DOI: 10.3901/CJME.2010.**.***, available online at www.cjmenet.com; www.cjmenet.com.cn

New Hybrid Parallel Algorithm for Variable-sized Batch Splitting Scheduling with Alternative Machines in Job Shops

ZHAO Yanwei[1, *], WANG Haiyan[1], WANG Wanliang[2], and XU Xinli[2]

1 *Key Laboratory of Mechanical Manufacture and Automation of Ministry of Education, Zhejiang University of Technology, Hangzhou 310014, China*

2 *College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310014, China*

Received July 28, 2009; revised May 10, 2010; accepted May , 2010; published electronically May , 2010

Abstract: The batch splitting scheduling problem has recently become a major target in manufacturing systems, and the researchers have obtained great achievements, whereas most of existing related researches focus on equal-sized and consistent-sized batch splitting scheduling problem, and solve the problem by fixing the number of sub-batches, or the sub-batch sizes, or both. Under such circumstance and to provide a practical method for production scheduling in batch production mode, a study was made on the batch splitting scheduling problem on alternative machines, based on the objective to minimize the makespan. A scheduling approach was presented to address the variable-sized batch splitting scheduling problem in job shops trying to optimize both the number of sub-bathes and the sub-batch sizes, based on differential evolution(DE), making full use of the finding that the sum of values of genes in one chromosome remains the same before and after mutation in DE. With considering before-arrival set-up time and processing time separately, a variable-sized batch splitting scheduling model was established and a new hybrid algorithm was brought forward to solve both the batch splitting problem and the batch scheduling problem. A new parallel chromosome representation was adopted, and the batch scheduling chromosome and the batch splitting chromosome were treated separately during the global search procedure, based on self-adaptive DE and genetic crossover operator, respectively. A new local search method was further designed to gain a better performance. A solution consists of the optimum number of sub-bathes for each operation per job, the optimum batch size for each sub-batch and the optimum sequence of sub-batches. Computational experiments of four test instances and a realistic problem in a speaker workshop were performed to testify the effectiveness of the proposed scheduling method. The study takes advantage of DE's distinctive feature, and employs the algorithm as a solution approach, and thereby deepens and enriches the content of batch splitting scheduling.

Key words: variable-sized batch splitting, differential evolution, alternative machines, local search

Notations

i—Job index,

j—Operation index,

k—Batch index,

l—Machine index,

N—Total number of jobs,

S_i—Original batch size of job i,

n_i—Total number of operations of job i,

M—Total number of machines,

b_{ij}—Total number of alternative machines for the jth operation of job i,

T_{ijl}^{PP}—Unit processing time for the jth operation of job i on machine l,

T_{ijl}^{SU}—Set-up time for the jth operation of job i on machine l,

S_{ijk}—Batch size of the kth sub-batch for the jth operation of job i,

M_{ijk}—Processing machine of the kth sub-batch for the jth operation of job i,

T_{ijk}^{SSU}—Start time for set-up procedure of the kth sub-batch for the jth operation of job i,

T_{ijk}^{FSU}—Finish time for set-up procedure of the kth sub-batch for the jth operation of job i,

T_{ijk}^{SPP}—Start time for processing procedure of the kth sub-batch for the jth operation of job i,

T_{ijk}^{FPP}—Finish time for processing procedure of the kth sub-batch for the jth operation of job i,

$T_{ij}(NM)$—Time when the number of parts that accomplish the processing procedure for the jth operation of job i reaches NM,

φ_{ijkl}—Need of set-up procedure on machine l for the kth sub-batch for the jth operation of job i,

ϕ_{ijkl}—Relationship between the processing machine of the kth sub-batch for the jth operation of job i

* Corresponding author. E-mail: zyw@zjut.edu.cn

This project is supported by National Hi-tech Research and Development Program of China (863 Program, Grant No. 2007AA04Z155), National Natural Science Foundation of China (Grant No. 60970021), and Zhejiang Provincial Natural Science Foundation of China (Grant No. Y1090592)

and machine l,

$\delta_{ijk\_i'j'k'}$—Processing sequence between the kth sub-batch for the jth operation of job i and the k'th sub-batch for the j'th operation of job i',

N_B—Total number of sub-batches,

L—Length of the chromosome,

N_p—Population size,

CR—Crossover probability,

N_{it}^{max}—The maximal iteration number.

1 Introduction

Under the batch production mode in a real manufacturing environment, a job consists of a batch of identical parts in the production scheduling problem, and there may exist alternative machines for operations. By splitting the original batch into many smaller sub-batches, these smaller sub-batches can be processed on different machines simultaneously, with all parts in one sub-batch processed altogether and the processing time of a sub-batch defined to be the sum of processing time of each part in that sub-batch, sharing only one set-up time, so that a faster completion can be obtained. That's how batch splitting, also called lot streaming in many researches, arises.

Batch splitting in flow shops can be equal (all sub-batches of a given job are of equal size), consistent (sub-batch sizes vary within a job but are the same for all machines), or variable (sub-batch sizes can change from machine to machine)[1]. And these three types of batch splitting can be applied to job shops too. Most of the existing researches on the batch splitting scheduling problem concerns the former two kinds of batch splitting. PAN, et al[2], studied the equal-sized batch splitting scheduling problem with set-up time and alternative machines, with the batch size for each sub-batch fixed in advance. SUN, et al[3], adopted a novel encoding method in genetic algorithm(GA) to optimize both the number of sub-batches for each job and the sub-batch processing order simultaneously when solving the equal-sized batch splitting job shop scheduling problem with set-up time and alternative machines. LOW, et al[4] performed comparisons between equal-sized batch splitting and consistent-sized batch splitting in the batch splitting scheduling problem with the number of sub-batches and the sizes of sub-batches fixed beforehand. MARTIN[1] proposed a heuristic to get the number of sub-batches for each job and the size of sub-batches in the consistent-sized batch splitting scheduling problem in flow shops. Although most of these researches solved the batch splitting scheduling problem by fixing the number of sub-batches, or the sub-batch sizes, or both, their achievements still provided important basis for a further study.

On the basis of the above researches, we focus our attention on variable-sized batch splitting, trying to obtain both the optimum number of sub-bathes for each operation per job and the optimum batch size for each sub-batch in the batch splitting scheduling problem with set-up time.

There is much scope for evolutionary algorithms for batch splitting scheduling problems. Among all kinds of evolutionary algorithms, DE is a newly-developed simple and efficient population-based heuristic, introduced by STORN, et al[5], and has been extensively investigated and improved to solve flow shop scheduling problems and job shop scheduling problems[6–9], but not including the batch splitting scheduling problem yet. Considering that the sum of values of genes in one chromosome remains the same before and after mutation in differential evolution (DE), when chromosomes are of equal length and have the same total value of genes, this paper adopts DE to solve the batch splitting problem, so that the sum of the batch sizes of all the sub-batches for any operation of any job remains the same as the original batch size of that job when values of genes in chromosomes represents batch sizes of sub-batches.

The paper is organized as follows. The formulation for the variable-sized batch splitting scheduling problem with alternative machines is established in section 2. In section 3, a new hybrid parallel algorithm, with a global search procedure, based on self-adaptive DE and genetic crossover operator, and a problem-dependent local search procedure, is brought forward to solve both the batch splitting problem and the batch scheduling problem. Simulation and results are presented and comparisons are drawn in section 4, followed by conclusions in section 5.

2 Problem Description and Formulation

Consider N jobs in a scheduling system. Each job consists of a batch of identical parts, and is planed to be processed in its predefined operation sequence. And there may exist alternative machines for operations.

To simplify the problem and make full use of alternative machines, we assume that jobs are all available at time zero, and the batch number of sub-batches for the jth operation of job i is equal to the total number of alternative machines for that operation, so that chromosomes in the algorithm can be of equal length. Each sub-batch requires one machine out of a set of its alternative machines, and a machine should be set up before it starts a processing procedure for a sub-batch.

A mathematical model for the variable-sized batch splitting scheduling problem is developed in this paper. Values of φ_{ijkl}, ϕ_{ijkl} and $\delta_{ijk\_i'j'k'}$ are listed as follows:

$$\varphi_{ijkl}=\begin{cases}1, \text{ if the } k\text{th sub-batch for the } j\text{th operation of job } i \\ \quad \text{needs set-up procedure on machine } l, \\ \quad 0, \text{ otherwise.}\end{cases}$$

$$\phi_{ijkl} = \begin{cases} 1, \text{ if the } k\text{th sub-batch for the } j\text{th operation} \\ \quad \text{of job } i \text{ is assigned to machine } l, \\ \quad 0, \text{ otherwise.} \end{cases}$$

$$\delta_{ijk\_i'j'k'} = \begin{cases} 1, \text{ if the } k\text{th sub-batch for the } j\text{th operation} \\ \quad \text{of job } i \text{ precedes the } k'\text{th sub-batch} \\ \quad \text{for the } j'\text{th operation of job } i', \\ 0, \text{ otherwise.} \end{cases}$$

The mathematical model is established as follows:

$$\min Z = \max_{i=1}^{N}\{\max_{k=1}^{b_{in_i}}\{T_{in_ik}^{\mathrm{FPP}}\}\}, \tag{1}$$

$$\sum_{k=1}^{b_{ij}} S_{ijk} = S_i, \ \forall\, i,\, j,\, S_{ijk} \in \mathbf{Z}, \tag{2}$$

$$T_{ijk}^{\mathrm{SSU}} \geqslant T_{i(j-1)}\left(\sum_{k'=1}^{k} S_{ijk'}\right) - \varphi_{ijk(M_{ijk})} T_{ij(M_{ijk})}^{\mathrm{SU}}, \quad \forall\, j > 1, \ \forall\, i, k, \tag{3}$$

$$T_{ijk}^{\mathrm{SSU}} \geqslant T_{i'j'k'}^{\mathrm{FPP}} \delta_{i'j'k'\_ijk} \phi_{i'j'k'(M_{ijk})}, \quad \forall\, i, j, k, i', j', k', \tag{4}$$

$$T_{ijk}^{\mathrm{FSU}} = T_{ijk}^{\mathrm{SSU}} + \varphi_{ijk(M_{ijk})} T_{ij(M_{ijk})}^{\mathrm{SU}}, \ \forall\, i, j, k, \tag{5}$$

$$T_{ijk}^{\mathrm{SPP}} = T_{ijk}^{\mathrm{FSU}}, \quad \forall\, i, j, k, \tag{6}$$

$$T_{ijk}^{\mathrm{FPP}} = T_{ijk}^{\mathrm{SPP}} + T_{ij(M_{ijk})}^{\mathrm{PP}} S_{ijk}, \ \forall\, i, j, k, \tag{7}$$

$$i, i' = 1, 2, \cdots, N; \ j = 1, 2, \cdots, n_i; \ j' = 1, 2, \cdots, n_{i'}; \\ k = 1, 2, \cdots, b_{ij}; \ k' = 1, 2, \cdots, b_{i'j'}. \tag{8}$$

Eq. (1) specifies the objective to minimize the makespan, defined by the maximum finish time of processing procedures for the latest operations. Eq. (2) ensures that the sum of the batch sizes of all the sub-batches for an operation of a job remains the same as the original batch size of that job. Eq. (3) shows that a machine is allowed to be set up for a sub-batch for an operation of a job before all the parts in the sub-batch finish the processing procedure for its predecessor operation of the same job, so that when all the parts in the sub-batch are ready, the machine can start processing procedure immediately. Note that there is an add-up notation in Eq. (3), owing to the rule presented in section 3.1 that all sub-batches within an operation of a job are sequenced in an increasing order of batch index. However, the machine starts the set-up procedure for a sub-batch only after, at least, it finishes the processing procedure for the predecessor sub-batch in the processing sequence on that machine, as is shown in Eq. (4). If the predecessor and successor sub-batches in the processing sequence on a machine are to deal with the same operation of the same job, the successor sub-batch does not need set-up procedure on that machine. Eq. (5) describes that when a sub-batch for an operation of a job needs set-up procedure, the set-up procedure couldn't be interrupted once started. Eq. (6) provides the relationship between the start time of processing procedure and the finish time of set-up procedure for a sub-batch, which certainly specifies the sequence between the set-up procedure and the processing procedure for any given sub-batch. Eq. (7) shows that processing procedure for a sub-batch couldn't be interrupted once started.

3 New Hybrid Parallel Algorithm

Since the batch number for an operation of a job is equal to the total number of alternative machines for that operation, there are $N_\mathrm{B} = \sum_{i=1}^{N}\sum_{j=1}^{n_i} b_{ij}$ sub-batches in all. Sizes and sequence of these sub-batches and machines allocated for these sub-batches are to be determined through algorithm. The proposed algorithm is detailed as follows, and the framework is illustrated in Fig. 1.

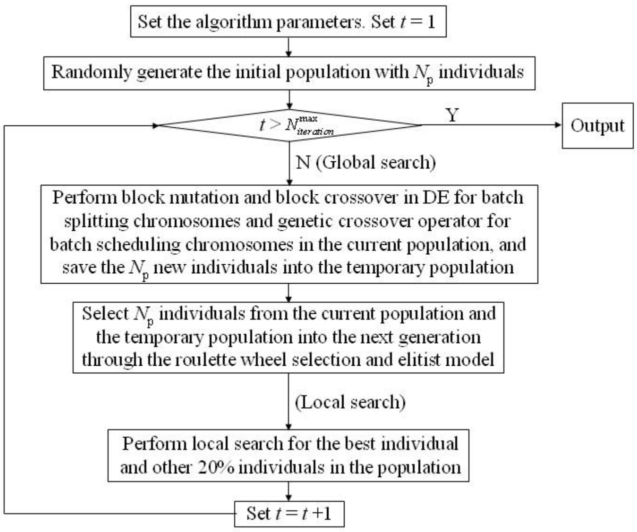

Fig. 1. Framework of the proposed algorithm

3.1 Individual representation

Parallel chromosome coding method is adopted to represent an individual, one called batch splitting chromosome, composed by batch sizes of N_B sub-batches, and the other called batch scheduling chromosome, containing sequence information of N_B sub-batches.

Randomly generate b_{ij} integers within the range $[0, S_i]$ that satisfy Eq. (2) as S_{ijk} for the jth operation of job i, where $i = 1, 2, \cdots, N$, $j = 1, 2, \cdots, n_i$, and $k = 1, 2, \cdots, b_{ij}$. All the batch sizes of these N_B sub-batches constitute the batch splitting chromosome, denoted by *chromosome*1, with its length $L = N_\mathrm{B}$, is shown as follows:

$$S_{111}\ S_{112}\cdots S_{11b_{11}}\ |\ S_{121}\ S_{122}\cdots S_{12b_{12}}\ |\cdots$$
$$|\ S_{ij1}\ S_{ij2}\cdots S_{ijb_{ij}}\ |\cdots|\ S_{Nn_N1}\ S_{Nn_N2}\cdots S_{Nn_Nb_{Nn_N}},$$

where S_{ijk} stands for the batch size of the kth sub-batch for the jth operation of job i. We denote $S_{ij1}\ S_{ij2}\cdots\ S_{ijb_{ij}}$ on *chromosome*1 as the sub-batch size array for the jth operation of job i, where $i=1,2,\cdots,N$ and $j=1,2,\cdots,n_i$. Values of zero are allowed in the batch splitting chromosome. The sequence of these N_B sub-batches in *chromosome*1 constitutes the batch scheduling chromosome, denoted by *chromosome*2, with the length L. For the batch splitting scheduling problem listed in Table 1, Fig. 2 and Fig. 3 are examples for the batch splitting chromosome and the batch scheduling chromosome for the problem respectively. Genes on *chromosome*2 are in "*ij*" format, referring to the jth operation of job i, and gene "*ij*" appears b_{ij} times overall. We specify the batch index to gene "*ij*" according to the time "*ij*" appears from left to right on *chromosome*2, which infers that all sub-batches within an operation of a job are sequenced in an increasing order of batch index. For example, the third gene "21" on *chromosome*2 in Fig. 3 form left to right represents the second sub-batch for the first operation of job 2, since the gene appears the second time from left to right, and the size is 4, which can be seen from *chromosome*1 in Fig. 2.

Table 1. Batch splitting scheduling problem

Job	Original batch size	Operation 1	Operation 2
J_1	20	b_{11}=2	b_{12}=3
J_2	15	b_{21}=3	b_{22}=2

13 7 | 10 0 10 | 5 4 6 | 7 8

Fig. 2. Batch splitting chromosome for the problem in Table 1

11 21 21 11 12 21 22 12 12 22

Fig. 3. Batch scheduling chromosome for the problem in Table 1

All the sub-batches for the jth operation of job i are required to be sequenced ahead of any sub-batch for the j'th operation of job i when $j<j'$, which means that all the sub-batches for all previous operations of a job are scheduled before any sub-batch for the current operation of that job. Then the time when parts with certain size accomplishes the processing procedure for the predecessor operation of a job can be obtained according to the completed arrangement of all the sub-batches for the predecessor operation of the job, which is needed in Eq. (3), when handling a sub-batch for the current operation of the same job.

A schedule can be obtained by decoding genes on *chromosome*2 from left to right, combined with *chromosome*1: If the gene on *chromosome*2 represents the kth sub-batch for the jth operation of job i, get the size for the sub-batch from *chromosome*1, and calculate the time when parts with size $\sum_{k'=1}^{k} S_{ijk'}$ accomplishes the processing procedure for the ($j-1$)th operation of the job if $j>1$. According to the tasks that are already allocated to alternative machines for the jth operation and constraints Eqs. (3)–(8), we select a machine as M_{ijk} that finishes the processing procedure with the earliest finish time and arrange the set-up procedure and the processing procedure for the sub-batch on machine M_{ijk}.

3.2 Fitness function

Eq. (1) is the objective to minimize the makespan. The fitness function is designed as

$$\max\ f=\frac{1}{Z}, \tag{9}$$

where $Z=\max_{i=1}^{N}\{\max_{k=1}^{b_{in_i}}\{T_{in_ik}^{\text{FPP}}\}\}$.

3.3 Global search procedure

An individual is composed of a batch splitting chromosome and a batch scheduling chromosome, due to the finding that the sum of values of genes in one chromosome remains the same before and after mutation in DE and powerful optimization ability of GA for scheduling, they evolve using DE and genetic crossover operator respectively. We denote *chromosome*1_h and *chromosome*2_h as the batch splitting chromosome and the batch scheduling chromosome from individual h respectively.

3.3.1 Evolution procedure for the batch splitting chromosome

A self-adaptive DE-based evolution procedure is designed for the batch splitting chromosome in this section.

(1) Evolution procedure. A DE with block mutation and block crossover is adopted for the batch splitting chromosome, and the current population evolves according to the following steps in one cycle.

Step 1: Set individual index h = 1.

Step 2: For individual h in the current population, randomly generate three integers within [1, N_p], denoted by d_1, d_2 and d_3, where N_p represents the population size. d_1, d_2 and d_3 are different from each other, and different from h. Carry out the evolution procedure for *chromosome*1 from individual h according to the following sub-steps:

Step 2.1: Randomly generate an integer within [1, N] as r_1 and another integer within [1, n_{r_1}] as r_2, and set job index i=1 and operation index j=1.

Step 2.2: If $i=r_1$ and $j=r_2$, execute step 2.3. Otherwise, execute step 2.4.

Step 2.3: Carry out block mutation and block crossover for the sub-batch size array for the jth operation of job i on $chromosome1_h$:

$$S'_{ijk}=S^{d_1}_{ijk}+K_{ij}(S^{d_2}_{ijk}-S^{d_3}_{ijk}),\ \ k=1,2,\cdots,b_{ij},\quad (10)$$

where S^{g}_{ijk} refers to the kth gene in the sub-batch size array for the jth operation of job i on $chromosome1$ from individual g (it further represents the batch size of the kth sub-batch for the jth operation of job i on $chromosome1_g$), where $g=d_1, d_2, d_3$. S'_{ijk} refers to the kth gene in the new array obtained through mutation. K_{ij} is the mutation probability for the sub-batch size array for the jth operation of job i on $chromosome1_h$.

To make sure that S'_{ijk} is within [0, S_i], the value of K_{ij} has to satisfy the following two equations. And we randomly choose a value that satisfies these two equations as K_{ij}:

$$K_{ij}\geqslant\max\left\{0,\max_{k=1}^{b_{ij}}\left\{\min\left\{\frac{S_i-S^{d_1}_{ijk}}{S^{d_2}_{ijk}-S^{d_3}_{ijk}},\frac{-S^{d_1}_{ijk}}{S^{d_2}_{ijk}-S^{d_3}_{ijk}}\right\}\right\}\right\},\quad (11)$$

$$K_{ij}\leqslant\min\left\{2,\min_{k=1}^{b_{ij}}\left\{\max\left\{\frac{S_i-S^{d_1}_{ijk}}{S^{d_2}_{ijk}-S^{d_3}_{ijk}},\frac{-S^{d_1}_{ijk}}{S^{d_2}_{ijk}-S^{d_3}_{ijk}}\right\}\right\}\right\}.\quad (12)$$

Since batch sizes in this paper are integer numbers, S'_{ijk} needs a delicate modification. Set SUM=0, and from k=1 to $k=b_{ij}$, perform $S'_{ijk}=[S'_{ijk}]$ and $SUM=SUM+[S'_{ijk}]$ if $k<b_{ij}$, and if $k=b_{ij}$, perform $S'_{ijk}=S_i-SUM$. "[•]" means to get the nearest integer number. In this way, all the S'_{ijk} in the new array is adjusted to integer numbers, and still satisfy Eq. (2), where $k=1, 2,\cdots, b_{ij}$. Select the new array into a temporary chromosome, denoted by $newchro1_h$. Execute step 2.5.

Step 2.4: Randomly generate a real number within [0, 1] as r_3. If $r_3\leqslant CR$, return to step 2.3. Otherwise, select the sub-batch size array for the jth operation of job i on $chromosome1_h$ into $newchro1_h$, and execute step 2.5.

Step 2.5: If $j<n_i$, perform $j=j+1$. Otherwise, perform $i=i+1$.

Step 2.6: If $i\leqslant N$, return to step 2.2. Otherwise, $newchro1_h$ now is a complete batch splitting chromosome. Put $newchro1_h$ into the temporary population.

Step 3: if $h<N_p$, perform $h=h+1$, return to step 2.

(2) Adaptive crossover probability CR. To improve the performance of the algorithm, we introduced the distribution variance of fitness value Ω in Ref. [10] that reflects the diversity of population to adjust the probability of crossover adaptively. Ω is defined as

$$\Omega=\frac{D_t}{D_{\max}},\quad (13)$$

$$D_t=\frac{1}{N_p}\sum_{h=1}^{N_p}(f^t_h-\bar{f}^t)^2,\quad (14)$$

where D_t represents the variance of fitness value, and f^t_h stands for the fitness value of individual h in the tth generation, while $\bar{f}^t$ refers to the average fitness value in the tth generation. $D_{\max}=\max\{D_{t'}, t'=1, 2,\cdots, t\}$, meaning the maximum variance of fitness value across generations.

We can see from Eq. (13) that the value of Ω varies from 0 to 1, and the higher Ω is, the better for the population in terms of the diversity of population. The adaptive crossover probability is designed in this paper as follows:

$$CR=\begin{cases}CR_0, & t=1,\\ CR_0 2^{1-\Omega}, & t>1.\end{cases}\quad (15)$$

The value of CR is adjusted adaptively within [CR_0, $2CR_0$] according to different value of Ω when evolving. When the diversity of population degrades, meaning that the value of Ω grows lower, CR is adjusted to a higher value to raise the exploration ability of the algorithm. Otherwise, when the diversity of population upgrades, meaning that the value of Ω grows higher, CR is then adjusted to a lower value to improve the exploitation ability of the algorithm.

3.3.2 *Evolution procedure for the batch scheduling chromosome*

Randomly divide the current population into $N_p/2$ groups in one cycle, with two batch scheduling chromosomes in each group, and genetic crossover operator is performed for two chromosomes in each group. Assume that two parent chromosomes in a group are denoted as $chromosome2_p$ and $chromosome2_q$, respectively. Perform the following steps.

Step 1: Randomly generate an integer within [1, N] as r_1 and another integer within [1, n_{r_1}] as r_2. Find two blocks on $chromosome2_p$ and $chromosome2_q$ that have least number of genes but contain all the sub-batches for the r_2th operation of job r_1. For the problem in Table 1, if parent chromosomes are shown in Fig. 4 and r_1=1 and r_2=2, blocks covered with shadow in Fig. 4 are the blocks that have least number of genes but contain all the sub-batches for the second operation of job 1.

$chromosome2_p$	11	21	21	11	12	21	22	12	12	22
$chromosome2_q$	11	11	12	21	12	21	21	22	12	22

Fig. 4. Parent chromosomes for the problem in Table 1

Step 2: Exchange two blocks on $chromosome2_p$ and *chromosome*2_q, and two offspring chromosomes are obtained, denoted by *chromosome*$2'_p$ and *chromosome*$2'_q$, respectively. Offspring chromosomes generated from the parent chromosomes in Fig. 4 are shown in Fig. 5.

chromosome$2'_p$	11	21	21	11	12	21	12	21	21	22	12	22
chromosome$2'_q$	11	11	12	21	22	12	12	22				

Fig. 5. Offspring chromosomes for the parent chromosomes in Fig. 4

Step 3: Delete redundant genes and insert missing genes within the newly inserted blocks on *chromosome*$2'_p$ and *chromosome*$2'_q$. When inserting a missing gene, insert the missing gene before the genes that represent any following operations of the same job and behind the genes that represent any previous operations of the same job within the block if those genes exist in the block. The revised offspring chromosomes, denoted by *newchro*2_p and *newchro*2_q respectively, are selected into the temporary population. The revised offspring chromosomes for chromosomes in Fig. 5 are presented in Fig. 6, where genes with underlines stand for the missing genes.

newchro2_p	11	21	21	11	12	12	21	22	12	22
newchro2_q	11	11	12	<u>21</u>	21	<u>21</u>	22	12	12	22

Fig. 6. Revised offspring chromosomes for chromosomes in Fig. 5

3.3.3 Selection procedure

After evolution procedures are performed for the current population, *newchro*1_h and *newchro*2_h construct a new individual h, and there are N_p new individuals in the temporary population. Select N_p individuals out of $2N_p$ individuals from the current population and the temporary population into the next generation through the roulette wheel selection and elitist model[11].

3.4 Local search procedure

An *Insert*-based local search method based on the individual representation proposed in this paper, is brought forward to gain a better performance. Suppose that individual h in the current population is selected to perform the local search, execute the following steps:

Step 1: Randomly generate an integer within [1, L] as *pos* and a natural number within [0, 1] as r_3, where L stands for the length of chromosome. Assume that the *pos*th gene on *chromosome*2_h from left to right represents the kth sub-batch for the jth operation of job i. Set $g=k$, *chromosome*$1'_h$ = *chromosome*1_h and *chromosome*$2'_h$ = *chromosome*2_h.

Step 2: If $r_3=1$, execute step 3. Otherwise, r_3 must be zero. Perform the forward search procedure:

Step 2.1: If $j>1$, find the gene that representing the $b_{i(j-1)}$th sub-batch for the $(j-1)$th operation of job i on *chromosome*$2'_h$, and denote the position as *pos*1. Else, if $j=1$, set *pos*1 $=-1$.

Step 2.2: If the (*pos*-1)th gene on *chromosome*$2'_h$ from left to right refers to a sub-batch for the jth operation of job i, exchange the values of S_{ijg} and $S_{ij(g-1)}$ on *chromosome*$1'_h$, and execute $g=g-1$. Execute *chromosome*$2'_h$ =*Insert*(*chromosome*$2'_h$, *pos*, *pos*-1).

Step 2.3: If f(*chromosome*$1'_h$ & *chromosome*$2'_h$)> f(*chromosome*1_h & *chromosome*2_h), set *chromosome*1_h = *chromosome*$1'_h$ and *chromosome*2_h= *chromosome*$2'_h$.

Step 2.4: Execute *pos*=*pos*-1. If *pos*>(*pos*1+1), return to step 2.2. Otherwise, stop the local search for individual h.

Step 3: Perform the backward search procedure:

Step 3.1: If $j<n_i$, find the gene that representing the first sub-batch for the $(j+1)$th operation of job i on *chromosome*$2'_h$, and denote the position as *pos*2. Else, if $j=n_i$, set *pos*2=$L+1$.

Step 3.2: If the (*pos*+1)th gene on *chromosome*$2'_h$ from left to right refers to a sub-batch for the jth operation of job i, exchange the values of S_{ijg} and $S_{ij(g+1)}$ on *chromosome*$1'_h$, and execute $g=g+1$. Execute *chromosome*$2'_h$ =*Insert*(*chromosome*$2'_h$, *pos*, *pos*+1).

Step 3.3: If f(*chromosome*$1'_h$ & *chromosome*$2'_h$)> f(*chromosome*1_h & *chromosome*2_h), set *chromosome*1_h = *chromosome*$1'_h$ and *chromosome*2_h= *chromosome*$2'_h$.

Step 3.4: Execute *pos*=*pos*+1. If *pos*<(*pos*2-1), return to step 3.2. Otherwise, stop the local search for individual h.

In the above steps, *Insert*(*chromosome*, u, v) means to insert the gene at the uth position in the vth position on *chromosome* from left to right.

3.5 Analysis of the complexity of the proposed algorithm

Consider a batch splitting scheduling problem, with population size N_p and maximal iteration number N_{it}^{max}. In the proposed algorithm, decoding procedure, calculation procedure for adaptive crossover probability *CR*, evolution procedure for the batch splitting chromosomes, evolution procedure for the batch scheduling chromosomes, selection and local search are concerned in one cycle, and their time complexities are $o(N_pL)$, $o(N_p)$, $o(N_pL)$, $o(0.5N_pL^2)$, $o(2N_p^2)$ and $o(0.2N_pL)$ respectively. Therefore, the total time complexity for the proposed algorithm is:

$$o(N_p, N_{it}^{max}, L) = N_{it}^{max}(o(N_pL) + o(N_p) + o(N_pL) + o(0.5N_pL^2) + o(2N_p^2) + o(0.2N_pL)) \approx 0.5N_{it}^{max}o(N_pL^2). \quad (16)$$

From Eq. (16), we can see that the maximal iteration number and population size, especially the length of chromosome affects the computational burden of the algorithm greatly.

4 Experiments and Analyses

4.1 Application of the proposed algorithm to test instances

To evaluate the performance of the proposed algorithm, we adopt scheduling data in Refs. [2, 12–13] to construct the following four test problems:

Problem 1 (denoted by *P*1): The problem is composed of 4 jobs and 6 machines, and the original batch size for each job is 8. The unit processing time for operations on alternative machines is shown in Table 2, and set-up time for an operation of a batch on a machine is equal to its unit processing time on the same machine.

Problem 2 (denoted by *P*2): The problem is composed of 4 jobs and 6 machines, and the original batch size for each job is 20. The unit processing time for operations on alternative machines is shown in Table 2, and set-up time for an operation of a batch on a machine is equal to its unit processing time on the same machine.

Problem 3 (denoted by *P*3): The problem is composed of 6 jobs and 6 machines, and the original batch size for each job is 10. The unit processing time and set-up time for operations on alternative machines is shown in Table 3.

Problem 4 (denoted by *P*4): The problem is composed of 6 jobs and 6 machines, and the original batch size for each job is 20. The unit processing time and set-up time for operations on alternative machines is shown in Table 3.

Table 2. Unit processing time for operations on alternative machines s

Job	Operation	Machine 1	Machine 2	Machine 3	Machine 4	Machine 5	Machine 6
	1	2	3	4	–	–	–
J_1	2	–	3	–	2	4	–
	3	1	4	5	–	–	–
	1	3	–	5	–	2	–
J_2	2	4	3	–	–	6	–
	3	–	–	4	–	7	11
	1	5	6	–	–	–	–
J_3	2	–	4	–	3	5	–
	3	–	–	13	–	9	12
	1	9	–	7	9	–	–
J_4	2	–	6	–	4	–	5
	3	1	–	3	–	–	3

Some related achievements concerned with these problems can be found in Refs. [2, 12–13]. The optimum makespan for *P*1 obtained in Ref. [12] was 87, and the optimum numbers of sub-batches were 1, 3, 5 and 4 for J_1, J_2, J_3 and J_4 respectively, using the proposed algorithm that first optimized the number of sub-batches for each job, and then scheduled those sub-batches based on equal-sized batch allocation method (PS: the optimum makespan in Ref. [12] for the problem with no batch splitting should be 141 instead of 140, as is shown in its gannt chart). In Ref. [2], the obtained optimum makespans for *P*2 were 345, 216, 196 and 194 when the number of sub-batches for each job was fixed to 1, 2, 4 and 5, respectively, using equal-sized batch allocation method. *P*3 was solved in Ref. [13] based on the genetic algorithm and the simulated annealing algorithm, with the number of sub-batches and the sizes of sub-batches fixed in advance, and the optimum makespan was 243 for *P*3 with no batch splitting.

To evaluate the performance of the proposed algorithm and confirm the effectiveness of the local search procedure and the adaptive crossover operator, we set $N_{\text{it}}^{\max}=200$, $N_{\text{p}}=50$ and $CR_0=0.3$, and solve the four test problems. The proposed algorithm has been coded with Visual C++ .NET 2003 and runs on a PC with a Pentium 2.53 GHz processor and a 1.00 GB RAM under Windows XP 2002. The results obtained over 20 runs are shown in Table 4, where *BMN*, *WMN*, *No.BMN* and *AvT.CPU* denote the best makespan, the worst makespan, the number of the best makespan obtained among 20 runs and the average computational time in seconds over 20 runs respectively. "*ALGRM*1" in Table 4 refers to the hybrid parallel algorithm proposed in this paper. And "*ALGRM*2" is the algorithm as same as *ALGRM*1, except that the local search procedure is not included. "*ALGRM*3" is the algorithm as same as *ALGRM*2, except that the crossover probability *CR* in *ALGRM*3 is fixed to CR_0 throughout evolution.

The increase of the size of problem from *P*1, *P*2 to *P*3, *P*4 adds to the complexity of the batch scheduling problem, while the increase of the original batch size for each job adds to the complexity of the batch splitting problem. It can be observed that the variable-sized batch splitting technique provides considerable makespan reduction, compared with results without batch splitting in Refs. [2, 12–13]. All these three algorithms can obtain excellent optimization outcomes, especially *ALGRM*1 and *ALGRM*2 that even surpass, or at least are not worse than the existing achievements in Refs. [2, 12–13]. From Table 4, we can see that *ALGRM*1 can always outperform the other two algorithms for all test problems in terms of optimization power, and *ALGRM*3 consumes least computation effort. Apparently, *ALGRM*1 derives great benefit from the local search procedure and the adaptive crossover operator, and provides better solutions within reasonable time limit, compared with *ALGRM*2 and *ALGRM*3.

Table 3. Unit processing time/set-up time for operations on alternative machines s

Job	Operation	Machine 1	Machine 2	Machine 3	Machine 4	Machine 5	Machine 6
J_1	1	2/1	–	–	–	–	–
	2	–	–	3/2	2/1	–	–
	3	–	2/1	–	2/2	3/2	2/1
	4	–	5/3	–	6/2	–	–
	5	–	–	2/1	–	–	2/1
	6	–	1/1	–	–	1/1	–
J_2	1	–	2/1	–	1/1	–	–
	2	–	–	4/2	–	–	–
	3	8/2	–	–	–	7/2	7/3
	4	–	4/2	5/1	5/2	–	–
	5	–	–	1/1	–	–	1/1
	6	–	4/2	–	–	5/1	–
J_3	1	4/2	–	5/2	–	–	–
	2	–	5/3	–	5/2	–	–
	3	–	–	1/1	–	1/1	–
	4	–	6/3	–	–	7/2	–
	5	–	2/2	2/1	–	–	3/1
J_4	1	4/2	–	–	4/1	–	–
	2	–	–	2/1	–	–	–
	3	–	4/1	–	3/1	–	3/1
	4	–	–	6/2	–	5/2	–
	5	6/1	–	–	–	–	–
	6	–	3/1	–	2/2	2/1	–
J_5	1	2/1	–	–	3/1	–	–
	2	–	5/1	–	–	4/1	–
	3	–	–	1/1	1/1	–	–
	4	–	–	3/1	–	–	2/1
	5	–	3/1	2/2	–	–	–
	6	–	–	–	–	2/2	–
J_6	1	2/1	–	3/1	–	2/2	–
	2	–	4/1	–	–	3/2	–
	3	–	–	–	6/2	–	6/1
	4	–	2/1	–	2/1	–	–
	5	–	–	1/2	–	–	–
	6	2/2	–	–	3/1	2/2	–

Table 4. Results for these four test problems through three algorithms

Problem	*ALGRM*1				*ALGRM*2				*ALGRM*3			
	BMN	*WMN*	*No. BMN*	*AvT. CPU*	*BMN*	*WMN*	*No. BMN*	*AvT. CPU*	*BMN*	*WMN*	*No. BMN*	*AvT. CPU*
*P*1	85	92	11	10.25	85	91	7	3	85	92	6	2.82
*P*2	183	196	8	12.3	185	203	5	3.84	188	210	3	3.65
*P*3	213	239	5	22.35	218	254	3	6.74	223	253	1	6.05
*P*4	415	464	4	31.8	431	485	2	8.1	432	491	1	7.48

The batch splitting approach in this paper tends to split the original batches into sub-batches of variable size, and compared with Ref. [2] and Ref. [12], we can get that variable-sized batch splitting is better than equal-sized batch splitting. This is probably because the variable-sized batch splitting in this paper takes care of different processing capabilities of alternative machines sufficiently, and the tradeoff between minimizing the total set-up time (achieved by reducing the number of sub-batches) and minimizing the idle time on machines (achieved by reducing batch sizes, or by increasing the number of sub-batches) is well dealt with. We consider that the conclusion acquired by LOW, et al[4] with the number of sub-batches and the sizes of sub-batches fixed in advance for experiment (five instances were prepared in advance covering these two batch allocation methods) is not suitable for general situations.

Optimum solutions to batch splitting for these four test problems through *ALGRM*1 are presented in Table 5, and the corresponding Gantt charts are shown in Figs. 7–10. From Table 5, we can see that the original batch of J_1 in *P*1 is split into 3, 3, 2 and 3 sub-batches for the four operations respectively, and the optimum batch sizes for the respective three sub-batches for the first operation are 0, 1 and 7, which means that the actual optimum batch number for the first operation of J_1 is 2. Sub-batches with size 0 are not displayed in Gantt charts.

Table 5. Solutions to batch splitting for these four test problems through *ALGRM*1

Problem	Operation	Optimum solutions to batch splitting					
		J_1	J_2	J_3	J_4	J_5	J_6
*P*1	1	0, 1, 7	3, 0, 5	5, 3	1, 2, 5	–	–
	2	5, 0, 3	2, 2, 4	0, 1, 7	2, 3, 3	–	–
	3	3, 1, 4	4, 0, 4	2, 3, 3	1, 6, 1	–	–
*P*2	1	6, 9, 5	4, 3, 13	11, 9	13, 4, 3	–	–
	2	3, 8, 9	2, 4, 14	4, 9, 7	3, 3, 14	–	–
	3	9, 6, 5	7, 5, 8	4, 10, 6	7, 10, 3	–	–
*P*3	1	10	9, 1	6, 4	8, 2	3, 7	0, 6, 4
	2	6, 4	10	2, 8	10	4, 6	5, 5
	3	0, 0, 1, 9	4, 3, 3	4, 6	5, 1, 4	7, 3	8, 2
	4	4, 6	6, 0, 4	4, 6	7, 3	4, 6	9, 1
	5	3, 7	9, 1	1, 0, 9	10	4, 6	10
	6	7, 3	6, 4	–	1, 0, 9	10	4, 0, 6
*P*4	1	20	11, 9	5, 15	13, 7	15, 5	1, 15, 4
	2	11, 9	20	8, 12	20	13, 7	10, 10
	3	0, 2, 1, 17	13, 1, 6	14, 6	7, 10, 3	5, 15	13, 7
	4	15, 5	11, 4, 5	9, 11	12, 8	8, 12	6, 14
	5	0, 20	12, 8	0, 2, 18	20	1, 19	20
	6	10, 10	12, 8	–	7, 8, 5	20	7, 2, 11

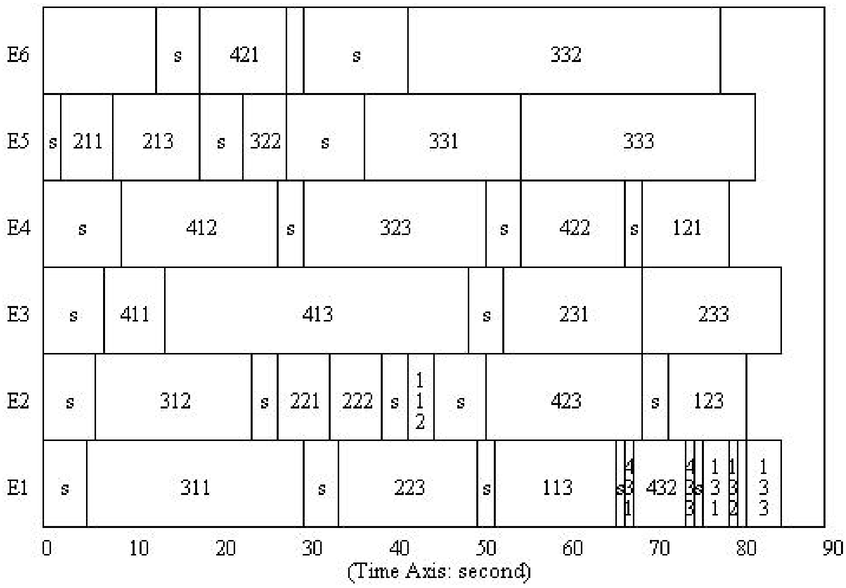

Fig. 7. Gantt chart for *P*1

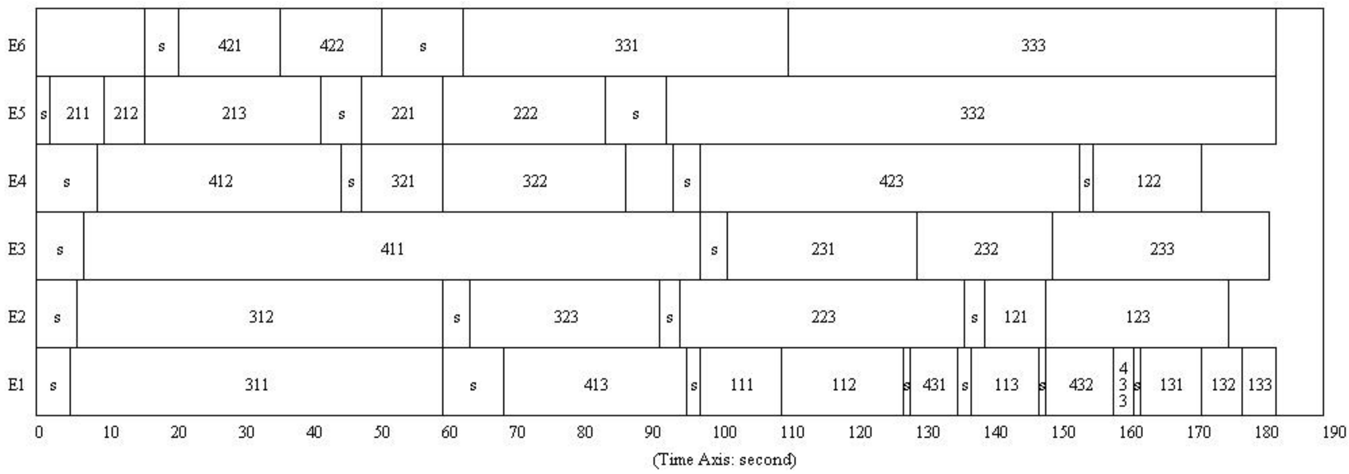

Fig. 8. Gantt chart for *P*2

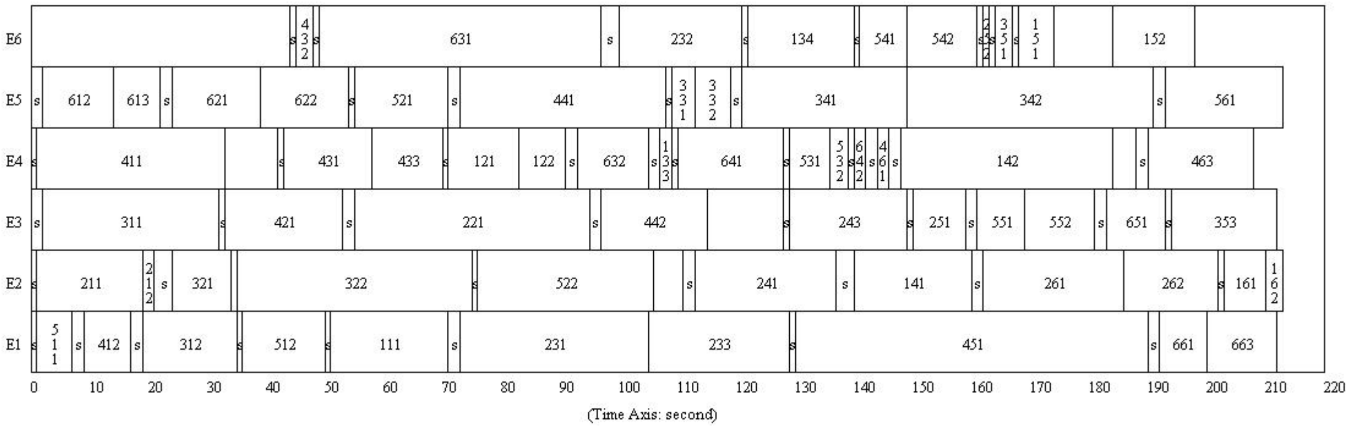

Fig. 9. Gantt chart for *P*3

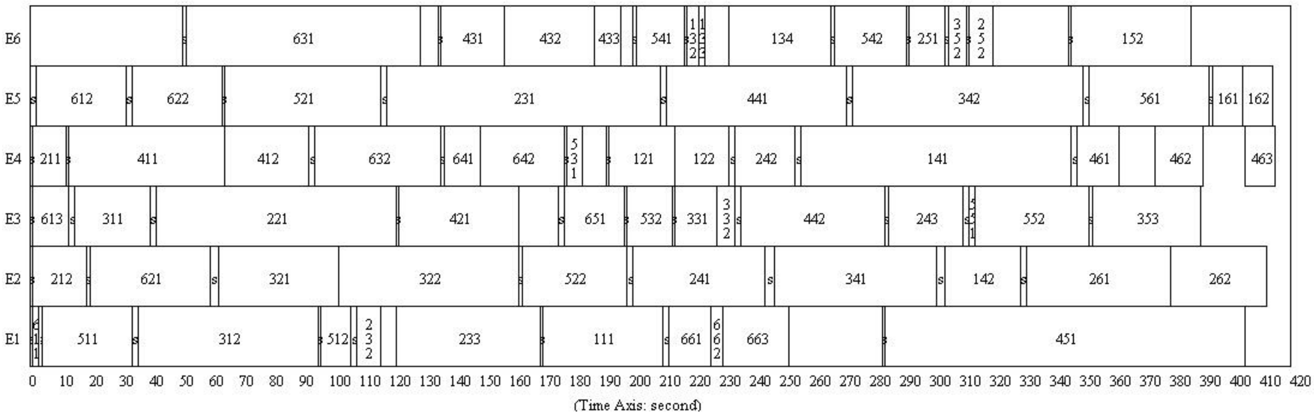

Fig. 10. Gantt chart for *P*4

*E*1 in Gantt charts means machine 1, and *E*2 means machine 2, etc. Panes labeled with "s" in Gantt charts stand for set-up procedures, while those labeled with notations in "abc" format represent processing procedures, where "a" refers to a job index, "b" refers to an operation index, and "c" represents a sub-batch. Take Fig. 7 for example. The pane that labeled with "311" on machine 1 in Fig. 7 stands for the processing procedure for the first operation of the first sub-batch of J_3, and the front pane labeled with "s" adjacent to it stands for the set-up procedure for the same operation. Combined with Table 5 and the time axis, we can get that the processing machine for the first operation of the first sub-batch of J_3, with sub-batch size 5, is machine 1 after machine allocation, and the set-up procedure starts at time 0 and ends at time 5, and the processing procedure starts at time 5 and ends at time 30. As for the pane labeled with "213" on machine 5 in Fig. 7, there is no front pane labeled with "s" adjacent to it, meaning that the third sub-batch for the first operation of J_2 dose not need set-up procedure on machine 5, since the predecessor sub-batch in the processing sequence on that machine is of the same job and the same operation.

4.2 Application of the proposed algorithm to the batch splitting scheduling problem in a speaker workshop

To evaluate the performance of the proposed algorithm in a realistic problem, we adopt a batch splitting scheduling problem in a speaker workshop. The scheduling data is listed in Table 6. Set parameters the same as *ALGRM*1's in section 4.1 and solve the problem through *ALGRM*1. We can obtain that *BMN*=43 256, *WMN*=48 151, *No.BMN*=5 and *AvT.CPU*=37.9 over 20 runs. The optimum solution to batch splitting for the realistic problem through *ALGRM*1 is presented in Table 7, and the corresponding Gantt charts are shown in Fig. 11. Since the set-up time consumed for a sub-batch is relatively short compared with the processing time of the whole sub-batch in this peoblem, set-up procedures are no longer illustrated with separate panes in Gantt chart. Panes labeled with "s" in Gantt chart stand for processing procedures together with set-up procedures.

The results of all the five problems above confirm the validity of the model established in this paper for the variable-sized batch splitting scheduling problem in job shops, as well as the excellent performance of the proposed algorithm. From the data analysis for all test problems, there are several important findings. First, a schedule can be greatly improved through the variable-sized batch splitting technique. Second, the local search procedure and the adaptive crossover operator designed in this paper work effectively to gain a better performance, and our algorithm is capable of providing desirable solution within reasonable time limit. Third, variable-sized batch splitting performs better than equal-sized batch splitting in batch splitting scheduling problem with alternative machines.

Table 6. Scheduling data in the batch splitting scheduling problem in a speaker workshop

Job	Original batch size	Operation		Unit processing time / set-up time for operations on alternative machines											s
				*E*1	*E*2	*E*3	*E*4	*E*5	*E*6	*E*7	*E*8	*E*9	*E*10	*E*11	*E*12
Cloth-edged paper cone (J_1)	600	1	Coat	–	–	–	–	–	5/5	7/5	8/5	–	–	–	–
		2	Stuff	–	–	–	–	10/2	–	–	–	–	–	–	–
		3	Heat-compress	–	–	–	2/4	–	–	–	–	–	–	–	–
		4	Die cut	–	–	–	–	5/2	–	–	–	–	–	–	–
		5	Final check	–	–	–	–	–	–	–	–	–	–	12/0	14/0
Foam-edged paper cone (J_2)	500	1	Paint internal circle	–	–	–	–	–	5/5	9/4	6/7	–	–	–	–
		2	Interim check	–	–	–	–	–	–	–	–	3/0	4/0	–	–
		3	Built paper cone	–	4/8	4/3	–	–	–	–	–	–	–	–	–
		4	Seal obversely	–	15/4	7/7	–	–	–	–	–	–	–	–	–
		5	Seal reversely	–	5/3	5/3	–	–	–	–	–	–	–	–	–
		6	Final check	–	–	–	–	–	–	–	–	–	–	10/0	8/0
Rubber-edged paper cone (J_3)	1 800	1	Paint internal circle	–	–	–	–	–	5/4	7/6	8/6	–	–	–	–
		2	Coat	–	–	–	–	–	6/6	6/8	10/6	–	–	–	–
		3	Interim check	–	–	–	–	–	–	–	–	4/0	5/0	–	–
		4	Built paper cone	–	15/5	14/3	–	–	–	–	–	–	–	–	–
		5	Seal reversely	–	5/3	3/1	–	–	–	–	–	–	–	–	–
		6	Final check	–	–	–	–	–	–	–	–	–	–	10/0	11/0
Paper sub-cone (J_4)	2 000	1	Make brass net	–	–	–	–	6/2	–	–	–	–	–	–	–
		2	Shape up with pulp	4/2	–	–	–	–	–	–	–	–	–	–	–
		3	Interim check	–	–	–	–	–	–	–	–	3/0	5/0	–	–
		4	Die cut	–	–	–	–	5/3	–	–	–	–	–	–	–
		5	Final check	–	–	–	–	–	–	–	–	–	–	6/0	9/0
Paper cap (J_5)	500	1	Paint internal circle	–	–	–	–	–	5/4	6/4	6/5	–	–	–	–
		2	Coat	–	–	–	–	–	5/6	5/6	6/6	–	–	–	–
		3	Interim check	–	–	–	–	–	–	–	–	5/0	5/0	–	–
		4	Heat-compress	–	–	–	6/4	–	–	–	–	–	–	–	–
		5	Seal obversely	–	9/2	10/4	–	–	–	–	–	–	–	–	–
		6	Seal reversely	–	5/2	4/2	–	–	–	–	–	–	–	–	–
		7	Final check	–	–	–	–	–	–	–	–	–	–	9/0	10/0

Table 7. Optimum solution to batch splitting for the realistic problem through *ALGRM*1

Job	Operation						
	1	2	3	4	5	6	7
J_1	120, 99, 381	600	600	600	177, 423	–	–
J_2	266, 79, 155	307, 193	371, 129	330, 170	115, 385	283, 217	–
J_3	855, 431, 514	944, 764, 92	1 323,4 77	674, 1 126	992, 808	930, 870	–
J_4	2 000	2 000	1 162, 838	2 000	947, 1 053	–	–
J_5	72, 264, 164	125, 187, 188	312, 188	500	143, 357	224, 276	426, 74

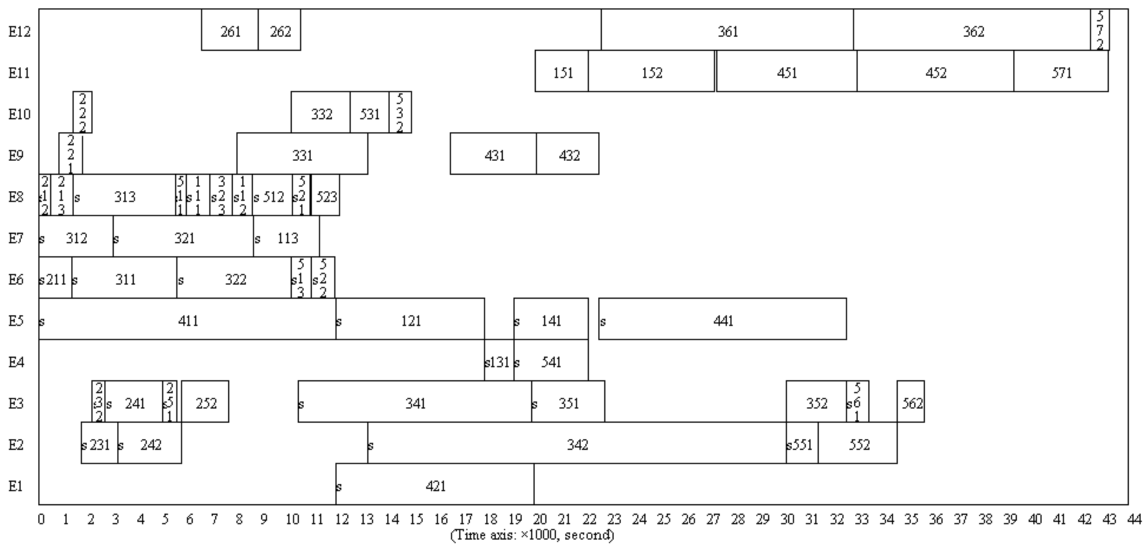

Fig. 11. Gantt chart for the realistic problem

5 Conclusions

(1) A variable-sized batch splitting scheduling problem model with alternative machines, on the basis of predefined batch numbers of sub-batches for each operation per job, is established.

(2) A new hybrid parallel algorithm is proposed to solve both the batch splitting problem and the batch scheduling problem, based on DE and genetic crossover operator. A problem-dependent local search procedure and an adaptive crossover operator are further designed for a better performance.

(3) The experiments of batch splitting job shop scheduling are performed, and the results confirm the validity of the problem model and the excellent performance of the proposed algorithm.

(4) Though the objective in this paper is to minimize the makespan, other objectives are also easily addressed by our approach.

(5) The proposed algorithm could also be used to solve batch splitting scheduling problems with bounded batch sizes by adjusting Eq. (11) and Eq. (12) to get batch sizes within bounds.

References

[1] MARTIN C H. A hybrid genetic algorithm/mathematical programming approach to the multi-family flowshop scheduling problem with lot streaming[J]. *Omega*, 2009, 37(1): 126–137.

[2] PAN Quanke, ZHU Jianying. Optimization method for a job-shop scheduling problem with alternative machines in the batch process[J]. *Chinese Journal of Mechanical Engineering*, 2004, 40(4): 36–39. (in Chinese)

[3] SUN Zhijun, AN Jin, HUANG Weiqing. Lot scheduling with multiple process routes in job shop[J]. *China Mechanical Engineering*, 2008, 19(2): 183–187. (in Chinese)

[4] LOW C, HSU C M, HUANG K I. Benefits of lot splitting in job-shop scheduling[J]. *International Journal of Advanced Manufacturing Technology*, 2004, 24(9–10): 773–780.

[5] STORN R, PRICE K. Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces[J]. *Journal of Global Optimization*, 1997, 11(4): 341–359.

[6] ONWUBOLU G, DAVENDRA D. Scheduling flow shops using differential evolution algorithm[J]. *European Journal of Operational Research*, 2006, 171(2): 674–692.

[7] QIAN Bin, WANG Ling, HUANG Dexian, et al. An effective hybrid DE-based algorithm for multi-objective flow shop scheduling with limited buffers[J]. *Computers & Operations Research*, 2009, 36(1): 209–233.

[8] NEARCHOU A C. A differential evolution approach for the common due date early/tardy job scheduling problem[J]. *Computers & Operations Research*, 2008, 35(4): 1329–1343.

[9] QIAN Bin, WANG Ling, HUANG Dexian *et al.* Scheduling multi-objective job shops using a memetic algorithm based on differential evolution[J]. *International Journal of Advanced Manufacturing Technology*, 2008, 35(9–10): 1 014–1 027.

[10] WANG Chengdong, ZHANG Youyun. Adaptive pseudo-parallel genetic algorithm based on real coding[J]. *Journal of Xi'an Jiaotong University*, 2003, 37(7): 707–710. (in Chinese)

[11] WANG Wanliang, WU Qidi. *Intelligence algorithms of production scheduling and application*[M]. Beijing: Science Press, 2007: 52–57. (in Chinese)

[12] AN Jin. *The job shop scheduling problem with alternative machine in the batch process*[D]. Nanjing: Nanjing University of Aeronautics & Astronautics, 2005. (in Chinese)

[13] LIN Nan, MENG Biao, FAN Yuqing. Hybrid genetic algorithm for multiple process and batch scheduling in job-shop[J]. *Journal of Beijing University of Aeronautics and Astronautics*, 2007, 33(12): 1 471–1 476. (in Chinese)

Biographical notes

ZHAO Yanwei, born in 1959, is currently an professor in Zhejiang University of Technology, China. She received her PhD degree from Shanghai University, China, in 2005. Her main research interests include digital design and manufacturing, scheduling.
Tel: +86-571-88320717; E-mail: zyw@zjut.edu.cn

WANG Haiyan, born in 1984, is currently a PhD candidate in Zhejiang University of Technology, China. Her research interests include differential evolution and production scheduling.
E-mail: haige102@163.com

WANG Wanliang, born in 1957, is currently a professor in Zhejiang University of Technology, China. He received his PhD degree from Tongji Universtiy, China, in 2001. His main research interests include automation and production scheduling.
E-mail: wangwanliang@zjut.edu.cn

XU Xinli, born in 1977, is currently an instructor in Zhejiang University of Technology, China. She received her PhD degree from Zhejiang University of Technology, China, in 2009. Her main research interests include computational intelligence and production scheduling.
E-mail: xxl@zjut.edu.cn