Neural network and genetic algorithm-based hybrid approach to expanded job-shop scheduling

Haibin Yu*, Wei Liang

Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang 110015, People’s Republic of China

Abstract

The expanded job-shop scheduling problem (EJSSP) is a practical production scheduling problem with processing constraints that are more restrictive and a scheduling objective that is more general than those of the standard job-shop scheduling problem (JSSP). A hybrid approach involving neural networks and genetic algorithm (GA) is presented to solve the problem in this paper. The GA is used for optimization of sequence and a neural network (NN) is used for optimization of operation start times with a fixed sequence.

After detailed analysis of an expanded job shop, new types of neurons are defined to construct a constraint neural network (CNN). The neurons can represent processing restrictions and resolve constraint conflicts. CNN with a gradient search algorithm, gradient CNN in short, is applied to the optimization of operation start times with a fixed processing sequence. It is shown that CNN is a general framework representing scheduling problems and gradient CNN can work in parallel for optimization of operation start times of the expanded job shop.

Combining gradient CNN with a GA for sequence optimization, a hybrid approach is put forward. The approach has been tested by a large number of simulation cases and practical applications. It has been shown that the hybrid approach is powerful for complex EJSSP. © 2001 Elsevier Science Ltd. All rights reserved.

Keywords: Job-shop scheduling; Neural network; Genetic algorithm; Gradient search

1. Introduction

Scheduling is an old discipline that originated and developed almost in step with operations research starting in the 1950s. Johnson’s algorithm for $n/2/F/C_{\text{max}}$ and parts of the special $n/3/F/C_{\text{max}}$ problems launched the research of scheduling theory in the 1950s. A series of representative scheduling problems were studied with pure integral programming, dynamic programming and branch and bound approach in the 1960s. Some heuristic algorithms were proposed for computing these complex problems in the last period of the 1960s. The intrinsic complexity of some scheduling problems was understood and many of them were proved to be intractable in the 1970s. The effective heuristics and their effectiveness have

* Corresponding author.

0360-8352/01/$ - see front matter © 2001 Elsevier Science Ltd. All rights reserved.

PII: S0360-8352(01)00010-9
been intensively conducted since then. It can be said that classical scheduling theory reached maturity as a branch of applied mathematics since the end of the 1970s (Bark, 1974).

On the other hand, scheduling is also a young discipline. With the development of production technology and management, many new types of practical scheduling problem are emerging constantly and need to be solved. Working emphases have been shifted to solving practical problems. New techniques emerging from the field of AI have been introduced to this discipline since the 1980s.

The job-shop scheduling problem (JSSP) is a simplified version of real production scheduling problems. In this paper, a more realistic problem, namely the expanded job-shop scheduling problem (EJSSP), is studied. EJSSP considers various constraints such as job delivery due dates and resource available times in addition to those considered by the standard JSSP.

In general, the research approach to production scheduling falls into the following three categories: heuristic priority rules, combinatorial optimization (Dubois, Fargier & Prade, 1995), and knowledge-based system with intelligence (Shaw, Park & Raman, 1992).

Each method has its own drawbacks. Although priority rules, especially some heuristic priority rules have been applied extensively because of their simplicity and ease of implementation, the resulting scheduling results cannot be predicted beforehand. There are no systematic methods of designing heuristics for a specific scheduling task. Analytic optimization is a desired method because it is efficient. Unfortunately, there is no generally analytic optimization algorithm for intrinsically complex NP-hard combinatorial optimization problems. In the intelligent scheduling information system (ISIS), a knowledge-based scheduling system first developed by Fox (1994), the constraint propagation technique is applied to solve a practical scheduling problem. There is a long way to go before a comprehensive theory and system for scheduling is fully developed, even though ISIS arouses the enthusiasm of those studying practical production scheduling problems.

Recently, much attention has been paid to applying neural networks (NNs) and genetic algorithms (GAs) to production scheduling problems. There are three existing ways of scheduling tasks using NNs. One way is to take advantage of the parallel computing ability of NNs to solve scheduling optimization problems (Jain & Meeran, 1998). This method’s drawback lies in its non-availability for universal scheduling problems, because there are no systematic means of constructing a proper energy function for them. The second way should be classified within the intelligent scheduling category, in which an NN is used to acquire scheduling knowledge by dint of its learning ability (Wang, 1995). The intelligent scheduling with NNs has given elementary results, but it needs a lot of supporting sample data and is not yet applicable to practical problems. The last way is to describe production constraints and encode scheduling rules in the NN. Intrinsically, this is one kind of heuristic scheduling. Only feasible schedules can result from the network’s operation (Foo & Takefuji, 1988; Chang & Jeng, 1995; Willems & Rooda, 1994). The major problems in this approach are (1) lack of ability to acquire optimal solutions; and (2) suitability only for specific simple scheduling problems.

In recent years, an interest in applying intelligent computing methods to job-shop scheduling has grown rapidly. Steinhofel, Albrecht and Wong (1999) present two simulated annealing-based algorithms for the classical JSSP where the objective is to minimize the makespan. To overcome the problem that convergence of simulated annealing does not hold in the application to job-shop scheduling, Kolonko (1999) proposed a new approach that uses a small population of SA runs in the GA framework. Moreover, an SA algorithm using three perturbation schemes, viz. pairwise exchange, insertion, and random insertion, for job-shop scheduling is put forward by Ponnambalam, Jawahar and Aravindan (1999). Meanwhile, GA is also an efficient method for JSSP. Cheng, Gen and Tsujimura (1999a,b) summarize
the representations and search strategies of GA only for standard JSSP. A hybrid approach combining
GA with multi-criteria decision-making (MCDM) and constraint logic programming (CLP) is proposed
by Mesghouni, Pesin, Trentesaux, Hammadi, Tahon and Borne (1999), where GA, assisted by MCDM
and CLP, plays the role of optimization. Additionally, Sakawa and Mori (1999) put forward a GA
algorithm for JSSP with fuzzy processing time and fuzzy due date. The approach based on genetized-
knowledge GA (gkGA) of Ombuki, Nakamura and Onaga (1999) encodes knowledge of heuristics as
genes of GA. Moreover, Ghedjati (1999) uses GA to solve the JSSP with unrelated parallel machines and
precedence constraints, which is another kind of EJSSP and is different from the problem of this paper.
In addition, Candido, Khator and Barcia (1998) present an algorithm based on GA for JSSP with multi-
constraints considering sub-assembly levels, where the problem is a specific example of our considered
problem. EJSSP of the paper is a special scheduling problem with no known solution techniques
involving GA and NN.

Job-shop scheduling can be decomposed into a constraint satisfactory part and an optimization part for
a specified scheduling objective. For this, an NN and GA-based hybrid scheduling approach is proposed
in this paper. First, several specific types of neuron are designed to describe these processing constraints,
detecting whether constraints are satisfied and resolving the conflicts by their feedback adjustments.
Constructed with these neurons, the constraint neural network (CNN) can generate a feasible solution for
the EJSSP. CNN here corresponds to the constraint satisfactory part. A gradient search algorithm can be
applied to guide CNN operations if an optimal solution needs to be found at a fixed sequence. For
sequence optimization, a GA is employed. Through many simulation experiments and practical applica-
tions, it is shown that the approach can be used to model real production scheduling problems and to
efficiently find an optimal solution. The hybrid approach is an ideal combination of the constraint
analysis initiated by Erschler, Roubellat and Vemhes (1976) and the optimization scheduling method
(Zhang & Yan, 1995).

This paper is organized as follows. In Section 2, an expanded job shop is described. CNN, a frame-
work for representing various constraints and resolving conflicts, is introduced in Section 3. A gradient
search algorithm is introduced to the CNN in Section 4. The hybrid approach is presented based on the
gradient CNN and GA in Section 5. A case study and our conclusions are presented in Sections 6 and 7,
respectively.

2. Expanded job-shop scheduling problem

EJSSP is a deterministic and static scheduling problem. There are $m$ distinct machines to process $n$
jobs that have their specific processing routines. Each job’s operation has its precedence and takes up a
deterministic time period at a specific machine. At one time, there is only one operation at a machine and
the job does not leave this machine until the operation is completed. The operation starting time of each
job must be within predefined regions, which are subject to the available time and due dates of jobs.
Enabling conditions of an operation are prescribed by technological planning including job and resource
requirement such as machines and fix tools, as well as cutting tools. It can be seen easily that EJSSP is
significantly more general than the standard JSSP.

The objectives considered are:

- to minimize the end time of the last completed job, or
• to minimize the total penalty for tardy and early jobs.

A complete scheduling solution should determine the processing sequence and all operation start times. For modeling convenience, it is helpful to clarify some important concepts here. The processing operation of any job needs some resources specified by technological planning, which are termed as operation enable conditions. Activity and operation are used interchangeably in this paper. Some of the processing activities of a job may have precedence sequential restrictions, decided by technological planning.

2.1. Notations for EJSSP

The symbols for modeling scheduling problems are as follows:

\[ n \] — number of jobs;
\[ n_i \] — number of operations of job \( i \);
\[ m \] — type number of various resources;
\[ r_s \] — number of resources of type \( s, s \in \{1, 2, \ldots, m\} \);
\[ R_i \] — set of pairs of operations \( \{k, l\} \) belonging to job \( i \), where operation \( k \) precedes operation \( l \);
\[ Q_i \] — set of pairs of operations \( \{k, l\} \) belonging to job \( i \), for any operation \( k \) and operation \( l \);
\[ N_q \] — set of operations requiring resource \( q, q \in \{1, 2, \ldots, r\} \);
\[ H \] — large enough positive number;
\[ t_{il} \] — processing time of operation \( l \) of job \( i, l \in \{l, \ldots, n_i\} \);
\[ x_{ik} \] — starting time of operation \( k \) of job \( i, k \in \{1, \ldots, n_i\} \);
\[ x_{il} \] — starting time of the first (or free) operation of job \( i \);
\[ x_{i} \] — completion time of the last (or free) operation of job \( i \);
\[ a_i \] — availability time of job \( i \);
\[ d_i \] — delivery due date of job \( i \);
\( [i, k] \) — the \( k \)th operation of job \( i \), also called operation \( k \) in short if no confusion is caused;

\[
y_{kl} = \begin{cases} 1 & \text{if operation} k \text{ precedes operation} l \\ 0 & \text{otherwise} \end{cases}
\]
where \( \{k, l\} \in Q_i, i = 1, 2, \ldots, n \);

\[
z_{ij} = \begin{cases} 1 & \text{if operation} i \text{ precedes operation} j \\ 0 & \text{otherwise} \end{cases}
\]
where \( \{i, j\} \in N_q, q = 1, 2, \ldots, r_s, s = 1, 2, \ldots, m \).

Note: \( i \in \{1, \ldots, n\}; s \in \{1, \ldots, m\} \); free operation means operation without precedence restrictions from technological planning.
2.2. Modeling and analysis of EJSSP

A feasible solution means that the scheduling satisfies all constraint conditions. There are two types of major constraints for any operation, precedence constraint and resource constraint.

**Precedence constraint.** Precedence constraint means that some jobs must be processed at different machines in fixed precedence sequence defined by technological planning. Concretely, the \(l\)th operation of job \(i\) must be before the \(k\)th operation of the same job, if \(\{k, l\}R_i\), i.e.

\[
x_{il} - x_{ik} - t_{ik} \geq 0 \quad \text{for all } i, \text{ if } \{k, l\} \in R_i
\]

(1)

**Resource constraint.** Resource constraint means that any resource can only provide service for one operation at a time; for example, resource \(q\) can only select one job to serve among jobs waiting to be processed in the queue at any time.

\[
x_{jl} - x_{ik} - t_{ik} + H(1 - z_{kl}) \geq 0 \quad \text{for all } q, \text{ if } \{k, l\} \in N_q
\]

(2a)

\[
x_{ik} - x_{jl} - t_{jl} + Hz_{kl} \geq 0 \quad \text{for all } q, \text{ if } \{k, l\} \in N_q
\]

(2b)

where \(z_{kl} = 0\) or 1. Eqs. (2a) and (2b) show that the operations \(k\) and \(l\) requiring the same resource \(q\) should not be processed at the same time.

**Job (hidden) constraint.** Although there may be no precedence constraint among some operations of a job, the constraint that the operations \(l\) and \(k\) could not be processed at the same time still exists because these two operations were done at the same job, i.e.

\[
x_{il} - x_{ik} - t_{ik} + H(1 - y_{kl}) \geq 0 \quad \text{for all } i, \text{ if } \{k, l\} \in Q_i
\]

(3a)

\[
x_{ik} - x_{il} - t_{il} + Hy_{kl} \geq 0 \quad \text{for all } i, \text{ if } \{k, l\} \in Q_i
\]

(3b)

where \(y_{kl} = 0\) or 1.

It is indicated that \(\{k, l\}R_i\), an operation of job \(i\), must satisfy Eq. (1) and \(\{k, l\}Q_i\), another operation of job \(i\), must satisfy either Eq. (3a) or Eq. (3b). It should be noted that only one of Eqs. (3a) and (3b) is active and the other will be satisfied naturally with \(y_{kl} = 0\) or 1.

**Starting and completion time constraint.** In practice, the starting time and the completion time of a job are restricted by the job available time and the due date of delivery. Mathematically, it can be depicted by Eqs. (4) and (5).

\[
x_{il} \geq a_i \quad \text{for all } i
\]

(4)

\[
x_{il} \leq d_i - t_{il} \quad \text{for all } i
\]

(5)

The scheduling objective is another important issue besides scheduling constraints. Scheduling problems with the following two objectives are tested in this paper although the hybrid scheduling approach is not restricted to a specific objective.
• Minimizing the completion time of the last completed job,

\[
\text{Min } Z = \max_i (x_i + t_i)
\]  

(6)

• Minimizing the total penalty for early and tardy jobs,

\[
\text{Min } Z = \sum_{i=1}^{n} [h_i \times \max(0, d_i - x_i) + w_i \times \max(0, x_i - d_i)]
\]  

(7)

where \( h_i \) and \( w_i \) are the early and tardy penalty weights for job \( i \). It should be noted that if Eq. (7) were served as scheduling objective, constraint equation (5) could be omitted, i.e. \( d_i \) should be set as a positive infinite number.

3. CNN model

The CNN of modeling Eqs. (1)–(5) is presented in this section. The CNN is introduced at three levels with neurons, neural blocks and NNs.

3.1. Neurons

Neurons are basic elements of NNs. A common neural cell or neuron is defined by the linearly weighted summation of its input signals, and serially connected non-linear activity function \( F(T_i) \).

\[
T_i = \sum_{j=1}^{n} (W_{ij}X_j)
\]  

(8)

\[
Y_i = F(T_i)
\]  

(9)

where \( W_{ij} \) is the connection weight of the \( j \)th input signal \( X_j \) and the \( i \)th neuron, \( T_i \) is the weighted summation of the \( i \)th neuron, \( F(T_i) \) is the activity function and \( Y_i \) is the output of the \( i \)th neuron.

Based on the common neuron model, \( S \) type neurons and \( C \) type neurons as well as \( I \) type neurons are further defined. They can describe the constraints in Eqs. (1)–(5).

\( S \) type neuron. It is designed to restrict the start time of an operation. In neuron \( S_i \) of type \( S \),

\[
T_{S_i}(t + 1) = \sum_j (W_{ij}Y_{C_j}(t)) + Y_{S_i}
\]  

(10)

\[
Y_{S_i}(t + 1) = \begin{cases} 
\text{LO}_i & T_{S_i}(t + 1) \leq \text{LO}_i \\
T_{S_i}(t + 1) & \text{LO}_i \leq T_{S_i}(t + 1) \leq \text{HI}_i \\
\text{HI}_i & T_{S_i}(t + 1) \geq \text{HI}_i 
\end{cases}
\]  

(11)

where \( Y_{S_i}(t) \) is the output of neuron \( S_i \) at time \( t \). \( \text{LO}_i \) and \( \text{HI}_i \) are used to depict the operation’s earliest starting time and latest completion time.
LO$_i$ is equal to $a_i$, the available time of job $i$, when neuron $S_i$ corresponds to the first operation of job $i$; otherwise LO$_i$ is set to 0. HI$_i$ is equal to $d_i$, the latest starting time of job $i$’s last operations when the neuron $S_i$ corresponds to the last operation of job $i$; otherwise HI$_i$ is set to $+\infty$. Weight $W_{ij}$ represents the connection from neuron $C_j$ of type C to neuron $S_i$ of type S. The value of $W_{ij}$ will be set in the sequence constraint (SC) block, resource constraint (RC) block and job constraint (JC) block later.

It is worth noting that there is a feedback coming from the neuron of S type itself. The feedback represents iterative adjustment of the starting time of the corresponding operation.

**C type neuron.** It is used to depict the precedence restriction of the operation. In neuron $C_i$ of type C,

$$T_{C_i}(t) = \sum_j (W_{ij}Y_{S_j}(t)) + \sum_j (W_{ij}Y_{I_j}(t)) + B_i$$  

(12)

$$Y_{C_i}(t) = \begin{cases} 
0 & T_{C_i}(t) \geq 0 \\
-T_{C_i}(t) & T_{C_i}(t) \leq 0 
\end{cases}$$  

(13)

where $B_i$ is the bias setting of neuron $C_i$, and $W_{ij}$ is the weight of neuron $C_i$. Neuron $C_i$ collects information from the S type and I type neurons through $W_{ij}$, whose values will be set in the SC, RC and JC blocks.

**I type neuron.** It is used to represent $Y_{kl}$, or $Z_{kl}$. In neuron $I_i$ of type I,

$$T_{I_i}(t) = \sum_j (W_{ij}Y_{S_j}(t))$$  

(14)

$$Y_{I_i}(t) = \begin{cases} 
1 & T_{I_i}(t) \leq 0 \\
0 & T_{I_i}(t) > 0 
\end{cases}$$  

(15)

where $W_{ij}$ is the weight of neuron $I_i$. Neuron $I_i$ collects information from the S type neuron through $W_{ij}$, whose value will be set in the RC block and the JC block.

### 3.2. Links of neurons

Links among neurons are through their weights. They represent the scheduling restrictions. They also reflect the adaptation or adjustment to resolve constraint conflicts through proper feedback links, when restrictions are not met.

**SC block for processing operations.** The SC block is constructed with three neurons, two of S type and one of C type, shown in Fig. 1(a). The C type neuron checks the precedence relation of the job’s operations. The SC block should be built for operations $k$ and $l$, where $\{k,l\}R_i$. The output $Y_{S_{ik}}$ of neuron $S_{ik}$ corresponds to $X_{ik}$, the starting time of the operation $[i,k]$.

The output of neuron C is equal to 0 if Eq. (1) is satisfied, otherwise it is equal to the overlapping time of the operations $[i,k]$ and $[i,l]$. The outputs of S type neurons will be adjusted through the feedback from neuron C iteratively. The neurons in the SC block run at the sequence of S type, at time $t \rightarrow$ C type, at time $t$, and then S type, at time $t + 1$. This process will iterate until all conflicts are resolved.

**RC block.** This block is designed to check constraints (2a) and (2b) and to adjust the associated job's
start times if the constraints are not met. It is constructed with one neuron of I type, two neurons of C type, and two neurons of S type, as shown in Fig. 1(b).

An RC block should be built for operations \([i, k]\) and \([j, l]\) of \([k, l]N_q\), because they all need resource \(q\) to enable them to start. The neuron \(C_1\) of type C collects the outputs of neurons \(S_{jl}\) and \(S_{ik}\) of type S through links with weights +1 and −1 and receive the adjustment −\(Y_{ik}H\) from neuron \(I_{kl}\) at the same time. The bias of neuron \(C_1\) is set to be \(H − t_{ik}\). The output of neuron \(C_1\) is 0 if Eq. (2a) is satisfied;
otherwise, it is 1. The outputs of neurons $S_{il}$ and $S_{ik}$ are adjusted through the feedback +0.1 and −0.1. Neuron $C_2$ is similar to $C_1$, but its bias is $-t_{il}$. Neurons $C_1$ and $C_2$ coordinate in this complementary mode to meet the constraints in Eqs. (2a) and (2b).

The neurons in RC block run at the sequence of $S_{type}$, at time $t \rightarrow I_{type}$, at time $t \rightarrow C_{type}$, at time $t$, setting $t = t + 1$. They repeat this process until all conflicts are resolved. It is worth noting that the output of neuron $I_{kl}$ indicates the sequence of operation $[i, k]$ and operation $[j, l]$. The output of $I_{kl}$ is $Y_{I_{kl}}$, which is an equivalent of $z_{kl}$. A simple scheduling strategy is represented with the dotted line from $S_{type}$ neurons to neuron $I$ in Fig. 1(b). The strategy here uses the current starting times of jobs to decide their sequences. Actually, a more efficient sequence strategy by means of GA is introduced in a later section. It is easy to know that fixing the output of $I_{kl}$ is equal to setting the sequence between the operation $[i, k]$ and operation $[j, l]$.

**JC block for processing operations.** This block is designed to check constraints (3a) and (3b) and to adjust the related job’s start time if the constraints are not met. It is constructed with one neuron of $I_{type}$, two neurons of $C_{type}$, and two neurons of $S_{type}$, as shown in Fig. 1(c).

Neuron $C_1$ of type C collects the outputs of neurons $S_{il}$ and $S_{ik}$ through connection weights +1 and −1 and receives the adjustment $-Y_{I_{kl}}H$ from neuron $I_{kl}$ at the same time.

The bias input of neuron $C_1$ is set to be $H - t_{ik}$. The output of neuron $C_1$ is 0 if Eq. (3a) is satisfied. Otherwise, it is 1. The outputs of neurons $S_{il}$ and $S_{ik}$ are adjusted through the feedback +0.1 and −0.1 to resolve their conflicts. The neuron $C_2$ is similar to $C_1$, but its bias is $-t_{il}$. Neurons $C_1$ and $C_2$ coordinate in a complementary mode to meet constraint equations (3a) and (3b), respectively. The running sequences of the neurons in the JC block follow the running $S_{type}$, at time $t \rightarrow I_{type}$, at time $t \rightarrow C_{type}$, at time $t$, setting $t = t + 1$. This process will be repeated until all conflicts are resolved. It is worth noting that the output of neuron $I_{kl}$ indicates the sequence of operations $[i, k]$ and $[i, l]$. $Y_{I_{kl}}$, the output of $I_{kl}$, is equivalent to $y_{kl}$. In Fig. 1(c), a simple scheduling strategy is presented with the dotted line from $S_{type}$ neurons to neuron $I_{kl}$. The initial start time of jobs will decide their sequence. A more efficient sequence strategy by means of GA is introduced in a later section.

### 3.3. Constraint neural network

A CNN is composed of three different levels of neurons. The outline of the general architecture of the CNN is shown in Fig. 2.

A CNN can be used to get a feasible or sub-optimal solution of an EJSSP. Certainly, neurons of $S_{type}$ will receive feedback from many $C_{type}$ neurons. Besides the activation of one constraint equation, interaction exists as well. The following three kinds of operating methods are investigated briefly:

1. **Synchronous mode.** In this mode, each neuron of the CNN will update at the same time. This concept is similar to that in the Hopfield network. The rationality of SC, RC and JC block evolution will be destroyed by the neuron’s synchronous updating, although it will truly be in favor of suppressing the man-made effect of the initialized starting time. Divergence occurs frequently in this situation.

2. **Asynchronous mode.** In this mode, each neuron of the CNN is triggered randomly to update its state. The major drawback lies in the fact that it will take too long to end the CNN’s iterations, i.e. until no conflicts exist.
(3) **Hybrid mode.** This hybrid mode works with synchronous and asynchronous characteristics. A block is taken as the basic synchronous unit. Asynchronous running happens within each block of SC, RC and JC as the prescribed sequence in Section 3.2.

Many experiments show that the hybrid mode is the best one to apply. The well-known $6 \times 6$ benchmark problem is used as an example to illustrate the CNN’s application in Section 6. It is indicated that the CNN is applicable to some practical scheduling problems although it is very poor at optimization of job sequencing and timing, especially for early and tardy scheduling problems.

### 4. Neural-based approach to production scheduling

To improve the CNN’s optimization ability and then its performance, a neural-based approach is proposed with the capability of optimizing the start time of expanded job-shop scheduling.

#### 4.1. Gradient search algorithm for scheduling optimization

Define an energy function.

$$E = Z$$  \hspace{1cm} (16)

The evolution of neuron $S_{ik}$ of type S follows,

$$S_{ik}(t + 1) = S_{ik}(t) - \lambda \frac{\partial E}{\partial S_{ik}}$$  \hspace{1cm} (17)

where $\lambda$ is a positive coefficient and $i \ (1, 2, \ldots, n)$. For

$$Z = \max_i (x_i + t_i)$$  \hspace{1cm} (18)
The first order of derivation of the energy function $E$,

$$\frac{\partial E}{\partial S_{ik}} = \begin{cases} 1 & \text{if } k \text{ is the last operation of job } i \\ 0 & \text{otherwise} \end{cases}$$  \hspace{1cm} (19)

For

$$Z = \sum_{i=1}^{n} [h_i \times \max(0, d_i - x_{il}) + w_i \times \max(0, x_{il} - d_i)]$$  \hspace{1cm} (20)

$$\frac{\partial E}{\partial S_{ik}} = \begin{cases} -h_i & \text{if } k \text{ is the last operation of job } i \text{ and } d_i - x_{il} > 0 \\ w_i & \text{if } k \text{ is the last operation of job } i \text{ and } d_i - x_{il} < 0 \\ 0 & \text{otherwise} \end{cases}$$  \hspace{1cm} (21)

Eqs. (20) and (22) are used to direct the CNN toward an optimum solution. $k = e$ means that operation $k$ is the last operation of the job.

4.2. Gradient CNN

Most scheduling problems can be decomposed into two parts, the constraint satisfaction part and the optimization part. The correspondent scheduling system is designed with two components, the constraint satisfaction neural network and the optimization algorithm.

The two components in the gradient CNN cooperate as follows:

1. Generate the start time for each operation randomly, and set these as the initial outputs of corresponding neurons of group $S$ at the bottom of the CNN.
2. Run SC blocks to force process sequences to be right and then run RC and JC blocks iteratively, to get a feasible scheduling solution.
3. Repeat Step 2 until all neurons of the $S$ group are in steady states without changes. Then a feasible solution is found.
4. Check whether the feasible solution is satisfied. If not, go to Step 5, otherwise stop the search process.
5. Adjust neurons of the $S$ group by the gradient search algorithm with Eq. (17).
6. Go back to Step 2.

The $6 \times 6$ benchmark problem is taken as a testing example in Example 2 and early/tardy (E/T) scheduling problem in Example 3 to show the gradient CNN’s characteristics. It is concluded that the gradient CNN still has poor sequence ability, although it shows great improvement in optimal performance. This issue will be dealt with in Section 5.

5. Hybrid approach to expanded job-shop scheduling

The GA is a new type of search algorithm which is analogous to the natural selection and genetic mechanism, which was put forward by Professor John Holland and his colleague at Michigan University
in the 1960s. The study of Dr K. D. Jong and D. E. Goldberg et al. made significant progress in theoretical study as well as practical applications (Goldberg, 1989).

There are distinguishing characteristics between the GA and the traditional search algorithm:

(1) the GA operates the codes of the variables, but not the variables themselves;
(2) the GA searches optimal points starting from a group of available points, but not from a signal point;
(3) the GA’s search is directed only by paying off information without the need of the derivative of this information;
(4) the GA is a kind of random search algorithm but not a definitive algorithm.

It is considered that the GA is a widely applicable and highly effective algorithm. The simple generic algorithm (SGA) representing the GA’s major characteristics was mainly composed of selection, crossover and mutation operators. 

The selection mechanism takes the function of embodying the principle of “natural selection and survival of the fittest”. The survival ability of each chromosome mainly depends on its adaptability to the optimal space. The crossover operator just contributes to the exploitation for new unknown space, so it is important to design an effective crossover operator for improving the GA’s efficiency. It is crucial to exploit the specific domain problem to design a good crossover operation. The mutation operator acts as diversifying function to prohibit the chromosome from falling into a local optimum early due to the loss of some genes.

The GA is applied to the sequence optimization of the scheduling problem in this paper. The related chromosome coding, crossover operator and mutation operator will be discussed in detail.
Coding strategy permutation code is applied for sequence optimization here. For example, string (b d c e a f) means the operations will be processed at the sequence ‘b’ first, ‘d’ second, ‘c’ third, ‘e’ fourth, ‘a’ fifth, and ‘f’ sixth.

It is only necessary to sequence operations that require common resources. It is not necessary and meaningful to sequence operations that do not have a common resource requirement, so it is enough to give out all permutation segments of the operations by resources. The form of permutation is somewhat like that in Fig. 3.

Certainly, it could represent all possible sequences for all schedulable operations. It is called a compatible permutation code if sequence conflicts exist among operations of different permutation segments.

It will be discussed how to generate the initial permutation for all operations in an expanded job-shop environment. It is known that there are needs of scheduling among the operations requiring common resources, so it is complex to sequence operations when there are many kinds of resource except for machine requirement. First, find out all operations [k, l] requiring a fixed common resource that needs to be scheduled, and then sequence them randomly to generate its permutation code. Select all operations requiring another fixed resource except for those pre-sequenced operations, and randomly sequence them to generate another segment of permutation code. Obviously, this procedure will ensure the compatibility of the permutation code.

The procedure of initialization of a chromosome for an operation sequence is as follows:

Step 1. Select a kind of resource S from the resource set S, from 1 to m. S_i is one of the Ss, i is from 1 to r_S.

Step 2. Randomly sequence the operations requiring resource S_i in S except for pre-sequenced operations.

Step 3. Randomly insert pre-sequenced operations into the permutation to form a complete permutation of resource S without destroying the sequence of pre-sequenced operations.

Step 4. S ← S + 1. Return to Step 2 until S > m.

Normally, major resources such as machines will be considered first to compute their operation permutations. Chromosomes after being operated may not be feasible because there may be conflicts among them. A similar procedure will be applied to modify the chromosomes after being operated to turn them into a feasible schedule. In this modification procedure, the sequence by resource will be randomly resequenced. Then a reasonable chromosome is formed.

Mapping is another important issue between permutation code and the outputs of type I neurons in the RC and JC blocks. For any operation [i, k] and [j, l] within an RC block, the output ZI_{kl} = 1 is fixed when [i, k] precedes [j, l] in permutation code and vice versa. For operations [i, k] and [i, l] within a JC block, the output YI_{kl} = 1 is fixed when [i, k] precedes [i, l] in permutation code and vice versa.

\[
ZI_{kl} = \begin{cases} 
1 & \text{if } [i, k] \text{ precedes } [j, l] \\ 
0 & \text{otherwise} 
\end{cases} 
\]  

\[
YI_{kl} = \begin{cases} 
1 & \text{if } [i, k] \text{ precedes } [i, l] \\ 
0 & \text{otherwise} 
\end{cases} 
\]
Crossover operator. Crossover performs a search for a new solution space. Crossover happens between the same segment of permutation code for each resource from two selected chromosomes.

For an expanded job shop with objective (6), order crossover (OX) (Poon & Carter, 1995) is applied. For E/T scheduling problems with objective (7), an E/T heuristic crossover, designed by the authors, is applied to a segment of the permutation code for machines. The E/T heuristic crossover is highly effective (Yu et al., 2000), for an E/T penalty scheduling problem and is detailed as follows:

**Step 1.** Randomly select two father chromosomes.

E.g. father1: (e, a, c, b, f, d) → (2, 5, 9, 10, 17, 23)
father2: (b, d, a, f, c, e) → (0, 5, 7, 14, 20, 21)

**Step 2.** Select genes from two father chromosomes to reserve randomly.

E.g. proto-child1: (e, , c, b, , ) → (2, , 9, 10, , )
proto-child2: ( , d, , f, , e) → ( , 5, , 14, , 21)

**Step 3.** Fill empty bit positions for one chromosome code with the complementary genes with the optimal starting times in another chromosome.

E.g. proto-child1’s complementary genes in proto-child2: ( , a, , f, d) → ( , 7, , 14, 5)
proto-child2’s complementary genes in proto-child1: (b, , a, , c, ) → (10, , 5, , 9, )
proto-child1: (e, a, c, b, f, d) → (2, 7, 9, 10, 14, 5)
proto-child2: (b, d, a, f, c, e) → (10, 5, 5, 14, 9, 21)

**Step 4.** Re-sort gene positions of these two proto-child chromosomes in ascending order, and generate two final child chromosomes.

E.g. child1: (e, d, a, c, b, f) → (2, 5, 7, 9, 10, 14)
child2: (d, a, f, b, f, e) → (5, 5, 9, 10, 14, 21)

Mutation operator. A swap operator is applied in this paper. The swap operator exchanges two stochastically selected genes. Taking (e, a, b, c, f, d) as an example to explain the swap operator, the result after swap is applied should be (e, c, b, a, f, d) if genes a and c are selected for mutation.

It has been verified that CNN is a powerful modeling framework and that gradient CNN is effective for time optimization in the prescribed sequence. It may not be optimal owing to the setting of the outputs $Y_{lkj}$, i.e. the operation sequence of I type neurons is mostly random.

Unification of GA and gradient CNN. A GA was introduced for optimizing sequence in the last section. Any chromosome code corresponds to a process sequence. The evaluation of a chromosome code should be decided by the performance of its best scheduling with optimal operation start time at the chromosome code’s sequence. The schedule resulting from the gradient CNN will be taken to represent the corresponding chromosome code’s performance to evaluate the chromosome. It can be said that the optimizing operations’ start time with gradient CNN is just a step of the chromosome’s performance evaluation.

The hybrid scheduling method is as follows:

**Step 1.** Initialization — assign a starting time to each operation.

**Step 2.** Prescribe the sequence of operations by chromosomes.

**Step 3.** Run the CNN in hybrid mode.

**Step 4.** Return to Step 3 if conflicts exist; otherwise, go to Step 5.

**Step 5.** Apply the gradient search algorithm to neurons of S type if starting time optimization is needed, then return to Step 3; otherwise, go to Step 6.
Step 6. Apply the GA’s crossover if an adjustment of the sequence of operations is needed, then return to Step 2; otherwise, end.

6. Case study

Scheduling problems are studied in this section to show the performance and characteristics of the method proposed in this paper. By the way, the following examples have been implemented using the C programming language on a personal computer with a 486/100 processor.

Example 1. The well-known 6 × 6 benchmark problem is taken from Muth and Thompson (1963). The 6 × 6 benchmark problem is shown in Table 1. It is known that the minimal completion time of the last job is 55.

Table 1  
The 6 × 6 benchmark scheduling data

<table>
<thead>
<tr>
<th></th>
<th>(m,t)</th>
<th>(m,t)</th>
<th>(m,t)</th>
<th>(m,t)</th>
<th>(m,t)</th>
<th>(m,t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job 1</td>
<td>3.1 →</td>
<td>1.3 →</td>
<td>2.6 →</td>
<td>4.7 →</td>
<td>6.3 →</td>
<td>5.6</td>
</tr>
<tr>
<td>Job 2</td>
<td>2.8 →</td>
<td>3.5 →</td>
<td>5.10 →</td>
<td>6.10 →</td>
<td>1.10 →</td>
<td>4.4</td>
</tr>
<tr>
<td>Job 3</td>
<td>3.5 →</td>
<td>4.4 →</td>
<td>6.8 →</td>
<td>1.9 →</td>
<td>2.1 →</td>
<td>5.7</td>
</tr>
<tr>
<td>Job 4</td>
<td>2.5 →</td>
<td>1.5 →</td>
<td>3.5 →</td>
<td>4.3 →</td>
<td>5.8 →</td>
<td>6.9</td>
</tr>
<tr>
<td>Job 5</td>
<td>3.9 →</td>
<td>2.3 →</td>
<td>5.5 →</td>
<td>6.4 →</td>
<td>1.3 →</td>
<td>4.1</td>
</tr>
<tr>
<td>Job 6</td>
<td>2.3 →</td>
<td>4.3 →</td>
<td>6.9 →</td>
<td>1.10 →</td>
<td>5.4 →</td>
<td>3.1</td>
</tr>
</tbody>
</table>

(1) The CNN and gradient CNN are applied to the benchmark problem. Different constraints or initial conditions and scheduling methods are applied to compare their performance. The results are shown in Table 2 where the completion time of the last job comes from experiments performed 400 times.

Table 2  
Results of 6 × 6 benchmark using CNN and gradient CNN

<table>
<thead>
<tr>
<th>Scheduling method applied</th>
<th>Due date setting</th>
<th>Initial starting time</th>
<th>Completion time of the last job</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN</td>
<td>+ ∞</td>
<td>0</td>
<td>73</td>
</tr>
<tr>
<td>Gradient CNN</td>
<td>+ ∞</td>
<td>Random</td>
<td>112.65/128/68</td>
</tr>
<tr>
<td>CNN</td>
<td>80</td>
<td>0</td>
<td>65</td>
</tr>
<tr>
<td>Gradient CNN</td>
<td>80</td>
<td>Random</td>
<td>63.4/65/58</td>
</tr>
<tr>
<td>CNN</td>
<td>60</td>
<td>0</td>
<td>66</td>
</tr>
<tr>
<td>Gradient CNN</td>
<td>60</td>
<td>Random</td>
<td>63/66/57</td>
</tr>
<tr>
<td>CNN</td>
<td>60</td>
<td>Random</td>
<td>58.2/65/55</td>
</tr>
</tbody>
</table>
Compared with results obtained using CNN, those obtained using gradient CNN are better. It is indicated that the average performance using gradient CNN is nearly independent of the due date restriction and the initial setting of operations’ start time. The chance of obtaining the best schedule is still very low. This means that gradient CNN is more powerful than CNN alone but it needs to be improved further.

(2) The hybrid method is applied to the benchmark problem also. It is seen from Table 2 that it is almost impossible to obtain optimal scheduling for the $6 \times 6$ benchmark problem using gradient CNN.
Table 3
Results of early/tardy scheduling on single machine

<table>
<thead>
<tr>
<th>Job no.</th>
<th>Processing time</th>
<th>Due date</th>
<th>$h_i$</th>
<th>$w_i$</th>
<th>Optimal start time</th>
<th>Processing sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14</td>
<td>100</td>
<td>4</td>
<td>5</td>
<td>87</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>93</td>
<td>1</td>
<td>3</td>
<td>28</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>121</td>
<td>2</td>
<td>3</td>
<td>112</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>126</td>
<td>5</td>
<td>2</td>
<td>145</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>13</td>
<td>16</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>43</td>
<td>5</td>
<td>1</td>
<td>39</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>110</td>
<td>5</td>
<td>1</td>
<td>101</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>12</td>
<td>31</td>
<td>5</td>
<td>4</td>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>132</td>
<td>4</td>
<td>3</td>
<td>122</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>55</td>
<td>3</td>
<td>2</td>
<td>45</td>
<td>5</td>
</tr>
<tr>
<td>11</td>
<td>8</td>
<td>137</td>
<td>3</td>
<td>3</td>
<td>132</td>
<td>13</td>
</tr>
<tr>
<td>12</td>
<td>11</td>
<td>87</td>
<td>3</td>
<td>2</td>
<td>76</td>
<td>8</td>
</tr>
<tr>
<td>13</td>
<td>12</td>
<td>77</td>
<td>5</td>
<td>3</td>
<td>64</td>
<td>7</td>
</tr>
<tr>
<td>14</td>
<td>5</td>
<td>150</td>
<td>2</td>
<td>5</td>
<td>140</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>8</td>
<td>62</td>
<td>3</td>
<td>3</td>
<td>55</td>
<td>6</td>
</tr>
</tbody>
</table>

The hybrid approach is employed to this benchmark problem and four optimal schedules are found with different sequences. The results are shown in Fig. 4. Note that the solutions are different from that given by Muth and Thompson (1963).

The average computing time is 6.7 s using gradient CNN. And the average computing time is 7.5 s using the hybrid method. Compared with CNN, we can see that the performance of the hybrid method exceeds that of the gradient CNN with a slight computing time cost.

(3) The effect of due date on performance. From the above, it is obvious that the due date HI is critical for scheduling with objective (6). If due date were too loose, it would be easy to obtain a feasible solution, but the solution’s performance is frequently poor on average. However, it is hard to find a feasible solution when the due date is set too strictly.

There is a direct relationship between performance and last due date in Fig. 5(a). The situation is different in Fig. 5(b), where the feasible solutions do not relate strongly to the last due date when it is too loose.

It is reasonable to adjust or decrease the due date setting because it is a kind of soft restriction (Fox, 1994). CNN is applicable to practical expanded job-shop scheduling with the objective in formula (6) by decreasing due date restrictions in step. It is concluded from the example that CNN is applicable to some expanded job-shop scheduling, but it is hard to find an optimal solution.

**Example 2.** Applying gradient CNN to an E/T scheduling problem. The E/T scheduling on a single machine is generated randomly with 15 jobs in Table 3. The processing sequence for the 15 jobs is supposed, shown in the first and last columns of Table 3. The optimal start time is obtained by gradient CNN, which is shown in the last-but-one column. According to the results using the algorithm given by Lee and Choi (1995), it is certified that this scheduling is optimal. It can be concluded that gradient CNN is applicable to E/T scheduling, which is superior to CNN alone for this situation.
The above two examples have shown that gradient CNN has improved scheduling performance but still lacks the ability to optimize the processing sequence.

**Example 3.** This example is taken from the work of Kusiak (1990) to show the modeling ability of CNN. The processing data is in Table 4. Sequence restriction is \( \{1, 2\} Q_1 \) for job 1; \( \{1, 2\} R_2 \) for job 2; \( \{1, 2\} R_3 \) and \( \{2, 3\} R_3 \) for job 3, decided by technological planning.

This is a special expanded JSSP because there is an assemblage relation for job 3 and there are two kinds of resource, machine and cutting tool. A method for original job-shop scheduling may not be applicable to this problem.

<table>
<thead>
<tr>
<th>Job no.</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation no.</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Processing time</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Machine no.</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Cutting tool no.</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

![Optimal scheduling](image.png)

**Fig. 6.** Optimal scheduling for manufacturing with multiple resource restriction.

It is easy to obtain the optimal schedule using the hybrid approach, which is better than that given in the work of Kusiak (1990), with completion time 10 of the last job. All of the above provides evidence that the hybrid approach is not only powerful in modeling complex manufacturing but also more efficient in achieving the optimal scheduling solution (Fig. 6).

7. **Conclusions**

The hybrid approach presented in this paper is based on the original idea of combining the constraint satisfaction neural network with scheduling optimization. CNN is easy to implement owing to its object-oriented character. The computational ability of the hybrid approach is strong enough to deal with complex scheduling problems, thanks to the NN’s parallel computability and the GA’s searching efficiency. Many experiments have shown that this hybrid approach is extremely promising for expanded job-shop scheduling.
Acknowledgements

This paper was supported by the Natural Science Foundation of China under contact 69974039, 59990470. This paper was supported also by the 863/CIMS program of Shenyang Blower Works.

References


