

An Improved Adaptive Neural Network for Job-Shop Scheduling

Shengxiang Yang

Department of Computer Science, University of Leicester
University Road, Leceister LE1 7RH, United Kingdom
s.yang@mcs.le.ac.uk

Abstract – *Job-shop scheduling is one of the most difficult production scheduling problems in industry. This paper presents an improved adaptive neural network together with heuristic methods for job-shop scheduling problems. The neural network is based on constraints satisfaction of job-shop scheduling and can adapt its structure and neuron connections during the solving. Several heuristics are also proposed to be combined with the neural network to guarantee its convergence, accelerate its solving process, and improve the quality of solutions. Experimental study shows that the proposed hybrid approach outperforms two classical heuristic algorithms regarding the quality of solutions.*

Keywords: Job-shop scheduling, constraint satisfaction, heuristics, adaptive neural network

1 Introduction

The job-shop scheduling problem (JSP) is one of the most difficult production scheduling problems. It aims to allocate a number of machines over time to perform a set of jobs with certain constraint conditions in order to optimize certain criterion, e.g., minimizing the makespan. Traditionally there are three kinds of approaches for solving JSPs: priority rules, combinatorial optimization and constraints analysis.

Due to the hardness of solving JSPs researchers have also investigated intelligent methods for JSPs. Foo and Takefuji [2, 3] first used a neural network to solve JSPs. Thereafter, several neural network architectures have been devised for JSPs [6, 7, 9]. Willems [6] first proposed a constraint satisfaction neural network for traditional JSPs. Yu [9] extended Willems's neural network by adding a job constraint block to deal with free operations. In [7] a constraint satisfaction adaptive neural network (CSANN) was proposed for generalized JSPs. CSANN has the ability to easily map the constraints of a JSP into its architecture and remove the violation of the mapped constraints during its processing. And CSANN can adaptively adjust the connection weights and biases of neurons according to the actual constraint violations during the run. In [7, 8] several heuristic algorithms were also proposed to improve the performance of CSANN and the quality of obtained solutions.

This paper proposes an improved CSANN by simplifying its resource constraint block. In the simplified CSANN, the resource constraint block is adaptively constructed from

the actual resource constraint satisfaction situation during the run and has reduced resource constraint units (and hence reduced computational complexity). In this paper new heuristics are also proposed to improve CSANN's performance. Experimental study shows that the simplified CSANN together with proposed heuristics has good performance with respect to the quality of solutions and the computing speed.

2 Job-shop scheduling problem

2.1 Formulation of the JSP

Usually, two types of constraints exist for JSPs, *sequence constraint* and *resource constraint*. The first type means that two operations of a job cannot be processed at the same time. The second type states that no more than one job can be handled on a machine at the same time. Job-shop scheduling can be viewed as an optimization problem, bounded by both sequence and resource constraints. Traditionally for a JSP, each job may have different number of operations that have a deterministic processing order on different machines. And the processing time of each operation on a machine is known and fixed. Operations can not be interrupted once started (non-preemption). This kind of scheduling is called deterministic and static, which is the focus of this paper.

Denote $J = \{J_1, \dots, J_n\}$ and $M = \{M_1, \dots, M_m\}$ as the job set and the machine set respectively, where n and m are the numbers of jobs and machines. Let n_i be the operation number of job i . O_{ikq} represents operation k of job i to be processed on machine q , T_{ikq} and P_{ikq} represent the starting time and processing time of O_{ikq} respectively, $T_{ie,q}$ and $P_{ie,q}$ represent the start time and process time of the last operation of job i respectively. Denote r_i and d_i as the release date (earliest starting time) and due date (latest ending time) of job i . Let S_i denote the set of operation pairs $[O_{ikp}, O_{ilq}]$ of job i , where O_{ikp} must be processed before O_{ilq} . Let R_q be the set of operations O_{ikq} to be processed on machine q . Taking minimizing the makespan as the optimization criterion, the JSP can be represented as follows:

Minimize $E = \max_{i \in J} (T_{ie,q} + P_{ie,q})$, subject to

$$T_{ilq} - T_{ikp} \geq P_{ikp},$$
$$[O_{ikp}, O_{ilq}] \in S_i, k, l \in \{1, \dots, n_i\}, i \in J \quad (1)$$

$$T_{jlq} - T_{ikq} \geq P_{ikq} \text{ or } T_{ikq} - T_{jlq} \geq P_{jlq},$$
$$O_{ikq}, O_{jlq} \in R_q, i, j \in J, q \in M \quad (2)$$

$$r_i \leq T_{ijq} \leq d_i - P_{ijq}, \quad i \in J, j \in \{1, \dots, n_i\}, q \in M \quad (3)$$

where the cost function is the complete time of the latest operation. Minimizing the cost function means minimizing the makespan. Eq. (1) represents the sequence constraint; Eq. (2) represents resource constraints in a disjunctive form; Eq. (3) represents the release and due date constraints.

2.2 Giffler and Thompson algorithms

Given a feasible schedule for JSPs, if an operation can be left-shifted (started earlier) without altering the processing sequences, such a left-shift is called a *local left-shift*. If a left-shift of an operation alters the processing sequences but does not delay any other operations, it is called a *global left-shift*. Based on the concept of local and global left-shift, feasible schedules for JSPs can be classified into four types: *inadmissible*, *semi-active*, *active* and *non-delay*. Inadmissible schedules are those that contain excess idle time and can be improved by local and/or global left-shift(s). Semi-active schedules are those that allow no local left-shift. Active schedules are those that allow neither local nor global left-shift. Non-delay schedules are active schedules in which no machine is kept idle while some operation can be processed. An optimal schedule is guaranteed to be an active one but not necessarily a non-delay one [1].

Giffler and Thompson [4] first proposed a systematic method, henceforth the *GT-Active* algorithm, to generate any active schedules for JSPs. Let $ES(O)$ and $EC(O)$ denote the earliest (possible) start time and earliest (possible) completion time of an operation O respectively. An active schedule is generated by repeating the algorithm until all operations are scheduled.

- 1). Let D be a set of all unscheduled operations. Find an operation O^* (with ties broken randomly) that has the minimum earliest (possible) completion time in D . Let M^* denote the machine that processes O^* .
- 2). Construct the *conflict set* C which contains unscheduled operations in D that are processed on M^* and whose processing will overlap with O^* . That is, $C := \{O \in D \mid O \text{ on } M^*, ES(O) < EC(O^*)\}$.
- 3). Select an operation $O \in C$ randomly and schedule it.

In [4] Giffler and Thompson also developed an algorithm, henceforth the *GT-ND* algorithm, to generate any non-delay schedules for JSPs iteratively as follows:

- 1). Let D be a set of all unscheduled operations. Find an operation O^* (with ties broken randomly) that has the minimum earliest (possible) start time in D . Let M^* denote the machine that processes O^* .
- 2). Construct the *conflict set* C which contains unscheduled operations in D that are processed on M^* and whose processing will overlap with O^* . That is, $C := \{O \in D \mid O \text{ on } M^*, ES(O) < EC(O^*)\}$.
- 3). Select an operation $O \in C$ randomly and schedule it.

Both the *GT-Active* and *GT-ND* algorithms have become the basis for many priority-rule based heuristics and hybrid

scheduling systems for JSPs. They will be used as peer algorithms in this paper for comparing the performance of CSANN, to be described next, for JSPs.

3 Original CSANN model for JSPs

3.1 Neurons of CSANN

Usually, a neural unit i consists of a linear summator and a nonlinear activation function $f(\cdot)$, serialized as below:

$$A_i = f(N_i) = f\left(\sum_{j=1}^n (W_{ij} \times A_j) + B_i\right), \quad (4)$$

where the summator sums a bias B_i and received activations A_j ($j = 1, \dots, n$) from connected units with connection weight W_{ij} from unit j to unit i . The output of summator is the net input N_i to neuron i , which is then passed to the activation function $f(\cdot)$ to obtain the activation A_i .

Based on the general neuron model, CSANN contains three kinds of units: *ST-units*, *SC-units* and *RC-units*. *ST-units* represent operations with the activation of each *ST-unit* representing the start time of an operation. *SC-units* and *RC-units* represent whether the sequence constraints and resource constraints are satisfied respectively. The net input and activation functions of an *ST-unit*, ST_i , are defined as:

$$N_{ST_i}(t) = \sum_j (W_{ij} \times A_{SC_j}(t)) + \sum_k (W_{ik} \times A_{RC_k}(t)) + A_{ST_i}(t-1) \quad (5)$$

$$A_{ST_i}(t) = \begin{cases} r_i, & N_{ST_i}(t) < r_i \\ N_{ST_i}(t), & r_i \leq N_{ST_i}(t) \leq d_i - P_{ST_i} \\ d_i - P_{ST_i}, & N_{ST_i}(t) > d_i - P_{ST_i} \end{cases} \quad (6)$$

where in Eq. (5) the net input of ST_i is summed from three parts. The first and second parts come from the weighted activations of *SC-units* and *RC-units* related to ST_i , which implement feedback adjustments due to sequence and resource violations respectively. The third part comes from previous activation of unit ST_i itself. The activation function in Eq. (6) is a linear-segmented function, where r_i and d_i are the release and due date of job i to which the operation, corresponding to ST_i , belongs. P_{ST_i} is the processing time of the operation. This activation function implements the release and due date constraints described by Eq. (3).

The net input and activation functions of an *SC-unit* SC_i or *RC-unit* RC_i have the same definition as shown below:

$$N_{C_i}(t) = W_1 \times A_{ST_1}(t) + W_2 \times A_{ST_2}(t) + B_{C_i} \quad (7)$$

$$A_{C_i}(t) = \begin{cases} 0, & N_{C_i}(t) \geq 0 \\ -N_{C_i}(t), & N_{C_i}(t) < 0 \end{cases} \quad (8)$$

where C_i represents SC_i or RC_i and B_{C_i} is the bias, which equals the processing time of a relative operation. The *ST-units*, ST_1 and ST_2 , represent two operations of the same job for an *SC-unit*, or two operations sharing the same machine for a *RC-unit*. The activation function is linear-segmented. When the activation of an *SC-unit* or *RC-unit* is greater than zero, it means the relevant sequence or resource constraint is violated and there will be feedback adjustments from SC_i or RC_i to connected ST_1 and ST_2 with adaptive weights.

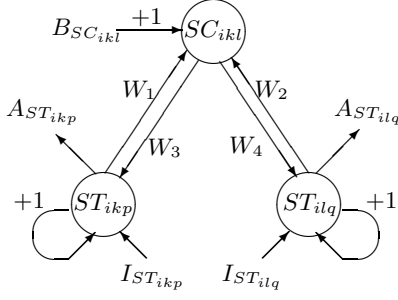


Figure 1: A SC-block unit SCB_{ikl} .

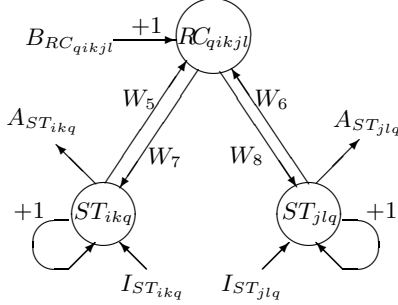


Figure 2: A RC-block unit RCB_{qikjl} .

3.2 Adaptive connection weights and biases

All neurons in CSANN are structured into two problem-specific constraint blocks: sequence constraint block (*SC-block*) and resource constraint block (*RC-block*). Each SC-block unit has two ST-units that represent two operations of a job and one SC-unit that represents whether the relevant sequence constraint is satisfied, see Figure 1. Similarly, each RC-block unit has two ST-units representing two operations on the same machine and one RC-unit representing whether the relevant resource constraint is satisfied, see Figure 2.

Figure 1 shows an example SC-block unit SCB_{ikl} . ST-units ST_{ikp} and ST_{ilq} represent two operations O_{ikp} and O_{ilq} of job i . Their activations $A_{ST_{ikp}}$ and $A_{ST_{ilq}}$ represent the start times T_{ikp} and T_{ilq} . The SC-unit SC_{ikl} represents the sequence constraint of Eq. (1) between O_{ikp} and O_{ilq} , with $B_{SC_{ikl}}$ being its bias. $I_{ST_{ikp}}$ and $I_{ST_{ilq}}$ represent the initial value for T_{ikp} and T_{ilq} , which are taken as the initial net input to ST_{ikp} and ST_{ilq} respectively. The weights and bias are valued as follows:

$$W_1 = -1, W_2 = 1, W_3 = -W, W_4 = W, B_{SC_{ikl}} = -P_{ikp} \quad (9)$$

where W , henceforth, is positive feedback adjustment factor. At time t during the run of CSANN, if the sequence constraint between O_{ikp} and O_{ilq} is satisfied, the activation $A_{SC_{ikl}}(t)$ of SC_{ikl} equals zero; otherwise, the activation of SC_{ikl} will be greater than zero and can be calculated by

$$\begin{aligned} A_{SC_{ikl}}(t) &= -N_{SC_{ikl}}(t) = A_{ST_{ikp}}(t) + P_{ikp} - A_{ST_{ilq}}(t) \\ &= T_{ikp}(t) + P_{ikp} - T_{ilq}(t) \end{aligned} \quad (10)$$

The feedback adjustments from SC_{ikl} to ST_{ikp} and ST_{ilq} are shown as follows:

$$A_{ST_{ikp}}(t+1) = T_{ikp}(t+1) = T_{ikp}(t) - W \times A_{SC_{ikl}}(t) \quad (11)$$

$$A_{ST_{ilq}}(t+1) = T_{ilq}(t+1) = T_{ilq}(t) + W \times A_{SC_{ikl}}(t) \quad (12)$$

where the feedback adjustments put backward the start time T_{ikp} of O_{ikp} and put forward T_{ilq} of O_{ilq} . Thus, the sequence constraint violation between O_{ikp} and O_{ilq} may be solved.

Figure 2 shows an example RC-block unit RCB_{qikjl} , representing the resource constraint of Eq. (2) between O_{ikq} and O_{jlq} on machine q . At time t during the run of CSANN, the weights and bias are adaptively valued with two cases.

Case 1: If $A_{ST_{ikq}}(t) \leq A_{ST_{jlq}}(t)$, i.e., $T_{ikq}(t) \leq T_{jlq}(t)$, Eq. (13) holds

$$W_5 = -1, W_6 = 1, W_7 = -W, W_8 = W, B_{RC_{qikjl}} = -P_{ikq} \quad (13)$$

In this case, RCB_{qikjl} represents a sequence constraint described by the first disjunctive equation of Eq. (2). If violation exists, the activation of RC_{qikjl} and feedback adjustments from RC_{qikjl} to ST_{ikq} and ST_{jlq} are calculated by

$$\begin{aligned} A_{RC_{qikjl}}(t) &= A_{ST_{ikq}}(t) + P_{ikq} - A_{ST_{jlq}}(t) \\ &= T_{ikq}(t) + P_{ikq} - T_{jlq}(t) \end{aligned} \quad (14)$$

$$\begin{aligned} A_{ST_{ikq}}(t+1) &= T_{ikq}(t+1) = A_{ST_{ikq}}(t) + W_7 \times A_{RC_{qikjl}}(t) \\ &= T_{ikq}(t) - W \times A_{RC_{qikjl}}(t) \end{aligned} \quad (15)$$

$$\begin{aligned} A_{ST_{jlq}}(t+1) &= T_{jlq}(t+1) = A_{ST_{jlq}}(t) + W_8 \times A_{RC_{qikjl}}(t) \\ &= T_{jlq}(t) + W \times A_{RC_{qikjl}}(t) \end{aligned} \quad (16)$$

Case 2: If $A_{ST_{ikq}}(t) \geq A_{ST_{jlq}}(t)$, that is, $T_{ikq}(t) \geq T_{jlq}(t)$, Eq. (17) holds

$$W_5 = 1, W_6 = -1, W_7 = W, W_8 = -W, B_{RC_{qikjl}} = -P_{jlq} \quad (17)$$

In this case RCB_{qikjl} represents a sequence constraint described by the second disjunctive equation of Eq. (2). If there exists violation, the activation of RC_{qikjl} and the feedback adjustments are calculated by

$$\begin{aligned} A_{RC_{qikjl}}(t) &= A_{ST_{jlq}}(t) + P_{jlq} - A_{ST_{ikq}}(t) \\ &= T_{jlq}(t) + P_{jlq} - T_{ikq}(t) \end{aligned} \quad (18)$$

$$\begin{aligned} A_{ST_{ikq}}(t+1) &= T_{ikq}(t+1) = A_{ST_{ikq}}(t) + W_7 \times A_{RC_{qikjl}}(t) \\ &= T_{ikq}(t) + W \times A_{RC_{qikjl}}(t) \end{aligned} \quad (19)$$

$$\begin{aligned} A_{ST_{jlq}}(t+1) &= T_{jlq}(t+1) = A_{ST_{jlq}}(t) + W_8 \times A_{RC_{qikjl}}(t) \\ &= T_{jlq}(t) - W \times A_{RC_{qikjl}}(t) \end{aligned} \quad (20)$$

3.3 Network complexity and run mechanism

The architecture of CSANN consists of two layers. The bottom layer consists of only ST-units. The top layer consists of SC-units and RC-units that are connected to ST-units at the bottom layer according to the JSP to be solved.

For a traditional JSP with m machines and n jobs where each job goes through all machines (i.e., $n_i = m$ for all i) in certain sequencing order, it requires mn ST-units representing the mn operations, $n(m-1)$ SC-units representing the $n(m-1)$ sequence constraints described by Eq. (1), and $mn(n-1)/2$ RC-units representing the $mn(n-1)/2$ resource constraints described by Eq. (2). Totally, there are $n(0.5mn + 1.5m - 1)$ neurons for the whole CSANN.

Given a problem-specific CSANN, it can be run iteratively: first, run SC-block units, and then run RC-block units in order till the activations of all SC- and RC-units equal 0. The final activations of ST-units form a feasible schedule.

4 Improved model — CSANN-II

4.1 Simplifying the RC-block for CSANN

In the original CSANN model, for each machine any combination of two jobs corresponds to one RC-block unit, which results in $n(n-1)/2$ RC-block units for each machine¹. This number can be greatly reduced to $n-1$ using the following dynamic adaptive scheme when CSANN is running.

- 1). Before each iteration of the RC-block, sort the ST-units related to each machine according to their activations, i.e., present start times of relevant operations to be processed on the machine, in a non-decreasing order;
- 2). From the first to the last in the ordered ST-unit list, construct one RC-block unit for two adjacent ST-units. This results in $n-1$ RC-block units for each machine.

The new CSANN with above adaptive RC-block constructing scheme is henceforth called *CSANN-II*. For CSANN-II, for a traditional JSP with m machines and n jobs where each job passes through all machines in certain sequencing order, it requires mn ST-units, $n(m-1)$ SC-units, and $m(n-1)$ RC-units. Totally, CSANN-II consists of $3mn - m - n$ neurons, in the order of $O(mn)$, instead of $n(0.5mn + 1.5m - 1)$ neurons, in the order of $O(mn^2)$, for CSANN, a deduction of magnitude n with respect to the network complexity.

For each iteration of CSANN-II, sorting the ST-units for each machine requires $O(n \log n)$ calculations by the quick sort algorithm. It also requires $n(m-1)$ SC-unit calculations and $m(n-1)$ RC-unit calculations, resulting in a computational complexity of $O(mn \log n)$. In contrast, each iteration of CSANN requires $n(m-1)$ SC-unit calculations and $mn(n-1)/2$ RC-unit calculations, which is in the order of $O(mn^2)$. Hence, for each iteration of the neural network, CSANN-II achieves a deduction of magnitude $n/\log n$ over CSANN with respect to the computational complexity.

4.2 Heuristic algorithms for CSANN

4.2.1 Swapping the order of two adjacent operations

This heuristics has two aspects: accelerating the solving process and guaranteeing feasible solutions [7, 8]. The former, called Heuristic Alg. 1(a), is for two adjacent operations of the same job, while the latter, called Heuristic Alg. 1(b), is for two adjacent operations on the same machine.

For Heuristic Alg. 1(a), assuming $[O_{ikp}, O_{ilq}] \in S_i$, at time t during the running of neural networks, if $A_{ST_{ikp}}(t) \geq$

$A_{ST_{ilq}}(t)$ (i. e., $T_{ikp}(t) \geq T_{ilq}(t)$), exchange the order of O_{ikp} and O_{ilq} by exchanging their start times as follows:

$$A_{ST_{ikp}}(t+1) = T_{ikp}(t+1) = T_{ilq}(t) \quad (21)$$

$$A_{ST_{ilq}}(t+1) = T_{ilq}(t+1) = T_{ikp}(t) \quad (22)$$

In fact, Eqs. (21) and (22) form a more direct method of removing sequence constraint violations than the feedback adjustment scheme in CSANN. Thus, the adjustment time for removing sequence constraint violations is shortened and the solving process is speeded up.

During the running of CSANNs, due to conflicts resulting from sequence constraint and resource constraint violation feedback adjustments, the phenomenon of *dead lock* may happen [7]. Dead lock will stop CSANNs from obtaining a feasible solution. Heuristic Alg. 1(b) was proposed to break dead lock (and hence guarantee obtaining feasible schedules) by exchanging the order of two adjacent operations on the same machine via exchanging their start times. Heuristic Alg. 1(b) works as follows.

For each RC-block unit RCB_{qikjl} , a variable $T_{qikjl}(t)$ is defined to count the number of continuous and similar feedback adjustments, accumulated over iterations, from RCB_{qikjl} to ST_{ikq} and ST_{jlq} due to the resource constraint violation between O_{ikq} and O_{jlq} on machine q . Two feedback adjustments are called *similar* if they have the same effect on ST_{ikq} and ST_{jlq} , e.g., both putting T_{ikq} forward and T_{jlq} backward. Whenever the resource constraint between O_{ikq} and O_{jlq} is satisfied or a different feedback adjustment occurs within RCB_{qikjl} , $T_{qikjl}(t)$ will be reset to zero. However, during the running of CSANNs, if dead lock happens $T_{qikjl}(t)$ will keep increasing over iterations of CSANNs. And when $T_{qikjl}(t)$ reaches a prescribed threshold T (e.g., $T = 5$), the equations below will take effect and swap the order, i.e., the start times, of O_{ikq} and O_{jlq} on machine q .

$$A_{ST_{ikq}}(t+1) = T_{ikq}(t+1) = T_{jlq}(t) \quad (23)$$

$$A_{ST_{jlq}}(t+1) = T_{jlq}(t+1) = T_{ikq}(t) \quad (24)$$

4.2.2 Adapting expected makespan

For a JSP without due date constraints, before running CSANNs, an expected makespan is prescribed, which is what the scheduler wants to achieve and can be used as the common due date of all jobs. The value of expected makespan affects the performance of CSANNs greatly: if set too loose, the quality of obtained schedules will be low, while set too tight, it will take CSANNs too long to obtain a schedule. This qualifies the importance of selecting a proper expected makespan for CSANNs to run.

In this paper an adaptive scheme, called Heuristic Alg. 2, is proposed to obtain a proper expected makespan by adding a preprocessing stage. Let $\sum P$ denote the total processing time of all operations and $\sum O$ total number of operations (e.g., $\sum O = mn$). Heuristic Alg. 2 is shown as follows:

- 1). Set the initial expected makespan $EM(0) = 0.5 \times \sum P$;

¹Assuming each of the n jobs of a JSP passes through all machines.

- 2). Run CSANN for τ times and calculate the mean iterations $\bar{I} = \sum I / \tau$ that CSANN used to reach a schedule;
- 3). If $\bar{I} < \rho \times \sum O$, $EM(k+1) = EM(k) - 0.01 \times \sum P$ and go to step 2; Otherwise, stop the preprocessing stage and return $EM(k)$ as the final expected makespan.

where in step 3, τ and ρ are parameters for the preprocessing stage. The aim of Heuristic Alg. 2 is to obtain a proper expected makespan that makes the average iterations CSANNs require for a schedule to be linear with respect to $\sum O$.

4.2.3 Improving the quality of solutions

The feasible schedules obtained by CSANN and CSANN-II are usually inadmissible, where there may exist many idle times for each machine while some operations are available to be processed. The schedules can be improved by compacting away these idle times. In [8], an algorithm, henceforth called Heuristic Alg. 3(a), is used to obtain a semi-active schedule from the schedule obtained by CSANN as follows:

- 1). Given a feasible schedule obtained by CSANN, sort all operations in non-decreasing order of their start times.
- 2). From the first to the last in the ordered operation list, each operation is moved forward to its earliest possible start time by carrying out local left-shift only.

In this paper another algorithm, henceforth called Heuristic Alg. 3(b), is proposed to generate an active schedule from the feasible schedule obtained by CSANN by replacing step 2 of Heuristic Alg. 3(a) with the following:

- 2). From the first to the last in the ordered operation list, each operation is moved forward to its earliest possible start time by first carrying out global left-shift (if possible) and then local left-shift.

4.3 Hybrid CSANN and heuristics for JSPs

The hybrid approach for JSPs consists of CSANN or CSANN-II and the proposed heuristics. Usually, the hybrid approach is executed many times to obtain a number of schedules and best of all schedules will be used as the final schedule. The running strategy is shown as follows:

- 1). Construct a neural network for a specific JSP, prescribe values for W , T , τ , ρ , and the maximum number of schedules $MaxSched$ to be calculated;
- 2). Perform the preprocessing stage with Heuristic Alg. 2 to obtain a proper expected makespan;
- 3). Run CSANN or CSANN-II for one schedule with above obtained expected makespan;
- 4). If $MaxSched$ is reached, stop; otherwise, go to step 3.

The procedure of running CSANN or CSANN-II for one schedule is shown as follows:

- 1). Randomly initialize $T_{ikp}(0)$ for each operation O_{ikp} , and take it as the initial net input $I_{ST_{ikp}}$ to ST_{ikp} ;
- 2). Run each SC-unit SC_{ikl} of the SC-block, calculate its activation with Eq. (10). $A_{SC_{ikl}}(t) \neq 0$ means the violation of related sequence constraint, then adjust activations of related ST-units with Eqs. (11) and (12) or Eqs. (21) and (22) if Heuristic Alg. 1(a) is triggered;

Table 1: Experimental results of comparing methods.

Measure	Alg.	FT06	FT10	FT20
Makespan (min/ave/std)	CSANN-II	55/55/0.0	982/1009/9.9	1292/1334/12.7
	CSANN	55/55/0.0	1053/1097/18.6	1450/1492/19.3
	GT-Active	55/56.2/0.8	1048/1102/17.9	1336/1383/16.7
	GT-ND	55/56.7/0.9	1035/1112/18.6	1301/1372/16.1
Time Used in Seconds (min/ave/std)	CSANN-II	31/129/166.5	222/753/607.1	820/941/168.6
	CSANN	26/854/1047.8	433/1062/526.7	1607/1848/382.7
	GT-Active	4/4.3/0.5	16/16.9/0.6	33/34.2/1.0
	GT-ND	4/4.18/0.4	16/17.1/0.8	32/33.1/0.9
Makespan		t-Test Result		
CSANN-II – CSANN		0.0	-29.56	-48.17
CSANN-II – GT-Active		-10.43	-32.21	-16.38
CSANN-II – GT-ND		-14.04	-34.43	-13.18
Time Used		t-Test Result		
CSANN-II – CSANN		-4.83	-2.72	-15.34
CSANN-II – GT-Active		5.30	8.57	38.04
CSANN-II – GT-ND		5.31	8.57	38.08

- 3). For CSANN-II, construct the RC-block adaptively;
- 4). Run each RC-unit RC_{qikjl} of the RC-block, calculate its activation with Eq. (14) or Eq. (18). $A_{RC_{qikjl}}(t) \neq 0$ means the violation of resource constraint corresponding to Eq. (2). Then adjust $A_{ST_{ikq}}(t+1)$ and $A_{ST_{jlq}}(t+1)$ with Eqs. (15) and (16) or Eqs. (19) and (20), or with Eqs. (23) and (24) if Heuristic Alg. 1(b) is triggered;
- 5). Repeat step 2 to 4 until all neurons become stable without changes, i.e., all sequence and resource constraints are satisfied and an feasible schedule is obtained;
- 6). Use Heuristic Alg. 3(a) or 3(b) to obtain a semi-active or active schedule from the feasible schedule obtained by CSANN or CSANN-II respectively.

5 Experimental study

The experimental study was finished on an Intel Pentium 4 PC running at 2.8GHz under GNU C++ programming environment in linux system. The benchmark problems, Muth and Thompson's FT06, FT10 and FT20 JSPs [5], were taken as the test problems to compare the performance of CSANN-II, CSANN, GT-Active and GT-ND. For CSANNs parameters are set as: $W = 0.5$, $T = 5$, $\tau = 10$ and $\rho = 2$.

For each run of an algorithm on a test JSP, 10^5 schedules² were calculated with the intermediate best-so-far schedule recorded every 100 schedule. And for each run the final best schedule and time used were also recorded. In order to avoid the effect a random seed may have, 50 runs with different random seeds were carried out for each algorithm on each test problem and the mean results over 50 runs are reported.

The experimental results regarding makespan of final best schedule and time used in second are given in Table 1, where *min/ave/std* means minimum, average and standard deviation over 50 runs of algorithms respectively. Statistical comparison of algorithms by one-tailed *t*-test is also given in Table 1,

²For CSANNs the schedules calculated during the preprocessing stage were also counted into the total 10^5 schedules.

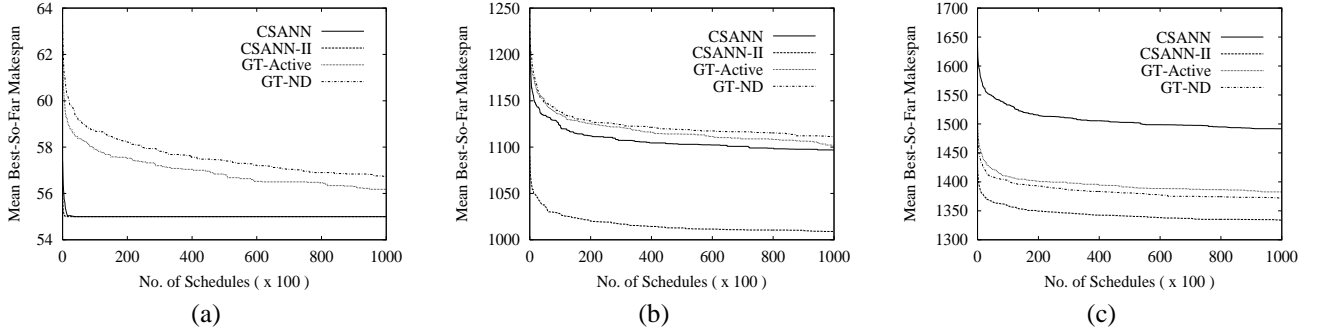


Figure 3: Experimental results on the test JSPs: (a) FT06, (b) FT10, and (c) FT20.

Table 2: Experimental results with fixed run time.

Alg.	Makespan (min/ave/std)		
	FT06	FT10	FT20
CSANN-II	55/55/0.0	982/1012/11.3	1292/1337/13.1
CSANN	55/55/0.0	1053/1105/17.9	1450/1507/22.7
GT-Active	55/55.1/0.3	1017/1075/13.5	1325/1351/11.8
GT-ND	55/55.7/0.5	1035/1073/15.2	1301/1339/13.7
t-Test Result			
CSANN-II – CSANN	0.0	-31.15	-45.79
CSANN-II – GT-Active	-2.33	-25.44	-5.44
CSANN-II – GT-ND	-9.75	-22.70	-0.49

where the t -test values shown in bold font are significant with 98 degrees of freedom at a 0.05 significance level. The experimental results regarding best-so-far makespan obtained by different methods against schedules are plotted in Figure 3, where the data were averaged over 50 runs.

From Table 1 and Figure 3 it can be seen that CSANN-II significantly outperforms CSANN regarding both the quality of obtained schedules on FT10 and FT20 and the solving speed on all test JSPs. CSANN-II also significantly outperforms GT-Active and GT-ND with respect to the quality of obtained schedules on all test JSPs but spends significantly more computational time.

In order to carry out a fairer comparison between algorithms regarding the computational time, further experiments were carried out to run algorithms on the test JSPs for certain fixed time. For each run, the algorithms are given a maximum of 60, 300, and 600 seconds on FT06, FT10, and FT20 respectively. The results regarding the final makespan are shown in Table 2, where the data were averaged over 50 runs. Table 2 shows that CSANN-II still significantly outperforms CSANN, GT-Active and GT-ND on nearly all test JSPs.

6 Conclusions

This paper proposes an improved CSANN, CSANN-II, by simplifying its RC-block. In CSANN-II, the RC-block is adaptively constructed according to the real resource constraint satisfaction situation when CSANN-II is running. Consequently, the size of the RC-block is deduced by a magnitude of n and the computational time per iteration is de-

duced by a magnitude of $n/\log n$, where n is the number of jobs in JSPs. In this paper new heuristics are also proposed to improve the performance of CSANNs, e.g., adapting the expected makespan parameter and improving the quality of obtained schedules. Experimental study shows that CSANN-II together with proposed heuristics outperforms not only the original CSANN but two typical heuristic algorithms for JSPs on the selected benchmark problems with respect to the quality of solutions and the computing speed.

CSANN-II can act as a good basis for constructing further hybrid intelligent systems for JSPs and other scheduling problems. For example, combining it with local search and genetic algorithms for JSPs is now under investigation.

References

- [1] K. R. Baker, *Introduction to Sequence and Scheduling*, John Wiley & Sons, New York, 1974.
- [2] S. Y. Foo and Y. Takefuji, “Neural networks for solving job-shop scheduling: Part 1. Problem representation,” *Proc. IEEE IJCNN*, vol. 2, pp. 275–282, 1988.
- [3] S. Y. Foo and Y. Takefuji, “Stochastic neural networks for solving job-shop scheduling: Part 2. Architecture and simulations,” *Proc. IEEE IJCNN*, vol. 2, pp. 283–290, 1988.
- [4] B. Giffler and G. Thompson, “Algorithms for solving production scheduling problems,” *Operations Research*, vol. 8, pp. 487–503, 1960.
- [5] J. F. Muth and G. L. Thompson, *Industrial Scheduling*, Prentice Hall, Englewood Cliffs, NJ, 1963.
- [6] T. M. Willems, “Neural networks for job-shop scheduling,” *Control Engg. Prac.*, vol. 2, no. 1, pp. 31–39, 1994.
- [7] S. Yang and D. Wang, “Constraint satisfaction adaptive neural network and heuristics combined approaches for generalized job-shop scheduling,” *IEEE Trans. on Neural Networks*, vol. 11, no. 2, pp. 474–486, 2000.
- [8] S. Yang and D. Wang, “A new adaptive neural network and heuristics hybrid approach for job-shop scheduling,” *Comp. & Oper. Res.*, vol. 28, no. 10, pp. 955–971, 2001.
- [9] H.-B. Yu, “Research of intelligent production scheduling methods and their applications”, *PhD Thesis*, Northeastern University, China, 1997.