Deep Reinforcement Learning Based Intelligent Job Batching in Industrial Internet of Things

Chengling Jiang, Zihui Luo, Liang Liu (✉), and Xiaolong Zheng

Beijing Key Lab of Intelligent Telecommunication Software and Multimedia, Beijing University of Posts and Telecommunications, Beijing, China
{jiangchengling,luozihui,liangliu,zhengxiaolong}@bupt.edu.cn

Abstract. The ever-developing Industrial Internet of Things (IIoT) is promoting the transformation of traditional manufacturers to intelligent manufacturing. Intelligent job batching, as one of the important parts of intelligent production in IIoT, is desired to group jobs with similar features into one batch under the constraint of batch capacity while considering the manufacturing target. This work formulates the job batching problem as a Markov Decision Process and proposes a deep reinforcement learning (DRL) based method to achieve intelligent job batching. The job batching model is based on the pointer network. The convergence of the model under different parameters is analyzed, and the performance of the method is evaluated by comparing it with the manual result, K-means algorithm, and multiple meta-heuristic algorithms via real production data. Experiments show that the proposed method can produce better solution in terms of the feature difference within a batch and the total batch number, especially in large-scale manufacturing scenarios.

Keywords: Job batching · Deep reinforcement learning · Industrial Internet of Things

1 Introduction

With the boom of Industrial Internet of Things (IIoT) [1, 3], the traditional industry is upgrading towards intelligent manufacturing. The realization of multi-variety and small-batch flexible production is an essential part of intelligent manufacturing. To improve the utilization of equipments and production efficiency, jobs with similar features are grouped into a batch for processing.

The job batching process is usually conducted by manpower in practical production. Due to the large number of jobs, unknown batch number, multiple job features and constraints, it is difficult for technicians to generate a reasonable solution in an efficient manner. As shown in Fig. 1, in the envisioned smart factory of IIoT, through the comprehensive perception of production data by massive intelligent devices, the real-time transmission of production data by wireless communication and wireless sensor network, and the rapid calculation and decision-making by job batching model in the cloud, the whole production process can be automatic, intelligent, and unmanned. Apparently, the results of

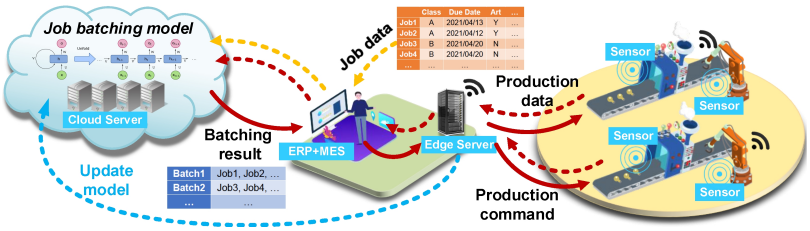


Fig. 1. The envisioned smart factory in Industrial Internet of Things

the batching model in the cloud determines the performance of production plan, which directly affects the production efficiency. To achieve the vision of IIoT, an efficient job batching model is desired. Existing studies on job batching mainly focus on clustering algorithm [15, 19] and meta-heuristic algorithm [5, 6, 10, 15, 20]. Both methods are not efficient in dealing with the batching problem with large number of jobs. Besides, the performance of both methods depends on human experience, which do not make use of the massive data to learn prior knowledge. Accordingly, designing an efficient and intelligent job batching method for IIoT scenario is of great urgency and practical significance.

The job batching problem can be regarded as a mapping from an input sequence of jobs to be batched to an output sequence of job batching result. Vinyals [18] proposed the pointer network base on the attention mechanism [8] to handle the sequence-to-sequence (Seq2Seq) scenario, which is consistent with the job batching process. However, the pointer network needs supervised training, whose performance heavily depends on the quality of labels. It is unrealistic to manually label massive data in IIoT scenario, making the supervised learning inapplicable. Reinforcement learning (RL) [17] is a data-driven method that can learns stable strategy by interacting with the environment without labels. However, each job has multi-dimensional features in job batching problem, which cannot be handled by RL. Recently, deep reinforcement learning (DRL) which combines the perceptual ability to process multi-dimensional features of deep learning (DL) [9] and the stable decision-making ability of RL, has successfully solved various practical problems [12–14, 21]. Therefore, this work adopts a DRL based method to approach the job batching problem based on pointer network.

The main contributions of this paper are as follows.

1) We formulate the job batching problem as a Markov Decision Process and employ a DRL based method which can process multi-dimensional input data and does not need labels to approach the job batching problem in IIoT.
2) We consider the job batching process as a mapping from one sequence to another and propose a job batching model based on pointer network, with the objective of minimizing the batch number and the feature difference within a batch under the batch capacity constraint.
3) We compare the proposed method with manual result, K-means algorithm and multiple meta-heuristic algorithms using real production data. The experimental results show the superiority of our method.

2 Problem Description and Mathematical Model

2.1 Notation

M	Total number of jobs to be batched.
N	Total number of batches (unknown in advance).
C	Maximum capacity of a batch.
K	Number of job features.
D	Total feature difference of all batches.
x_i	The job i.
w_i	Weight of job i.
f_k	The $k\,th$ feature of job.
c_k	Score of $k\,th$ feature of job.
α, β	Score of two sub objectives.
U_n	Set of jobs contained in batch n.
$\lvert U_n \rvert$	Number of jobs contained in batch n.
V_n	Remaining capacity of batch n.
μ_{in}	Decision variable: is job i in batch n.

2.2 Problem Formulation

In this paper, we mainly consider a typical job batching problem that must be faced under the multi-variety and small-batch flexible production in IIoT. Specifically, given a set of M jobs $X = \{x_i, i = 1, 2, \cdots, M\}$, each job x_i has K features $f_i = \{f_{ik}, k = 1, 2, \cdots, K\}$ such as the the delivery date, length, width etc. which are defined by the actual application scenarios, and a weight w_i. So each job is defined as $x_i = \{f_i, w_i\}$. The maximum capacity of a batch is C. The total batch number is unknown in advance. The purpose of job batching problem is to group jobs with similar features into a batch under the constraint of batch capacity, so as to minimize the feature difference D of all batches and the total number of batches N. The mathematical model is as follows.

$$\min target = \alpha D + \beta N \tag{1}$$

$$\text{where } D = \sum_{n=1}^{N} \sum_{i=1}^{|U_n|} \sum_{j \neq i}^{|U_n|} \sqrt{c_1(f_{i1} - f_{j1})^2 + c_2(f_{i2} - f_{j2})^2 + \cdots + c_k(f_{ik} - f_{jk})^2}$$

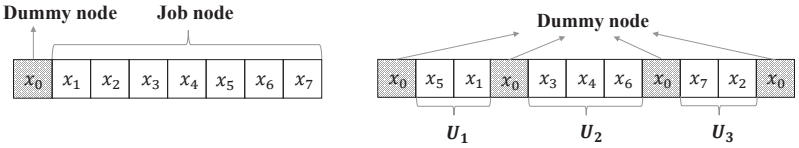
$$0 \leq \alpha \leq 1,\ 0 \leq \beta \leq 1 \tag{2}$$

$$\sum_{k=1}^{K} c_k = 1 \tag{3}$$

$$0 < \sum_{i \in U_i} w_i \leq C \tag{4}$$

$$0 < \sum_{n=1}^{N} \mu_{in} \leq 1, \text{where } \mu_{in} = \begin{cases} 1, \text{job } i \text{ belongs to batch } n \\ 0, \text{otherwise} \end{cases} \tag{5}$$

Equation (1) is the objective function; Constraint (2) and (3) represent the importance of the two sub objectives in Equation (1) and the importance of each feature respectively; Constraint (4) indicates that the total weight of jobs in a batch cannot exceed the maximum capacity; Constraint (5) indicates that one job can only be grouped into one batch.



(a) Input sequence (b) Output sequence

Fig. 2. A specific instance of input sequence and output sequence.

3 Job Batching Model

3.1 Markov Decision Process

This paper formulates the job batching problem as a Markov Decision Process (MDP). At each time step t, the agent observes current environment state S^t and take the action y_t. The agent can obtain the reward R from the environment according to its action. Then the environment state moves from S^t to S^{t+1}. The agent continuously interacts with the environment and batches jobs successively so as to obtain the maximum cumulative reward.

State. It is defined that when a job x_i is grouped into one batch, the weight of job x_i would change from w_i to 0 and the remaining capacity V_n of current batch n would change from C to $C - d_i$. So each job can be redefined as $x_i = \{f_i, w_i^t\}$, where f_i refers to the static features of job x_i, which remain unchanged at any time, w_i^t refers to the weight of job x_i, which changes with the action of agent. Accordingly, the current environment state at time t is $S^t = \{f, w^t, {V_n}^t\}$, where $f = \{f_i, i = 1, 2, \cdots, M\}$ refers to the static features of all jobs, $w^t = \{w_i^t, i = 1, 2, \cdots, M\}$ refers to the weights of all jobs at time t, and ${V_n}^t$ refers to the remaining capacity of the current batch n at time t.

Action. The dummy node $x_0 = \{f_0, w_0\}$ is defined to separate each batch, the static features and weight of the dummy node are always 0. The dummy node x_0 and all jobs x_i $(i = 1, 2, \cdots, M)$ serve as the input sequence. At each time step t, the action of agent is to select a node from the input sequence as the output node. The first output node is always dummy node, indicating the beginning of the batching. If the agent considers there are no available jobs to group into current batch, it would select the dummy node to end the current batch and start a new batch. When all jobs are batched, an output sequence can be obtained. A specific instance is shown in Fig. 2. The input sequence is x_i $(i = 0, 1, \cdots, 7)$. Note that the dummy node is always at the first position in the input sequence. The output sequence is $\{x_0, x_5, x_1, x_0, x_3, x_4, x_6, x_0, x_7, x_2, x_0\}$, indicating that jobs are grouped into 3 batches: $U_1 = \{x_5, x_1\}$, $U_2 = \{x_3, x_4, x_6\}$, $U_3 = \{x_7, x_2\}$.

Reward. The reward reveals the quality of the action taken by the agent under the current environment state. The goal of the agent is to minimize the total number of batches (N) and the sum of feature difference of all batches (D). The smaller the target value, the better job batching decisions the agent makes and the greater the reward to the agent. Accordingly, the reward R is designed

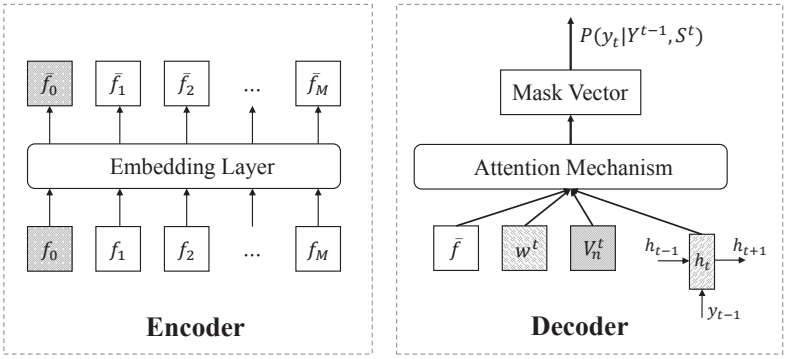


Fig. 3. Overall structure of job batching model

in Equation (6):

$$R = -target = -(\alpha D + \beta N) \tag{6}$$

3.2 Overall Structure of The Model

The overall structure of the job batching model proposed is shown in Fig. 3. It is based on the pointer network, which includes an encoder and a decoder.

Encoder. In pointer network, the encoder is implemented based on Recurrent Neural Network (RNN) [16]. However, RNN is meaningful only when the permutation of the input sequence conveys certain information. In the input sequence, except that the first node is dummy node, the order of other job nodes has no meaning. Therefore, the job batching model omits the RNN in the encoder, and directly use a one-dimensional convolution layer as the embedding layer to map the static features $f = \{f_i, i = 0, 1, \cdots, M\}$ of nodes to a matrix $\bar{f} = \{\bar{f}_i, i = 0, 1, \cdots, M\}$, which reduces the computational complexity.

Decoder. The decoder includes a Long Short-Term Memory (LSTM) [4, 23], an attention mechanism and a mask vector, the last two parts would be illustrated further in Section 3.3. At each time step t, the LSTM fetches hidden state h^{t-1} of LSTM and the output node y^{t-1}, and outputs the hidden state h^t. The output matrix $\bar{f}$ of the encoder, the dynamic weight feature w^t of all nodes in input sequence, the remaining capacity ${V_n}^t$ of the current batch n, and the hidden state h^t of LSTM are input to the attention mechanism. Then the probability distribution of each node can be obtained with the help of mask vector. Finally, the node with the highest probability is selected as the output node at time t. After the agent makes the decision at time t, the weights of jobs would be updated from w^t to w^{t+1}, the remaining capacity of the current batch would be updated from ${V_n}^t$ to ${V_n}^{t+1}$ and the mask vector would be updated according to the decision of agent. The updated vectors are input to the model at next time step $t + 1$.

Algorithm 1 DRL Based Job Batching Algorithm

initialize the parameters of actor network θ and critic network φ randomly
for $epoch = 1, 2, \cdots$ **do**
  reset gradients: $d\theta \leftarrow 0$, $d\varphi \leftarrow 0$
  sample J instances randomly from the training set
  for $j = 1, 2, \cdots, N$ **do**
    initialize time step $t \leftarrow 0$
    initialize the mask vector: $mask = [1, 0, \cdots 0]$
    construct the input sequence X_i $(i = 0, 1, 2, \cdots, M)$
    repeat
      observe state S_j^t, select output node y_t according to $P(y_t|\, Y^{t-1}, S^t)$
      update state $S_j^t \leftarrow S_j^{t+1}$, mask vector and time step $t \leftarrow t + 1$
    until all jobs are grouped into corresponding batches
    calculate actual reward R_j and the estimated reward $V(S_j^0; \varphi)$
  end for
  calculated $d\theta$ and $d\varphi$ and update actor network and critic network separately
end for

3.3 Attention Mechanism and Mask Vector

At each time step t, the attention mechanism calculates the probability distribution a^t of each node in the input sequence via Equation (7), where v_a and ω_a are trainable variables. The probability distribution serves as the pointer to the node in input sequence.

$$a^t = \text{Softmax}(v_a \tanh(\omega_a \left[\bar{f}; h^t; d^t; {V_n}^t\right])) \tag{7}$$

A mask vector which has the same length with the input sequence is introduced to constraint the agent decision. each element of the mask vector has one-to-one correspondence to the node in the input sequence. Each element value of the mask vector is either 0 or 1. The first element value of the mask vector which corresponds to the dummy node is always 1, enabling the agent to end the partition of the current batch at any time. At time step t, if one of the following situations are satified, the element values of the mask vector corresponding to the job node x_i $(i = 1, 2, \cdots, M)$ are set to zero: *a*) job node x_i is selected as the output node; *b*) the weight ${w_i}^t$ of job node x_i is larger than the remaining capacity ${V_n}^t$; *c*) at $t = 0$, the agent can only select the dummy node.

Combined with the mask vector, the final probability distribution of each node at time t is calculated in Equation (8), where v_b are trainable variable and a^t is probability distribution calculated by Equation (7). From Equation (8), If the element value of the mask vector is 0, the probability of the node corresponding to the mask vector element as the output node is also 0. Finally, the node with the highest probability is taken as the output node.

$$P(y_t|\, Y^{t-1}, S^t) = \text{Softmax}(v_b a^t - \ln(mask)) \tag{8}$$

3.4 DRL Based Algorithm

A DRL based job batching algorithm is employed to train the job batching model. It contains two network: an actor network and a critic network. The actor network is used to predict the output probability of each node in the input sequence at each time step t. Assuming the parameter of actor network is θ. The critic network is used to calculate the estimated reward of each input sequence. Assuming that the parameter of critic network is φ.

The specific steps of the DRL based job batching algorithm are shown in Algorithm 1. Firstly, the parameters of actor network and critical network are initialized randomly. At each epoch, J instances are sampled randomly from the training set (each instance is a set of M jobs). For each instance j, the time step t and mask vector is initialized, the input sequence is constructed. At each time step t, actor network observes current environment state S_j^t and selects the output node y_t according to Equation (8), then the environment state, the mask vector and the time step are updated accordingly. When all jobs are grouped into corresponding batches in instance j, the actual reward R_j and the estimated reward $V(S_j^0;\varphi)$ are calculated respectively. After the batching task of J instances are terminated, the gradients of actor network and critic network are calculated and updated according to Equation (9) and (10).

$$d\theta \leftarrow \frac{1}{J}\sum_{j=1}^{J}(R_j - V(S_j^0;\varphi))\nabla_\theta \log P(Y_j|\, S_j^0) \tag{9}$$

$$d\varphi \leftarrow \frac{1}{J}\sum_{j=1}^{J}\nabla_\varphi (R_j - V(S_j^0;\varphi))^2 \tag{10}$$

In Equation (9) and (10), S_j^0 refers to the environment state of jth instance at time step $t = 0$, Y_j refers to the output sequence with regard to S_j^0, R_j refers to the actual reward obtained in the jth instance, $P(Y_j|\, S_j^0)$ refers to the probability distribution of each node, $V(S_j^0;\varphi)$ refers to the estimated reward in the jth instance with regard to S_j^0.

4 Experimental Results

The experiments focus on two parts: analyzing convergence of the proposed method using different parameter settings; and comparing performance of the proposed method with manual result, K-means algorithm, and other meta-heuristic algorithms in terms of the total batch number, the feature difference of all batches, the target value, and the inference time.

4.1 Experimental Environment

The experiments are conducted on real production data provided by Nanjing Iron and Steel Co., Ltd. (NISCO). The production process of the steel plants in

(a) (b) (c) (d)

Fig. 4. Main production process and physical equipments of the steel plants in NISCO. (a) Iromaking; (b) Steelmaking; (c) Continuous casting; (d) Hot rolling.

NISCO mainly includes four stages (Fig. 4): ironmaking, steelmaking, continuous casting and hot rolling. The production machines in the first three stages are batch processing machines (BPMs). According to the capacity constraint of BPMs (the maximum capacity of a batch), jobs with similar features are grouped into a batch to improve production efficiency. The job data of NISCO in the past six months are used for training the model, the results of manual batching of 50, 100, 200, 300 jobs are used for testing the model. We train the proposed model on NVIDIA Tesla P100-PCIE 16GB GPU and evaluate the performance on Lenovo Intel(R) Core(TM) i7-6700 8GB CPU @ 3.40GHz.

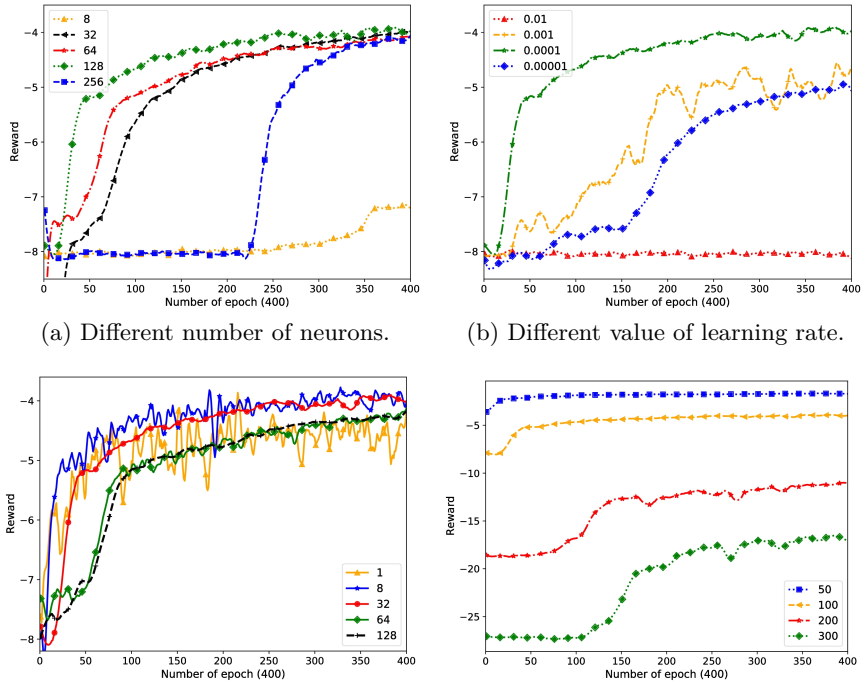
The parameters used in this work are divided into two categories: *i*) Parameters that are set according to specific application scenario: the score of job's each feature; the score of two sub objectives. According to the NISCO's job batching requirements, each job has six features ($K = 6$), which are category, thickness, length, width, delivery date and processing technology. The score of job's features are: $c_1 = 0.2$, $c_2 = 0.2$, $c_3 = 0.08$, $c_4 = 0.16$, $c_5 = 0.16$, $c_6 = 0.2$; the score of two sub objectives are: α is $0.1 \sim 0.5$, β is $0.1 \sim 0.5$; the maximum capacity of a batch C is 180. *ii*) The parameters used in the DRL based algorithm for training: the number of neurons in the hidden layer is $10 \sim 256$; the learning rate for updating the actor network and critic network is $0.00001 \sim 0.01$; the batch size for one training is $1 \sim 128$.

4.2 Convergence Analysis

This subsection analyzes the convergence of training the proposed model using different settings of four factors: number of neurons in the hidden layer, learning rate, batch size and different number of jobs.

Firstly, we analyze the convergence effect using different number of neurons in the hidden layer. Consider the problem instance with $M = 100$. Fig. 5a shows the convergence effect when the number of neurons in the hidden layer is 8, 32, 64, 128, 256. Within limits, the more the number of neurons is, the more quickly the convergence is achieved. However, when the number of neurons is 256 which is too large, the model converges after 200 epoches. Accordingly, the number of neurons in the hidden layer is set to 128 which has the fastest convergence speed.

Secondly, we analyze the convergence effect using different learning rate. Consider the problem instance with $M = 100$. Fig. 5b shows the convergence



(a) Different number of neurons.

(b) Different value of learning rate.

(c) Different value of batch size.

(d) Different number of jobs (M).

Fig. 5. Convergence analysis on different parameter settings.

effect when the learning rate is 0.01, 0.001, 0.0001, 0.00001. When the learning rate is 0.00001, which is too small, it takes a very long time to converge (after 400 epoches). When the learning rate is 0.01 which is too large, the model cannot converge. So we set the learning rate to 0.0001 which converges fastest.

Thirdly, we conduct the convergence analysis on different value of batch size. Fig. 5c shows the convergence analysis of the instance with $M = 100$ when the batch size is 1, 8, 32, 64, 128. The smaller the batch size is, the larger the gradient oscillation amplitude of the model is, which is not conducive to the convergence of the model; the larger the batch size is, the smaller the gradient oscillation amplitude of the model is, but the training is easy to fall into local optimum. When the batch size is 32, the oscillation amplitude of gradient is small and the model converges fast. Accordingly, the batch size is set to 32.

Lastly, we conduct the convergence analysis on different number of jobs. Fig. 5d shows the convergence analysis of M is 50, 100, 200, 300. The more the number of jobs is, the more slowly the convergence is achieved. Although it takes hours to train the model, once the model is well trained, it can quickly solve the problems it has not encountered before in seconds or minutes.

Table 1. Comparasion of the experimental results using the proposed method, manual result, the K-means method, and three meta-heuristic algorithms.

Size	Parameter (α,β)	Metrics	Method					
			Manual	K-means	GA	ACO	TS	Ours
50	(0.5, 0.2)	Batch Number	**4**	**4**	**4**	**4**	**4**	**4**
		Difference Value	5.79	5.70	**1.35**	4.02	1.37	**1.35**
		Target Value	3.70	3.65	**1.48**	2.81	1.49	**1.48**
		Inference Time (s)	-	**2.78**	10.08	6.45	86.94	13.00
100	(0.1, 0.3)	Batch Number	**10**	**10**	**10**	**10**	**10**	**10**
		Difference Value	6.34	5.58	8.37	7.44	6.42	**3.09**
		Target Value	3.63	3.56	3.84	3.74	3.64	**3.31**
		Inference Time (s)	-	**3.77**	34.61	46.25	406.58	35.00
200	(0.1, 0.5)	Batch Number	22	**21**	22	22	22	**21**
		Difference Value	16.00	16.61	66.88	14.30	14.05	**13.99**
		Target Value	12.60	12.16	17.69	12.43	12.41	**11.90**
		Inference Time (s)	-	**5.70**	135.19	380.31	2261.31	57.00
300	(0.1, 0.5)	Batch Number	34	**32**	34	34	34	**32**
		Difference Value	24.41	25.29	122.41	17.63	17.48	**17.36**
		Target Value	19.44	18.53	29.24	18.76	18.75	**17.74**
		Inference Time (s)	-	307.64	314.43	1386.12	6574.58	**93.00**

4.3 Performance Comparison

To evaluate the performance of proposed method, we compare our method with the manual result, K-means algorithm and several typical meta-heuristic algorithms which are Genetic Algorithm (GA), Ant Colony Optimization (ACO), and Tabu Search (TS). The experiments are conducted on different problem sizes with M is 50, 100, 200, 300. Table 1 shows the comparison results.

K-means algorithm needs the batch number in advance, which is unknown in practical production. For experimental comparison, this paper set the batch number of K-means algorithm to $n = \left\lceil \sum_{i=1}^{M} w_i/M \right\rceil$, which is the ideal minimum batch number. Since the results of meta-heuristic algorithm are unstable for its introduction of random factors, we run each meta-heuristic algorithm 10 times, and compare their optimal results with our proposed method. From Table 1, we know that our proposed method performs better than the manual result, K-means algorithm and other meta-heuristic algorithms in terms of the total batch number, the total feature difference of all batches and the target value. Especially, the inference time of our method does not increase significantly as the number of jobs increases. Therefore, our method can be used to deal with the job batching problem with large job number in actual production.

5 Related Work

Job batching problem exists widely in chemical, steelmaking, textile, semiconductor, and other fields. In recent years, scholars at home and abroad have made extensive researches on this problem.

In chemical field, [11] proposed the best priority and variable neighborhood search algorithm for furnace grouping in NdFeB enterprises; [22] proposed a novel Particle Swarm Optimization algorithm with Von Neumann Topology for the furnace-grouping in special aluminum alloy production. In steelmaking field, [20] proposed an improved compact genetic algorithm for the charge design in steelmaking and continuous casting; [15] proposed a fuzzy clustering algorithm based on an improved particle swarm optimization for the order batching in cold rolling. In textile field, [5] proposed a hybrid multi-subpopulation genetic algorithm for textile batch dyeing scheduling; [2] proposed a Multi-Agent Recurrent Proximal Policy Optimization (MA-RPPO) reinforcement learning based fully reactive scheduling method for the vats batching problem of textile dyeing workshop, which simplified the problem by only considering one job feature. In semiconductor field, [7] proposed an algorithm based on the hybrid ant colony optimization algorithm for the semiconductor furnace operation.

Most of the researches on job batching problem are based on meta-heuristic algorithms, whose performance depends heavily on human experience. Only few works attempt to use DRL to solve simplified job batching problem which deviates from actual production.

6 Conclusion

In this paper, we build a job batching model based on pointer network. We propose a DRL based method to achieve intelligent job batching, which can learn the stable strategy by making full use of the massive unlabeled data in IIoT and can handle multi-dimensional data. We analyze the convergence of the model and compare the performance with multiple methods. The experiments demonstrate that our method can produce stable and better solution. Especially, our method can produce faster solution even in the the problem with large job number. In the future, we will focus on the study of using DRL method to achieve intelligent scheduling combined with the job batching problem in IIoT.

Acknowledgements. This work was supported in part by the National Key R&D Program of China 2017YFB1003000, NSFC 62061146002, 61632008, 61921003, and Fundamental Research Funds for the Central Universities (2019XD- A14).

References

1. Gilchrist, A.: Industry 4.0: the industrial internet of things. Springer (2016)
2. He, J.J., Zhang, J., et al.: Multi-agent reinforcement learning based textile dyeing workshop dynamic scheduling method. CIMS pp. 1–31 (2021)

3. He, Y., Guo, J.C., Zheng, X.L.: From surveillance to digital twin: Challenges and recent advances of signal processing for industrial iot. IEEE Signal Processing Magazine **35**(5), 120–129 (2018). https://doi.org/10.1109/MSP.2018.2842228
4. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997). https://doi.org/10.1162/neco.1997.9.8.1735
5. Huynh, N.T., Chien, C.F.: A hybrid multi-subpopulation genetic algorithm for textile batch dyeing scheduling and an empirical study. Computers and Industrial Engineering **125**(NOV.), 615–627 (2018). https://doi.org/10.1016/j.cie.2018.01.005
6. Jia, S.J., Yi, J., Du, B.: A charge optimization algorithm based on column generation and linear programming. Metallurgical Industry Automation **v.44;No.262**(03), 35–40 (2020)
7. Jiang, X.K., Zhang, P., Lv, Y.L., et al.: Hybrid ant colony algorithm for batch scheduling in semiconductor furnace operation. Journal of Shanghai Jiaotong University (8), 792–804 (2020). https://doi.org/10.16183/j.cnki.jsjtu.2018.232
8. Kool, W., Van, H., Welling, M.: Attention, learn to solve routing problems! In: ICLR (2018)
9. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature **521**(7553), 436–444 (2015). https://doi.org/10.1038/nature14539
10. Liu, X.H., Zhang, T., et al.: Batch scheduling of parallel machines with different capacity for sterilization process in medical equipment enterprises. Industrial Engineering Journal **24**(01), 82–89 (2021)
11. Liu, Y.F., Chai, T.Y.: The best priority and variable neighborhood search algorithm for production furnace grouping in ndfeb enterprises. CIESC Journal (2018)
12. Luo, S.: Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. Applied Soft Computing **91**, 106208 (2020). https://doi.org/10.1016/j.asoc.2020.106208
13. Mnih, V., Kavukcuoglu, K., et al.: Playing atari with deep reinforcement learning. Computer Science (2013)
14. Nazari, M., Oroojlooy, A., et al.: Reinforcement learning for solving the vehicle routing problem. In: NeurIPS (2018)
15. Pan, R.L., Wang, X.M., et al.: Optimization method of order batching for cold rolling based on fuzzy clusting. Control and Decision (1), 141–148 (2017). https://doi.org/10.13195/j.kzyjc.2015.1238
16. Schuster, M., Paliwal, K.: Bidirectional recurrent neural networks. IEEE transactions on Signal Processing **45**(11), 2673–2681 (1997). https://doi.org/10.1109/78.650093
17. Sutton, R., Barto, A.: Reinforcement learning: An introduction. MIT press (2018)
18. Vinyals, O., Fortunato, M., Jaitly, N.: Pointer networks. In: NIPS (2015)
19. Wang, L., Gao, X.W., Wang, W., Wang, Q.: Order production scheduling method based on subspace clustering mixed model and time-section ant colony algorithm. IEEE/CAA JAS **40**(009), 1991–1997 (2014)
20. Yi, J., Du, B., et al.: An imporved compact genetic algorithm for charge design problem. In: CCDC (2019)
21. Zhang, C., Song, W., et al.: Learning to dispatch for job shop scheduling via deep reinforcement learning. NIPS **33** (2020)
22. Zhang, H., Ku, T., Zhang, D.Y.: Furnace-grouping optimization with order-grouping for special aluminum ingots. Control Theory and Applications (2019)
23. Zhang, Y., Zheng, X.L., Liu, L., Ma, H.D.: Multivariate and multi-frequency lstm based fine-grained productivity forecasting for industrial iot. IEEE MSN (2020). https://doi.org/10.1109/MSN50589.2020.00058