Policy Network for Solving Flexible Job Shop Scheduling Problem With Setup Times and Rescoure Constraints

Ning Xu
Software Engineering Institute
Shanghai, China
51194501195@stu.ecnu.edu.cn

Tian-Ming Bu*
Shanghai Key Laboratory of Trustworthy Computing
Shanghai, China
tmbu@sei.ecnu.edu.cn

ABSTRACT

The flexible job shop scheduling problem (FJSP) is one of the most popular scheduling problems which allows an operation to be completed on several machines in a specific order and is well known as NP-hardness. And in today's industrial manufacturing environment, this problem is becoming more complex. We should take sequence-dependent setup times and resource constraints into account during the manufacturing process. In this article, a policy network is designed to solve the extended FJSP, with the goal of minimizing the total completion time (makespan). Three dispatching rules are proposed to simultaneously select an operation and assign it on a feasible machine at each scheduling point. To describe the manufacturing status at a scheduling stage, six state features are extracted as the input of the network. After the calculation of the network, the distribution of choosing each dispatching rule is obtained. After all of the operations are completed, the policy network is trained using *policy gradient*, a well-known reinforcement learning (RL) method. Finally, we generate ten benchmark data instances with different scales to compare the performance and efficiency of policy network with other algorithms. The proposed policy network outperforms its competitors, according to the results of the experiments.

CCS CONCEPTS

• **Applied computing → Industry and manufacturing**.

KEYWORDS

flexible job shop scheduling problem, sequence-dependent setup times, resource constraints, policy network, reinforcement learning

ACM Reference Format:
Ning Xu and Tian-Ming Bu. 2022. Policy Network for Solving Flexible Job Shop Scheduling Problem With Setup Times and Rescoure Constraints. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/1122445.1122456

*Corresponding author

1 INTRODUCTION

Scheduling plays a very important role in industrial manufacturing. JSP is one of the most classic scheduling problems, and it has been proven to be NP-hard. Flexible job shop problem (FJSP) which allows each operation to be processed on one or more machines is a more general version of JSP. In today's highly integrated modern production environment, it is not enough to model the practical problems with traditional FJSP. Sequence-dependent setup times and resource constraints should be considered when describing the manufacturing environment. The sequences of alternative operations and machines are two typical variables in different production environments and project scheduling. In this paper, the aim of the extended FJSP is to determine the allocation of each operation and to specify the sequence of operations on each machine in order to minimize the completion time of the last operation.

As a hot research field in industry and academia, FJSP has been deeply studied in the past few decades. Due to its computational complexity, many heuristic approaches are used to solve FJSP, which include genetic algorithm (GA) [9], ant colony algorithm (ACO) [21], artificial bee colony (ABC) [19], particle swarm optimization (PSO) [14] and so on. Tabu search (TS) [2] is also an effective method to solve FJSP, which is composed of two parts: a procedure that searches for the best sequence of job operations, and a procedure that finds the best choice of machine alternatives. As a result, many researchers use hybrid algorithm which combines intelligence algorithm with local search to get better results such as [5]. In recent years, with the emerging of RL, many researches use RL-based methods to solve FJSP. For example, [4] uses RL to find the optimal parameters of the intelligence algorithm. [13] uses RL to find the proper dispatching rule for scheduling.

In this paper, we propose a policy network to solve the extended FJSP with setup times and rescoure constraints. The contributions of this paper will be listed as follows:

(1) Extract six generic features with values in range [0, 1] to represent the state at each scheduling time.

(2) Design three dispatching scheduling rules to select the next operation to be processed and the next machine to be assigned on.

(3) Construct a neural network trained by *policy gradient.*

(4) Bulid 10 benchamark instances based on Brandimarte's FJSP instances and conduct many experiment on them, and the results show that the algorithm in this paper is more effective and has better performance compared to other algorithms.

The rest of this article is structured as follows. Section 2 briefly reviews related work in literature. Section 3 defines FJSP with setup times and rescoure constraints formally. Section 4 introduces reinforcement learning. Section 5 proposes the policy network. Section

6 presents a large number of experimental results. Finally, the conclusion is presented in Section 7.

2 RELATED WORK

Intelligent optimization algorithms have become the most widely used algroithm for solving FJSP with different constraints. In 2002, Kacem et al. [8] suggested GA for controlling the operation and machine sequences to solve traditional FJSP. In 2008, Pezzella et al. [15] proposed an approach that combines several methods for generating original populations and selecting individuals to reproduce new individuals, resulting in improved outcomes. In 2014, Li et al. [11] presented a novel discrete artificial bee colony (DABC) algorithm for solving the multi-objective flexible job shop scheduling problem with maintenance activities. In 2015, Gao et al. [6] used a discrete harmony search to solve FJSP with fuzzy processing time. In 2016, Li et al. [12] proposed a hybrid algorithm which hybridizes GA and TS to minimize the makespan. In 2018, Jiang et al. [7] applied gray wolf optimization (GWO) to solve FJSP, which is originally proposed to solve continuous optimization problems inspired from the social hierarchy and hunting behaviors of grey wolves. In 2019, Tang et al. [18] presented a hybrid discrete particle swarm optimization integrated with simulated annealing algorithm (PSO-SA) to solve FJSP with tolerated time interval and limited starting time interval. In 2020, Li et al. [10] proposed an imporved Jaya (IJaya) algorithm to solve FJSP with setup time and transportation time.

In the field of artificial intelligence, RL has achieved great success. Researchers are increasingly putting this technology in different directions, such as playing Go, playing electric games, autonomous driving, etc. After episodes of training, the agent learns how to maximize rewards through interaction with the environment. Many RL-based approaches have been applied to various types of complex scheduling problems, due to RL's ability to learn the right action at each decision point and respond to dynamic events entirely in real time. In 1999, Riedmiller et al. [16] first proposed a RL network to learn dispatching rules for JSP. In this paper, a neural network based agent optimizes its local dispatching strategy autonomously to achieve a global optimization target for the entire scheduling. In 2000, Aydin et al. [1] developed an intelligent agent based dynamic scheduling system which is trained by Q-III learning which consists of the agent and the environment. In real time, the agent chooses the most suitable priority rule according to shop conditions. At the same time, the simulated environment schedules operations was chosen by the agent according to the rule. In 2004, Wang et. al [20] designed a single machine agent which employs the Q-learning algorithm to develop a decision-making policy of choosing rules to minimize mean tardiness. In 2017, Shahrabi et al. [17] used Q-learning to find the optimal parameters of the variable neighborhood search. The states of scheduling are divided into several numerical intervals according to the number of jobs and the mean processing time of operations in the current. In 2020, Chen et al. [3] proposed GA-Q, GA-SARSA,SLGA which use both Q-learning and Sarsa algorithm to adjust GA parameters. According to the fitness of the current generation, states are divided into 20 intervals.

Most RL algorithms for scheduling problems use discrete and finite state Q-learning, which involves keeping track of the function values of multiple actions in each discrete state and performing and adjusting the action with the largest function value. However, for a practical problem, the number of states may be infinite, thus it is impossible to use Q-list based RL method like Q-learning and Sarsa. A direct solution is to divide state into numerical pieces. The shortcomings of state discretization are obvious. First, there is no effective way to choose the proper number of states. Second, the precision is lost. To address this problem, in this paper, we introduce a neural network with six features of the state as input. It can deal with continuous spatial states.

3 PROBLEM FORMULATION

FJSP is a classic complicated scheduling problem, which has been proven to be NP-hard. It can be defined as follows. There are n jobs to be processed on m different machines. Each job i ($1 \leq i \leq n$) has N_i operations to be processed orderly. After each machine has processed an operation, it needs an operator to change the tool to continue processing the next operation. The number of operators who can change tools is w. If there is no free operator, the machine that needs to be changed the tool is in a waiting state. Each operation O_{ij} (the jth operation of job i) can be processed on the machine set M_{ij}. The processing time T_{ijk} of operation O_{ij} on machine k ($k \in M_{ij}$) may be different. The time of the operator to change the tool is sequence-dependent, which is represented by L_{ijmnk}. The objective is to minimize the makespan, namely, the finish time of the last completed operation.

3.1 Parameters

n: the number of jobs;
m: the number of machines;
w: the number of operators who can change tools;
N_i: the number of operations of job i;
O_{ij}: the jth operation of job i;
M_{ij}: the machine set of O_{ij};
T_{ijk}: the processing time of operation O_{ij} on machine k;
L_{mnijk}: the time of the operator to change the tool from operation O_{mn} to operation O_{ij} on machine k;
C_{ij}: the completion time of operation O_{ij}.

3.2 Constraints

(1) (Capacity Constraint) Each machine can process at most one operation at a time;

(2) (Precedence Constraint) Operations belonging to the same job should be processed one after another in a fixed order;

(3) (Setup times Constraint) After each machine has processed an operation, it needs an operator to change the tool to continue processing the next operation.

(4) (Resource Constraint) The number of operator is limited.

(5) (Consistency Constraint) Each operation should be assigned to one machine and should not be interrupted;

3.3 Objective

$$\text{Minimize} \max_{i,j} C_{ij}$$

3.4 Example

In order to illustrate the problem clearly , Table 1 gives the processing times of operation of each job in an instance with 3 machines and 3 jobs. Due to space constraints, only the sequence-dependent setup times of machine 1 are listed in Table 2. The number of operators is 2. The gant chart of a solution of the instance is shown in Figure 1. The blue rectangle is the setup time, the white rectangle represents the processing time. Since there are only 2 opeartors, Operation O_{21} can't be processed at the beginning on machine 3 until any operator is available(time 1). If there is no limit number of operators, the completion time of job 2 can be reduced by 1.

Table 1: Processing times

Job	O_{ij}	M_1	M_2	M_3
J_1	O_{11}	5	5	4
	O_{12}	4	2	3
J_2	O_{21}	2	2	1
	O_{22}	3	4	3
J_3	O_{31}	2	4	5
	O_{32}	2	-	2

Table 2: Setup times on machine 1

	O_{11}	O_{12}	O_{21}	O_{22}	O_{31}	O_{32}
start	1	1	1	2	2	1
O_{11}	-	1	2	1	1	2
O_{12}	-	-	2	2	2	1
O_{21}	1	2	-	1	1	1
O_{22}	1	2	-	-	2	2
O_{31}	1	2	1	2	-	1
O_{32}	2	2	2	1	-	-

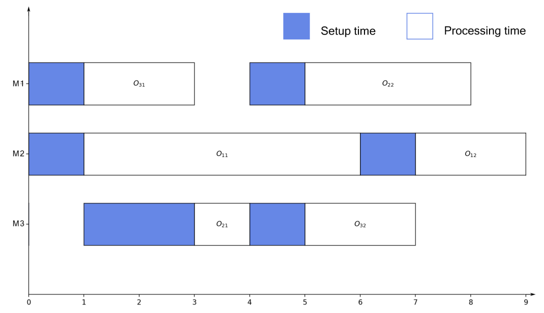

Figure 1: Gantt chart of the example

4 REINFOCEMENT LEARNING

4.1 Markov Decision Process

The most classic model of RL is called Markov Decision Process (MDP). The MDP model can generally be represented by a 5-tuple (S, A, P, γ, R).

S: The state set;

A: The action set;

P: The probability of moving from state S_t to the next state S_{t+1} by making action A_t and getting reward R_{t+1} is defined as follows.

$$Pr[S_{t+1} = s', R_{t+1} = r | S_t = s, At = a],$$
$$\pi(a|s) = Pr[A_t = a | S_t = s];$$

γ: The discount factor determines how to trade off between recent rewards and future rewards;

R: Rewards.

In the MDP model, according to the strategy π, the agent interacts with the environment. At each decision time t, the agent observes the current state S_t, follows the strategy π, and makes an action A_t. After interacting with the environment,the agent moves from S_t to S_{t+1}, and gets a reward r.

4.2 Bellman Equations

The purpose of RL is to find a strategy that maximizes returns. Assuming that the turn-based task reaches the end state at step T, the return G can be defined as:

$$G = R_0 + R_1 + \cdots + R_T.$$

The value of the optimal strategy π is defined as follows:

$$v_\pi(s) = E_\pi[G_t | S_t = s], \tag{1}$$

$$q_\pi(s, a) = E_\pi[G_t | S_t = s, A_t = a]. \tag{2}$$

From Eq.1 and Eq.2, Bellman Equations can be derived as follows:

$$v_\pi(s) = \sum_a \pi(a|s)\,[r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_\pi(s'))], s \in S, \tag{3}$$

$$q_\pi(s,a) = \sum_{s',r} p(s', r|s, a)[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a')], s \in S, a \in A. \tag{4}$$

4.3 Policy Gradient

The policy gradient theorem gives the relationship between the expected return and the policy. In the turn-based task, the expected return of the strategy $\pi(\theta)$ can be represented as $E_{\pi(\theta)}[G_0]$. Using Eq.3 and Eq.4, the theorem derives the gradient of the policy parameters as follows:

$$\nabla E_{\pi(\theta)}[G_0] = E[\sum_{t=0}^{+\infty} \gamma^t G_t \nabla \ln \pi(A_t | S_t; \theta)] \tag{5}$$

The policy gradient theorem shows that changing the strategy parameters along the direction of the strategy gradient has the opportunity to increase the expected return. Based on this conclusion, the algorithm of *policy gradient* is proposed in Algorithm 1.

Policy gradient chooses an action through observation, and uses a reward to directly increase/decrease he possibility of choosing good/bad behaviors. Since the target of FJSP is determined by the behaviors of the whole round and *policy gradient* can adjust strategy after the whole schedule, we choose it to train the network.

5 POLICY NETWORK FOR FJSP WITH SETUP TIMES AND RESOURCE CONSTRAINTS

FJSP can be regarded as an MDP where an intelligent agent should determine the optimal action, i.e., which rule to choose, after comprehensively observing the information from the current production state and gets a reward after evaluating the performance

Algorithm 1 Policy Gradient

```
Input: Observations of state.
Output: Optimal strategy parameters estimation.
1: Initial the stragety parameters $\theta$.
2: for each episode do
3:     Generate $\{s_1, a_1, r_2, \cdots, s_{T-1}, a_{T-1}, r_T\}$ according to the strategy $\pi_\theta$
4:     for $t = T - 1$ to 1 do
5:         $G \leftarrow \gamma G + r_{t+1}$ ;
6:         $\theta \leftarrow \theta + \alpha\gamma^t G \nabla \ln \pi(A_t|S_t;\theta)$
7:     end for
8: end for
```

of each scheduling action. So, in the policy network, the state S refers to the features at each scheduling time. Action A refers to choosing which rule. Reward R refers to the reward or punishment of the action.

5.1 Framework

At each scheduling time, we use six state features as the input to the policy network. The probability of which rule should be chosen is generated as the output. Following the selected rule, we execute the corresponding action which helps us scheduling an operation on a specific machine. After all operations have been scheduled, the policy network will be trained with *policy gradient*. With several episodes' training, the estimation of policy parameter θ gets its optimization and the makespan of scheduling is obtained accordingly. Figure 2 gives an illustration of the proposed policy network. The procedure of the policy network is given in Algorithm 2.

Algorithm 2 Policy Network For FJSP

```
1: Initialize the observation, the action, the reward memory.
2: Initialize the online network with random weights $\theta$.
3: for episode 1 to $L$ do
4:     Generate the initial state $s_1$ with feature vector $\phi_1$
       $=\{U_{ave}(1), U_{std}(1), CRO(1), JCR_{ave}(1), JCRstd(1), UPM(1)\}$
       $=\{1, 0, 0, 0, 0, 1\}$
5:     for 1 to $T$ do ($t$: decision time, $T$: terminal time)
6:         Select an action $a_t$ with softmax policy.
7:         Execute the dispatching rule according to the action $a_t$.
8:         Observe the next state $s_{t+1}$.
9:         Calculate the immediate reward $r_t$.
10:        Store observation, action, reward in memory.
11:    end for
12:    /* policy gradient to adjust network parameter $\theta$ */
13:    for $t = T - 1$ to 1 do
14:        $G \leftarrow \gamma G + r_{t+1}$
15:        $\theta \leftarrow \theta + \alpha\gamma^t G \nabla \ln \pi(a_t|s_t;\theta)$
16:    end for
17: end for
```

The policy network used in this paper is a deep neural network consisting of four fully connected layers with one input layer, one output layer and two hidden layers. The numbers of nodes belonging to input layer and output layer are the same as the number of state features and available actions, respectively. The number of nodes in each hidden layer is 20. We use "tanh" activation function for the input, hidden and output layers. In the action selection policy, we utilize a "softmax" policy to select an action according to the output of the network. For any state $s \in S$, it is always satisfied that $\sum_a \pi(a|s;\theta) = 1$. Thus, the action preference function $h(s, a; \theta)$ is introduced. Its softmax value is defined in Eq.6:

$$\pi(a|s;\theta) = \frac{\exp h(s, a; \theta)}{\sum_{a'} \exp h(s, a'; \theta)} \tag{6}$$

5.2 Details of the implementation

How to define state, action and reward is the most important in RL. How to define them determines the performance of scheduling. We will first describe the state features in this subsection. The descriptions of dispatching rules and rewards are then given.

Firstly, we need to define some parameters.

$C_{M_k}(t)$: the completion time of the last operation on machine k at scheduling time t;

$C_i(t)$: the completion time of the last completed operation of job i at scheduling time t;

$Oper_i(t)$: the number of uncompleted operations of job i at scheduling time t;

$U_k(t)$: $\frac{\sum_{i=1}^{n}\sum_{j=1}^{Oper_i(t)} T_{i,j,k} X_{i,j,k}}{C_{M_k}(t)}$:
the utilization rate of machine k at scheduling time t,
where $X_{ijk} = \begin{cases} 1, \text{ if operation } O_{ij} \text{ is assigned on machine } k \\ 0, \text{ otherwise} \end{cases}$;

$Tave_{ij}$: $\frac{\sum_{k \in M_{ij}} T_{ijk}}{|M_{ij}|}$, the average proceeing time of M_{ij}.

5.2.1 *Features of State.*

In most RL-based scheduling methods, state features are always defined as some parameters of the production state, i.e the remaining number of uncompleted jobs, the remaining number of operations, the remaining processing time and so on. However, in real application, these parameters are unlimited and vary in a wide range. If we use these parameters as input, the performance of *policy gradient* will be very poor. To address this problem, six state features valued in [0, 1] are designed and extracted at each scheduling time.

(1) The average utilization rate of each mahcine U_{ave}, as defined in Eq.7.

$$U_{ave} = \frac{\sum_{k=1}^{m} U_k(t)}{m} \tag{7}$$

(2) The standard deviation of machine utilization rate U_{std}, as defined in Eq.8.

$$U_{std} = \sqrt{\frac{\sum_{k=1}^{m}(U_k(t) - U_{ave})^2}{m}} \tag{8}$$

(3) The completion rate of operations CRO, as defined in Eq.9.

$$CRO = \frac{\sum_{i=1}^{n} Oper_i(t)}{\sum_{i=1}^{n} N_i} \tag{9}$$

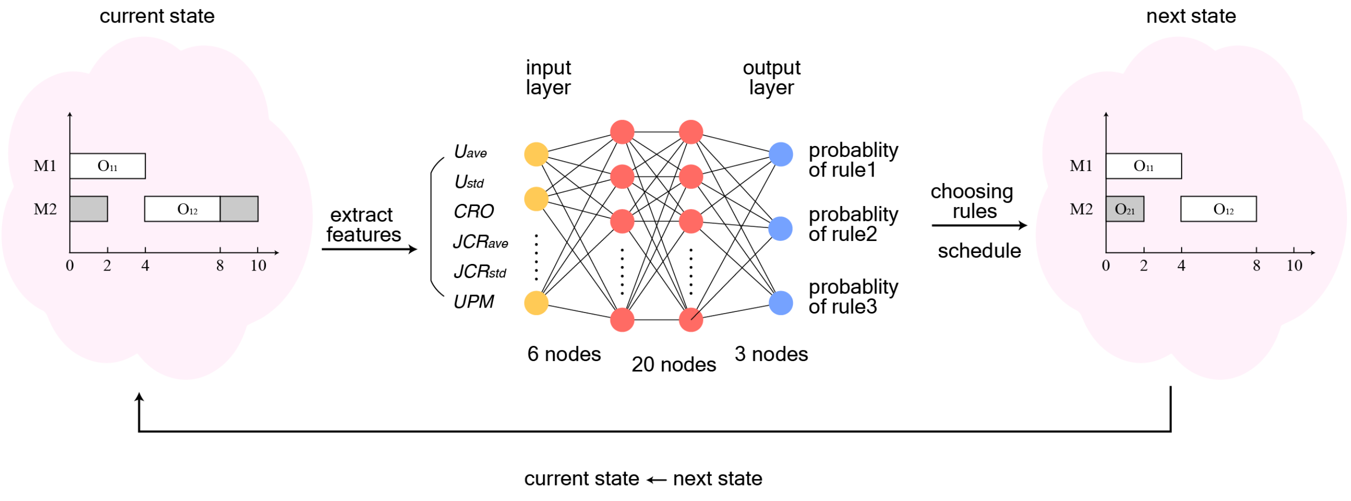

Figure 2: Structure of the proposed policy network

(4) The average of job completion rate JCR_{ave}, as defiend in Eq.10.

$$JCR_{ave} = \frac{\sum_{i=1}^{n} \frac{Oper_i(t)}{Ni}}{n} \tag{10}$$

(5) The standard deviation of job completion rate JCR_{ave}, as defined in Eq.11.

$$JCR_{std} = \sqrt{\frac{\sum_{i=1}^{n} (\frac{Oper_i(t)}{Ni} - JCR_{ave})^2}{n}} \tag{11}$$

(6) The average utilization per makespan, UPM, as defined in Eq.12.

$$UPM = \frac{\sum_{i=1}^{n} \sum_{j=1}^{Oper_i(t)} T_{ijk} X_{ijk}}{\max_i C_i(t) \sum_{i=1}^{n} \sum_{j=1} Oper_i(t)} \tag{12}$$

5.2.2 *The proposed dispatching rules.*

In FJSP, different scheduling strategies will have an impact on the makespan. Rules must be applied to different states. So three composited dispatching rules are elaborately designed to assign operations on suitable machine to make the makespan as small as possible. The three comprehensive dispatching rules are described in the following.

We need to define the earliest available completion time $Cea_{ijk}(t)$ of operation O_{ij} on the machine k at the scheduling time t before choosing rules. The calculation process is described in Alogrithm 3.

Algorithm 3 The earliest available completion time $Cea_{ijk}(t)$

```
Input: all available intervals $[t_s, t_e]$ on the machine $k$, $L_{mnijk}$
Output: $Cea_{ijk}(t)$
1: for each intervals $[t_s, t_e]$ do
2:     if interval $[t_s, t_s + L_{mnijk}]$ is also aviable for operator then
3:         if max $[C_i(t), t_s + L_{mnijk}] + T_{ijk} < t_e$ then
4:             $Cea_{ijk}(t) = \max [C_i(t), t_s + L_{mnijk}] + T_{ijk}$
5:             return
6:         end if
7:     end if
8: end for
9: $Cea_{ijk}(t) = C_k(t) + L_{mnijk} + T_{ijk}$.
10: return
```

Algorithm 4 Procedure of dispatching rule 1

```
Input: $C_i(t), Oper_i(t), M_{ij}, T_{ijk}$
Output: $X_{ijk}$
1: $i \leftarrow \arg\min_{i \in n} \frac{Oper_i(t)}{N_i}$
2: $j \leftarrow Oper_i(t) + 1$
3: $k \leftarrow \arg\min_{k \in M_{ij}} Cea_{ijk}(t)$
4: assign $O_{ij}$ on machine $k$
5: set $X_{ijk} = 1$
```

(1) Dispatching rule 1

First, job i with the minimum operation completion rate is selected. Then we assign job i on a proper machine k which leads to minimum completion time. If there are more then one machine having the minimum available completion time, we randomly choose one of them. The procedure is described in Algorithm 4.

(2) Dispatching rule 2

First, job i with the minimum $C_i(t)$ is selected. Then we assign job i on a proper machine k which leads to minimum completion time. If there are more then one machine having the minimum available completion time, we randomly choose one of them. The procedure is described in Algorithm 5.

(3) Dispatching rule 3

Fisrt, random choose job i. Then we assign job i on a proper machine k which leads to minimum completion time. If there are more then one machine having the minimum available completion time, we randomly choose one of them.

5.2.3 *Rewards.*

Algorithm 5 Procedure of dispatching rule 2

```
Input: $C_i(t), Oper_i(t), M_{ij}, L_{mnijk}, T_{ijk}$
Output: $X_{ijk}$
1: $i \leftarrow \arg\min_{i \in n} C_i(t)$
2: $j \leftarrow Oper_i(t) + 1$
3: $k \leftarrow \arg\min_{k \in M_{ij}} Cea_{ijk}(t)$
4: assign $O_{ij}$ on machine $k$
5: set $X_{ijk} = 1$
```

Since the makespan, which is determined by the whole actions, the reward r_t is defined by successively considering the makespan of last round and this round. The procedure is given in Algorithm 6.

Algorithm 6 Reward r_t at each decision point t

```
Input: $Makespan_b$ (Makespan of last round) , $C_i(t)$
1: if the schdeluing is not completed then
2:     $r_t \leftarrow 0$
3: else
4:     $Makespan_{now} \leftarrow \max_{i \in n} C_i(t)$
5:     if $Makespan_{now} \leq Makespan_b$ then
6:         $r_t \leftarrow 1$
7:     else
8:         $r_t \leftarrow -1$
9:     end if
10:    $Makespan_b \leftarrow Makespan_{now}$
11: end if
```

6 EXPERIMENTS

In this section, a large number of experiments are carried out to test the performance and effectiveness of the proposed policy network. We bulid 10 benchamark instances based on Brandimarte's FJSP instances [2] and these instances are named Ins. We generate random integers between 1 and 5 as the setup times for each machine in each instance. We set the number of operators according to the scale of each instance. These experiments are implemented in Python 3.7 on a PC with Intel Core i7 @ 2.6 GHz and 16 GB RAM. The algorithm's parameter settings are listed in Table 3.

Table 3: parameter settings of policy network

Parameter	Value
Number of episodes L	200
Learning rate α	0.01
Discount factor γ	1
Number of layers	3
Number of nodes of hidden layers	20

6.1 Convergence

We choose Ins02 (10 jobs * 6 machines * 3 operators) for experiment. The training steps are shown in Figure 3. At the beginning, the makespan obtained by randomly choosing dispatching rules is very large. With the training of the network, the curve of makespan decreases smoothly. We also use a linear regression model to model the curve, the slope of the line is negative, which indicates that the policy network has learned how to select the proper rule in an efficient way at various scheduling states.

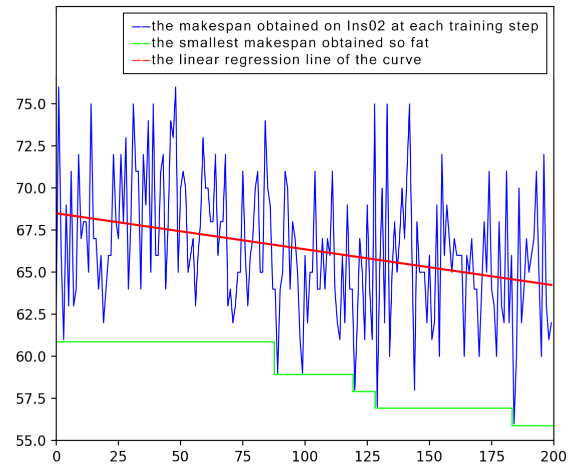

Figure 3: Makespan obtained at each training episode. X axis: the number of training episodes, Y axis: makespan.

The result of scheduling instance Ins02 is shown in Figure 4. The blue rectangle is setup time, and the white rectangle represents processing time. We can see from the chat that the utilization of each machine is very high, which indicates that it is a good solution for this problem.

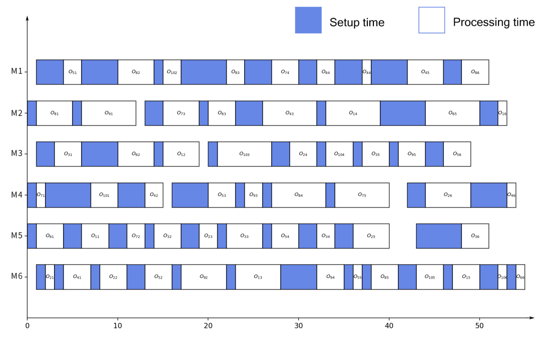

Figure 4: Gant chart of Ins02

6.2 Comparison with the proposed dispatching rules

To verify the effectiveness of the proposed policy network, we compare the performance of policy network with each dispatching

Table 4: Experimental results obtained by policy network and dispatching rules

Instance	Best Value				Mean Value				Sd Value			
(J*M*O)	Rule 1	Rule 2	Rule 3	PN	Rule 1	Rule 2	Rule 3	PN	Rule 1	Rule 2	Rule 3	PN
Ins01(10*6*3)	75	82	72	**71**	79.9	108.9	79.8	**74.0**	**3.1**	9.9	4.5	9.6
Ins02(10*6*3)	59	75	57	**55**	64.0	88.0	65.0	**57.0**	**2.5**	5.7	4.8	6.5
Ins03(15*8*3)	248	304	239	**228**	258.9	352.5	252.7	**236.5**	**6.6**	22.8	7.2	22.0
Ins04(15*8*3)	122	145	114	**110**	132.6	186.7	125.9	**116.4**	**5.6**	15.8	5.9	27.7
Ins05(15*4*3)	262	313	257	**251**	269.4	330.3	266.7	**256.0**	**3.8**	11.5	5.4	18.2
Ins06(10*15*3)	143	187	141	**137**	152.6	208.0	151.3	**142.8**	**4.8**	11.6	5.8	23.7
Ins07(20*5*4)	213	259	210	**208**	244.5	273.1	223.6	**212.8**	**6.0**	10.0	6.7	28.6
Ins08(20*10*4)	656	754	655	**640**	665.3	799.0	667.7	**650.6**	**5.4**	22.8	6.9	59.1
Ins09(20*10*4)	400	558	408	**400**	418.6	618.8	424.1	**419.1**	**6.9**	28.5	8.0	55.4
Ins10(20*15*4)	310	400	305	**303**	319.1	442.7	**320.9**	321.4	**4.0**	20.1	7.5	70.9

Table 5: Experimental results obtained by policy network and other intelligent algorithms

Instance	Best Value				Mean Value				Sd Value			
(J*M*O)	PSO-SA	DABC	IJaya	PN	PSO-SA	DABC	IJaya	PN	PSO-SA	DABC	IJaya	PN
Ins01	90	102	99	**71**	106.7	117.5	108.8	**74.0**	35.7	36.1	31.2	**9.6**
Ins02	97	106	93	**55**	107.4	116.1	110.3	**57.0**	27.7	23.8	33.6	**6.5**
Ins03	457	478	458	**228**	493.8	534.5	504.6	**236.5**	129.2	112.1	100.0	**22.0**
Ins04	162	190	173	**110**	185.5	203.4	187.9	**116.4**	53.2	31.9	42.2	**27.7**
Ins05	306	320	324	**251**	329.8	345.5	347.4	**256.0**	64.8	64.2	66.3	**18.2**
Ins06	286	318	299	**137**	310.4	339.2	332.0	**142.8**	85.8	67.3	52.6	**23.7**
Ins07	302	333	306	**208**	326.2	360.9	340.8	**212.8**	71.5	88.5	98.0	**28.6**
Ins08	769	797	813	**640**	814.4	841.9	856.5	**650.6**	142.7	114.1	114.5	**59.1**
Ins09	668	747	710	**400**	740.6	783.6	770.4	**419.1**	158.0	103.8	162.8	**55.4**
Ins10	611	646	667	**303**	655.0	705.8	703.3	**321.4**	170.3	131.7	113.8	**70.9**

rule proposed in this paper. On each instance, the policy network and each rule are repeatedly run independently for 30 times. The results of expeirment obtained by each method are provided in Table 4 with the best experiment results highlighted in bold font. J,M,O means number of Job, Machine and Operator. PN means the proposed policy network in this paper.

From Table 4, the proposed policy network gets the best value of 10 instances, compared with other dispatching rules. This implies that the proposed policy network is able to learn how to select the best dispatching rule for different problems. In addition, it also gets the best mean value of 9 instances as opposed to other dispatching rules, which indicates that the proposed policy network's performance is superior to that of other rules.

Furthermore, from the comparisons of three dispatching rules, no rule can always win and get the best results of the 10 instances. Thus it can be concluded that single-rule based scheduling is not a good strategy. We should select the best rule by thoroughly investigating the current state through the proposed policy network.

6.3 Comparisons with other Intelligent algorithms

In this section, experiments are carried out to verify the performance of the policy network compared to other intelligent algorithms including PSO-SA[18] , DABC[11] and IJaya [10] as mentioned in related work. Each instance is run for 30 times indepenently. The summary of computational results for the makespan is given in Table 5. The best results are highlighted in bold font.

From Table 5, it is obviously seen that the policy network gets the best results for all instances in best value and mean value compared with other three algorithms. And for larger scales, the proposed policy network improves the solution quality obviously. The larger the scale, the more obvious it is. These results indicate that the proposed policy network is more effictive in solving FJSP with setup-time and resource constraints, especially large-scale problems which is more suituable for real manufacturing environment. However the time consumption of the proposed policy network has increased compared with intelligent algorithms as shown in Table 6, because we use neural network for training policy nertork to generate strategy. As far as perfomance is concerned, it is worthwill to sacrifice the time of solving problem.

Table 6: Time consumption (second) by policy network and intelligent algorithms

Instance	PSO-SA	DABC	IJaya	PN
Ins01	1.9	1.8	3.1	5.7
Ins02	2.0	1.9	3.1	6.3
Ins03	5.2	6.7	8.3	17.5
Ins04	2.9	2.9	5	9.2
Ins05	5.2	3.4	8.3	9.6
Ins06	6.3	4.3	10.1	16.2
Ins07	4.6	3.3	6.1	11.2
Ins08	11.3	7.5	15.2	27.3
Ins09	11.9	8.4	15.3	30.7
Ins10	10.5	8.1	14.2	29.3

7 CONCLUSION

In this paper, a policy network is proposed to address FJSP with setup-time and resource constraints. Three dispatching rules are designed to pick an appropriate operation and assign it to a suitable machine at each scheduling state. Six elaborately-designed features valued in [0, 1] are extracted from the current state as the input of the policy network. After all of the operations have been arranged, the makespan is used to determine the reward, which influences how to choose dispatching rules at next episode. When training the policy network, a "softmax" policy is utilized to choose rules.

We generate ten examples from ten Bandimarte's instances with random setup-time and operator number. Numerical experiments are carried out to verify the effectiveness and performance of the proposed policy network. The results indicate that the policy network performs better than other dispatching rules, intelligence algorithms for solving FJSP with setup times and resource constraints.

The dispatching rule is a crucial factor in scheduling. In the future, we will design more suitable dispatching rules for the policy network to choose from. Meanwhile, there are many other state-of-art RL-based approaches for optimization, such as A3C, TRPO, and PPO. We will run more tests on them and compare their performance with the policy network.

REFERENCES

[1] M Emin Aydin and Ercan Öztemel. 2000. Dynamic job-shop scheduling using reinforcement learning agents. *Robotics and Autonomous Systems* 33, 2-3 (2000), 169–178.

[2] Paolo Brandimarte. 1993. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations research* 41, 3 (1993), 157–183.

[3] Ronghua Chen, Bo Yang, Shi Li, and Shilong Wang. 2020. A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem. *Computers & Industrial Engineering* 149 (2020), 106778.

[4] Eid Emary, Hossam M Zawbaa, and Crina Grosan. 2017. Experienced gray wolf optimization through reinforcement learning and neural networks. *IEEE transactions on neural networks and learning systems* 29, 3 (2017), 681–694.

[5] Jie Gao, Linyan Sun, and Mitsuo Gen. 2008. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers & Operations Research* 35, 9 (2008), 2892–2907.

[6] Kai Zhou Gao, Ponnuthurai Nagaratnam Suganthan, Quan Ke Pan, and Mehmet Fatih Tasgetiren. 2015. An effective discrete harmony search algorithm for flexible job shop scheduling problem with fuzzy processing time. *International Journal of Production Research* 53, 19 (2015), 5896–5911.

[7] Tianhua Jiang and Chao Zhang. 2018. Application of grey wolf optimization for solving combinatorial problems: job shop and flexible job shop scheduling cases. *Ieee Access* 6 (2018), 26231–26240.

[8] Imed Kacem, Slim Hammadi, and Pierre Borne. 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)* 32, 1 (2002), 1–13.

[9] Nilsen Kundakcı and Osman Kulak. 2016. Hybrid genetic algorithms for minimizing makespan in dynamic job shop scheduling problem. *Computers & Industrial Engineering* 96 (2016), 31–51.

[10] Jun-qing Li, Jia-wen Deng, Cheng-you Li, Yu-yan Han, Jie Tian, Biao Zhang, and Cun-gang Wang. 2020. An improved Jaya algorithm for solving the flexible job shop scheduling problem with transportation and setup times. *Knowledge-Based Systems* 200 (2020), 106032.

[11] Jun-Qing Li, Quan-Ke Pan, and M Fatih Tasgetiren. 2014. A discrete artificial bee colony algorithm for the multi-objective flexible job-shop scheduling problem with maintenance activities. *Applied Mathematical Modelling* 38, 3 (2014), 1111–1132.

[12] Xinyu Li and Liang Gao. 2016. An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. *International Journal of Production Economics* 174 (2016), 93–110.

[13] Shu Luo. 2020. Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. *Applied Soft Computing* 91 (2020), 106208.

[14] Tao Ning, Ming Huang, Xu Liang, and Hua Jin. 2016. A novel dynamic scheduling strategy for solving flexible job-shop problems. *Journal of Ambient Intelligence and Humanized Computing* 7, 5 (2016), 721–729.

[15] Ferdinando Pezzella, Gianluca Morganti, and Giampiero Ciaschetti. 2008. A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research* 35, 10 (2008), 3202–3212.

[16] Simone Riedmiller and Martin Riedmiller. 1999. A neural reinforcement learning approach to learn local dispatching policies in production scheduling. In *IJCAI*, Vol. 2. Citeseer, 764–771.

[17] Jamal Shahrabi, Mohammad Amin Adibi, and Masoud Mahootchi. 2017. A reinforcement learning approach to parameter estimation in dynamic job shop scheduling. *Computers & Industrial Engineering* 110 (2017), 75–82.

[18] Hongtao Tang, Rong Chen, Yibing Li, Zhao Peng, Shunsheng Guo, and Yuzhu Du. 2019. Flexible job-shop scheduling with tolerated time interval and limited starting time interval based on hybrid discrete PSO-SA: An application from a casting workshop. *Applied Soft Computing* 78 (2019), 176–194.

[19] Ling Wang, Gang Zhou, Ye Xu, Shengyao Wang, and Min Liu. 2012. An effective artificial bee colony algorithm for the flexible job-shop scheduling problem. *The International Journal of Advanced Manufacturing Technology* 60, 1-4 (2012), 303–315.

[20] Yi-Chi Wang and John M Usher. 2004. Learning policies for single machine job dispatching. *Robotics and Computer-Integrated Manufacturing* 20, 6 (2004), 553–562.

[21] J Wu, GD Wu, JJ Wang, et al. 2017. Flexible job-shop scheduling problem based on hybrid ACO algorithm. *International Journal of Simulation Modelling* 16, 3 (2017), 497–505.