2006 International Joint Conference on Neural Networks
Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

Job-Shop Scheduling with an Adaptive Neural Network and Local Search Hybrid Approach

Shengxiang Yang, *Member, IEEE*

***Abstract*— Job-shop scheduling is one of the most difficult production scheduling problems in industry. This paper proposes an adaptive neural network and local search hybrid approach for the job-shop scheduling problem. The adaptive neural network is constructed based on constraint satisfactions of job-shop scheduling and can adapt its structure and neuron connections during the solving process. The neural network is used to solve feasible schedules for the job-shop scheduling problem while the local search scheme aims to improve the performance by searching the neighbourhood of a given feasible schedule. The experimental study validates the proposed hybrid approach for job-shop scheduling regarding the quality of solutions and the computing speed.**

I. 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 to solve the JSPs, priority rules, combinatorial optimization and constraints analysis [5]. Due to the hardness of solving JSPs researchers also investigated intelligent methods for JSPs [8].

Foo and Takefuji [6], [7] first used a neural network to solve JSPs. Thereafter, several neural networks have been devised by researchers for JSPs. Willems [12] first proposed a constraint satisfaction neural network for traditional JSPs. Yu [16] extended Willems's neural network by adding a job constraint block to deal with free operations. Yang and Wang in [13] devised a constraint satisfaction adaptive neural network (CSANN) for generalized JSPs where there may exist free sequence operation pairs or free operations of each job. CSANN is constructed from the constraints of a JSP and works by resolving constraint violations during its running. CSANN can adapt the connection weights and biases of neurons according to the actual constraint violations during the running of CSANN.

Recently, Yang [15] further proposed an improved CSANN model, called *CSANN-II*. In CSANN-II, the resource constraint block is constructed adaptively from actual resource constraint satisfactions during the running, which is achieved by quick sorting the jobs on each machine according to their starting time and then orderly pair two neighbouring jobs into resource constraint units. CSANN-II has reduced number of resource constraint units in the resource constraint block, which leads to reduced network complexity and hence reduced computational complexity. Several heuristics have been combined with CSANN-II to guarantee its convergence, accelerate its solving speed, and improve the quality of obtained solutions.

In this paper, a local search mechanism is further proposed to be combined into CSANN-II for JSPs. In the local search mechanism, given a feasible schedule obtained by CSANN-II, we first relax the starting time of the operations to obtain an relaxed schedule and then we perform local search from the relaxed schedule iteratively as follows. We first swap the starting times of the operation with the largest completion time and a randomly selected different operation on each machine and then use CSANN-II to obtain a feasible schedule from this resulting possibly infeasible schedule. In order to test the efficiency of the proposed hybrid approach, experiments are carried out to compare the hybrid approach with CSANN-II only and two classical heuristics on three benchmark JSPs by Muth and Thompson [11]. The experimental results show that CSANN-II with the local search scheme has good performance regarding the quality of solutions and the computing speed.

The remaining of this paper is organized as follows. Section II describes the mathematical formulation, the classification of feasible solutions, and Giffler and Thompson's classic heuristics [9] for JSPs. Section III presents in detail the model of CSANN-II, including its neuron model and its adaptive connections and architecture. In Section IV, we describe the proposed local search scheme that can be combined with CSANN-II for better performance. Section V presents the experimental results of comparing the proposed hybrid approach with CSANN-II and two classical heuristics for JSPs. Finally, Section VI concludes this paper with some discussions on the relevant future work.

II. Job-Shop Scheduling Problem

A. Description of the Job-Shop Scheduling Problem

Traditionally, the JSP can be stated as follows [1]: given n jobs to be processed on m machines in a prescribed order. The objective is to optimally arrange the processing order and the start times of operations to optimize certain criteria. Usually, for JSPs there are two types of constraints: *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. JSP can be viewed as an optimization problem, bounded by both sequence and resource constraints. In this paper we consider traditional JSPs and assume each job pass through

Shengxiang Yang is with the Department of Computer Science, University of Leicester, University Road, Leicester LE1 7RH, United Kingdom (Tel: 0044-116-2515341; Fax: 0044-116-252 3915; Email: s.yang@mcs.le.ac.uk).

0-7803-9490-9/06/$20.00/©2006 IEEE

all machines in certain sequencing order. The processing time of each operation on a machine is known and fixed. Operations can not be interrupted once started, i.e., non-preemptive. The JSP is formally described as follows.

Let $J = \{J_1, \cdots, J_n\}$ and $M = \{M_1, \cdots, M_m\}$ denote the job and machine set respectively, where n and m are the number of jobs and machines. Each job has m operations. O_{ikq} represents operation k of job i to be processed on machine q, S_{ikq} and P_{ikq} represent the start time and processing time of O_{ikq} respectively, S_{ie_iq} and P_{ie_iq} 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 SS_i be the sequence set of operation pairs $[O_{ikp}, O_{ilq}]$ of job i, where operation O_{ikp} must precede O_{ilq}. Let RS_q be the set of operations O_{ikq} to be processed on machine q.

Taking minimizing the makespan as the optimization criterion, the JSP considered can be formulated as follows:

$$Minimize \;\; E = max_{i \in J}(S_{ie_iq} + P_{ie_iq})$$
$$subject\ to$$

$$S_{ilq} - S_{ikp} \geq P_{ikp}, \quad [O_{ikp}, O_{ilq}] \in SS_i,\ k, l \in \{1, \cdots, m\},\ i \in J \qquad (1)$$

$$S_{jlq} - S_{ikq} \geq P_{ikq}\ or\ S_{ikq} - S_{jlq} \geq P_{jlq}, \quad O_{ikq}, O_{jlq} \in RS_q,\ i, j \in J,\ q \in M \qquad (2)$$

$$r_i \leq S_{ijq} \leq d_i - P_{ijq},\ \ i \in J,\ j \in \{1, \cdots, m\},\ q \in M \qquad (3)$$

In the above formulation, the cost function E is the complete time of the latest operation. Minimizing the cost function means minimizing the makespan. Eqn. (1) represents the sequence constraint between two operations of a job. Eqn. (2) represents the resource constraints between two jobs on a machine in a disjunctive format. Eqn. (3) represents the release and due date constraints of jobs.

B. Classification of Feasible Solutions for JSPs

For a given JSP, there are in fact infinite feasible schedules since arbitrary excess idle times can be inserted into a feasible schedule to create new feasible ones. 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). obviously, these kind of schedules are not of practical usefulness. Semi-active schedules are those that allow no local left-shift, but there may be allowable global left-shift(s). Active schedules are those that allow neither local left-shift(s) nor global left-shift(s). 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].

TABLE I
A LIST OF JOB-SHOP DISPATCH RULES.

Rule	Description
SPT (Shortest Processing Time)	Select an operation with the shortest processing time
LPT (Longest Processing Time)	Select an operation with the longest processing time
MWR (Most Work Remaining)	Select an operation for the job with the most total remaining processing time
LWR (Least Work Remaining)	Select an operation for the job with the least total remaining processing time
MOR (Most Operations Remaining)	Select an operation for the job with the greatest number of operations remaining
LOR (Least Operations Remaining)	Select an operation for the job with the greatest number of operations remaining

C. Giffler and Thompson Heuristics for JSPs

Giffler and Thompson [9] first proposed a systematic method, denoted *GT-Random* in this paper, to generate any active schedules for JSPs as described below. 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 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) completion time in D. That is, $O^* := \arg\min\{EC(O) | O \in D\}$. 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 on M^* with its completion time equal to $EC(O)$.

In Step 3 of the above GT-Random algorithm, if all possible choices are considered, all active (non-delay) schedules will be generated respectively, but the total number of schedules will be very large. Researchers have developed a large number of heuristic priority rules to be used in the Giffler and Thompson algorithm to select an operation from the schedulable set to be dispatched next. An extensive summary and discussion can be found in [3], [10]. Table I lists some priority rules commonly used in practice.

It is also quite common that the priority rules can be combined into the Giffler and Thompson algorithm: when dispatching an operation, one priority rule is first randomly selected from a pre-defined set of priority rules and then applied to select an operation. This hybrid method is denoted *GT-Rule* in this paper.

Both the GT-Random and GT-Rule algorithms have become the basis for many priority-rule based heuristics and hybrid scheduling systems for JSPs. They will be used as

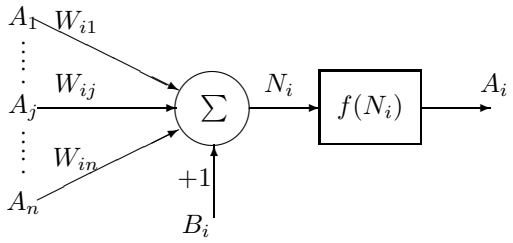

Fig. 1. General neural unit model.

peer algorithms in this paper[1] for comparing the performance of CSANNs, to be described next, for JSPs.

III. Improved Constraint Satisfaction Adaptive Neural Network — CSANN-II

A. Neurons of CSANN-II

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

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

where the summator sums a bias B_i and received activations $A_j(j = 1, \cdots, n)$ 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. Fig. 1 shows the model of a general neural unit.

Based on the general neuron model, CSANN-II contains three kinds of neurons: *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:

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

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

where in Eqn. (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 Eqn. (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 Eqn. (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:

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

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

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- or RC-unit is greater than 0, 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.

B. Adaptive Connection Weights and Biases

Usually for a neural network for constraint satisfactory problems, the connection weights between neurons are problem-specific and set in advance before it is run. In CSANN-II, the connection weights and biases are adaptive in accordance with the actual activations of ST-units while CSANN is running, together with the sequence and resource constraints of the specific JSP.

All neurons of CSANN-II are structured into two problem-specific constraint blocks: sequence constraint block (*SC-block*) and resource constraint block (*RC-block*). Each SC-block unit consists of two ST-units that represent two operations of a job and one SC-unit that represents whether relevant sequence constraint is satisfied, see Fig. 2. Similarly, each RC-block unit has two ST-units that represent two operations on a machine and one RC-unit representing whether relevant resource constraint is satisfied, see Fig. 3.

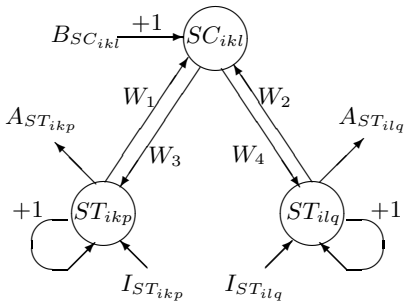

Fig. 2. A SC-block unit SCB_{ikl}.

Fig. 2 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 S_{ikp} and S_{ilq}. The SC-unit SC_{ikl} represents the sequence constraint of Eqn. (1) between O_{ikp} and O_{ilq}, with $B_{SC_{ikl}}$ being its bias. In Fig. 2, $I_{ST_{ikp}}$ ($I_{ST_{ilq}}$) represents

[1]The GT-Rule algorithm studied in this paper uses exactly the six rules in Table I as the set of priority rules

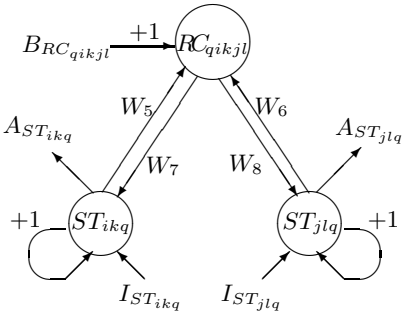

Fig. 3. A RC-block unit RCB_{qikjl}.

the initial value for S_{ikp} (S_{ilq}) that is taken as the initial net input to ST_{ikp} (ST_{ilq}). The weights and bias are valued as:

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

where W, henceforth, is positive feedback adjustment factor. At time t during the run of CSANN-II, 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

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

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

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

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

where the feedback adjustments put backward S_{ikp} of O_{ikp} and put forward S_{ilq} of O_{ilq}. Hence, the sequence constraint violation between O_{ikp} and O_{ilq} may be solved.

Fig. 3 shows an example RC-block unit, RCB_{qikjl}, which represents the resource constraint of Eqn. (2) between O_{ikq} and O_{jlq} on machine q. At time t during the run of CSANN-II, the weights and bias are adaptively valued according to the following two cases.

Case 1: If $A_{ST_{ikq}}(t) \le A_{ST_{jlq}}(t)$, i.e., $S_{ikq}(t) \le S_{jlq}(t)$, Eqn. (13) holds

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

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

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

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

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

Case 2: If $A_{ST_{ikq}}(t) \ge A_{ST_{jlq}}(t)$, that is, $S_{ikq}(t) \ge S_{jlq}(t)$, Eqn. (17) holds

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

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

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

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

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

The architecture of CSANN-II 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 a specific JSP.

C. Adaptive RC-Block Scheme

CSANN-II uses an adaptive scheme to construct the RC-block when it is running, as described below:

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.

With the above adaptive scheme, for a traditional JSP with m machines and n jobs where each job passes trough all machines in certain order, the number of RC-block units is reduced to $n-1$ for each machine instead of $n(n-1)/2$ in the original CSANN. Totally, CSANN-II requires mn ST-units, $n(m-1)$ SC-units, and $m(n-1)$ RC-units, which gives a total number of $3mn - m - n$ neurons in the order of $O(mn)$, while CSANN consists of $n(0.5mn + 1.5m - 1)$ neurons in the order of $O(mn^2)$. Hence, CSANN-II achieves a deduction of magnitude n with respect to the network complexity.

Given a problem-specific CSANN-II, it is run iteratively: first run SC-block units and then RC-block units in a fixed order until the activations of all SC-units and RC-units equal zero. The final activations of ST-units form a feasible schedule. For each iteration, CSANN-II needs to sort the ST-units for each machine, which can be done in $O(n \log n)$ time by the quick sort algorithm. It also needs calculating $n(m-1)$ SC-unit and $m(n-1)$ RC-units, resulting in a computational complexity of $O(mn \log n)$ for each iteration.

In contrast, CSANN requires $n(m-1)$ SC-unit and $mn(n-1)/2$ RC-unit calculations, totally in the order of $O(mn^2)$. Hence, computationally CSANN-II achieves a deduction of $O(n/\log n)$ over CSANN for each iteration.

D. Combined Heuristic Algorithms for CSANN-II

Several heuristics have been devised and combined into CSANN-II to guarantee its convergence, accelerate its solving speed, and improve the quality of solutions. They are described as follows.

1) Swapping the order of adjacent operations: This heuristics has two aspects: to accelerate the solving process and to guarantee obtaining feasible solution [13], [14]. 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 $[O_{ikp}, O_{ilq}] \in SS_i$, at time t during the running of neural networks, if $A_{ST_{ikp}}(t) \geq A_{ST_{ilq}}(t)$ (*i. e.*, $S_{ikp}(t) \geq S_{ilq}(t)$), exchange the order of O_{ikp} and O_{ilq} by exchanging their start times as follows:

$$A_{ST_{ikp}}(t+1) = S_{ikp}(t+1) = S_{ilq}(t) \tag{21}$$

$$A_{ST_{ilq}}(t+1) = S_{ilq}(t+1) = S_{ikp}(t) \tag{22}$$

In fact, Eqns. (21) and (22) form a more direct method of removing sequence constraint violations than the feedback adjustment scheme in CSANN-II. 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[2]. 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 RC_{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 pushing S_{ikq} forward and S_{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 threshold parameter T, the equations below will swap the order of O_{ikq} and O_{jlq} by swapping their start times.

$$A_{ST_{ikq}}(t+1) = S_{ikq}(t+1) = S_{jlq}(t) \tag{23}$$

[2]See [13] for more detailed explanation on the reason to dead lock.

$$A_{ST_{jlq}}(t+1) = S_{jlq}(t+1) = S_{ikq}(t) \tag{24}$$

2) Obtaining a proper expected makespan adaptively: For a JSP without due date constraints, before running CSANN-II, 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 CSANN-II greatly: if set too loose, the quality of obtained schedules will be low, while set too tight, it will take CSANN-II too long to obtain a schedule. This qualifies the importance of selecting a proper expected makespan for CSANN-II to run.

In [15], an adaptive scheme was proposed to obtain a proper expected makespan by adding a preprocessing stage, which is denoted Heuristic Alg. 2 in this paper. 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 $EM(0) = 0.5 \times \sum P$;
2) Run CSANN-II for τ times and compute the mean iterations $\bar{I} = \sum I/\tau$ that CSANN-II uses to obtain 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 $\sum O$.

3) Improving the quality of schedules: The feasible schedules obtained by CSANNs are usually inadmissible, which can be improved by deleting machine idle times. In [15], an algorithm, henceforth called Heuristic Alg. 3, is used to obtain an active schedule from the one obtained by CSANNs as follows:

1) Given a feasible schedule obtained by CSANN-II, sort all operations in the 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 first carrying out global left-shift (if possible) and then local left-shift.

IV. CSANN-II and Local Search Hybrid Approach

A. Local Search Mechanism

Local search mechanisms have proved helpful for improving the performance of many optimization algorithms. The basic principle of local search mechanisms is as follows: pick a (possibly bad) feasible solution, change it a little bit (usually according to a defined neighbourhood), and check whether a better solution is obtained. If so, the algorithm moves to this new solution; otherwise, just make another change to the old solution. The procedure continues until some termination condition becomes true, e.g., the maximum number of changes without improvement has reached.

The local search mechanism can be integrated into CSANNs for better performance. The idea is described as follows. From a feasible schedule obtained by CSANN-II, find the last job on each machine and swap it with one randomly selected different job on the machine by exchanging their start times. Then, run CSANN-II from the resulting (possibly infeasible) schedule to obtain a new feasible schedule. If the new schedule is better, the local search moves to this new schedule; otherwise, the local search continues from the old solution. This progress continues until a pre-set maximum number of local searches have been done.

Here, one problem lies in that if we use the schedule obtained by CSANN-II directly as the base for local search, it may take a long time to run CSANN-II for a new schedule because the original schedule is quite tight. To solve this problem, we propose a schedule relaxing technique to obtain a quite relaxed schedule from a schedule obtained by CSANN-II. Then, local search is carried out from the relaxed schedule instead. The schedule relaxing technique is described below.

B. Schedule Relaxing Technique

Given a feasible schedule obtained by CSANN-II, a relaxed schedule can be obtained in the follow steps:

1) Calculate the distance between the expected makespan and the makespan of the given schedule. Let Δ denote the distance.
2) Find a critical path in the schedule and let n denote the number of critical operations in the critical path and $d = \Delta/(n-1)$ denote the relax distance.
3) Sort all operations in the schedule in the non-decreasing order of their start times.
4) Transverse from the first to the last in the ordered operation list and postpone the start time of operations as follows: when we meet an operation with its start time larger than the start time of the ith critical operation but smaller than the start time of the $i+1$th critical operation in the critical path, its start time is right shifted by $i * d$ units of time.

Fig. 4 shows an example schedule relaxing on a JSP with three machine and three jobs. The original schedule is a tight active schedule with the unique critical path consisting of four critical operations. In the relaxed schedule, the critical path is lengthened by Δ with a gap of d inserted between two original critical operations Other non-critical operations are also postponed accordingly.

C. Framework of the Hybrid Approach for JSPs

The hybrid approach consists of CSANN-II and the proposed local search mechanism, denoted *CSANN-LS* in this paper. In practice, the hybrid approach is executed a number of times to obtain schedules and the best schedule will be used as the final schedule. The running strategy is shown as follows:

1). Construct a CSANN-II for a specific JSP, set values for W, T, τ, ρ, and the maximum number of schedules $MaxSched$ to be calculated;

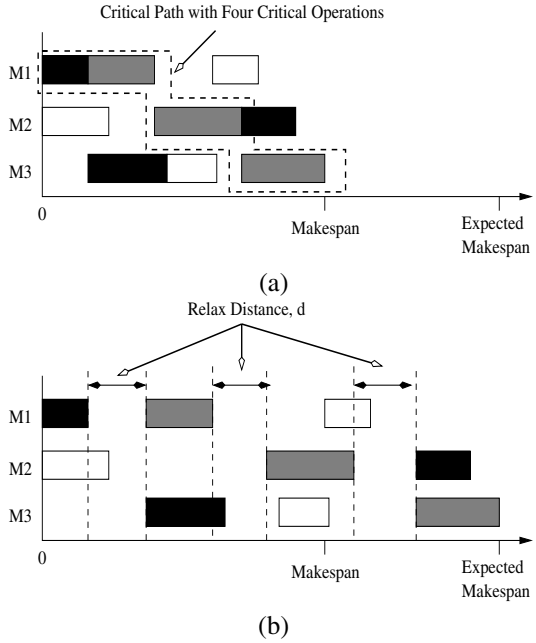

Fig. 4. Illustrating a relaxing operation: (a) before and (b) after.

2). Perform the pre-processing stage with Heuristic Alg. 2 to obtain a proper expected makespan;
3). Run CSANN-II for one schedule with the above obtained expected makespan;
4). Create a relaxed schedule from the schedule obtained by CSANN-II;
5). Perform a local search from the relaxed schedule;
6). If the maximum number of schedules, $MaxSched$, is reached, stop; otherwise, go to step 4.

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

1). Randomly initialize $S_{ikp}(0)$ or set $S_{ikp}(0)$ according to a given schedule for each operation O_{ikp}, and take it as the initial input $I_{ST_{ikp}}$ to ST-unit ST_{ikp};
2). Run each SC-unit SC_{ikl} of the SC-block: calculate its activation with Eqn. (10). If $A_{SC_{ikl}}(t) \neq 0$, it means the violation of related sequence constraint, then adjust activations of related ST-units with Eqns. (11) and (12) or Eqns. (21) and (22) if Heuristic Alg. 1(a) is triggered;
3). Construct the RC-block adaptively;
4). Run each RC-unit RC_{qikjl} of the RC-block, calculate its activation with Eqn. (14) or Eqn. (18). $A_{RC_{qikjl}}(t) \neq 0$ means the violation of resource constraint corresponding to Eqn. (2). Then adjust $A_{ST_{ikq}}(t+1)$ and $A_{ST_{jlq}}(t+1)$ with Eqns. (15) and (16) or Eqns. (19) and (20), or with Eqns. (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 to obtain an active schedule from the feasible schedule obtained by CSANN-II.

V. Experimental Study

A. Experimental Setting

The experimental study was finished on a 2.8GHz Intel Pentium 4 PC under GNU C++ programming environment. The benchmark problems, Muth and Thompson's FT06, FT10 and FT20 JSPs [11], were taken as the test problems to compare the performance of CSANN-LS, CSANN-II, GT-Random and GT-Rule. The parameters for CSANNs 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 schedules[3] 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.

B. Experimental Results and Analysis

The experimental results regarding makespan of final best schedule and time used in second are given in Table II, 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 II, 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 of algorithms against schedules are plotted in Fig. 5, where the data was averaged over 50 runs.

TABLE II
EXPERIMENTAL RESULTS OF COMPARING METHODS.

Measure	Algorithm	FT06	FT10	FT20
Makespan	CSANN-LS	55/55/0.0	971/999/16.1	1221/1269/25.8
(Min/Ave/Std)	CSANN-II	55/55/0.0	982/1009/9.9	1292/1334/12.7
	GT-Random	55/56.2/0.8	1048/1102/17.9	1336/1383/16.7
	GT-Rule	55/56.8/0.8	1073/1116/15.7	1333/1379/17.4
Time Used	CSANN-LS	16/78.3/105.5	68/675/916.2	49/100/61.5
in Seconds	CSANN-II	31/129/166.5	222/753/607.1	820/941/168.6
(Min/Ave/Std)	GT-Random	4/4.3/0.46	16/16.9/0.56	33/34.2/1.0
	GT-Rule	4/4.6/0.49	17/18.0/0.45	35/38.9/3.9
t-Test Results Regarding Markspan				
CSANN-LS – CSANN-II		0.0	**-3.82**	**-15.89**
CSANN-LS – GT-Random		**-10.43**	**-30.35**	**-26.07**
CSANN-LS – GT-Rule		**-15.92**	**-36.78**	**-24.95**
t-Test Results Regarding Time Used				
CSANN-LS – CSANN-II		**-1.83**	-0.5	**-33.14**
CSANN-LS – GT-Random		**4.96**	**5.08**	**7.59**
CSANN-LS – GT-Rule		**4.94**	**5.07**	**7.03**

From Table II and Fig. 5, it can be seen that CSANN-LS significantly outperforms CSANN-II regarding both the quality of obtained schedules on FT10 and FT20 and the solving speed on all test JSPs. CSANN-LS also significantly outperforms GT-Random and GT-Rule with respect to the quality of obtained schedules on all test JSPs but spends significantly more computational time.

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

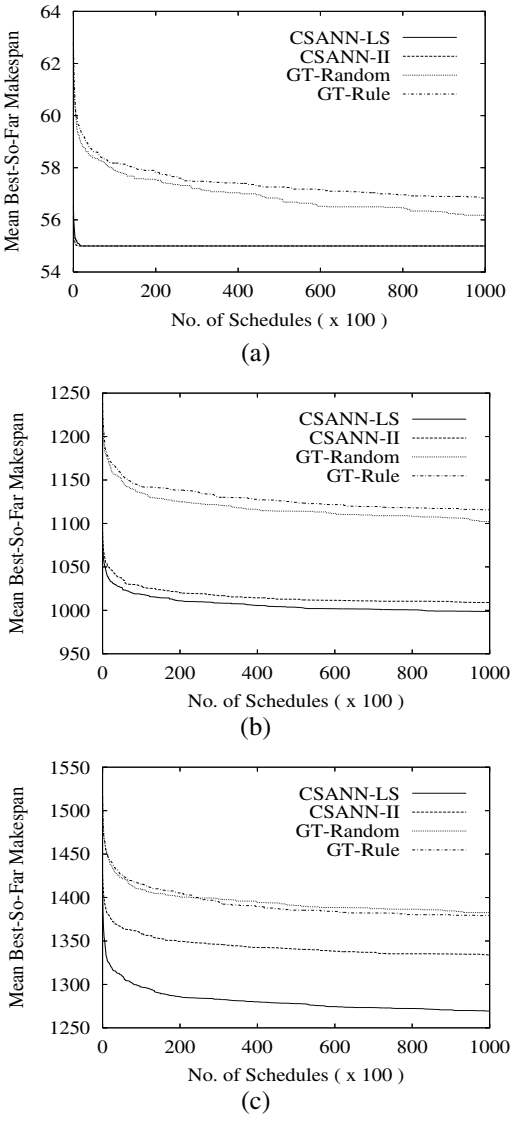

Fig. 5. Test results on (a) FT06, (b) FT10 and (c) FT20.

From Fig. 5, it can be seen that on FT06 both CSANN-II and CSANN-LS reach the optimum makespan 55 very quickly while both GT-Random and GT-Rule fail to achieve the optimum within 10^5 schedules. On FT10, CSANN-LS perform better than CSANN-II and on FT20, CSANN-LS clears wins over CSANN-II. On all the three problems, CSANN-II and CSANN-LS outperform GT-Random and GT-Rule clearly with respect to the best-so-far makespan against schedules tried.

In order to carry out a fairer comparison between the algorithms in terms of the computational time, further experiments were carried out to run the 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

TABLE III

EXPERIMENTAL RESULTS OF COMPARING ALGORITHMS WITH THE MAXIMUM ALLOWABLE RUN TIME SET TO 60, 300, AND 600 SECONDS FOR FT06, FT10, AND FT20 RESPECTIVELY.

	Makespan (Min/Ave/Std)		
Algorithm	FT06	FT10	FT20
CSANN-LS	55/55.04/0.3	971/999/12.8	1201/1250/19.6
CSANN-II	55/55/0.0	982/1012/11.3	1292/1337/13.1
GT-Random	55/55.1/0.3	1017/1075/13.5	1325/1351/11.8
GT-Rule	55/55.4/0.6	1060/1087/11.2	1325/1350/12.8
t-Test Result			
CSANN-LS – CSANN-II	1.0	**-5.43**	**-26.26**
CSANN-LS – GT-Random	-1.02	**-29.06**	**-31.23**
CSANN-LS – GT-Rule	**-7.89**	**-36.87**	**-30.31**

makespan, also averaged over 50 runs, are shown in Table III. From Table III it can been seen CSANN-LS still significantly outperforms CSANN-II, GT-Random and GT-Rule on nearly all test JSPs.

VI. CONCLUSIONS AND FUTURE WORK

This paper proposes a local search scheme for the improved adaptive neural network, CSANN-II. In the hybrid approach CSANN-LS, CSANN-II together with some heuristics is used to obtain feasible schedules for JSPs while the local search scheme is used to iterate the obtained schedules for better schedules. The schedule relaxing technique helps shorten the run time for CSANN-II to obtain a new schedule from a possibly infeasible schedule that has just been changed by the local search scheme.

For JSPs on the selected benchmark problems, experimental study shows that CSANN-LS yields higher quality solutions than CSANN-II and the two classical Giffler and Thompson's heuristic algorithms. CSANN-II is also much faster in computing speed than CSANN-II, roughly by a factor of 2 to 15, but CSANN-LS is 2 to 4 times slower than the two classical approaches to find a fixed number of solutions for the benchmark JSPs. When the solving time is fixed, both CSANN-II and CSANN-LS can find higher quality solutions than the two classical approaches.

CSANN-II can act as a good base for constructing further hybrid systems for JSPs and other production scheduling problems. It is noticeable that many other advanced intelligent methods for JSPs, e.g., hybrid genetic algorithms [4], are in fact based on the Giffler and Thompson's heuristics. We believe integrating CSANN-II into those advanced intelligent methods to replace the Giffler and Thompson's heuristics will further improve the performance for JSPs. This work is now under investigation.

REFERENCES

[1] K. R. Baker, *Introduction to Sequence and Scheduling*, New York: John Wiley & Sons, 1974.

[2] C. Bierwirth and D. Mattfeld, "Production scheduling and rescheduling with genetic algorithms," *Evolutionary Computation*, Vol. 7, No. 1, pp. 1–17, 1999.

[3] J. Blackstone, D. Phillips, and G. Hogg, "A state-of-the-art survey of dispatching rules for manufacturing job shop operations," *International Journal of Production Researches*, Vol. 20, pp. 27–45, 1982.

[4] R. Cheng, M. Gen and Y. Tsujimura, "A tutorial survey of job-shop scheduling problems using genetic algorithms, Part II: Hybrid genetic search strategies," *Computers and Industrial Engineering*, Vol. 36, pp. 343–364, 1999.

[5] D. Dubois, H. Fargier and H. Prade, "Fuzzy constraints in job-shop scheduling," *Journal of Intelligent Manufacturing*, Vol. 6, pp. 215–234, 1995.

[6] S. Y. Foo and Y. Takefuji, "Neural networks for solving job-shop scheduling: Part 1. Problem representation," *Proc. IEEE IJCNN*, Vol. II, pp. 275–282, 1988.

[7] S. Y. Foo and Y. Takefuji, "Stochastic neural networks for solving job-shop scheduling: Part 2. Architecture and simulations," *Proc. IEEE IJCNN*, Vol. II, pp. 283–290, 1988.

[8] M. S. Fox and M. Zweben, *Knowledge-based Scheduling*, San Manteo, CA: Morgan Kaufmann Publishers, 1993.

[9] B. Giffler and G. Thompson, "Algorithms for solving production scheduling problems," *Operations Research*, Vol. 8, pp. 487–503, 1960.

[10] R. Haupt, "A survey of priority-rule based scheduling problem," *OR Spektrum*, Vol. 11, pp. 3–16, 1989.

[11] J. F. Muth and G. L. Thompson, *Industrial Scheduling*, Prentice Hall, Englewood Cliffs, NJ, 1963.

[12] T. M. Willems, "Neural networks for job-shop scheduling," *Control Engineering Practice*, Vol. 2, No. 1, pp. 31–39, 1994.

[13] 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.

[14] S. Yang and D. Wang, "A new adaptive neural network and heuristics hybrid approach for job-shop scheduling," *Compters & Operations Research*, Vol. 28, No. 11, pp. 955–971, 2001.

[15] S. Yang, "An improved adaptive neural network for job-shop scheduling," *Proc. of IEEE 2005 Int. Conf. on Systems, Man and Cybernetics*, Vol. 2, pp. 1200–1205, 2005.

[16] H.-B. Yu, *Research of intelligent production scheduling methods and their applications*, PhD Thesis, Northeastern University, China, 1997.