PERGAMON

Computers & Operations Research 28 (2001) 955–971

computers & operations research

www.elsevier.com/locate/dsw

A new adaptive neural network and heuristics hybrid approach for job-shop scheduling☆

Shengxiang Yang[a,*], Dingwei Wang[b]

[a]*Department of Computer Science, King's College London, University of London, London, WC2R 2LS, UK*
[b]*Department of Systems Engineering, Northeastern University, Shenyang 110006, People's Republic of China*

Received 1 May 1998; received in revised form 1 April 1999

Abstract

A new adaptive neural network and heuristics hybrid approach for job-shop scheduling is presented. The neural network has the property of adapting its connection weights and biases of neural units while solving the feasible solution. Two heuristics are presented, which can be combined with the neural network. One heuristic is used to accelerate the solving process of the neural network and guarantee its convergence, the other heuristic is used to obtain non-delay schedules from the feasible solutions gained by the neural network. Computer simulations have shown that the proposed hybrid approach is of high speed and efficiency. The strategy for solving practical job-shop scheduling problems is provided.

Scope and purpose

Job-shop scheduling is usually a strongly NP-complete problem of combinatorial optimization problems and is the most typical one of the production scheduling problems. It is usually very hard to find its optimal solution. Practically researchers turn to search its near-optimal solutions with all kind of heuristic algorithms. The scope of this paper is to present a new hybrid approach in dealing with this job-shop scheduling problem based on adaptive neural network and heuristics.

Keywords: Job-shop scheduling; Adaptive neural network; Heuristics

☆This research was supported by the National Nature Science Foundation (No. 69684005) and National High-Tech Program of People's Republic of China (No. 863-511-9609-003) and was done when Shengxiang Yang was pursuing his Ph.D. degree.

* Corresponding author. Tel.: + 44-207 848 2009; fax: + 44-207 848 2851.
E-mail address: yangs@dcs.kcl.ac.uk (S. Yang)

0305-0548/01/$ - see front matter
PII: S0305-0548(00)00018-6

1. Introduction

It is well known, the job-shop scheduling problem is the most complicated and typical problem of all kinds of production scheduling problems, the allocation of resources over time to perform a collection of tasks [1]. Job-shop scheduling can be stated as follows [2]: given n jobs that have to be processed on m machines in a prescribed order under certain restrictive assumptions, the objective is to decide how to arrange the processing orders and starting times of operations sharing the same machine for each machine, in order to optimize certain criteria. Manufacturing systems with different objectives require different optimization criteria [3], such as stock size, due-date reliability, mean lead time and makespan.

Traditionally, there are three kinds of approaches to solve job-shop scheduling problems: priority rules, combinatorial optimization and constraints analysis [4]. The first kind of method has the merit of being computationally very efficient and easy to be applied to real cases, but there is no guarantee with respect to the quality of the obtained solution. Especially if some temporary constraints should be respected [5]. The optimization methods are much more rigorous but are not tractable in large size problems if the optimal solution is required [6]. The third method, originated from Erschler et al. [7], looks for a set of feasible solutions that meet several technological constraints for the user to choose the final solution.

It has been demonstrated [8] that job-shop scheduling is usually an NP-complete (nondeterministic polynomial time complete) problem. Because of the NP-complete characteristics of job-shop scheduling, it is usually very hard to find its optimal solution, and an optimal solution in the mathematical sense is not always necessary in practices [6]. Researchers turned to search its near-optimal solutions with all kind of heuristic algorithms [9]. Fortunately, the searched near-optimal solutions usually meet requirements of practical problems very well. Recently, several knowledge-based scheduling systems have been presented [10,11], which are much general than the above traditional methods because of its using constraints systematically, its implementing heuristic knowledge and its generality as a framework for stating and solving combinatorial optimization problems.

Since Hopfield [12] first used a neural network to solve an optimization problem, Hopfield networks have been successfully applied to solving a variety of problems, such as the analog-to-digital conversation problem [13], the traveling-salesman problem [14], the combinatorial optimization problem [15], the linear and non-linear programming problems [16]. However, Hopfield networks have the drawbacks of non-convergence to valid solutions, inability to locate the global minimum and poor scaling properties due to the use of quadratic energy functions, as pointed out by DARPA [17]. Since Foo and Takefuji [18,19] first used neural networks to solve job-shop scheduling problems, several neural network architectures have been presented to solve job-shop scheduling (see e.g., Foo and Takefuji [20], Foo et al. [21], Zhou et al. [22] and Willems and Brandts [23]). All the above-mentioned neural networks are basicaly non-adaptive networks with the connection weights and biases prescribed in advance before the networks begin to work.

In Yang and Wang [24] we have proposed an efficient constraint satisfaction adaptive neural network (CSANN) and heuristics combined approach for job-shop scheduling problems. CSANN differs itself from the above-mentioned networks in its adaptivity. CSANN has the property of adaptively adjusting its weights of connections and biases of neural units according to the actual constraint violations during its processing to remove these violations for obtaining feasible

solutions. In order to improve the performance of CSANN several heuristics are presented in Yang and Wang [24].

In this paper we present a new heuristic based on the property of non-delay schedules. This new heuristic together with one of the heuristics presented in Yang and Wang [24] can be combined with CSANN to form a new hybrid approach for job-shop scheduling problems. In the hybrid approach, CSANN is used to obtain feasible solutions, the heuristics from Yang and Wang [24] is used to accelerate the solving process of CSANN and guarantee feasible solutions, the new heuristic is used to obtain the non-delay solution from the feasible solution obtained by CSANN with determined orders of operations. The new hybrid approach presented in this paper is simpler and equivalently efficient (see e.g. Yang and Wang [24]). The computational simulations have shown that the proposed hybrid approach has good performance with respect to the quality of solution and the speed of computation.

This paper is organized as follows. Section 2 presents a mathematical formulation of the job-shop scheduling problem. The model of CSANN is presented in Section 3. In Section 4 the heuristics used are described, the hybrid approach is also described in this section. Section 5 presents the computer simulation results with two examples to show the performance of the proposed new hybrid approach for job-shop scheduling. Finally, the conclusions about the hybrid approach are presented in Section 6.

2. Formulation of the job-shop scheduling problem

Generally for the job-shop scheduling problem there are two types of constraints: *sequence* constraint and *resource* constraint. The first type states 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. For a job-shop scheduling problem, each job may consist of different number of operations, subjected to some precedence restrictions. Commonly the processing orders of each job by all machines and the processing time of each operation are known and fixed. Once started operations cannot be interrupted (non-preemption). This kind of scheduling is usually called deterministic and static scheduling. In this paper we consider the deterministic and static job-shop scheduling problem.

Denote $N = \{1, \ldots, n\}$ and $M = \{1, \ldots, m\}$ as the job set and the machine set, 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 (which is known in advance) of O_{ikq}, respectively, T_{ie_iq} and P_{ie_iq} represent the starting time and processing 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 $[O_{ikp}, O_{ilq}]$ with precedence restriction of job i, where operation O_{ikp} must precede operation O_{ilq}. Let R_q be the set of operations O_{ikq} that will be processed on machine q. Commonly, the starting time and the processing time of an operation are assumed to be integers.

We use the pure integer representation model to transfer the sequence constraints, resource constraints, the release date and due date constraints of jobs into integer linear inequalities. Taking minimizing the makespan as the optimization criterion, the mathematical formulation of the

job-shop scheduling problem considered is presented as follows:

$$\text{Minimize } E = \max_{i \in N} (T_{ie_iq} + P_{ie_iq})$$

subject to

$$T_{ilq} - T_{ikp} \geqslant P_{ikp}, \quad [O_{ikp}, O_{ilq}] \in S_i, \quad k, l \in \{1, \ldots, n_i\}, \quad i \in N, \tag{1}$$

$$T_{jlq} - T_{ikq} \geqslant P_{ikq} \text{ or } T_{ikq} - T_{jlq} \geqslant P_{jlq}, \quad O_{ikq}, O_{jlq} \in R_q, \quad i, j \in N, \quad q \in M, \tag{2}$$

$$r_i \leqslant T_{ijq} \leqslant d_i - P_{ijq}, \quad i \in N, \quad j \in \{1, \ldots, n_i\}, \quad q \in M, \tag{3}$$

where the cost function is the ending time of the latest operation, i.e., maximal complete time of the job-shop scheduling problem. Minimizing the cost function means minimizing the makespan. Eq. (1) represents the sequence constraint; Eq. (2), in a disjunctive type, represents resource constraints; Eq. (3) represents the release date and due date constraints.

For an $n/m/J/C_{max}$ (notation system of Conway [2]) problem, there are at most $n(m-1)$ sequence constraint inequalities of Eq. (1) type, at most $mn(n-1)$ resource constraint inequalities of Eq. (2) type, at most mn starting time constraint inequalities of Eq. (3) type, resulting in a total number of at most $n(mn+m-1)$ constraint inequalities. There are also at most mn number of variables T_{ikq}s. The objective of job-shop scheduling is to solve these variables so that they satisfy all the constraint inequalities while minimizing the makespan.

3. Model of CSANN

To solve the job-shop scheduling problem, the previous pure integer representation model has to be mapped onto the CSANN. The proposed CSANN will be discussed in detail with respect to its basic components of units and connections, its architecture and its solving process for job-shop scheduling.

3.1. Neural units of CSANN

Generally a neural unit consists of a linear summator and a nonlinear activation function which are serialized [25] (see e.g., Fig. 1). The summator of unit i receives all activations A_j $(j = 1, \ldots, n)$ from connected units and sums the received activations, weighted with connection weight W_{ij}, together with a bias B_i. The output of summator is the net input N_i, this net input N_i is passed through an activation function $f(.)$, resulting in the activation A_i of unit i. The summator and the activation function are defined as follows:

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

where W_{ij} is the connection weight from unit j to unit i.

Usually, for neural units to perform different functional behaviors, there are several types of activation functions, such as linear threshold function, linear-segmented function and S-shaped

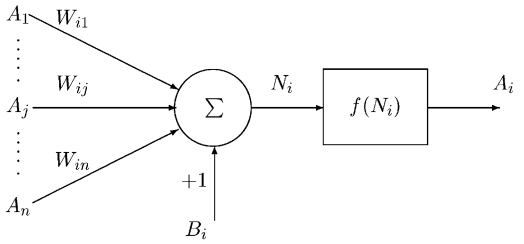

Fig. 1. General neural unit model.

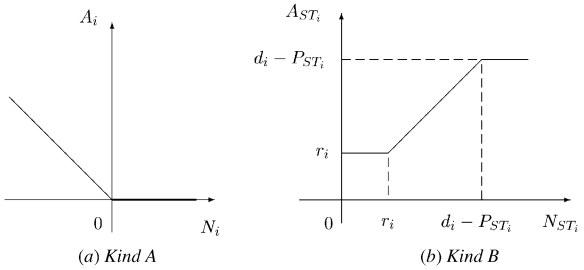

Fig. 2. Linear-segmented activation functions.

function [26]. In this paper two kinds of linear-segmented function A and B (see e.g., Fig. 2(a) and (b)) are used as the activation functions of neural units.

Based on the general neural unit, CSANN contains three kinds of units: *ST-units*, *SC-units* and *RC-units*. The first kind of units represent the starting times of all operations. Each ST-unit represents one operation of job-shop scheduling problem with its activation representing the starting time of the operation. The second represents whether the sequence constraints are violated. The third represents whether the resource constraints are violated.

The net input of an ST-unit (e.g., ST_i) is calculated by

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

where the net input of unit ST_i is summed from three parts. The first part comes from the weighted activations of SC-units connected with ST_i, which implements feedback adjustments because of sequence violations. The second part comes from the weighted activations of RC-units connected with ST_i, implementing feedback adjustments because of resource violations. The third part comes from the previous activation, with weight being $+1$, of unit ST_i itself.

The activation function of ST-units is deterministic linear-segmented function of type B (as shown in Fig. 2(b)) and is defined as follows:

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

where r_i and d_i are the release date and due date of job i to which the operation, corresponding to unit ST_i, belongs. P_{ST_i} is the processing time of the operation relevant to unit ST_i. This activation function implements the release date and due date constraints described by Eq. (3).

SC-units receive the incoming weighted activations from the connected ST-units, representing operations of the same job. The RC-units receive the incoming weighted activations from the connected ST-units, representing operations to be processed on the same machine. The net input of an SC-unit or RC-unit has the same definition form as follows:

$$N_{C_i}(t) = \sum_j (W_{ij} \times A_{ST_j}(t)) + B_{C_i}, \tag{7}$$

where C_i means SC_i or RC_i, and B_{C_i} is the bias of the neural unit SC_i or neural unit RC_i. The bias B_{C_i} is added to the incoming weighted activations of the connected ST-units ST_j's and equals the processing time of a relative operation, described in Eq. (7).

The activation function of an SC or RC-unit is a deterministic linear-segment function of type A (as illustrated in Fig. 2(a)), defined as follows:

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

The activation of an SC-unit or RC-unit being greater than zero means that the corresponding sequence constraint or resource constraint is violated. Hence there are feedback adjustments from this SC-unit or RC-unit to connected ST-units through adaptive weighted connections.

3.2. Connections of adaptive weights and biases

Generally for neural networks performing constraint satisfaction, the determination of connection weights between the neural units is executed by the designer of the neural network and the weights are set according to the constraint satisfaction problem in advance before the network begins to work. In CSANN, the connection weights and biases are adaptive in accordance with the actual activations of ST-units while the network is running, together with the sequence and resource constraints of the specific problem.

All units of CSANN are connected according to the two kinds of sequence and resource constraints of the specific job-shop scheduling problem, resulting in two blocks: SC-block (sequence constraints block) and RC-block (resource constraints block). Each unit of SC-block contains two ST-units, responding to two operations of a job, and one SC-unit, representing whether the sequence constraint between these two operations is satisfied (see, e.g., Fig. 3). Each unit of RC-block contains two ST-units, responding to two operations sharing the same machine, and one RC-unit, representing whether the resource constraint between these two operations is satisfied (see, e.g., Fig. 4).

Fig. 3 presents an example of SC-block unit, representing the constraint equation $T_{ilq} - T_{ikp} \geqslant P_{ikp}$, denoted by 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 starting times T_{ikp} and T_{ilq} of O_{ikp} and

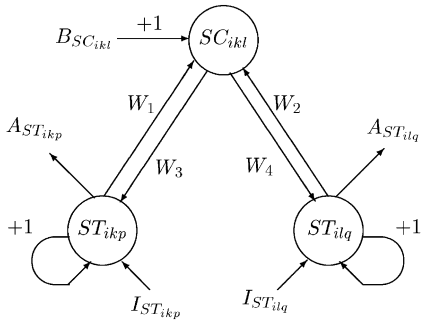

Fig. 3. A SC-block unit SCB_{ikl}.

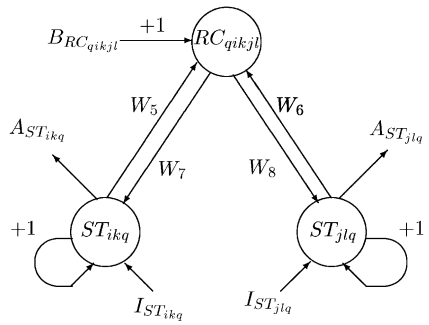

Fig. 4 . A RC-block unit RCB_{qikjl}.

O_{ilq}. The SC-unit SC_{ikl} represents whether the sequence constraint of Eq. (1) between O_{ikp} and O_{ilq} is violated, with $B_{SC_{ikl}}$ being its bias. The weights and bias are valued as follows:

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

where W is the positive feedback adjustment parameter (the same with subsequent equations where W appears).

At time t during the processing of CSANN, when the sequence constraint between O_{ikp} and O_{ilq} is satisfied, the activation $A_{SC_{ikl}}(t)$ of SC_{ikl} equals zero. If the constraint is violated, the activation of SC_{ikl} becomes greater than zero and can be calculated by

$$A_{SC_{ikl}}(t) = -N_{SC_{ikl}}(t) = A_{ST_{ikp}}(t) + P_{ikp} - A_{ST_{ilq}}(t) = T_{ikp}(t) + P_{ikp} - T_{ilq}(t) \tag{10}$$

and $A_{SC_{ikl}}(t)$ should be applied as a corrective signal for ST_{ikp} and ST_{ilq}. The feedback adjustments from SC_{ikl} to ST_{ikp} and ST_{ilq} are shown as follows:

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

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

From the above equations we can see the feedback adjustments from unit SC_{ikl} puts back the starting time T_{ikp} of operation O_{ikp} in time axis, while putting forward T_{ilq} of O_{ilq}. Thus, the sequence violation between O_{ikp} and O_{ilq} can be removed.

Fig. 4 presents an example of RC-block unit, denoted by RCB_{qikjl}, which embodies the resource constraint Eq. (2), representing the resource constraint between O_{ikq} and O_{jlq} on machine q. At time t during the processing of network, the weights and bias are adaptively valued as the following two cases show.

Case 1: If $A_{ST_{ikq}}(t) \leqslant A_{ST_{jlq}}(t)$, that is, $T_{ikq}(t) \leqslant T_{jlq}(t)$, Eq. (13) holds

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

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 the feedback adjustments from RC_{qikjl} to ST_{ikq} and ST_{jlq} are calculated by

$$A_{RC_{qikjl}}(t) = A_{ST_{ikq}}(t) + P_{ikq} - A_{ST_{jlq}}(t) = T_{ikq}(t) + P_{ikq} - T_{jlq}(t), \tag{14}$$

$$A_{ST_{ikq}}(t+1) = T_{ikq}(t+1) = A_{ST_{ikq}}(t) + W_7 \times A_{RC_{qikjl}}(t) = T_{ikq}(t) - W \times A_{RC_{qikjl}}(t), \tag{15}$$

$$A_{ST_{jlq}}(t+1) = T_{jlq}(t+1) = A_{ST_{jlq}}(t) + W_8 \times A_{RC_{qikjl}}(t) = T_{jlq}(t) + W \times A_{RC_{qikjl}}(t). \tag{16}$$

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

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

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

$$A_{RC_{qikjl}}(t) = A_{ST_{jlq}}(t) + P_{jlq} - A_{ST_{ikq}}(t) = T_{jlq}(t) + P_{jlq} - T_{ikq}(t), \tag{18}$$

$$A_{ST_{ikq}}(t+1) = T_{ikq}(t+1) = A_{ST_{ikq}}(t) + W_7 \times A_{RC_{qikjl}}(t) = T_{ikq}(t) + W \times A_{RC_{qikjl}}(t), \tag{19}$$

$$A_{ST_{jlq}}(t+1) = T_{jlq}(t+1) = A_{ST_{jlq}}(t) + W_8 \times A_{RC_{qikjl}}(t) = T_{jlq}(t) - W \times A_{RC_{qikjl}}(t). \tag{20}$$

3.3. Architecture and running mechanisms of CSANN

The architecture of CSANN consists of two layers. The bottom layer consists of only ST-units, corresponding to the starting times of all operations. The top layer contains SC-units and RC-units, which represent sequence and resource constraints respectively and provide feedback information to adjust ST-units in order to satisfy sequence and resource constraints through SC-block and RC-block respectively.

For an $n/m/J/C_{max}$ problem, where $n_i = m$ for all $i \in N$ and each job passes through all machines in a sequencing order there are mn ST-units representing mn number of operations, $n(m-1)$ SC-units representing $n(m-1)$ sequence constraints described by Eq. (1), $mn(n-1)$ RC-units representing $mn(n-1)$ resource constraints described by Eq. (2). There are a total number of $n(mn+m-1)$ units of the whole network.

To a specific job-shop scheduling problem, CSANN can be built up as follows: first calculate the number of ST-units according to the specific problem, which equals $\sum_{i=1}^{n} n_i$, then build up the two sets of P_i and R_q according to the actual sequence and resource constraints, finally form the SC-block and RC-block, resulting in the problem-specific neural network.

There are three mechanisms of running CSANN (see e.g., Yang and Wang [24]). The first one is an asynchronous processing mode which calculates the activation of units in a fixed order. The second one is an asynchronous processing mode which calculates the activation of units in a random order. The third mechanism is a synchronous parallel processing mode. In this paper the first mechanism is used, under which from one given initial solution CSANN has to converge only to a determined solution.

4. Description of heuristics and hybrid approach

This section first gives out the descriptions of two heuristics, which are used to improve the performance of CSANN for job-shop scheduling problems. One is used to accelerate the solving process of CSANN and guarantee feasible solutions, the other is used to obtain the local optimal solution from feasible solution solved by CSANN with determined orders of operations. Secondly the hybrid approach for job-shop scheduling problems is presented in this section.

4.1. Heuristics

Heuristics 1: Exchange the orders of two adjacent operations. This heuristics has two aspects of function: to accelerate the solving process and to guarantee feasible solution. The former is for two adjacent operations coming from the same job, while the latter is for two adjacent operations sharing the same machine.

On the one hand, assume $[O_{ikp}, O_{ilq}] \in S_i$. In order to accelerate the solving speed of CSANN, at time t during its processing, if $A_{ST_{ikp}}(t) \geqslant A_{ST_{ilq}}(t)$ (i.e., $T_{ikp}(t) \geqslant T_{ilq}(t)$), exchange the orders of O_{ikp} and O_{ilq} by exchanging their starting times as follows:

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

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

In fact, Eqs. (21) and (22) are a more direct method of removing sequence violation than that of the feedback adjustment of CSANN. Thus the adjustment time from removing sequence violations may be shortened and the solving process of CSANN for feasible solution is accelerated.

On the other, hand, during the processing of CSANN there may appear the phenomenon of "dead lock" which can result in no feasible solution. In order to remove "dead lock", we use the following heuristic: exchange the orders of two near operations sharing the same machine by exchanging their starting times.

Assuming O_{ikq} and $O_{ijq} \in R_q$, during the processing of CSANN, if $T_{qikjl}(t) \geqslant T$, the following equations begin to work:

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

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

where the parameter T is a prescribed positive integer, variable $T_{qikjl}(t)$ is the summed continuous change times between the starting times of operation pairs O_{ikq} and O_{jlq} (sharing machine q) because of their resource violation. That is, at time t, the starting times of O_{ikq} and O_{jlq} have already continuously changed $T_{qikjl}(t)$ times because of their resource violation on machine q, and the changing effects are the same (e.g., always putting T_{ikq} forwards and T_{jlq} backwards). When $T_{qikjl}(t)$ reaches T, Eqs. (23) and (24) begin to work.

The above heuristic can be used together with CSANN to guarantee the feasible solution. The phenomenon of "dead lock" results from the conflicts of feedback adjustments while removing sequence and resource constraint violations. For example, assuming $[O_{ikp}, O_{ilq}] \in S_i$ and $[O_{ilq}, O_{jmq}] \in R_q$. During the processing of CSANN, the SC-unit SC_{ikl} may put forward the starting time T_{ilq} of operation O_{ilq} along the positive direction of time axis through feedback adjustment because of sequence violation, while the RC-unit RC_{qiljm} may put back T_{ilq} through feedback adjustment because of resource violation. Thus there may exist conflicts resulting from this two kinds of adjustments which result in "dead lock". "Dead lock" results in the nonconvergence of CSANN to its stable station, which corresponds to the feasible solution of the specific job-shop scheduling problem. By using the proposed heuristic, when the phenomenon of "dead lock" happens and T_{ilq} has been continuously put back T times because of resource violation between O_{ilq} and O_{jmq}, that is, at time t $T_{qikjl}(t)$ reaches T, the starting time T_{ilq} of O_{ilq} may be exchanged with T_{jmq} of O_{jmq}. Thus "dead lock" can be effectively avoided and the feasible solution is guaranteed.

Heuristics 2: Obtain a non-delay schedule from the feasible solution solved by CSANN. A schedule is *non-delay* if no machine lies idle when there is at least one job waiting to be operated on that machine [9]. A non-delay schedule is a local optimal schedule with orders of operations to be operated on each machine already determined. A schedule is *active* if no operation can be started earlier without delaying another operation or violating the sequence constraints. It is evident that an optimal schedule is an active one. The set of non-delay schedules is a proper subset of the active set. So when the obtained non-delay schedule falls in the active schedule optimal subset, the optimal schedule is achieved, and this is the implicit theory base of heuristics 2. CSANN can obtain feasible solutions quickly, but there may be many idle times for each machine with operations available to be operated. Obvious, these idle times heavily degrade the quality of feasible schedule and should be compacted away in order to shorten makespan or improve the quality of schedule. The detailed heuristics is as follows.

Assuming a feasible solution $\{T_{ikp}, i \in N, k \in \{1, \ldots, n_i\}, p \in M\}$ have been obtained by CSANN. Sort them in non-decreasing order. Then from the minimal to the maximal, each T_{ikp} is adjusted as follows:

$$T'_{ikp} = \begin{cases} T_{ljp} + P_{ljp}, & T_{ljp} + P_{ljp} \geqslant T_{i(k-1)q} + P_{i(k-1)q}, \\ T_{i(k-1)q} + P_{i(k-1)q}, & T_{ljp} + P_{ljp} < T_{i(k-1)q} + P_{i(k-1)q}, \end{cases} \tag{25}$$

where T'_{ikp} is the starting time of O_{ikp} in the obtained nondelay schedule after the heuristics is run. $O_{i(k-1)q}$ is the precedence operation of O_{ikp} from the same job i, and O_{ljp} is the precedence operation of O_{ikp} sharing the same machine p. Eq. (25) means to shorten each starting time T_{ikp} to the completion time of $O_{i(k-1)q}$ or the completion time of O_{ljp}, depending on whichever is smaller. The adjustments of all starting times are dynamic, i.e., the starting time of the previous operation that has been adjusted works while adjusting the latter operations. For example, supposing that T_{ikp} has been adjusted into T'_{ikp}, when computing $T'_{i(k+1)q}$ of operation $O_{i(k+1)q}$ which is just next to O_{ikp} of the same job i, T'_{ikp} is used in Eq. (25) instead of T_{ikp}. Thus each operation needs to be adjusted only once to obtain a non-delay schedule.

4.2. Hybrid approach for job-shop scheduling

The hybrid approach for job-shop scheduling consists of CSANN and the two proposed heuristics. The solving process of the hybrid approach is iterative. The main steps of the hybrid approach are as follows:

Step 1: Build up CSANN model, set values for parameters T and W, prescribe the maximal runtime restriction MT and the initial expected makespan;

Step 2: Randomly initialize the starting time $T_{ikp}(0)$ for each operation O_{ikp}, and take it as the initial net input $I_{ST_{ikp}}$ of each ST-unit ST_{ikp};

Step 3: Run each SC-unit SC_{ikl} of SC-block, calculate its activation with Eq. (10). $A_{SC_{ikl}}(t) \neq 0$ means the dissatisfaction of sequence constraint, then adjust activations of relative ST-units with Eqs. (11) and (12) or with Eqs. (21) and (22) under the condition of heuristic 1;

Step 4: Run each RC-unit RC_{qikjl} of RC-block, calculate its activation with Eq. (14) or (18). $A_{RC_{qikjl}}(t) \neq 0$ means the dissatisfaction 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) under the condition of heuristic 1;

Step 5: Repeat step 3 and step 4 until all units are in stable states without changes, which means that all the sequence and resource constraints are satisfied and the feasible solution is obtained;

Step 6: Use heuristics 2 to obtain a non-delay schedule solution from the feasible solution obtained in Step 5;

Step 7: If the makespan of the obtained non-delay schedule is shortened, or continuously keeps unchanged less than the prescribed times (e.g., X times) and the run time is less than MT, take the makespan of newly obtained non-delay solution as the new expected makespan and return to step 2; Otherwise, stop the program and output the best solution.

In the solving process of hybrid approach, expected makespan is usually used as the common due date for all jobs. The initial expected makespan is prescribed to be big enough for obtaining feasible solution, maybe greater than the sum of processing times of all operations. The solving

process of hybrid approach is iterative, with the makespan of newly obtained non-delay solution used as the new expected makespan of next iteration. During each iteration, CSANN is used to obtain a feasible solution, which may have a shorter makespan than that of the previous iteration. Thus, the obtained schedule is getting better and better. When the prescribed maximal runtime is achieved, or the obtained makespan is kept continuously the same for X times, the iterating process is stopped.

We take the aforementioned whole iteration process as a "run". In practical application, we can execute a batch of runs and take the best of all obtained best solutions as the final schedule.

5. Simulation study

5.1. Simulation examples

Example 1. We take the benchmark $6/6/J/C_{max}$ problem from Muth and Thompson [27] as the first experimental problem. This example has an optimum (i.e., minimal makespan) of 55.

Example 2. Table 1 presents a $10/10/J/C_{max}$ problem measured from the feasible schedule given in Zhou et al. [22], where (m, t) means that the relevant operation of some job will be processed on machine m with its processing time being t. The sequence constraints of all jobs are the same: in order from operation 1 to operation 10. The makespan of the feasible schedule given in Zhou et al. [22] is 98.

5.2. Simulation results

The simulations are finished on an Intel 586 PC running at 133 MHz under Microsoft Visual C++ 5.0 development environment.

Table 1
Original data of Example 2

Job no.	Operation No.									
	1	2	3	4	5	6	7	8	9	10
1	3,1	1,3	2,5	4,8	6,3	5,7	7,5	8,8	9,8	10,4
2	2,8	3,5	5,10	6,9	7,10	8,4	1,5	4,3	10,5	9,7
3	3,5	4,4	7,8	8,9	2,1	5,8	6,3	10,7	9,10	1,3
4	7,5	8,5	2,5	1,4	3,8	4,10	10,7	9,4	5,7	6,10
5	3,8	7,4	8,5	2,4	5,1	10,1	9,7	6,7	1,8	4,7
6	2,3	4,3	7,8	9,10	10,4	6,1	8,7	1,9	5,7	3,5
7	5,7	6,7	3,7	10,5	9,1	4,10	7,10	8,4	2,3	1,9
8	4,5	9,7	10,10	6,4	3,4	5,8	1,5	2,10	8,4	7,5
9	5,3	10,8	9,4	6,7	4,7	1,5	2,9	3,5	7,10	8,10
10	6,8	2,1	1,5	5,7	8,9	3,3	4,7	7,5	10,9	9,4

Table 2
Simulation results of Example 1 by hybrid approach

Prescribed maximal runtime (s)	Runtime for best solution (s) (ave/min/max)	Iteration times per run (ave/min/max)	Makespan (E) (ave/min/max)	Percentage of obtaining optimal solution (%)
15	8.4/3/14	6/3/12	55.40/55/57	66
30	10.6/3/27	8/3/13	55.25/55/56	75
60	20.5/3/55	9/3/15	55.01/55/56	99

For Example 1, the simulations are finished with the maximal runtime prescribed to be 15, 30 and 60 s, respectively. For each maximal runtime, 100 experiments or runs are carried out. For all experiments, the parameters are valued as follows: $T = 5$, $W = 0.5$ and $X = 5$, and the initial expected makespan is set to be 500, which is much greater than the sum of processing times of all operations, being 197. And for each iteration of all experiments, the initial solution for CSANN is randomly determined with the initial starting times of all operations valued in a randomly uniform distribution between [0,100]. And the expected makespan is used as the common due date for all jobs and the release dates for all jobs are set to zero. Table 2 shows the statistics of simulation results with respect to average, minimum and maximum of runtime for obtaining the last feasible solution or the best solution per run, iteration times per run, makespan of the obtained best solution, and the percentage of obtaining optimal solution for each prescribed maximal runtime respectively.

From Table 2 we can see: with different maximal runtime restriction, the hybrid approach can always quickly obtain good near-optimal or optimal solutions within several iterations. For the given Example 1, when the maximal runtime is prescribed to 15, 30 and 60 s, the hybrid approach obtains good near-optimal or optimal solutions within 6, 8 and 9 iteration times on average, respectively. The percentages of obtaining optimal solutions are 66, 75 and 99%, respectively, all being quite high. In fact, when the maximal runtime is prescribed to be 60 s, only one of the executed 100 experiments obtained a best solution with the makespan being 56, all the other 99 runs resulted in optimal solutions. The average makespans of the obtained best solutions are 55.40, 55.25 and 55.01 respectively, all being very near the optimal value 55. For the three cases, the longest makespan of the obtained best solutions is 57 when MT equals 15 s, which is only a little longer than the optimal value. The solving speed of hybrid approach is very high. The average runtimes of obtaining best solutions are 8.4, 10.6 and 20.5 s, respectively. For all the three cases, the shortest runtime of obtaining best solutions, also optimal solutions, is only 3 s within the three iterations.

Fig. 5 presents the iteration process of a run with MT prescribed to be 60 s. During this run CSANN is used 8 times to obtain the feasible solutions, of which the best solution is also the optimal solution. With the initial expected makespan being 500, the first feasible solution is obtained with makespan being 76. Then 76 is used as the new expected makespan in the second iteration of CSANN, resulting in the second feasible solution with the makespan being 68. And so on, the iteration process continues. During the fifth to seventh iteration, makespans of obtained

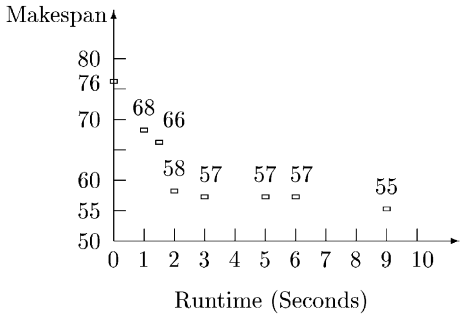

Fig. 5 . The iteration process of a run of Example 1.

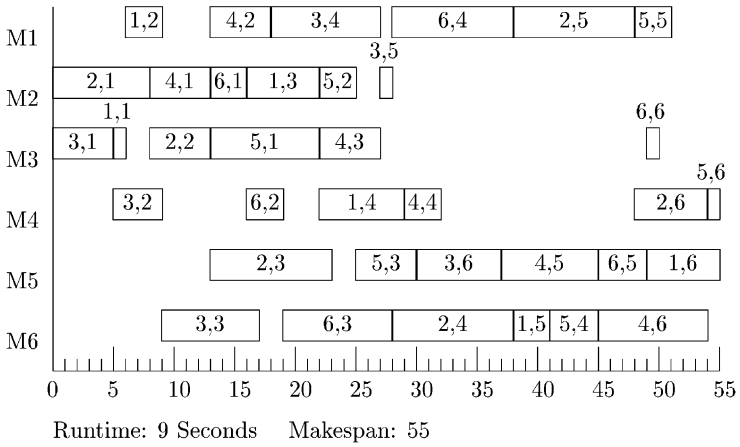

Fig. 6. An optimal schedule of Example 1.

solutions keep the same of 57 for three times, less than $X = 5$ times. In the 8th iteration when the runtime reaches 9 s, the feasible solution with makespan being 55 is obtained, which is the optimal solution of the example problem. For the nineth iteration, 55 is used as the expected makespan and the program stopped when the runtime reaches 60 s before new feasible solution can be obtained.

Fig. 6 shows the relative Gantt chart of the best solution obtained in the run shown in Fig. 5. In Fig. 6, a block means an operation with the length of the block equivalent to its processing time, the number pairs (i, j), inside or above the block, means that the relative operation is the jth operation of job i.

For Example 2, the simulation is finished with the maximal runtime prescribed to be 100 s, with the parameters valued as follows: $T = 5$, $W = 0.5$ and $X = 5$. The initial expected makespan is set to be 1000, which is much greater than the sum of processing times of all operations. Fig. 7 shows a simulation result Gantt chart. From Fig. 7, we can see that the makespan of the obtained best solution is 97, which is better than the schedule result given in Zhou et al. [22].

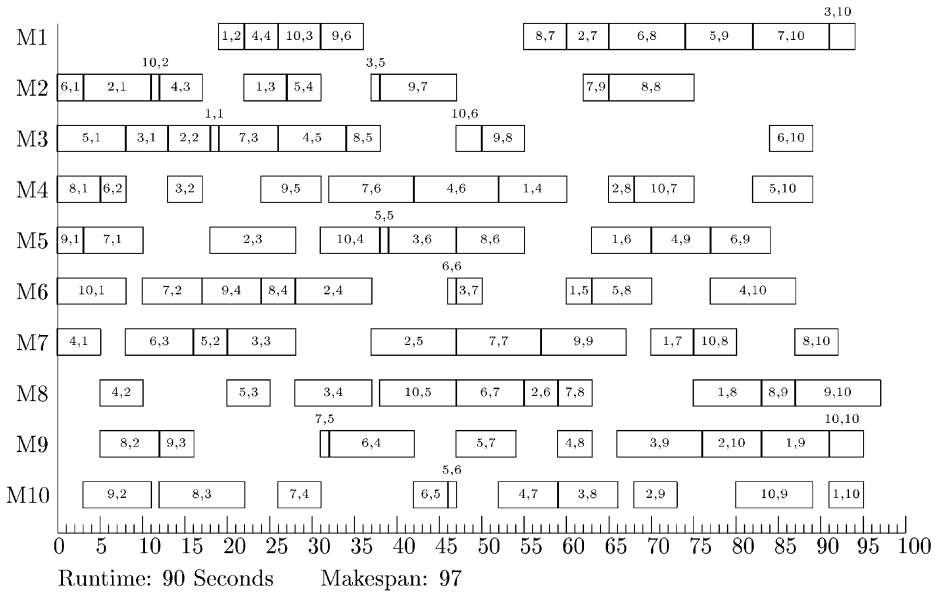

Fig. 7 . A solution of Example 2.

6. Conclusions

In this paper we proposed a new hybrid approach, combining CSANN and two heuristics, for job–shop scheduling. CSANN is used to obtain feasible solutions during the iterations of hybrid approach, while the two heuristics are used to improve CSANN's property and obtain better solutions. Simulations have shown that the proposed hybrid approach for job-shop scheduling has excellent performance with respect to the quality of solutions and the speed of calculation.

While the proposed hybrid approach is used for practical job-shop scheduling problems, we can take the following strategy. Execute the hybrid approach to solve practical job-shop schedule problem from an appropriate small maximal runtime restriction. Then gradually enlarge the value of maximal runtime by an appropriate increment (e.g., 10 s) and run the hybrid approach. If the makespans of the obtained best solutions are kept to be the same continuously for several runs, usually they are the near-optimal or optimal solutions of the problem. Thus, we can stop the program and use them as practical schedules.

Acknowledgements

The helpful comments and suggestions from the anonymous referees are much appreciated by the authors.

References

[1] Baker KR. Introduction to sequence and scheduling. New York: Wiley, 1974.
[2] Conway RW, Maxwell WL, Miller LW. Theory of scheduling. Reading, MA: Addison-Wesley, 1967.
[3] Stephen C. Graves. A review of production scheduling. Operations Research 1981;29(24):646–75.
[4] Dubois D, Fargier H, Prade H. Fuzzy constraints in job–shop scheduling. Journal of Intelligent Manufacturing 1995;6:215–34.
[5] Montazeri H. Analysis of scheduling rules for an FMS. International Journal of Production Research 1990;28:785–802.
[6] Erschler JF, Roubellat JP, Vernhes. Finding some essential characteristics of the feasible solutions for a scheduling problem. Operations Research 1976;24:774–83.
[7] Bellman RE, Esogbue AO, Nabeshima I. Mathematical aspects of scheduling and applications. Oxford: Pergamon Press, 1982.
[8] Garey MR, Johnson DS, Sethi R. The complexity of flowshop and job–shop scheduling. Math. Opl. Res. 1976;1:117–29.
[9] French S. Sequencing and scheduling: an introduction to the mathematics of the job–shop. New York: Wiley, 1982.
[10] Hentenryck PV. Constraint satisfaction and logic programming. Cambridge, MA: The MIT Press, 1989.
[11] Fox MS, Zweben M. Knowledge-based scheduling. San Manteo, CA: Morgan and Kaufman, 1993.
[12] Hopfield JJ, Tank DW. 'Neural' computation of decisions in optimization problems. Biological Cybernetics 1985;52:141–52.
[13] Tank DW, Hopfield JJ. Simple neural optimization networks: an A/D converter, single decision circuit and a linear programming circuit. IEEE Transactions on Circuits and Systems 1986;33:533–41.
[14] Park JH, Jeong H. Solving the TSP using an effective Hopfield network. Proceedings of IEEE International Conference on Neural Networks, 1990.
[15] Looi Chee-Kit. Nueral network methods in combinatorial optimization. Computers and Operations Research 1992;19:191–208.
[16] Kenndey MP, Chua LO. Neural networks for nonlinear programming. IEEE Transactions on Circuits and Systems 1988;35:554–62.
[17] DARPA. DARPA nueral network study. AECEA International Press, 1988.
[18] Foo SY, Takefuji Y. Neural networks for solving job–shop scheduling: Part 1. problem representation. Proceedings of IEEE IJCNN, San Diego, vol. II. 1988. p. 275–82.
[19] Foo SY, Takefuji Y. Stochastic neural networks for solving job–shop scheduling: Part 2. architecture and simulations. Proceedings of IEEE IJCNN, San Diego, vol. II. 1988. p. 283–90.
[20] Foo SY, Takefuji Y. Integer linear programming neural networks for job–shop scheduling. Proceedings of IEEE IJCNN, San Diego, vol. II. 1988. p. 341–8.
[21] Foo SY, Takefuji Y, Szu H. Job-shop scheduling based on modified Tank-Hopfield linear programming networks. Engineering Application and Artificial Intelligent 1994;7(3):321–7.
[22] Zhou DN, Charkassky V, Baldwin TR, Hong DW. Scaling neural network for job–shop scheduling. Proceedings of IEEE International Joint Conference on Neural Networks, New York, vol. 3. 1989. p. 889–94.
[23] Willems TM, Brandts LEMW. Implementing heuristics as an optimization criterion in neural networks for job–shop scheduling. Journal of Intelligent Manufacturing 1995;6:377–87.
[24] Yang S, Wang D. Constraint satisfaction adaptive neural network and heuristics combined approaches for generalized job–shop scheduling. IEEE Transactions on Neural Networks 2000;11.
[25] Nielsen RH. Neurocomputing. Reading, MA: Addison-Wesley, 1989.
[26] Lippmann RP. An introduction to computing with neural nets. IEEE ASSP Magazine 1987: 4–22.
[27] Muth JF, Thompson GL. Industrial scheduling. Englewood Cliffs, NJ: Prentice-Hall, 1963.

Shengxiang Yang is now a Post-doctoral Research Associate at Department of Computer Science, King's College London, University of London. He received his Ph.D. in Systems Engineering at Northeastern University, Shenyang, P. R. China. His current research interests include artificial neural networks, production scheduling, combinational optimization, genetic algorithms, network flow theory and algorithms.

Dingwei Wang is currently Professor at Department of Systems Engineering, Northeastern University, Shenyang, P. R. China. He received the Ph.D. degree in Control Theory and Application at Northeastern University, Shenyang, P. R. China. His research interests include MRP-II, JIT manufacturing systems, production planning and scheduling, fuzzy optimization and genetic algorithms.