Int j simul model 18 (2019) 4, 699-707
ISSN 1726-4529 Original scientific paper

A NOVEL JOB-SHOP SCHEDULING STRATEGY BASED ON PARTICLE SWARM OPTIMIZATION AND NEURAL NETWORK

Zhang, Z.; Guan, Z. L.; Zhang, J. & Xie, X.[#]
School of Economics and Management, Beijing Jiaotong University, Beijing 100044, China
E-Mail: xxie@bjtu.edu.cn ([#] Corresponding author)

Abstract
This paper innovatively introduces particle swarm optimization (PSO) and neural network (NN) to solve the job-shop scheduling problem (JSP). Each particle in the swarm was treated as a connection in the NN. Then, the connection weight was iteratively updated according to the latest position of the corresponding particle. In this way, the NN no longer falls into the local optimum trap. Then, the PSO-optimized NN was applied to solve the JSP with a single objective: minimizing the maximum makespan. Through experiments on benchmark problems, it is confirmed that the proposed strategy outperforms the other scheduling methods in fulfilling the optimization objective.
(Received, processed and accepted by the Chinese Representative Office.)

Key Words: Job-Shop Scheduling Problem (JSP), Particle Swarm Optimization (PSO), Neural Network (NN), Maximum Makespan

1. INTRODUCTION

Since the middle of the 20th century, job-shop scheduling problem (JSP) has been a thorny issue among manufacturing enterprises [1, 2]. This non-deterministic polynomial-time (NP) hard problem [3] is extremely difficult to model or optimize. The modelling of a JSP must fully consider the machines, job-shop scale and the difficulty of the production process. However, the production process is highly diverse. The operation sequence varies from job to job, the machines might fail, and the order quantity is sometimes dynamic. Any of these variations will cause changes to the JSP model, and the entire schedule. For most JSPs, it is very tedious to find the global optimal solution.

This type of problems cannot be solved effectively by a single algorithm. For example, the neural network (NN) method consumes an increasingly long time, when it approaches a solution to the JSP, and cannot converge to a desirable solution [4, 5]. To overcome the defects, a viable option is to integrate heuristic algorithms with the NN to solve the JSP. For example, particle swarm optimization (PSO) or genetic algorithm (GA) could be adopted to train the NN. The PSO-based training can improve the rule adjustment and extraction, as well as the selection of parameters and attributes of the NN. The GA-based training can reduce the computing load of the NN. However, an NN trained by the GA will consume a long time in solving largescale problems, due to the complex operations of the GA.

This paper introduces the PSO to optimize the network weight such that the NN no longer falls into the local optimum trap. Next, the optimized NN was applied to solve the JSP with the aim to minimize the maximum makespan.

2. LITERATURE REVIEW

The research of the JSP can be traced back to the 1970s. Being pioneers in JSP research, Garey et al. [6] analysed the JSP with two conflicting machines, detailed the production process and highlighted the complexity of problem-solving. In real-world scenarios, the JSPs vary greatly with the job-shop environments, production objectives and schedule types. In

some JSPs, the production process needs to be scheduled under certain constraints, e.g. fixed working hours of each machine; in some other JSPs, the scheduling should be implemented under uncertain constraints. By the number of scheduling objectives, the JSPs can be divided into single-objective JSP and multi-objective JSP.

Li and Gao [7] developed a hybrid algorithm based on the GA and tabu search that can effectively solve the single-objective flexible JSP (FJSP) under uncertain constraints. Targeting multi-objective flexible JSP, Jun et al. [8] improved the bee colony algorithm (BCA) and combined it with simulated annealing (SA) algorithm, creating a hybrid strategy with strong global and local search abilities. Pang et al. [9] minimized the makespan of single-objective JSP with the improved PSO: the improvement diversified the swarm, prevented the local optimum trap, and optimized the convergence speed and quality of the solution. Hao et al. [10] proposed a multi-objective estimation of distribution algorithm to solve the bi-criteria stochastic JSP with uncertain processing time; the proposed algorithm minimizes the expected average makespan and the expected total tardiness within a reasonable computing time. Zhang et al. [11] studied the single-objective FJSP with energy consumption constraints, and proposed a new imperialist competition algorithm to optimize the makespan and total tardiness.

In the late 1980s, scholars started to apply the NN to solve the JSPs. For instance, Satake et al. [12] minimized the makespan of static JSP based on the Hopfield interconnected NN model. However, the computing efficiency of the model is poor, because too many variables are used to set up the energy function. Based on the backpropagation (BP) NN and the SA, Wu [13] established a prediction model for the completion rate of jobs in the job-shop. Li et al. [14] solved the FJSP with the NN and other methods, using the NN to control the problem constraints. In general, slow convergence and proneness to local optimum trap are common when the JSP is solved by the NN only.

To overcome the defects, the PSO has often been adopted to train the NN, thanks to its simple principle and relatively few parameters. For example, Lin et al. [15] optimized the input weight of the artificial neural network (ANN) with the PSO, and thus achieved the expected accuracy and avoided premature convergence to the local optimum. Mao et al. [16] put forward an accurate hybrid prediction algorithm PSO-NN, in which the PSO searches for the suitable NN parameters. Zhang and Wu [17] solved the minimum root-mean-square error (*RMSE*) of the NN through the PSO, and implemented the improved NN to optimize complex problems.

The above analysis fully demonstrates the advantages of the NN trained by the PSO. However, there is no report that applies the PSO-optimized NN in the JSP. To make up for the gap, this paper combines the PSO and the NN to solve the JSP with the objective of minimizing the maximum makespan. Specifically, the JSP was modelled by the NN, and the PSO was used to optimize the network weight.

3. PSO-OPTIMIZED NN

The NN selected for our research is an error feedback NN [18]. This type of NN computes the error between calculated output and expected output after forward calculation, and completes the reverse calculation (adjusts the weights of node connections) to achieve the required accuracy, if the error is unacceptable.

In general, the accuracy of an NN model depends on the training effect. As an information processing paradigm, the NN can learn from the environment and iteratively adjust its internal weights. However, the NN often faces the following disadvantages: (1) With the maintenance of the learning rate, the convergence rate will decrease continuously, pushing up the time cost of training; (2) The network is highly sensitive to the initial value, and thus prone to local

optimums; (3) There is no mature theory on the selection of hidden layers or the number of neurons.

The key to PSO-based NN optimization is to associate the position of each particle in the swarm with the weight of each connection in the NN. Once the PSO finds the optimal position of a particle, the weight of the corresponding connection in the NN will be modified accordingly, thus minimizing the learning error.

Let $W = \{w_1, w_2, \ldots, w_n\}$ be the set of position vectors of the particles in the swarm, where w_i is the position vector of the i^{th} particle and n is the number of connections in the neural network. Suppose the NN has two hidden layers. Then, the value of n, i.e. the swarm size W, can be computed by:

$$W = (I_n + 1) \times F_n + (F_n + 1) \times S_n + (S_n + 1) \times Q_n \tag{1}$$

where, I_n is the number of input layer neurons; F_n is the number of neurons in the first hidden layer; S_n is the number of neurons in the second hidden layer; Q_n is the number of output layer neurons.

Next, the NN was trained by the PSO iteratively. In each iteration, the training data are input into the NN, and each particle in the swarm corresponds to a connection in the network. The weight of each connection is determined by the current position of the corresponding particle. Once an element changes in W, the corresponding connection weight will be updated.

Using the objective function of the PSO, the actual outputs of the NN is compared with the expected outputs according to the updated positions of particles at the end of each iteration, aiming to iteratively minimize the learning error. The objective function F_o of swarm o can be defined as:

$$F_o = \frac{1}{M} \sum_{i=1}^{M} (E(j) - O(j))^2 \tag{2}$$

where, M is the number of particles; $E(j)$ and $O(j)$ are expected output of the j^{th} particle of swarm o and the actual output of the corresponding weight of the NN.

In each iteration, the position of each particle is updated based on the individual best-known position P_{id}^{best} and the global best-known position P_{gd}^{best}.

As shown in Fig. 1, the PSO-based optimization of the NN can be summed up as follows:

Step 1. Randomly initialize the position of each particle in the swarm.

Step 2. Import data into the input layer of the NN and obtain the output data from the output layer of the NN. Determine the connection weights of the NN based on the current positions of particles.

Step 3. Calculate the fitness of each particle by Eq. (2).

Step 4. Evaluate the new individual best-known position P_{id}^{best} and the global best-known position P_{gd}^{best}.

Step 5. Update the speed of each particle V_{id}^{t+1}.

Step 6. Update the position of each particle P_{id}^{t+1}.

Step 7. Return to Step (2) if all particles have been updated.

Step 8. Return to Step (2) after meeting the termination condition.

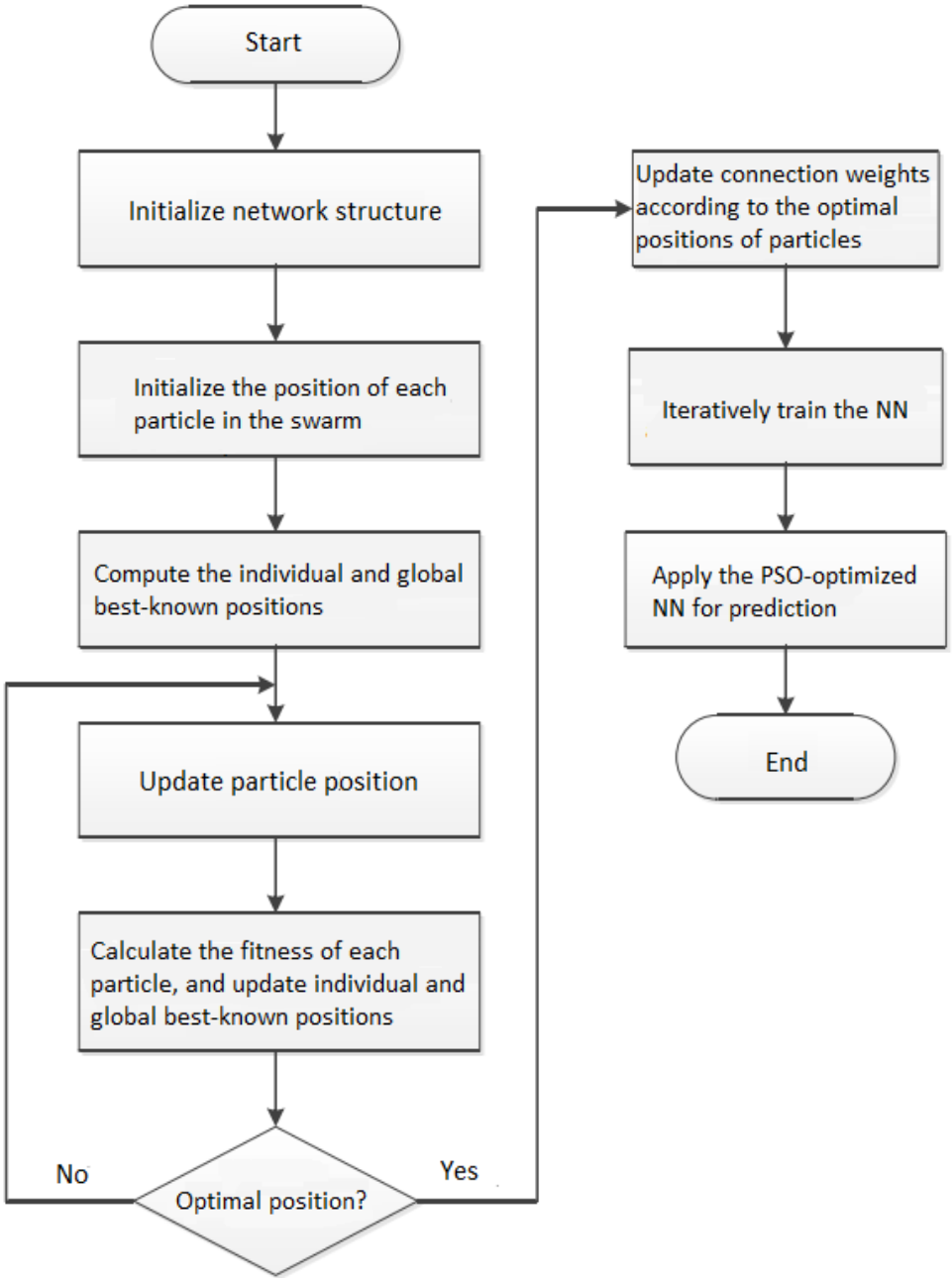

Figure 1: Flow chart of PSO-based NN optimization.

4. SINGLE-OBJECTIVE JOB-SHOP SCHEDULING BASED ON PSO-OPTIMIZED NN

This section establishes an NN model of the JSP with a single objective: minimizing the maximum makespan, and then solves the model by the PSO-optimized NN.

4.1 NN modelling

This paper selects a 4-layer NN to model the JSP, which includes 1 input layer, 2 hidden layers and 1 output layer. Suppose the job-shop production involves 6 jobs and 6 machines. Then, the GA solution to the JSP is a sequence of 36 operations: {1, 2, 2, 3, 5, 2, 1, 3, 2, 6, 5, 2, 6, 5, 4, 3, 2, 1, 5, 3, 4, 1, 3, 6, 5, 3, 4, 6, 2, 3, 4, 3, 5, 4, 6, 1}. Each number in the sequence represents the number of job and the number of the current operation.

In priority rules for scheduling, four common attributes are usually adopted as input features: process number, processing time, remaining time, and machine load [19]. Specifically, process number is the number of operations for a job, processing time is the time to process a job on the machines, remaining time is the total processing time of the remaining operations of a job, and machine load is the total time required for all operations on a machine. Drawing on concept hierarchy, the JSPs can be divided into {1}, {2, 3}, {4, 5} and {6} based on process number, short, general and long based on processing time, low, medium and high based on remaining time, and light and heavy based on machine load. Therefore, the total number of input layer neurons can be obtained as: 4 process numbers + 3 processing times + 3 remaining times + 2 equipment loads = 12.

In the NN, more hidden layers help to reduce error and enhance accuracy. However, the addition of new hidden layer(s) may complicate the NN structure and prolong network training. Considering these factors, two hidden layers were selected for our NN, forming a 4-layer NN. The number of hidden layer neurons H is generally determined empirically by the following formulas:

$$H < I - 1 \tag{3}$$

$$F_n = \sqrt{I + O} + D \tag{4}$$

$$S_n = \log_2 I \tag{5}$$

where, F_n and S_n are the number of neurons in the first and second hidden layers, respectively; $D \in [1, 10]$ is an adjustment constant; I and O are the number of neurons in the input layer and output layer, respectively.

The output layer of the NN can be set freely in the light of the expression form of the JSP. The number of output layer neurons has little impact on the NN. Let L be the total number of operations of a job. Then, the position of each operation falls into the interval [1: L]. In this paper, the L operations are classified into several different priorities, and the operation numbers are ranked in descending order according to the priority level. In FT06 benchmark problem, there are a total of 36 operations (Table I). Thus, the number of output layer neurons was defined as 6.

Table I: The 36 operations in FT06 benchmark problem.

Process number	**Priority**
1-6	1
7-12	2
13-18	3
19-24	4
25-30	5
31-36	6

4.2 Problem solving

To enhance the NN's ability to solve the JSP, two sub-problems, namely, process scheduling and machine selection, were optimized by elite retention and neighbourhood search.

The initial operation sequence *array* was obtained according to the number of jobs and the operations for each job. The code of machine sequence corresponds to the fixed job process. According to the job sequence 1~*n*, the index values of each operation was saved, using the machine in the set of available machines. The machines were selected through global and local searches at random probabilities. 50 %, 30 % and 20 % of machines were selected iteratively, respectively.

In the PSO, the particle position is represented by the quantum superposition state of probability amplitude quantum, the swarm size is set as *N*, the spatial dimension, i.e. the

particle sequence size is the total number of operations M, and the initialization results in two sequences belong to the range of [0, M]. The initial state of the swarm can be described as:

$$P_i = \left[\begin{vmatrix}\sin\theta_{i1}\\ \cos\theta_{i1}\end{vmatrix} \ldots \begin{vmatrix}\sin\theta_{iM}\\ \cos\theta_{iM}\end{vmatrix}\right] \quad i = (1, 2,\ldots, N),\ j = (1, 2,\ldots, M) \tag{6}$$

where, $\theta_{ij}=2\pi \times \mathrm{rand}()$ is the quantum bit phase. As shown in Fig. 2, there is a one-to-one correspondence between the initial particle sequence and initial operation sequence.

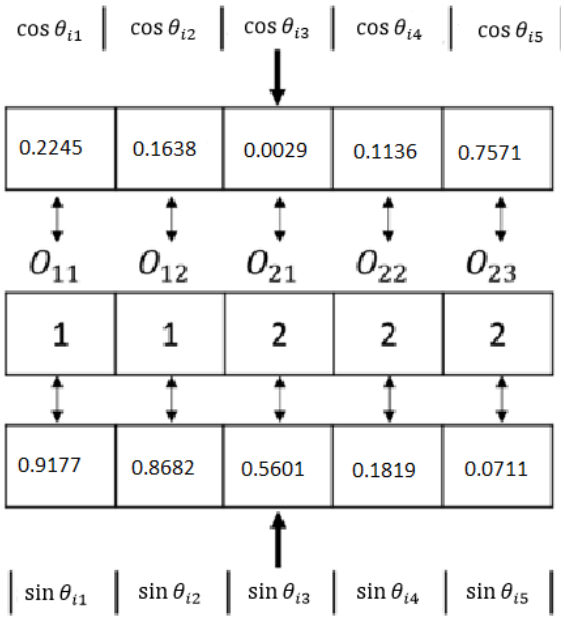

Figure 2: The correspondence between initial particle sequence and initial operation sequence.

To minimize the maximum makespan, the maximum makespan was taken as the fitness function. Then, the operation sequence *array* was transformed according to different particle sequences (Fig. 3).

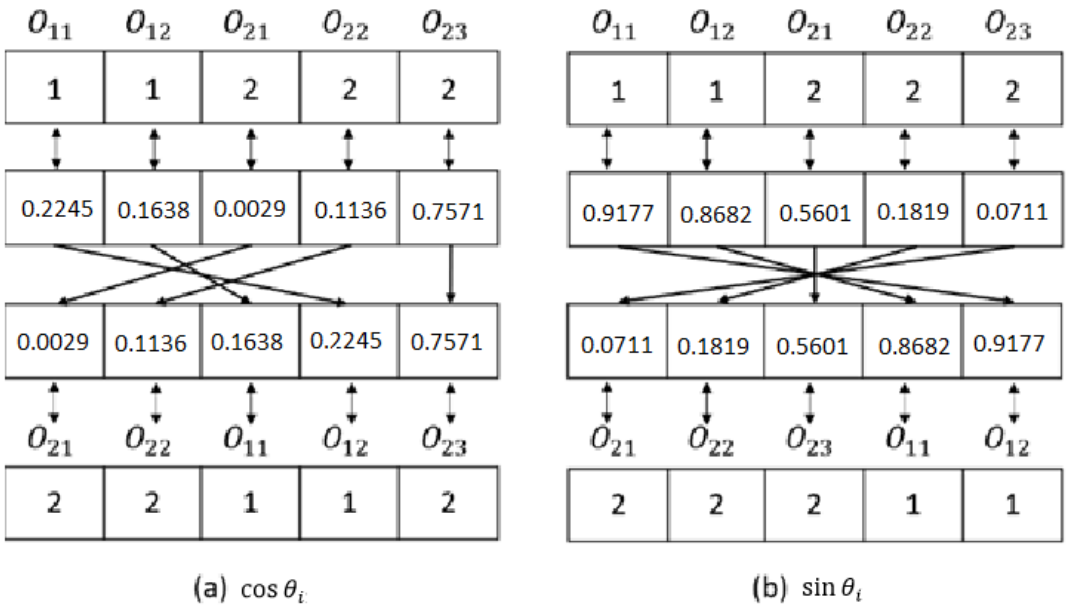

Figure 3: Operation sequence transformation.

In problem solving, the update and mutation of the swarm could change the quantum phase of the particle through the quantum rotary gate, and realize the simultaneous movement

of two positions, which greatly improves the solving efficiency. Once a particle is updated, two new sequences were obtained. Then, the new sequences were ranked in ascending order again. According to the new index, the sequence *array* was rearranged.

During the iteration, some operations have only a few available machines, sowing the risk of falling into the local optimum. In this paper, the elite retention is combined with neighbourhood search to eliminate the risk. The combined strategy is implemented as follows:

Step 1. Initialization: The particle sequence is randomly generated by Eq. (6). The individual best-known solution and global best-known solution are both obtained. The two solutions save the fitness and rotation angle of each particle, which facilitates the position update in the iterative process.

Step 2. Position update: In each iteration, the control parameter α is used to change the particle position.

Step 3. Swarm mutation: The particles in the swarm are mutated at the mutation probability.

Step 4. Fitness calculation: The particle sequence is ranked in ascending order, and the corresponding index sequence is obtained. Then, the initial sequence *array* is changed according to the index, creating the operation sequence. Finally, the machine sequence is generated.

Step 5. Elite retention: At the end of each iteration, the top 20 % of the particles in the swarm are retained, and the top 80 % of the particles are subjected to neighbourhood search to generate a new population.

Step 6. Comparative update: The individual best-known solution and global best-known solution are compared, and the better one is retained.

Step 7. Termination: Steps 2~6 are repeated until reaching the termination condition.

5. SIMULATION AND RESULTS ANALYSIS

The proposed PSO-optimized NN was applied to minimize the maximum makespan of the FT06 benchmark problem, and compared with other scheduling algorithms [19, 20]. The parameters of our algorithm were configured as follows: swarm size $N = 50$, weight = 0.8, learning factors C_1 and $C_2 = 2$, maximum number of iterations = 1,000. The simulation results of each algorithm are listed in Table II, where deviation refers to the error between the minimum makespan of the current scheduling and that of the best-known solution.

Table II: Simulation results of different algorithms on FT06 benchmark problem.

Scheduling algorithm	**Makespan**	**Deviation** (%)
NN	66	7.72
Our algorithm	64	6.97
Attribute oriented induction	83	22.85
Shortest processing time	76	13.88

It can be seen from Table II that our algorithm outshines the NN in terms of the makespan. Our algorithm achieved the minimum makespan of 64 units, 5 units (0.75 %) more than the optimal solution. The deviation was 0.75 % lower than that of the NN. The deviations of the other algorithms fell between 17 and 24 units (13.88~22.85 %), far greater than our deviation. Hence, our algorithm boasts the best effect on the 6×6 benchmark JSP.

To verify the performance of our algorithm on largescale problems, two hundred 20×10 benchmark problems, three hundred 20×15 benchmark problems, three hundred 20×20 benchmark problems were selected to test our algorithm. The simulation results are shown in Table III. The results are plotted into a histogram (Fig. 4) to make the contrast more intuitive.

Table III**:** Comparison of makespan of different algorithms.

n*×*m	**Size**	**Optimal result**	**Shortest processing time**	**NN**	**Our algorithm**
20 × 10	200	920	1,870	1,560	1,505
20 × 15	300	725	1,295	1,010	960
20 × 20	300	890	1,580	1,255	915

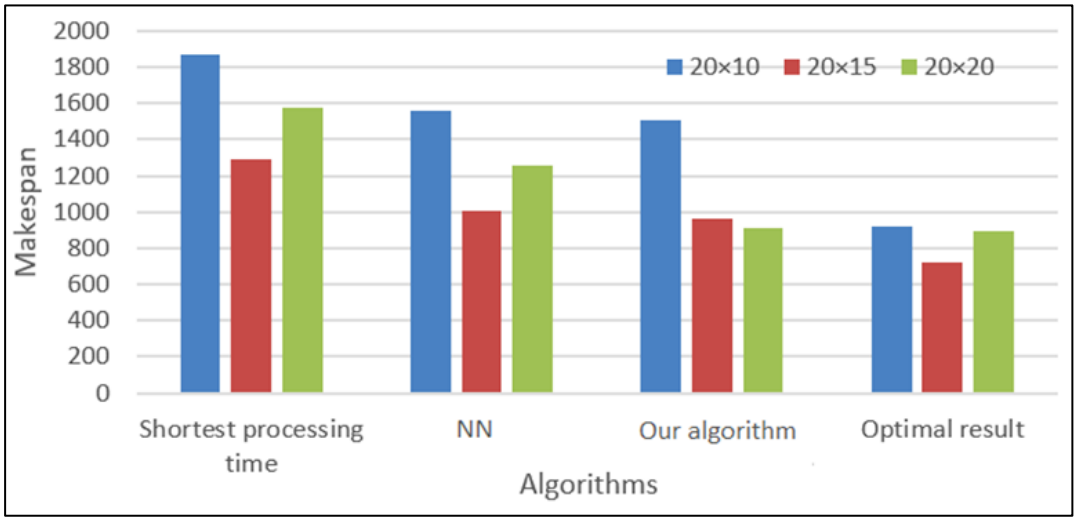

Figure 4: Comparison of makespan of different algorithms.

According to Table III and Fig. 4, with the increase in the problem scale (computing load), our algorithm exhibited a growing advantage over the contrastive algorithms in makespan. This means our algorithm is suitable for largescale JSPs.

6. CONCLUSIONS

This paper applies the PSO-optimized NN to minimize the maximum makespan of the JSP. First, a JSP model was established based on the NN. The data coding of input and output layers was explained in details. Then, the network weights were optimized by the PSO to avoid the local optimum trap. Finally, our algorithm was compared with other scheduling methods through simulation of small-scale test on FT06 benchmark problem and several largescale JSPs. The results show that our algorithm achieved the shortest makespan on FT06 benchmark problem and effectively solved largescale problems.

REFERENCES

[1] Li, Y.; Shi, S. Y.; Huang, Q. D. (2018). Three-machine job shop scheduling with intermediate transfer, *International Journal of Simulation Modelling*, Vol. 17, No. 2, 359-368, doi:10.2507/IJSIMM17(2)CO10

[2] Zhong, Y.; Li, J. M.; Zhu, S. Z. (2017). Research on the multi-objective optimized scheduling of the flexible job-shop considering multi-resource allocation, *International Journal of Simulation Modelling*, Vol. 16, No. 3, 517-526, doi:10.2507/IJSIMM16(3)CO13

[3] Hanen, C. (1994). Study of a NP-hard cyclic scheduling problem: the recurrent job-shop, *European Journal of Operational Research*, Vol. 72, No. 1, 82-101, doi:10.1016/0377-2217(94)90332-8

[4] Munoz-Guijosa, J. M.; Riesco, E.; Olmedo, M. (2017). Neural network and training strategy design for train drivers' vibration dose simulation, *International Journal of Simulation Modelling*, Vol. 16, No. 1, 72-83, doi:10.2507/IJSIMM16(1)6.370

[5] Koochaki, M.; Lotfi, M. (2017). Design of a neural network controller for the electrode control system in the electric arc furnace, *Journal Européen des Systèmes Automatisés*, Vol. 50, No. 3, 299-311, doi:10.3166/JESA.50.299-311
[6] Garey, M. R.; Johnson, D. S.; Sethi, R. (1976). The complexity of flowshop and jobshop scheduling, *Mathematics of Operations Research*, Vol. 1, No. 2, 117-129, doi:10.1287/moor.1.2.117
[7] Li, X.; Gao, L. (2016). An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem, *International Journal of Production Economics*, Vol. 174, 93-110, doi:10.1016/j.ijpe.2016.01.016
[8] Li, J.-Q.; Pan, Q.-K.; Tasgetiren, M. F. (2014). A discrete artificial bee colony algorithm for the multi-objective flexible job-shop scheduling problem with maintenance activities, *Applied Mathematical Modelling*, Vol. 38, No. 3, 1111-1132, doi:10.1016/j.apm.2013.07.038
[9] Pang, S.; Li, T.; Dai, F.; You, M. (2013). Particle swarm optimization algorithm for multi-salesman problem with time and capacity constraints, *Applied Mathematics & Information Sciences*, Vol. 7, No. 6, 2439-2444, doi:10.12785/amis/070637
[10] Hao, X.; Gen, M.; Lin, L.; Suer, G. A. (2017). Effective multiobjective EDA for bi-criteria stochastic job-shop scheduling problem, *Journal of Intelligent Manufacturing*, Vol. 28, No. 3, 833-845, doi:10.1007/s10845-014-1026-0
[11] Zhang, Q.; Manier, H.; Manier, M.-A. (2012). A genetic algorithm with tabu search procedure for flexible job shop scheduling with transportation constraints and bounded processing times, *Computers & Operations Research*, Vol. 39, No. 7, 1713-1723, doi:10.1016/j.cor.2011.10.007
[12] Satake, T.; Morikawa, K.; Nakamura, N. (1994). Neural network approach for minimizing the makespan of the general job-shop, *International Journal of Production Economics*, Vol. 33, No. 1-3, 67-74, doi:10.1016/0925-5273(94)90119-8
[13] Wu, K.-B. (1999). An efficient configuration generation mechanism to solve job shop scheduling problems by the simulated annealing algorithm, *International Journal of Systems Science*, Vol. 30, No. 5, 527-532, doi:10.1080/002077299292263
[14] Li, B.; Guo, C.; Ning, T. (2018). An improved bacterial foraging optimization for multi-objective flexible job-shop scheduling problem, *Journal Européen des Systèmes Automatisés*, Vol. 51, No. 4-6, 323-332, doi:10.3166/JESA.51.323-332
[15] Lin, T.-L.; Horng, S.-J.; Kao, T.-W.; Chen, Y.-H.; Run, R.-S.; Chen, R.-J.; Lai, J.-L.; Kuo, I.-H. (2010). An efficient job-shop scheduling algorithm based on particle swarm optimization, *Expert Systems with Applications*, Vol. 37, No. 3, 2629-2636, doi:10.1016/j.eswa.2009.08.015
[16] Mao, C.; Lin, R.; Xu, C.; He, Q. (2017). Towards a trust prediction framework for cloud services based on PSO-driven neural network, *IEEE Access*, Vol. 5, 2187-2199, doi:10.1109/ACCESS.2017.2654378
[17] Zhang, Y.; Wu, L. (2011). Crop classification by forward neural network with adaptive chaotic particle swarm optimization, *Sensors*, Vol. 11, No. 5, 4721-4743, doi:10.3390/s110504721
[18] Waheeb, W.; Ghazali, R.; Herawan, T. (2016). Ridge polynomial neural network with error feedback for time series forecasting, *Plos One*, Vol. 11, No. 12, Paper e0167248, 34 pages, doi:10.1371/journal.pone.0167248
[19] 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
[20] Huang, C.; Zhou, X.; Hou, D. (2018). Online no-wait scheduling of leather workshop supply chain based on particle swarm optimization, *Journal Européen des Systèmes Automatisés*, Vol. 51, No. 1-3, 153-167, doi:10.3166/JESA.51.153-167