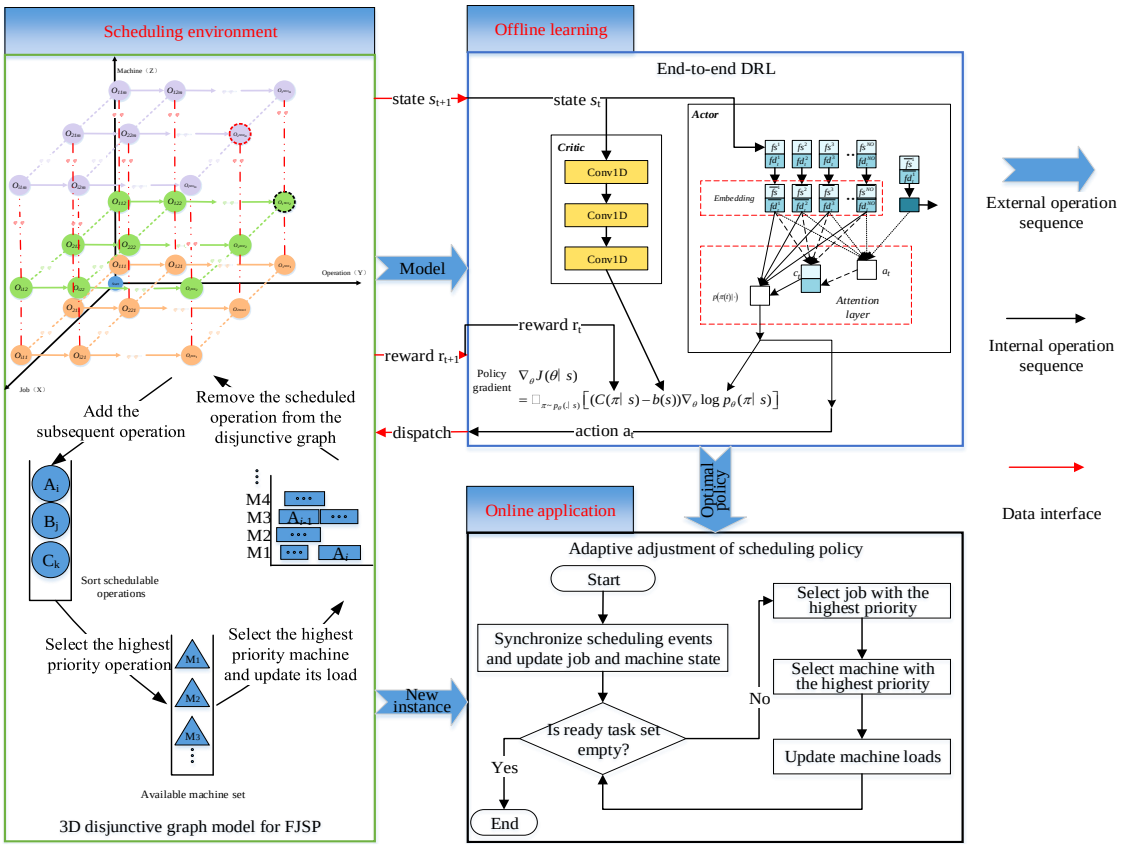


Figure 1: Scheduling framework with DRL including scheduling environment, offline learning, and online application.

3. SCHEDULING ENVIRONMENT

3.1 Problem description

The FJSP can be described as follows. There are NJ jobs to be processed on NM machines. Each job J_i consists of NO_i operations where O_{ih} is the h^{th} operation of J_i. Each operation O_{ih} can be processed on any machine M_m selected from an available machine set $\boldsymbol{M}_{ih}$. The processing time of operation O_{ih} on machine M_m is denoted as P_{ihm}. The goal is to minimize the maximum completion time for all jobs, while satisfying the following constraints:

(1) Each machine can process only one job at a time (capacity constraint).

(2) All operations belonging to the same job should be processed in a predetermined order (precedence constraint).

(3) Once the operation has started, no interruption is allowed.

(4) Transportation times and setup times are negligible.

The notations and indices for various parameters used are given in Table I.

Table I: Notations and indices for various parameters.

Indices	
i, j	Index of jobs, $i, j = 1, 2, 3, \ldots, NJ$
h, l	Index of operations for a job, $h = 1, 2, 3, \ldots, NO_i$, $l = 1, 2, 3, \ldots, NO_j$
o	Index of all operations, $o = 1, 2, 3, \ldots, NO$
m, n, r	Index of machines, $m, n, r = 1, 2, 3, \ldots, NM$
Parameters	
$\boldsymbol{J}$	Job set
$\boldsymbol{O}$	Operation set
$\boldsymbol{O}_m$	Set of operations assigned to machine M_m

$\boldsymbol{O}_{\text{ready}}$	Ready operation set
$\boldsymbol{M}$	Machine set
$\boldsymbol{M}_{ih}$	Available machine set for operation O_{ih}
NJ	Total number of jobs
NO	Total number of all operations
NO_i	Total number of operations belonging to job J_i
NM	Total number of machines
NM_{ih}	Total number of available machines for operation O_{ih}
O_{ih}	h^{th} operation of J_i
P_{ihm}	Processing time of O_{ih} on machine M_m
S_{ihm}	Start time of O_{ih} on machine M_m
C_{ihm}	Completion time of O_{ih} on machine M_m
C_i	Completion time of job J_i
L	A large positive number
Variables	
x_{ihm}	Binary variable that is equal to 1 if machine M_m is selected for O_{ih} and 0 otherwise
y_{ihjlm}	Binary variable that is equal to 1 if O_{ih} is processed before O_{jl} on machine M_m and 0 otherwise

The mathematical representation of the FJSP is formulated as follows.

$$\text{Minimize } \sum_{i=1}^{n} max\{C_i\} \tag{1}$$

$$s.t. \begin{cases} C_i \geq 0, C_{ihm} \geq 0, \forall i,j,m & (a) \\ \sum_{m \in \boldsymbol{M}_{ih}} x_{ihm} = 1, \forall i,h & (b) \\ C_{ihm} \leq S_{i(h+1)n, i \in [1,NJ], h \in [1,NO_i - 1]} & (c) \\ S_{ihm} + P_{ihm} \leq S_{jlm} + L(1 - y_{ihjlm}), \forall i,h,m,j,l & (d) \end{cases} \tag{2}$$

Objective (1) is to minimize the maximum completion time of all jobs. Eq. (2)(a) indicates that the completion time of each job and operation must be non-negative. Eq. (2)(b) signifies that each operation can only be assigned on one machine. Eq. (2)(c) ensures that the latter operation can only be processed after the former operation has been finished. Eq. (2)(d) guarantees that at most one process can be processed at a time on each machine.

3.2 3D disjunctive graph model

To express the FJSP, this paper proposes a 3D disjunctive graph model that can be expressed by $G=(V \cup V', C \cup D \cup E)$, where V denotes all operations O_{ihm} of jobs that can be processed on machine M_m, V' denotes the operations $O_{i'h'm'}$ that do not exist or cannot be processed on machine $M_{m'}$, C denotes a set of directed solid lines which represent the precedence constraints between every two consecutive operations of the same job, D denotes a set of undirected dotted lines connecting mutually unordered tasks executed on the same machine, E denotes a set of double-dotted lines meaning that each operation can only be processed on one machine.

According to the above description, the three-dimensional general scheduling model for FJSP as shown in Fig. 2 was established.

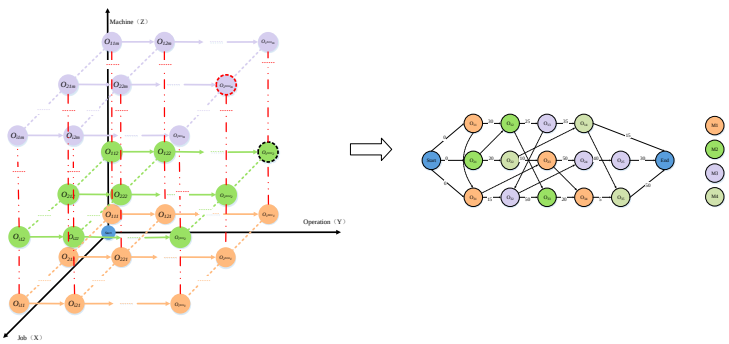


Figure 2: 3D disjunctive graph model.

When solving with this model, first determine the machine for each operation, and then the three-dimensional model is reduced to a two-dimensional one.

4. SOLVING FJSP BASED ON END-TO-END DRL

In this section, we formally define the problem and our proposed framework for FJSP with a given set of input operations $\boldsymbol{O}$. Given an input 3D disjunctive graph, represented as a sequence of NO operations in a d-dimensional space $SF = \{f_o, o \in \boldsymbol{O}\}$, where $f_o = \{f_1, \cdots, f_d\} \in \mathbb{R}^d$, each input operation O can be represented by a sequence of feature tuples $\{f_o^t \doteq (f_{S_o}, fd_o^t),\ t = 0, 1, \ldots\}$, where f_{S_o} and fd_o^t are the static and dynamic features of the input operation, respectively. SF_t denotes the set of all input states at a fixed time t. We are concerned with finding a permutation of the input operations π that has the minimum makespan.

We start from an arbitrary input in SF_0, where the pointer $\pi(0)$ is used to refer to that input. At each decoding time $t(t = 0, \cdots, NO - 1)$, $\pi(t+1)$ points to one of the available inputs SF_t to determine the input of the next decoding step; The process is repeated until all operations are scheduled and will generate a sequence π of length NO. The goal is to find a stochastic policy $p(\pi \mid SF_0)$ that generates the sequence π in a way that minimizes the makespan while satisfying the constraints. The neural network architecture uses the probability chain rule to factorize the probability of generating sequence π as:

$$p(\pi \mid SF_0) = \prod_{t=0}^{NO-1} p(\pi(t) \mid \pi(< t), SF_t) \tag{3}$$

and

$$SF_{t+1} = F(\pi(t), SF_t) \tag{4}$$

is a recursive update of the scheduling representation with the state transition function F. Each term on the RHS of Eq. (3) is computed by the attention mechanism, i.e.,

$$p(\pi(t) \mid \pi(< t), SF_t) = \text{softmax}\left(g(h, SF_t)\right) \tag{5}$$

where g is an affine function that outputs an input-sized vector, and h_t is the state of the RNN decoder, which summarizes the information of all previous decoding steps $\pi(< t)$.

4.1 State, action, and reward

We propose the dynamic and static features describing the scheduling states from the perspectives of overall, task, and machine, as shown in Table II. The overall information, which often is in the form of proportion, average, standard deviation, etc., is used to describe the full picture of the scheduling problem. Task information reflects the job and operation related attributes of each task and the Max-Min scaling is adopted to normalize among different tasks. Machine information describes the processable information and scheduled information of each machine, and the Max-Min scaling is also used for normalization.

Most RL selects problem-specific rules as the execution actions according to the system states, but limited heuristic rules can always not completely cover various sorting results. A remedy is to select an operation from the ready tasks set at each decision step based on the disjunctive graph and each selection corresponds to an action. Since there are precedence constraints between different operations of the same job, effective actions can only be selected from the ready tasks set with a new masking scheme to set the log-probabilities of selecting any operation outside the ready tasks set to $-\infty$. As mentioned earlier, in addition to the operation sequencing, there is also a problem of machine assignment. For this reason, we argue that the operation is different when processed on different machines so that the total number of operations in the scheduling problem will change from $\sum_{i=1}^{NJ} NO_i$ to $\sum_{i=1}^{NJ} \sum_{h=1}^{NO_i} NM_{ih}$.

Table II: Scheduling features.

Index	Object	Attribute	Value
f_1	whole	static	Std. of job P.T./average job P.T.
f_2			Std. of O.N. of job/average O.N. of job
f_3			Std. of operation P.T./average operation P.T.
f_4			Std. of O.N. available for machine/average O.N. available for machine
f_5			Std. of P.T. available for machine/average P.T. available for machine
f_6		dynamic	Total remaining time/total P.T.
f_7			Average R.T. of job/total P.T.
f_8			Average M.U.
f_9			Std. of M.U./average M.U.
f_{10}			Std. of M.L./average M.L.
f_{11}			SM x M.L./average M.L.
f_{12}			Min. M.L./average M.L.
f_{13}	task	static	Total P.T. of job
f_{14}			Total O.N. of job
f_{15}			Average operation P.T. of job
f_{16}			Std. of operation P.T. of job
f_{17}			Average M.N. of each operation of job
f_{18}			All the M.N. of job
f_{19}			Operation P.T.
f_{20}			Operation P.T./job P.T.
f_{21}			Number of pre-operations
f_{22}			Total P.T. of pre-operations
f_{23}			Number of post-operations
f_{24}			Total P.T. of post-operations
f_{25}			Available M.N.
f_{26}		dynamic	Total remaining P.T. of job
f_{27}			Total remaining O.N. of job
f_{28}			Average P.T. of remaining operations
f_{29}			Std. of P.T. of remaining operations
f_{30}			Average M.N. of remaining operations of job
f_{31}			All the M.N. of remaining operations of job
f_{32}			Early start of operation
f_{33}			Average M.U. of available machines
f_{34}			Average O.N. assigned for available machines
f_{35}			Std. of O.N. assigned for available machines
f_{36}			Average P.T. assigned for available machines
f_{37}			Std. of P.T. assigned for available machines
f_{38}			Max. completion time of available machines
f_{39}	machine	static	P.T. available for machine
f_{40}			O.N. available for machine
f_{41}		dynamic	Current M.L.
f_{42}			O.N. assigned
f_{43}			M.U.
f_{44}			Max. completion time

Remark: P.T. denotes processing time, O.N. the number of operations, R.T. remaining time, M.U. and M.L. machine utilization and loads.

4.2 Network model

The Pointer Network method can be summarized as using a neural network model to achieve sequence-to-sequence mapping. The core idea is to use an encoder to encode the input sequence to obtain a feature vector, and then use a decoder to combine an attention mechanism to construct the solution in an autoregressive way. RNNs are necessary only when the inputs transfer sequential information. But for combinatorial optimization problems like FJSP, any random permutation of operations contains the same information as the original inputs and has no influence on the final scheduling result. Therefore, we simply discard the encoder RNN and directly replace the RNN hidden states with the embedded inputs.

As illustrated in Fig. 3, the model contains two main components. The first is an embedding set, which maps the inputs into a *D*-dimensional vector space. The second component is a decoder, which points to an input at each decoding step. Here, GRU RNN is used to model the decoder network.

An attention mechanism is a differentiable structure used to address different parts of the input. Fig. 3 illustrates the employed attention mechanism. At each decoding step *t*, the context-based attention mechanism with a glimpse is utilized to extract relevant information from the inputs using a variable-length alignment vector a_t. Generally speaking, a_t specifies how relevant each input data point is in the next decoding step *t*.

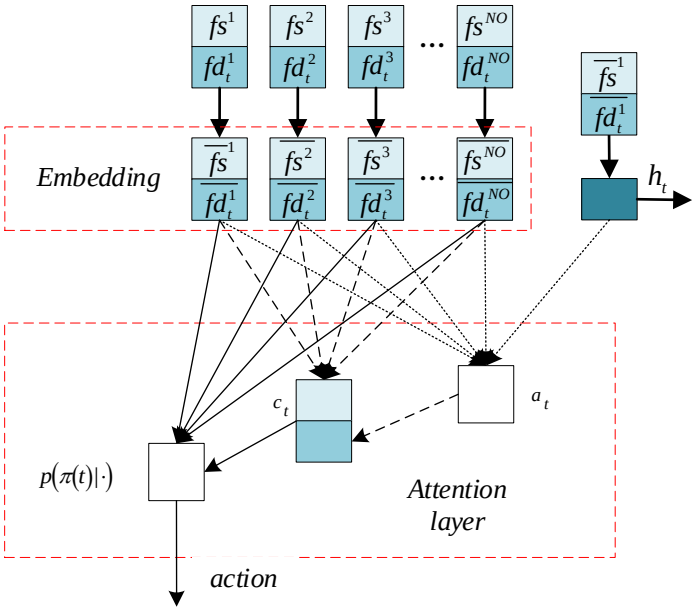


Figure 3: Our adopted model.

Let $\overline{f_t^o} = (\overline{fs^o}, \overline{fd_t^o})$ be the embedded input *o*, and $h_t \in \mathbb{R}^D$ be the memory state of the RNN cell at decoding step *t*. The alignment vector a_t is calculated as:

$$a_t = a_t(\overline{f_t^0}, h_t) = \text{softmax}\,(u_t), \text{ where } u_t^o = v_a^T \tanh\left(W_a[\overline{f_t^0}; h_t]\right) \tag{6}$$

Here ";" represents the concatenation of two vectors. The context vector c_t essentially computes a linear combination of the embedded inputs $\overline{f_t^0}$ weighted by the attention probability a_t^o, as follows:

$$c_t = \sum_{i=1}^{M} a_t^o \overline{f_t^o} \tag{7}$$

We compute the conditional probabilities by combining the context vector c_t with the embedded inputs, and then normalizing the values with the softmax function, as following:

$$P(\pi(t) \mid \pi(<t), SF_t) = \text{softmax}\,(\tilde{u}_t^o), \text{ where } \tilde{u}_t^o = v_c^T \tanh\left(W_c[\overline{f_t^o}; c_t]\right) \tag{8}$$

In Eqs. (6) to (8), v_a, v_c, W_a and W_c are all trainable variables.

4.3 Training method

An RL method using model-free policy is proposed to optimize the parameters $\boldsymbol{\theta}$ of a pointer network. The training objective is the expected makespan for a given input graph s, which is defined as:

$$J(\boldsymbol{\theta} \mid s) = \mathbb{E}_{\pi \sim p_\theta(\cdot|s)} C(\pi \mid s) \tag{9}$$

During training, the graphs are drawn from a distribution $\mathcal{S}$, and the overall training objective involves sampling from the distribution of graphs, i.e. $J(\boldsymbol{\theta}) = \mathbb{E}_{s \sim \mathcal{S}} J(\boldsymbol{\theta} \mid s)$.

We utilize policy gradient methods and stochastic gradient descent to optimize the network parameters. Using the REINFORCE algorithm, the gradient can be formalized as:

$$\nabla_\theta J(\theta \mid s) = \mathbb{E}_{\pi \sim p_\theta(|s)}[(C(\pi \mid s) - b(s)) \nabla_\theta \log\, p_\theta(\pi \mid s)] \tag{10}$$

where $b(s)$ denotes a baseline function independent of π and estimates the expected makespan, whose addition reduces the variance of the gradients without affecting the value of $\nabla_\theta J(\theta \mid s)$.

By drawing sample graphs $s_1, s_2, \ldots, s_B \sim \mathcal{S}$ under independent and identical distribution and sampling a single solution per graph, i.e. $\pi_i \sim p_\theta(\cdot \mid s_i)$, the gradient in Eq. (10) is approximated by Monte Carlo sampling as follows:

$$\nabla_\theta J(\theta) \approx \frac{1}{B} \sum_{i=1}^{B} \big(C(\pi_i \mid s_i) - b(s_i)\big) \nabla_\theta \log\, p_\theta(\pi_i \mid s_i) \tag{11}$$

Using a parameterized baseline to estimate the expected makespan usually improves learning efficiency. Therefore, an auxiliary network called a critic and parameterized by θ_v, is introduced to learn the expected makespan found by the current policy p_θ given a state.

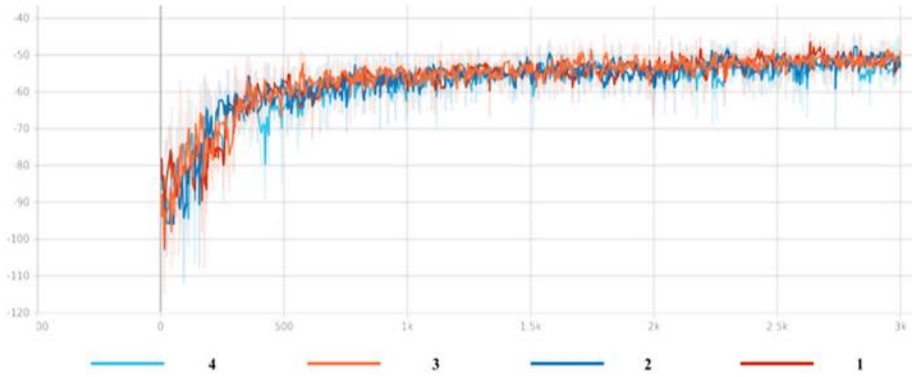
Stochastic gradient descent is used to train the critic on a mean squared error objective between the predicted value $b_{\theta_v}(s)$ and the actual makespan. The objective is computed as

$$\mathcal{L}(\theta_v) = \frac{1}{B} \sum_{i=1}^{B} \left\| b_{\theta_v}(s_i) - C(\pi_i \mid s_i) \right\|_2^2 \tag{12}$$

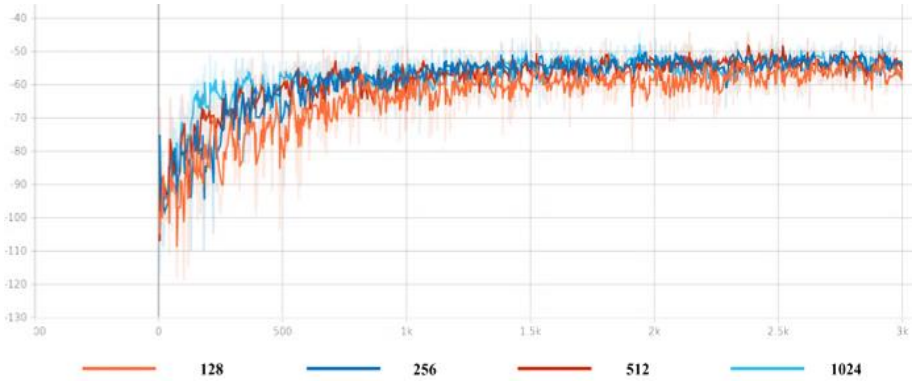
5. EXPERIMENTS

5.1 Experimental conditions

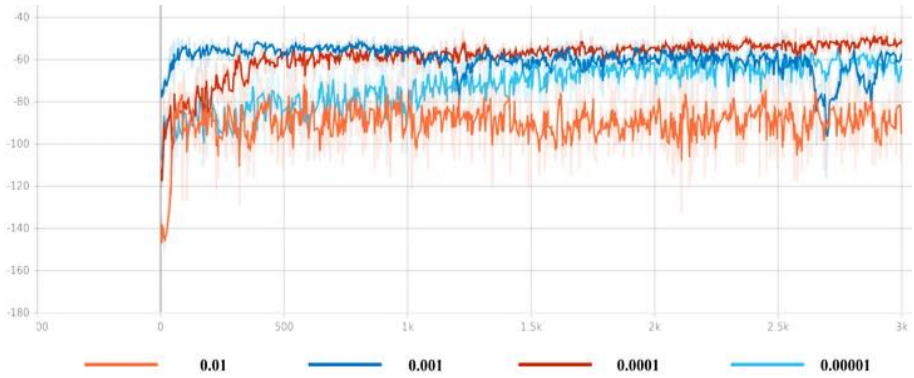
In the study, an improved pointer network is used as an actor to model the policy. The core of the pointer network is the encoder and decoder. Since the input order in FJSP does not affect the result, a one-dimensional convolutional (Conv1D) layer with 1024 hidden units and a kernel size of 1 is used directly instead of RNN; the decoder uses a GRU RNN with 1 hidden layer with 1024 hidden units each. The critic network contains 3 Conv1D layers with the hidden units (80, 80, 1) and a kernel size of 1 each. Both models are trained by Adam optimizer with a learning rate of 10^{-4}. To prevent the gradient explosion, the L_2 norm of the gradients is clipped to 2. The sensitivity analysis on Mk01 was carried out for the number of actor hidden layers, the number of hidden units per layer and the learning rate for both critic and actor network as shown in Fig. 4. The following conclusions can be drawn: (1) The number of actor hidden layers has little effect on the scheduling result. (2) 1024 hidden layer nodes guarantee faster convergence to the optimal result. (3) When the learning rate is 0.0001, the learning process is more stable. Other parameters directly use the empirical parameter values in [19]. The proposed algorithm is implemented by Pytorch 1.7 and runs on a PC platform with Windows 10, 64 bit operating system, 32 Gb RAM, Intel i9 3.6 GHz CPU.



a) Number of the actor hidden layers



b) Number of hidden units in each layer



c) Learning rate of actor and critic network

Figure 4: Parameter sensitivity analysis on the Mk01.

5.2 Experimental results

To verify its effectiveness, the proposed algorithm was applied to train a single model to solve FJSPs in different scales, and compared with nine heuristic rules (three job dispatching rules: shortest processing time – SPT, shortest remaining processing time – SRPT, fewest operation number remaining – FOPNR; three machine assignment rules: earliest finish – EF, shortest processing time – SPT, shortest processing time plus machine work – SPTW; pairwise combination to get nine rules). The author in [19] only used static features as the input of the decoder network, but in FJSP, dynamic features will also have a very profound impact on scheduling results. Therefore, it is necessary to input both dynamic and static features into the network and be compared with the situation where only the static features are input. We call the former AC-SD and the latter AC-S.

We selected Brandimarte's Data benchmarks for verification, which included 10 scheduling instances in different sizes. The scheduling results are shown in Table III. The following

conclusions can be drawn: (1) The optimal scheduling rules of different scheduling instances are marked in red. It can be seen that scheduling rules have different performance. (2) The two RL methods are significantly superior to each scheduling rule on all scheduling instances, which indicates that our proposed algorithm has a stronger generality. (3) The average performance of AC-SD is better than that of AC-S. This is because AC-SD considers both static and dynamic features, and can better grasp the essence of the scheduling problem.

To further verify the generalization performance of the algorithm, scheduling cases were randomly generated for training, and then their performance on 100 unknown cases was tested. Two test scenarios were designed. Test scenario 1 had the same parameter distribution as the training scenario, and test scenario 2 adopted parameter distribution different from the training scenario. Parameter settings for training and test scenarios are shown in Table IV.

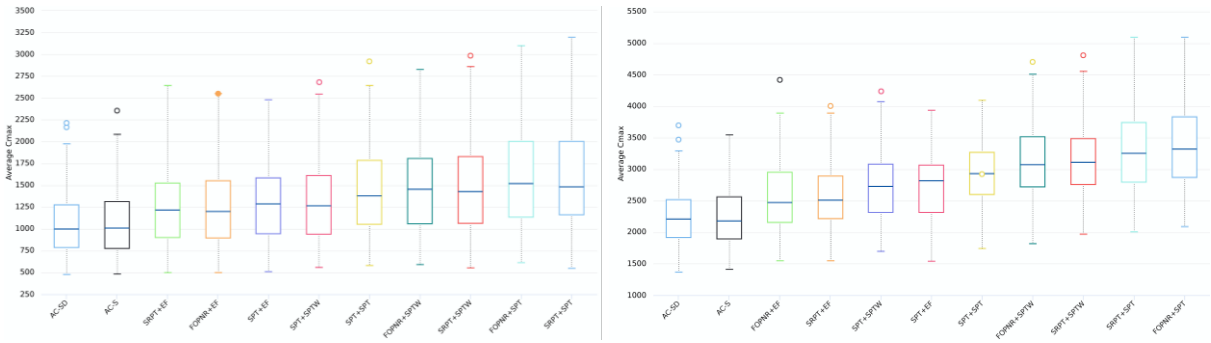
Table III: Scheduling results comparison.

Instances	**Mk01**	**Mk02**	**Mk03**	**Mk04**	**Mk05**	**Mk06**	**Mk07**	**Mk08**	**Mk09**	**Mk10**	**Ave.**
Size	10×6	10×6	15×8	15×8	15×4	10×15	20×5	20×10	20×10	20×15	
SPT+EF	81	85	465	112	250	161	325	717	563	509	326.8
SPT+SPT	80	76	553	195	264	269	371	836	722	585	395.1
SPT+SPTW	81	84	477	106	249	160	328	766	564	524	333.9
SRPT+EF	71	60	374	120	236	126	278	643	535	373	281.6
SRPT+SPT	73	65	579	205	265	264	446	777	716	517	390.7
SRPT+SPTW	69	71	381	120	265	178	295	728	525	414	304.6
FOPNR+EF	76	69	374	123	242	149	278	661	559	404	293.5
FOPNR+SPT	74	65	579	198	259	278	446	761	712	540	391.2
FOPNR+SPTW	59	80	381	111	224	162	295	717	550	460	303.9
LB	**36**	**24**	**204**	**48**	**168**	**33**	**133**	**523**	**299**	**165**	163.3
UB	**42**	**32**	**211**	**81**	**186**	**86**	**157**	**523**	**369**	**296**	198.3
AC-S	49	47	257	86	222	149	227	581	473	405	249.6
AC-SD	44	28	245	74	193	123	216	523	386	337	216.7

Table IV: Parameter distribution in training and test scenarios.

Parameter	**Training scenario**	**Test scenario 1**	**Test scenario 2**
Number of jobs	U[5, 20]	U[5, 20]	N(15, 4)
Number of operations in each job	U[5, 15]	U[5, 15]	U[3, 20]
Number of machines	U[5, 15]	U[5, 15]	N(10, 4)
Number of available machines for each operation	U[2, 5]	U[2, 5]	U(2, 8)
Operation processing time	U[1, 99]	U[1, 99]	N(120, 20)

The parameters as set before (the number of episodes is increased to 100,000) are used to train the algorithms AC-S and AC-SD with two different state expressions. To compare the training effects of the two algorithms, 50 random cases generated according to the distribution of training scenario are selected as the verification set. The optimal scheduling strategies of the two algorithms on the verification set are selected (AC-S and AC-SD got their best strategies in the generations 82000 and 93000, respectively), and their parameters are restored into the neural network structure, which are used to solve the scheduling problems in the two test scenarios respectively, and to compare with the heuristic rules. The results are shown in Fig. 5. The following conclusions can be drawn: (1) Both RL scheduling strategies are significantly better than heuristic rules in scenario 1, and AC-SD is better than AC-S. (2) Both RL scheduling strategies are significantly better than heuristic rules in scenario 2, indicating that RL agents still have strong generalization in test cases from different distributions.



a) Test scenario 1 b) Test scenario 2

Figure 5: Results comparison between two end-to-end RL strategies and heuristics in two test scenarios.

6. CONCLUSION

This research proposed an end-to-end DRL scheduling framework based on a 3D disjunctive graph model, in which the FJSP was formulated as a sequence decision-making problem. With the improved pointer network, each operation was encoded into a high-dimensional embedded vector. Through the attention mechanism, an input was pointed as an action at each decoding step. 20 static features and 24 dynamic features from the perspectives of overall, task and machine were selected, and the input operations were expanded through feature combinations to ensure that only one action is needed to simultaneously determine the job and machine with the highest priority. Experimental results showed that our algorithm can get significantly better scheduling results than heuristic rules with a single trained model, indicating that the model can adapt to different scheduling instances. Besides, the performance of the two models with the input of static and dynamic features, and only the static features was compared, and the results showed that the former had a better average performance.

Compared with the traditional RL method, this method essentially searches in the solution space instead of the rule space and improves the solution quality. The advantages of this algorithm over meta-heuristics lie that through offline training, it can be applied online to scheduling problems of different scales without retraining, which has strong generalization and adaptability. Future research will focus on the following aspects. Firstly, we will perform sensitivity analysis on each hyperparameter to further improve the solution quality. Secondly, we will try to use more advanced policy gradient methods, such as A3C, TRPO, etc. to enhance the existing actor-critic. Finally, the proposed method should be applied to reactive scheduling in a dynamic scheduling environment.

ACKNOWLEDGEMENT

This work was supported by Guangdong science and technology innovation strategy plan (广州市科技计划项目资助), China with the project number 202011020005.

REFERENCES

[1] Brucker, P.; Schlie, R. (1990). Job-shop scheduling with multi-purpose machines, *Computing*, Vol. 45, No. 4, 369-375, doi:10.1007/bf02238804

[2] Panwalkar, S. S.; Iskander, W. (1977). A survey of scheduling rules, *Operations Research*, Vol. 25, No. 1, 45-61, doi:10.1287/opre.25.1.45

[3] Pezzella, F.; Morganti, G.; Ciaschetti, G. (2008). A genetic algorithm for the flexible job-shop scheduling problem, *Computers & Operations Research*, Vol. 35, No. 10, 3202-3212, doi:10.1016/j.cor.2007.02.014

[4] Nouiri, M.; Bekrar, A.; Jemai, A.; Niar, S.; Ammari, A. C. (2015). An effective and distributed particle swarm optimization algorithm for flexible job-shop scheduling problem, *Journal of Intelligent Manufacturing*, Vol. 29, No. 3, 603-615, doi:10.1007/s10845-015-1039-3
[5] Kacem, I.; Hammadi, S.; Borne, P. (2002). Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems, *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, Vol. 32, No. 1, 1-13, doi:10.1109/TSMCC.2002.1009117
[6] Sutton, R. S.; Barto, A. G. (2018). *Reinforcement Learning: An Introduction*, 2nd edition, The MIT Press, Cambridge
[7] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. (2013). Playing Atari with deep reinforcement learning, *arXiv*, Paper 1312.5602, 9 pages
[8] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; Hassabis, D. (2015). Human-level control through deep reinforcement learning, *Nature*, Vol. 518, No. 7540, 529-533, doi:10.1038/nature14236
[9] Zhang, W.; Dietterich, T. G. (1995). A reinforcement learning approach to job-shop schedulling, *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 1114-1120
[10] Riedmiller, S.; Riedmiller, M. (1999). A neural reinforcement learning approach to learn local dispatching policies in production scheduling, *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, 764-771
[11] Aydin, M. E.; Öztemel, E. (2000). Dynamic job-shop scheduling using reinforcement learning agents, *Robotics and Autonomous Systems*, Vol. 33, No. 2-3, 169-178, doi:10.1016/S0921-8890(00)00087-7
[12] Paternina-Arboleda, C. D.; Das, T. K. (2005). A multi-agent reinforcement learning approach to obtaining dynamic control policies for stochastic lot scheduling problem, *Simulation Modelling Practice and Theory*, Vol. 13, No. 5, 389-406, doi:10.1016/j.simpat.2004.12.003
[13] Gabel, T.; Riedmiller, M. (2012). Distributed policy search reinforcement learning for job-shop scheduling tasks, *International Journal of Production Research*, Vol. 50, No. 1, 41-61, doi:10.1080/00207543.2011.571443
[14] Wang, Y.-F. (2020). Adaptive job shop scheduling strategy based on weighted Q-learning algorithm, *Journal of Intelligent Manufacturing*, Vol. 31, No. 2, 417-432, doi:10.1007/s10845-018-1454-3
[15] Ren, J. F.; Ye, C. M.; Yang, F. (2020). A novel solution to JSPS based on long short-term memory and policy gradient algorithm, *International Journal of Simulation Modelling*, Vol. 19, No. 1, 157-168, doi:10.2507/IJSIMM19-1-CO4
[16] Pan, R.; Dong, X.; Han, S. (2020). Solving permutation flowshop problem with deep reinforcement learning, *Proceedings of the 2020 Prognostics and Health Management Conference*, 349-353, doi:10.1109/PHM-Besancon49106.2020.00068
[17] Vinyals, O.; Fortunato, M.; Jaitly, N. (2015). Pointer networks, *arXiv*, Paper 1506.03134, 9 pages
[18] Bello, I.; Pham, H.; Le, Q. V.; Norouzi, M.; Bengio, S. (2017). Neural combinatorial optimization with reinforcement learning, *arXiv*, Paper 1611.09940, 15 pages
[19] Nazari, M.; Oroojlooy, A.; Snyder, L.; Takác, M. (2018). Reinforcement learning for solving the vehicle routing problem, *arXiv*, Paper 1802.04240, 21 pages