Int j simul model 19 (2020) 1, 157-168
ISSN 1726-4529 Original scientific paper

A NOVEL SOLUTION TO JSPs BASED ON LONG SHORT-TERM MEMORY AND POLICY GRADIENT ALGORITHM

Ren, J. F.[*,**]; Ye, C. M.[*,#] & Yang, F.[***]
[*] School of Business, University of Shanghai for Science and Technology, Shanghai 200093, China
[**] School of Computer and Information Engineering, Henan University of Economics and Law, Zhengzhou 450018, China
[***] School of Management, Henan University of Chinese Medicine, Zhengzhou 450018, China
E-Mail: yechm@usst.edu.cn ([#] Corresponding author)

Abstract
Based on long short-term memory (LSTM) and policy gradient algorithm, this paper proposes a novel solution to the job-shop scheduling problems (JSPs). Firstly, two LSTM networks with identical structures were established, serving as the encoding and decoding networks, respectively. Next, a pointer network was introduced to determine the job with the highest priority in the current state, creating a job sequence. Another neural network (NN) was constructed to evaluate the current job sequence. The evaluation results were taken as the baseline of the policy gradient algorithm for reinforcement learning. Then, the job sequence was optimized and updated by gradient descent. The effectiveness of our method was demonstrated through contrastive experiments on benchmark problems.
(Received in October 2019, accepted in January 2020. This paper was with the authors 2 months for 1 revision.)

Key Words: Job-Shop Scheduling Problem (JSP), Long Short-Term Memory (LSTM), Pointer Network, Policy Gradient Algorithm

1. INTRODUCTION

Currently, the manufacturing industry is faced with a host of problems, ranging from overcapacity, shrinking profit margins to increasingly unhealthy competition. These problems pose a severe challenge to manufacturing enterprises. In addition, there are some unique problems with the manufacturing enterprises in China: the industrial structure is distorted, the demographic dividends are diminishing, and the labour cost is soaring. Almost every manufacturing enterprise in China is operating under the pressure of high cost and low output. Against this backdrop, the manufacturing industry has reached the consensus to reshape the development mode with information technology.

In the course of production, the complexity of scheduling directly bears on the cost and profit of the enterprise. The production process should be scheduled efficiently to make full use of machines, complete tasks on time and reduce the inventory cost. For manufacturing enterprises, many business processes (e.g. production, transport planning and network communication) can be abstracted into job-shop scheduling problems (JSPs), which are non-deterministic polynomial-time (NP) hard. This NP-hard problem severely bottlenecks the implementation of the production plan.

The JSPs have long been a research hotspot among researchers and technicians. Akram and Kamal [1] hybridized simulated annealing (SA) with quenching to solve the JSPs. Aiming to maximize the makespan, the hybrid approach executes the quenching process as per the quenching plan, performs an enhanced search on the hyperplane of the local hyper-fitness function, and converges to the global optimum solution by increasing the number of iterations. Considering various factors related to job sequence (e.g. machines and unexpectedness), Roshanaei et al. [2] developed a mixed-integer linear programming model to solve the JSPs, minimized the maximum makespan by adaptively modifying a metaheuristic algorithm, and proved the superiority of the algorithm through experiment. Based on tabu search, Gonzalez et al. [3] designed an advanced scatter search algorithm to solve the JSPs, and attributed the good performance of the algorithm to a new neighbourhood structure, which relies on a non-anticipatory feature map with set time. Pongchairerks [4] proposed a two-level meta-heuristic algorithm for the JSPs. On the upper level, a population-based algorithm controls the parameters of the lower level; on the lower level, a local search algorithm searches for the optimal schedule in the solution space.

Many scholars have introduced neural networks (NNs) to solve the JSPs. For example, Foo and Takefuji [5] was the first to apply the Hopfield neural network to the JSPs, creating a model with the structure of a stochastic neural network (SNN). Yang and Wang [6] solved the JSPs with a constrained adaptive neural network and several heuristic algorithms. Jain and Meeran [7] improved the adaptability of an NN with a trained mechanism of backward error propagation, and effectively solved the JSPs with a Hopfield neural network. Yang and Wang [8] combined an NN with a heuristic algorithm to adaptively adjust the connection weights, and verified the feasibility of the adjustment strategy through simulation. Taking the genetic algorithm (GA) as the benchmark, Weckman et al. [9] designed an NN-based hybrid intelligent system to create an optimal job-shop scheduler, which maps the job operations to the decision features and solves the scheduling with the trained NN. Silva et al. [10] predicted the makespan and delivery deadline (DD) of a hypothetical dynamic job-shop, using an artificial neural network (ANN), constructed an allocation model based on the ANN and two dynamic allocation rules, and confirmed the effectiveness of the model through contrastive experiment.

With the development of computing power and artificial intelligence (AI), sequence problems have been successfully solved by recurrent neural networks (RNNs), shedding new light on the solution to the JSPs. Considering the excellence of deep NNs in complex learning, Sutskever et al. [11] presented an end-to-end sequential learning method with the fewest hypotheses on sequence structure, in which the input sequence is mapped to a fixed-dimension vector by a multilayer long short-term memory (LSTM), and the target sequence is decoded from the vector by another deep LSTM. Fonseca and Navaresse [12] created an innovative ANN that can effectively replace traditional methods in job-shop simulation: the ANN was trained by the reverse error propagation mechanism of the multilayer neural network meta-model, producing a general JSP analysis framework; the ANN-based simulation effectively captured the relationship between the machine sequence of a job and the mean flow time of the sequence.

In addition, heuristic algorithms have achieved good results in solving the JSPs. For instance, Mnih et al. [13] applied the deep reinforcement learning (DRL) successfully in the application of Atari games. The DRL was proved effective in solving to continuous and discrete tasks, through the training with a variant of the asynchronous actor-critic framework. Pfau and Vinyals [14] tackled the optimization problem related to generative adversarial networks (GANs) and reinforcement learning, summed up the generalization strategies of the two types of models, and came up with a scalable and stable multi-level optimization algorithm based on deep networks. Zhang et al. [15] combined particle swarm optimization (PSO) and the NN to solve the JSPs. Each particle in the swarm was considered a connection in the NN, and the connection weights were updated iteratively according to the latest positions of the corresponding particles. In this way, the NN no longer falls into the local optimum trap, and applies to the minimization of the maximum makespan of a single-objective JSP.

There are two major drawbacks of the heuristic algorithms: limited solution quality and poor adaptability. The solution quality of a heuristic algorithm is limited, because the algorithm must traverse the solution space by the heuristic rules of the problem. The heuristic algorithm lacks adaptability, in that any change to the problem or problem scale requires a modification of the algorithm. Moreover, most existing deep or reinforcement learning algorithms are adopted to handle problems like chess games and travelling salesman problem (TSP). Only a few studies have applied deep or reinforcement learning algorithms to solve the JSPs. In these studies, deep or reinforcement learning either adopts heuristic algorithms or mimics the search process of heuristic algorithms. However, no meaningful result has been achieved by these studies.

This paper puts forward a novel solution to the JSPs, which fully exploits the structural features of deep learning and reinforcement learning. Without requiring the heuristic rules of the problem, our solution maps the JSP to a problem that suits the learning framework, and conducts reinforcement learning by strategy gradient optimization [16-18]. Besides, the common features of jobs were mined through learning and training, and used to enhance the adaptability of our solution. The research results provide new insights into the JSPs.

2. PROBLEM DESCRIPTION

Being the hardest NP-complete problems, the JSPs are difficult to solve in polynomial time. In common JSPs, there is a unique machine, which is known in advance, for each operation. A flow shop scheduling problem (FSSP) is a special case of the JSP, while a flexible JSP (FJSP) is an extension of the JSP, in which a number of machines are optional for each operation. Thus, the solution to the JSP can be easily extended to relevant scheduling problems.

In general, a JSP can be described as using m machines to process n independent jobs, each of which needs to go through h operations. The following constraints must be satisfied to solve the JSP:

Constraint 1. The operations of different jobs are independent of each other, i.e. there is no sequential constraint between operations. The operation sequence and makespan of each job are known in advance, and remain the same throughout the production.

Constraint 2. The processing of each job starts at time 0, and no interruption is allowed once the processing starts.

Constraint 3. Each machine can only process one job at a time. The subsequent operation cannot begin before the completion of the current operation. The machine failure is as rare as negligible.

Constraint 4. Each job can only be processed on one machine at a time, and can be processed only once by each machine.

Let $J = \{1, 2,\ldots, n\}$ be the set of jobs, $M = \{1, 2,\ldots, m\}$ be the set of machines, and $O = \{1, 2,\ldots, O\}$ be the set of operations, where O_{ij} is the j^{th} operation of the i^{th} job. The set of operations O can also be defined as the collection of the operation subset of each job (the subset of the operations of each job).

For a given job-shop scheduling task, it is necessary to search the space of all possible operation sequences, and find the operation sequence that meets the objectives under various constraints.

Unlike heuristic algorithms, our solution is a data-driven, end-to-end approach to solve the JSP. In our solution, the constraint relationships between the operation sequences are learned under a deep reinforcement learning framework. Without following heuristic rules, the objectives of the JSP were achieved by optimizing and training the learning framework parameters, and solving new problems with the trained framework. Our solution boasts a high adaptability, for the heuristic search strategy of the trained framework is not limited to a specific problem.

Machine learning falls into three categories: supervised learning, unsupervised learning, and reinforcement learning. Over the years, supervised learning has attracted more attention than the other two types, especially in such fields as gaming, image recognition and voice recognition. For two reasons, supervised learning has not been widely applied in combinatory optimization of the JSPs: (1) It is costly and difficult to obtain the labels of samples; (2) Considering the various evaluation indices for job-shop scheduling, different targets should be evaluated by the mixed use of sample labels, which prolongs the training time and slows down convergence. Experiments have shown that supervised learning is not as effective as reinforcement learning in solving the JSPs.

In the light of the features of reinforcement learning, the difference between operation sequences was taken as a feedback reward signal, and a hybrid solution for the JSPs was designed based on both deep learning and reinforcement learning. In the hybrid solution, the strategy gradient optimization is performed to refine the parameters of a RNN, and the target of job-shop scheduling target was treated as a long-term reward. Next, the Monte Carlo tree search (MCTS) was added to improve the learning performance.

3. RNN FRAMEWORK

This paper attempts to solve the JSP with deep reinforcement learning. The set of operations was modelled as a sequence-to-sequence problem, i.e. the sequential relationship between operations was determined based on their constraint relationship.

As shown in Fig. 1, the NN framework of deep reinforcement learning consists of an encoding network and a decoding network. The two networks are LSTM RNNs with the same structure, and connected by a pointer network.

In each time step, the decoding network outputs the most probable operation and feedbacks the information on the output operation via the pointer network to the corresponding time step of the encoding network. Upon receiving the information, the encoding network will take it as the input for the next time step.

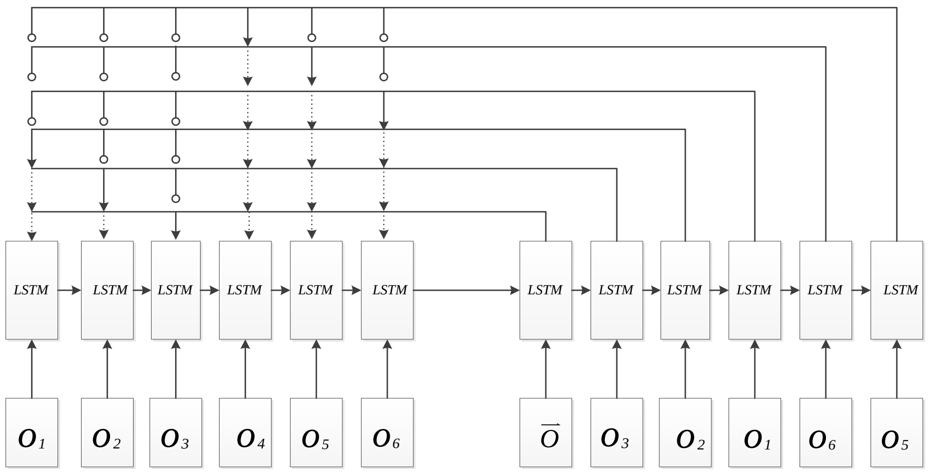

Figure 1: The NN framework of deep reinforcement learning.

All the operations in O are imported to the LSTM of the encoding network. In each time step, the d-dimensional embedding vector $\bar{O}$ of an operation is inputted into the LSTM, and the operation is converted into a d-dimensional memory state sequence $\{enc_i\}_{i=1}^{n}$. Meanwhile, the corresponding unit state is outputted for the next time step.

For the encoding network, three items are inputted in the next time step: the output of the previous time step, the embedding vector of another operation, and the most probable operation (acquired in the decoding network and fed back via the pointer network). The first step of encoding needs two inputs, namely, the embedding vector of the first operation and the most probable operation.

For the decoding network, the input in the first time step is the d-dimensional memory state sequence $\{dec_i\}_{i=1}^{n}$ from the encoding network. Each step of decoding has two inputs and one output. However, the inputs in the first step include the unit state of the last step of the encoding network, and the d-dimensional embedding vector of the operation. The data in each dimension of the vector can be used as a set of trainable parameters. Each step of decoding generates an operation with the highest pointing probability in the current state, serving as the input of the next step of decoding.

4. POINTER NETWORK

The pointer network is a simplified version of the attention mechanism. Coupled with a RNN, the pointer network can effectively solve combinatorial optimization problems by mapping them to sequence problems. The coupled strategy outshines the pointer networks proposed by Vinyals et al. [19] and Bahdanau et al. [20] in two aspects: (1) the operation to be outputted can be positioned, replacing the simple index output; (2) the computing complexity is greatly reduced by the first-step calculation of the attention mechanism.

In general, the minimal maximum makespan of the JSP can be taken as the evaluation target:

$$C(\pi|O) = \min\{\max C_i,\ i = 1, 2, \ldots, n\} \tag{1}$$

where, C_i is the makespan of job J_i.

The JSP can be considered a special sequence-to-sequence problem. If the stochastic strategy $p(\pi|O)$ is parameterized (i.e. probabilities are assigned to the operations in O), then the operation with shorter maximum makespan will be assigned higher probability. The probability assignment ensures that each step of decoding outputs the most probable operation:

$$p(\pi\,|\,O) = \prod_{i=1}^{n} p(\pi_i\,|\,\pi_{1:i-1}, O) \tag{2}$$

Drawing on the attention mechanism, the non-parameterized *softmax* function can be adopted to express the pointing result of the current operation as a probability. Hence, the operation position of the encoding network is determined through the attention mechanism.

In our deep reinforcement learning framework, the reference vectors of the attention mechanism can be described as $ref = \{enc_1, enc_2, \ldots, enc_f\}$, and the query vector of the attention mechanism as $q = dec_i$. Then, the attention scoring function can be expressed as an additive model:

$$s(enc_i, q) = v^T \cdot tan(W_{ref} \cdot enc_i + W_q \cdot q) \tag{3}$$

where, v is the attention vector of the d-dimensional scoring function; W_{ref} and W_q are the attention matrices of the $d \times d$-dimensional scoring function. v, W_{ref} and W_q are trainable parameters.

Note that, in a computing cycle, a job can only be processed once on a machine at a time, according to the JSP constraints. Thus, the score of the operation pointed by the pointer network must be set to infinitely small, such that the same operation will not be pointed at again.

Next, the attention distribution of each operation can be computed by:

$$\begin{aligned} att_i &= softmax(s(enc_i, q)) \\ &= \frac{exp(s(enc_i, q))}{\sum_{j=1}^{N} exp(s(enc_J, q))} \end{aligned} \tag{4}$$

The following can be derived from Eq. (5):

$$
\begin{aligned}
p(\pi \mid \boldsymbol{O}) &= \prod_{i=1}^{n} p(\pi_i \mid \pi_{1:i-1}, \boldsymbol{O}) \\
&= softmax(s(enc_i, q)) \\
&= \frac{exp(s(enc_i, q))}{\sum_{j=1}^{N} exp(s(enc_J, q))}
\end{aligned}
\tag{5}
$$

5. ATTENTION MECHANISM

Facing the JSPs, the heuristic algorithms have a common defect: their performance will degrade significantly if the problem is large, resulting in premature convergence or non-convergence. This defect is inherited by our solution, which adopts the deep reinforcement learning mechanism. To solve the defect, a viable option is to calculate the weighted mean of query vectors based on the attention distribution vector of the pointer network, i.e. to sum up the input information. This option helps to reveal the potential correlations between the operations.

The attention distribution function has the same structure and input as Eq. (4). The only difference is that the attention vector v^{att} and attention matrices W_{ref}^{att} and W_q^{att} of the function are independent trainable parameters:

$$
S(enc_i, q) = v^{att^T} \cdot tan(W_{ref}^{att} \cdot enc_i + W_{ref}^{att} \cdot q) \tag{6}
$$

$$
\begin{aligned}
ATT_i &= softmax(S(enc_i, q)) \\
&= \frac{exp(S(enc_i, q))}{\sum_{j=1}^{N} exp(S(enc_J, q))}
\end{aligned}
\tag{7}
$$

The degree of attention paid to an operation in the current time step can be described as an information summary function:

$$
Glimps(ref, q; W_{ref}^{att}, W_q^{att}, v^{att}) = \sum_{i=1}^{f} enc_i \cdot ATT_i \tag{8}
$$

Since the attention vector v^{att} and the attention matrices W_{ref}^{att} and W_q^{att} are trainable parameters, an iterative calculation method can be introduced:

$$
G_0 = enc_i \tag{9}
$$

$$
G_p = Glimps(ref, G_{p-1}; W_{ref}^{att}, W_q^{att}, v^{att}) \tag{10}
$$

The final attention vector G_p obtained by Eq. (10) is fed to the next time step in parallel with the hidden unit state of the current time step.

6. STRATEGY GRADIENT OPTIMIZATION

The parameters of the learning framework were optimized by a model-free strategy gradient algorithm in reinforcement learning. First, the strategy gradient method was implemented to find a set of optimal parameters of the pointer network. Then, the expectation of operation sequence for the objective function in the JSP was minimized:

$$
X(\theta \mid \boldsymbol{O}) = \min_{\theta} E_{\pi \sim \mathrm{p}(\cdot|o)} C(\pi \mid \boldsymbol{O}) \tag{11}
$$

Thus, the objective function can be defined as:

$$
X(\theta) = E_{o \sim O} C(\theta \mid \boldsymbol{O}) \tag{12}
$$

Suppose the distribution function $p_\theta(O)$ of operation set O is differentiable, the integral form of Eq. (12) can be updated as:

$$
X(\theta) = \int p_\theta(o) C(o) d_o \tag{13}
$$

The gradient can be expressed as:

$$\nabla_\theta X(\theta) = \int \nabla_\theta p_\theta(o) C(o) d_o \tag{14}$$

According to the logarithmic derivative identity $\nabla logf(x) = \frac{\nabla f(x)}{f(x)}$, we have:

$$p_\theta(o)\nabla_\theta \log p_\theta(o) = p_\theta(o)\frac{\nabla_\theta p_\theta(o)}{p_\theta(o)} = \nabla_\theta p_\theta(o) \tag{15}$$

Substituting Eq. (15) into Eq. (14), the stochastic gradient descent formula that optimizes the reinforcement learning parameters can be defined as:

$$\nabla_\theta X(\theta|O) = \int p_\theta(o)\nabla_\theta p_\theta(o) C(o) d_o = E_{\pi\sim \mathrm{p}_\theta(o)}\left[C(\pi|o)\nabla_\theta logp_\theta(\pi|o)\right] \tag{16}$$

When the JSP is solved by deep reinforcement learning, if the learning framework is trained by small-scale samples, the algorithm may face difficulty in convergence, due to the excessively large gradient in strategy optimization. Therefore, a baseline should be introduced to improve Eq. (16) as:

$$\nabla_\theta X(\theta|O) = E_{\pi\sim \mathrm{p}_\theta(o)}\left[(C(\pi|o) - b(o))\nabla_\theta logp_\theta(\pi|o)\right] \tag{17}$$

The addition of the baseline reduces the variance of the gradient in strategy optimization, without affecting the value of $\nabla_\theta X(\theta|O)$. Without loss of generality, it is assumed that $b(o)$ is a constant. Then, the last term in Eq. (17) should be proved as zero.

Proof:

$$\begin{aligned} & E_{\pi\sim \mathrm{p}_\theta(o)}\left[\nabla_\theta logp_\theta(\pi|o) b(o)\right] \\ & = \int p_q(o)\nabla_\theta logp_\theta(o) b(o) d_o \\ & = \int \nabla_\theta p_\theta(o) b(o) d_o \\ & = b(o)\nabla_\theta \int p_\theta(o) d_o \end{aligned}$$

The value of $\int P_\theta\,(o) d_o$ is obvious one. Thus, ∇_θ^l must be zero. Then, the last term in Eq. (17) must be zero.

As shown in Fig. 2, two LSTMs and two fully-connected layers were designed to obtain the baseline needed to solve the JSPs. Parallel computing was performed on the hidden results of the two LSTM layers, followed by the calculations on the fully-connected layers. The resulting scalar value was taken as the baseline. The *Tanh* activation function was adopted in the fully-connected layers.

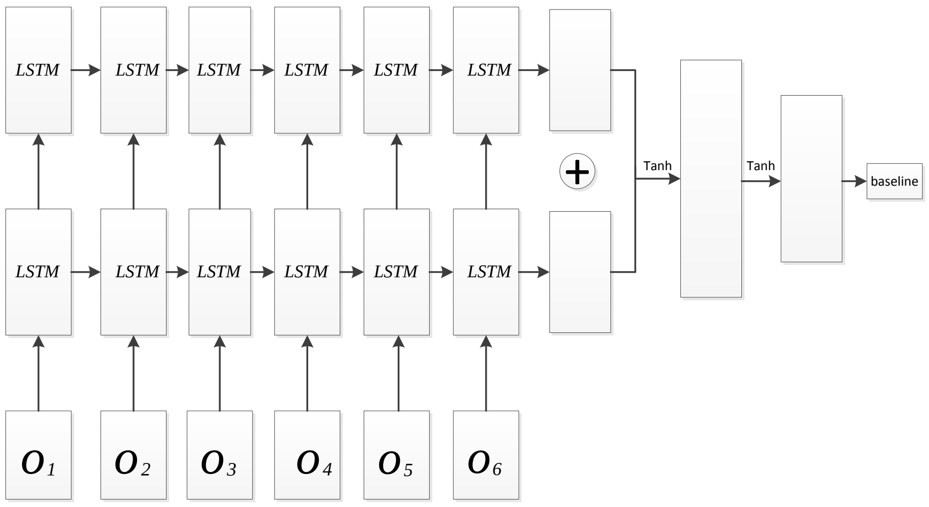

Figure 2: Baseline solution network.

The actor-critic algorithm is a reinforcement learning method based on strategy gradient optimization. The actor uses the strategy function to generate actions and interact with the environment, while the critic uses the value function to evaluate the actor's performance and corrects the next action of the action. The actor-critic algorithm has stringent requirements on the state sequence, and iteratively updates the policy function separately. As a result, the algorithm often has difficulty in convergence.

To overcome the difficulty, this paper proposes the asynchronous advantage actor-critic (A3C) algorithm (Algorithm 1) to reduce the correlation between empirical data of scheduling and enhance the training result. To cope with the complexity of the JSP, the A3C implements multi-threaded learning through the interaction with the environment, aiming to achieve asynchronous and simultaneous learning. The parameters of the strategy function and the value function were updated by gradient ascent and gradient descent, respectively.

Algorithm 1: The A3C for the JSP

```
Initialize parameters $\theta$ and $\theta_b$
Initialize step counter $t \leftarrow 1$
Repeat
Set $d\theta \leftarrow 0$ and $d\theta_b \leftarrow 0$
Get current state $s_t$
Perform $a_t$ according to $\pi(a_t \mid s_t; \theta')$
Receive rewards $r_t$ and $s_{t+1}$
Until terminal state or the maximum number of iterations
Do Loop
 $R \leftarrow r_i + \gamma R$
 $d\theta \leftarrow d\theta + \nabla_\theta \log p_\theta(\pi \mid o)(C(\pi|o) - b(o))$
 $d\theta_b \leftarrow d\theta_b + \partial(C(\pi \mid o) - b(o_i; \theta_b'))/\partial\theta_b'$
End Loop
```

In the A3C, the results $C(\pi|o)$ and $b(o)$ are solved by the pointer network and the baseline solution network, respectively, and then substituted into Eq. (17). During the calculation of the scheduling algorithm, the $C(\pi|o)$ is performed by Monte Carlo sampling in each small batch, and the baseline calculation is performed by the common exponential moving average (EMA) method:

$$\nabla_\theta X(\theta) \approx \frac{1}{N}\sum_{1}^{N}\left[(C(\pi_i|o_i) - b)\nabla_\theta \log p_\theta(\pi_i|o_i)\right] \tag{18}$$

where, N is the size of a small batch.

$$b = \alpha * b + (1-\alpha) * (\frac{1}{N}\sum_{1}^{N} b_i) \tag{19}$$

where, α is the smoothing index. The workflow of the A3C algorithm is shown in Fig. 3.

7. CONTRASTIVE EXPERIMENTS

7.1 Key parameters

Considering its importance in our solution, the parameters of the LSTM network must be configured carefully before verifying the performance of the proposed algorithm. During the experiments, the information of a job is inputted in each time step, a new internal state is formed through gate setting, and the linear information is transferred cyclically to provide the long-term memory of the encoding network. The nonlinear output of each time step was taken as the external state of the hidden layer. Meanwhile, once the new job information was inputted in each time step, the internal state and input information were converted to nonlinear form to obtain the candidate state. After the information of the last job was inputted, the network formed the memory of a complete internal state, facilitating the feature mapping between jobs. This memory is very helpful to solve complex JSPs.

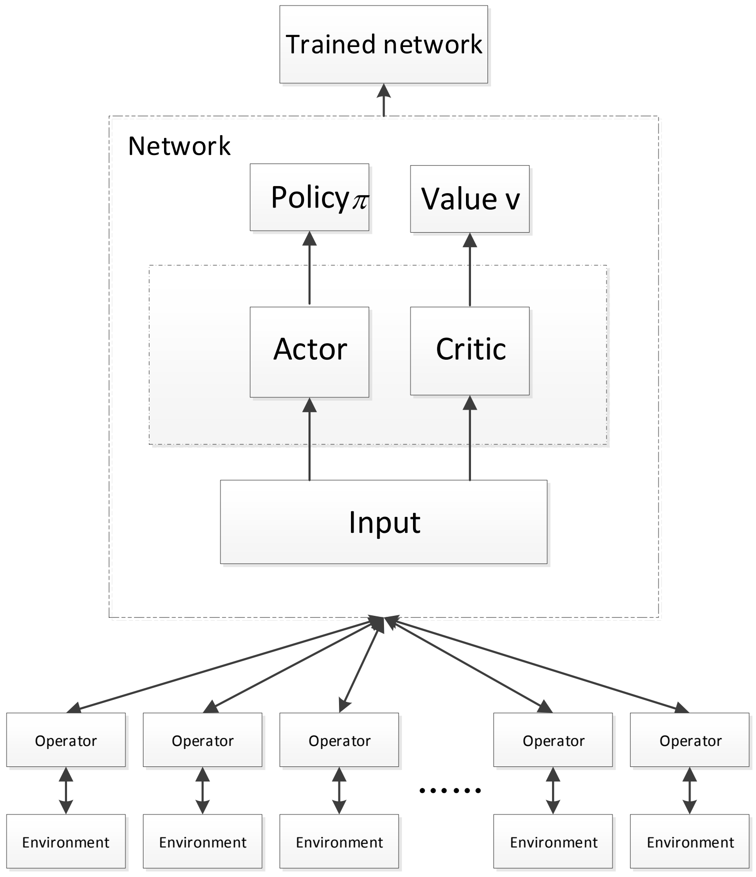

Figure 3: The flow chart of the A3C algorithm.

Our experiments verify the hyper-parameters of the hidden cells of the LSTM. For comparison, the number of hidden units was set to 128, 256, and 512, respectively, in different rounds. Finally, the number of hidden units of the LSTM cells was set to 256, i.e. each job was converted into a 256-dimensional embedded vector by the encoding network.

The training of the LSTM network shows that an excessively small value will reduce the value of the forget gate, i.e. only a small portion of the information in the previous time step has been memorized. Then, it is difficult for the network to capture long-range dependent information. Besides, the excessively small gradient between adjacent time intervals would cause the gradient dispersion problem. For this reason, the forgotten parameter was initialized within [0.8, 1.0] and its offset was set to 1.

In addition, *sigmoid* function was selected as the activation function of the forget gate, input gate, and output gate in the LSTM, while the *softsign* function was taken as the unit activation function for hidden state calculation. The *softsign* function can output a flatter curve, and slow down the falling of the derivative, making the learning more efficient and avoiding the vanishing gradient problem. Finally, the initial learning rate was set to 10^{-5}.

7.2 Experimental procedure and results analysis

To verify its effectiveness, our algorithm was applied to solve 15 benchmark problems, namely, LA16-20, TA36-40, and TA46-40. For comparison, the reinforcement learning uses the REINFORCE and A3C architectures separately in different rounds, and a glance mechanism (G) was added to the learning part of the pointer network. The benchmark problems were solved separately by the improved tabu search-simulated annealing (TSSA) and the improved genetic algorithm-tabu search (GATS); the better of the two results was taken as the baseline.

Each benchmark problem was run 30 times, and the mean value of each problem was taken as the final result (Table I). It can be seen that the A3C with G achieved the best performance: the results of LA16, LA18, TA36 and TA37 were close to or the same with the result of the benchmark example; the results of TA40, TA47, TA48 and TA50 fell between those of TSSA and GATS; the results of other examples were slightly better than those of TSSA. Overall, our algorithm performed roughly the same as TSSA, and slightly worse than GATS.

Table I: The results on benchmark problems.

Benchmark problems	Scale	LSTM-RL				TSSA	GATS
		REINFORCE – with G	A3C – with G	REINFORCE – no G	A3C – no G		
LA16	10×10	946.2	945.0*	950.4	948.1	945	945
LA17	10×10	785.6	784.5	795	788.6	784	784
LA18	10×10	850.2	848.2	855.1	852.1	848	848
LA19	10×10	846.4	843.0	850.9	848.9	842	842
LA20	10×10	906.2	903.5	912.4	909.7	902	902
TA36	30×15	1,822.2	1,819.3	1,835.3	1,827.6	1,819	1,819
TA37	30×15	1,780.1	1,778.0*	1,789.4	1,783.4	1,778	1,771
TA38	30×15	1,675.2	1,674.0	1,675.3	1,674.9	1,673	1,673
TA39	30×15	1,797.4	1,797.2	1,805.2	1,800.1	1,795	1,795
TA40	30×15	1,679.3	1,676.2+	1,685.3	1,684.1	1,676	1,673
TA46	30×20	2,018.5	2,012.4	2,022.8	2,020.9	2,010	2,009
TA47	30×20	1,908.2	1,904.4+	1,917.3	1,915.6	1,903	1,898
TA48	30×20	1,963.9	1,958.2+	1,974.0	1,970.4	1,955	1,946
TA49	30×20	1,973.0	1,968.4	1,980.2	1,977.4	1,967	1,965
TA50	30×20	1,938.3	1,933.1+	1,950.8	1,945.2	1,931	1924

Note: * indicates that the result reaches the same level as the results of TSSA and GATS;
+ indicates that the result falls between the results of TSSA and GATS.

Fig. 4 compares the results of our algorithm with the benchmark results. Thirty results of each benchmark example were used. Considering the scale difference between the problems, the performance of our algorithm was verified by the relative error between the result of our algorithm and that of the benchmark problem.

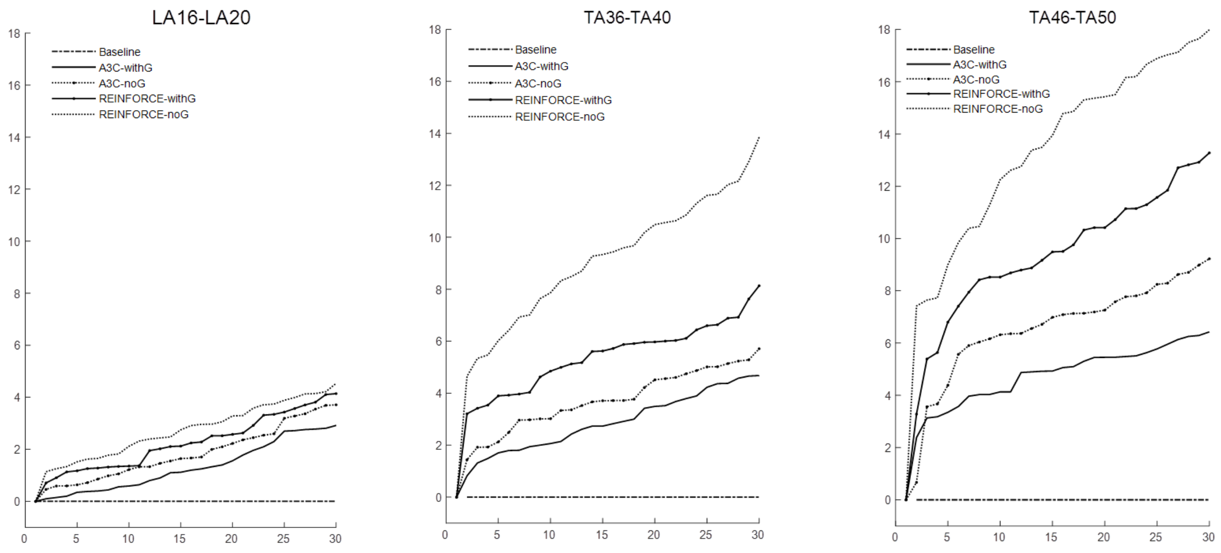

Figure 4: Comparison with benchmark results.

Among the examples, LA16-20 are relatively small in scale. The relative error between $A3C-withG$ and these five benchmark examples was the smallest. With the growing complexity of problem, the relative error exhibited a slow increase. The relative errors of three other algorithms all fell in the acceptable range. These algorithms could be ranked as $A3C-noG$, $REINFORCE-withG$, and $REINFORCE-noG$, in ascending order of relative error. The experimental results show that the A3C algorithm had an obvious advantage over REINFORCE in solving JSPs, and that the addition of glance mechanism to the pointer network could improve the learning ability of the reinforcement learning framework.

For medium-large scale examples (e.g. TA36-40), the relative errors of $REINFORCE-withG$ and $REINFORCE-noG$ relative to benchmark results increased much faster than those of $A3C-withG$ and $A3C-noG$. This means the A3C remained superior over the REINFORCE algorithm in solving largescale scheduling problems.

8. CONCLUSIONS

The heuristic algorithms are often adopted to solve the JSPs. This paper explores the search patterns and constraints of these algorithms in solving the JSPs. Thanks to the rapid advances in ANN, reinforcement learning, and computing techniques, the latest results of the AI technology were absorbed to solve combinatory optimization, sequence decision-making, and job-shop scheduling. Besides, the authors introduced the LSTM, pointer network, and strategy gradient optimization to the JSPs. The long-term memory of the LSTM was effectively employed to acquire and store the digital features and correlations between jobs, while a pointer network was adopted to determine the priority probability distribution of the jobs in the current state, creating an effective scheduling sequence. To improve the solution quality, an NN was constructed to predict the baseline in parallel with the strategy gradient optimization, thereby reducing the optimization variance and enhancing convergence efficiency. In addition, a glance mechanism was added to the pointer network to further improve solution quality.

Our algorithm provides a brand-new method to solve the JSPs, while avoiding the difficulty in sample collection through supervised learning. The future research will further streamline the algorithm structure, improve solution quality, and minimize the deviation from the benchmark results, laying a solid basis for intelligent manufacturing.

ACKNOWLEDGEMENT

The study was supported by National Natural Science Foundation, China (No. 71840003); Technology Development Project of University of Shanghai for Science and Technology, China (No. 2018KJFZ043).

REFERENCES

[1] Akram, K.; Kamal, K. (2015). Hybridization of simulated annealing with quenching for job shop scheduling, *Proceedings of the 2015 International Conference on Fluid Power and Mechatronics*, 825-829, doi:10.1109/FPM.2015.7337228

[2] Roshanaei, V.; Balagh, A. K. G.; Esfahani, M. M. S.; Vahdani, B. (2010). A mixed-integer linear programming model along with an electromagnetism-like algorithm for scheduling job shop production system with sequence-dependent set-up times, *International Journal of Advanced Manufacturing Technology*, Vol. 47, No. 5-8, 783-793, doi:10.1007/s00170-009-2210-9

[3] Gonzalez, M. A.; Vela, C. R.; Varela, R.; González-Rodríguez, I. (2015). An advanced scatter search algorithm for solving job shops with sequence dependent and non-anticipatory setups, *AI Communications*, Vol. 28, No. 2, 179-193, doi:10.3233/AIC-140631

[4] Pongchairerks, P. (2019). A two-level metaheuristic algorithm for the job-shop scheduling problem, *Complexity*, Vol. 2019, Paper 8683472, 11 pages, doi:10.1155/2019/8683472

[5] Foo, S. Y.-P.; Takefuji, Y. (1988). Stochastic neural networks for solving job-shop scheduling. I. problem representation, *Proceedings of the IEEE 1988 International Conference on Neural* Networks, 275-282, doi:10.1109/ICNN.1988.23939

[6] Yang, S. X.; Wang, D. W. (1999). Using constraint satisfaction adaptive neural network and efficient heuristics for job-shop scheduling, *Information and Control*, Vol. 28, No. 2, 121-126, doi:10.3969/j.issn.1002-0411.1999.02.009

[7] Jain, A. S.; Meeran, S. (1998). Job-shop scheduling using neural networks, *International Journal of Production Research*, Vol. 36, No. 5, 1249-1272, doi:10.1080/002075498193309

[8] Yang, S.; Wang, D. (2000). Constraint satisfaction adaptive neural network and heuristics combined approaches for generalized job-shop scheduling, *IEEE Transactions on Neural Networks*, Vol. 11, No. 2, 474-486, doi:10.1109/72.839016

[9] Weckman, G. R.; Ganduri, C. V.; Koonce, D. A. (2008). A neural network job-shop scheduler, *Journal of Intelligent Manufacturing*, Vol. 19, No. 2, 191-201, doi:10.1007/s10845-008-0073-9

[10] Silva, C.; Ribeiro, V.; Coelho, P.; Magalhães, V.; Neto, P. (2017). Job shop flow time prediction using neural networks, *Procedia Manufacturing*, Vol. 11, 1767-1773, doi:10.1016/j.promfg.2017.07.309

[11] Sutskever, I.; Vinyals, O.; Le, Q. V. (2014). Sequence to sequence learning with neural networks, Ghahramani, Z.; Welling, M.; Cortes, C.; Lawrence, N. D.; Weinberger, K. Q. (Eds.), *Advances in Neural Information Processing Systems*, Curran Associates Inc., New York, 3104-3112

[12] Fonseca, D. J.; Navaresse, D. (2002). Artificial neural networks for job shop simulation, *Advanced Engineering Informatics*, Vol. 16, No. 4, 241-246, doi:10.1016/S1474-0346(03)00005-3

[13] Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Harley, T.; Lillcrap, T. P.; Silver, D.; Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning, *Proceedings of the 33rd International Conference on Machine Learning*, 1928-1937

[14] Pfau, D.; Vinyals, O. (2016). Connecting generative adversarial networks and actor-critic methods, *arXiv:1610.01945 [cs.LG], NIPS Workshop on Adversarial Training*, Barcelona, 10 pages

[15] Zhang, Z.; Guan, Z. L.; Zhang, J.; Xie, X. (2019). A novel job-shop scheduling strategy based on particle swarm optimization and neural network, *International Journal of Simulation Modelling*, Vol. 18, No. 4, 699-707, doi:10.2507/IJSIMM18(4)CO18

[16] Yang, Z.; Liu, C. (2018). A multi-objective genetic algorithm for a fuzzy parallel blocking flow shop scheduling problem, *Academic Journal of Manufacturing Engineering*, Vol. 16, No. 2, 3-11

[17] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiler, 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

[18] Zhu, J.; Shao, Z. H.; Chen, C. (2019). An improved whale optimization algorithm for job-shop scheduling based on quantum computing, *International Journal of Simulation Modelling*, Vol. 18, No. 3, 521-530, doi:10.2507/IJSIMM18(3)CO13

[19] Vinyals, O.; Fortunato, M.; Jaitly, N. (2015). Pointer networks, *Proceedings of the 27th Conference Advances in Neural Information Processing Systems*, 2692-2700

[20] Bahdanau, D.; Cho, K.; Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate, *arXiv:1409.0473 [cs.CL]*, *Proceedings of the 2015 International Conference on Learning Representations*, 15 pages