Available online at www.sciencedirect.com
ScienceDirect
European Journal of Operational Research 196 (2009) 78–92
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/ejor

Discrete Optimization

Using a family of critical ratio-based approaches to minimize the number of tardy jobs in the job shop with sequence dependent setup times

Tsung-Che Chiang [a,1], Li-Chen Fu [a,b,*]

[a] Department of Computer Science and Information Engineering, National Taiwan University, No. 1, Sec. 4, Roosevelt Road, Taipei, Taiwan, ROC
[b] Department of Electrical Engineering, National Taiwan University, No. 1, Sec. 4, Roosevelt Road, Taipei, Taiwan, ROC

Received 17 March 2005; accepted 5 December 2007
Available online 23 February 2008

Abstract

This paper addresses the job shop scheduling problem to minimize the number of tardy jobs, considering the sequence dependent setup time. This problem is taken as a sequencing problem, and a family of approaches with different levels of intricacy is proposed. The simplest form is a critical ratio-based dispatching rule, which leads to satisfactory solutions by taking into account the group information rather than only the individual information of jobs. Then, an enhanced approach consisting of an iterative schedule refining mechanism will be given. Its feature is to iteratively adjust the estimation of the remaining processing times of jobs in a dynamic and operation-specific manner. Finally, a genetic algorithm which takes the dispatching rule and the refining mechanism as the core is proposed. The performance of these approaches is carefully examined by a comprehensive experimental study.

Keywords: Scheduling; Job shop; Sequence dependent setup; Dispatching rules; Genetic algorithms

1. Introduction

Job shop scheduling is essentially a task of allocation of machines over time to the jobs in such a way that the selected performance criteria are optimized. For the shops running in the make-to-order fashion, the most important goal is to complete the customers' orders within the specified due dates. In this paper, we focus on minimizing the number of tardy jobs as the objective, reflecting the recent emphasis given to customer satisfaction in the industry (Balogun and Popplewell, 1999). Most previous research works assumed no setup times when solving the job shop scheduling problem. However, in practical applications like semiconductor manufacturing, the setup times are often not negligible and are sequence dependent. Hence, we consider sequence dependent setup (SDS) times in our problem. In the following, we give the mathematical formulation of the target problem.

$$\text{Minimize} \quad \sum_{i \in N} U_i,$$

* Corresponding author. Address: Department of Electrical Engineering, National Taiwan University, No. 1, Sec. 4, Roosevelt Road, Taipei, Taiwan, ROC. Tel.: +886 2 23622209; fax: +886 2 23657887.
E-mail addresses: tcchiang@ieee.org (T.-C. Chiang), lichen@ntu.edu.tw (L.-C. Fu).
[1] Tel.: +886 918891498; fax: +886 2 23657887.

0377-2217/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2007.12.042

Nomenclature

N, M	the set of jobs, the set of machines
n_i	the number of operations of job i
o_{ij}; o_i^*	the jth operation of job $i, i \in N, j \in \{1,\ldots,n_i\}$; the imminent operation of job i
o_{00}	the dummy operation, used to indicate the initial status of machines
O	the set of all operations (excluding o_{00})
m_{ij}	the machine required to process operation o_{ij}
p_{ij}; p_i	the processing time of operation o_{ij}; $p_i = p_i^*$
r_{ij}; r_i	sum of processing times of operation o_{ij} and its succeeding operations; $r_i = r_i^*$
s_{klij}	the required setup time to start operation o_{ij} right after processing of operation o_{kl}; $s_{00kl} = 0\ \forall k \in N$ and $l \in \{1,\ldots,n_k\}$
x_{klij}	the indicator for the processing order between operations o_{kl} and o_{ij}; $x_{klij} = 1$ iff operation o_{ij} is processed immediately after operation o_{kl}
c_{ij}; c_i	the completion time of operation o_{ij} ($c_{00} = 0$); $c_i = c_{ij}$ where $j = n_i$
d_i	the due date of job $i \in N$
U_i	the indicator for whether job i is tardy or not

subject to

$$\forall o_{ij} \in O, \quad x_{00ij} + \sum_{o_{kl} \in O, m_{kl} = m_{ij}} x_{klij} = 1, \tag{1}$$

$$\forall o_{ij} \in O, \quad c_{ij} \geqslant p_{ij} + \sum_{o_{kl} \in O \cup \{o_{00}\}} (x_{klij} \cdot (c_{kl} + s_{klij})), \tag{2}$$

$$\forall i \in N, \quad j \in \{2,\ldots,n_i\}, \quad c_{ij} \geqslant c_{i(j-1)} + p_{ij} + \sum_{o_{kl} \in O \cup \{o_{00}\}} (x_{klij} \cdot s_{klij}), \tag{3}$$

$$\forall o_{kl} \in O \cup \{o_{00}\}, \quad o_{ij} \in O, \quad x_{klij} \in \{0,1\}, \tag{4}$$

$$\forall o_{ij} \in O, \quad x_{ijij} = 0, \tag{5}$$

$$\forall o_{ij}, o_{kl} \in O \wedge m_{ij} \neq m_{kl}, \quad x_{klij} = 0, \tag{6}$$

$$\forall i \in N, U_i \in \{0,1\}, \quad \text{and} \quad U_i = 1 \text{ iff } c_i > d_i, \tag{7}$$

For each operation o_{ij}, constraint (1) is used to indicate its previously processed operation on machine m_{ij}. Constraint (2) expresses the capacity constraint that one machine can process only one job at a time. Constraint (3) represents the technological precedence constraint. Constraints (4)–(6) define the possible values of x_{klij}, and constraint (7) sets the values of U_i. If the setup times are negligible or are sequence independent, the model is still feasible just by setting $s_{klij} = 0\ \forall o_{kl}$, $o_{ij} \in O$ or by setting $s_{xyij} = s_{klij}\ \forall o_{xy}$, o_{ij}, o_{kl}, $\in O$, respectively.

Job scheduling is usually reduced to a sequencing problem, and the dispatching rule is a common solution in the industry. Lee et al. (1997) designed a three-phase heuristic to minimize the total weighted tardiness on a single machine in the presence of SDS. Kim et al. (2001) proposed several rules for scheduling in the wafer fabrication facilities to minimize mean tardiness. Chern and Liu (2003) proposed the family-based scheduling rules to address SDS occurred in the photolithography stage in the wafer fabrication system. Duwayri et al. (2006) developed a threshold-based rule to schedule the ion implanters, which are the bottleneck workstations with significant amount of setup time in the semiconductor manufacturing system. Many other dispatching rules can be found in the survey and simulation-based reports by Chang et al. (1996), Jayamohan and Rajendran (2000), and Chiang and Fu (2006). Besides the efforts to the development of efficient rules, there were also research works on rule combination (Dabbas and Fowler, 2003) and by-machine rule selection (Yang et al., 2007) to solve the job scheduling problems.

In addition to dispatching rules, several kinds of meta-heuristics, like genetic algorithms (GA), are also popular in the field of job scheduling. Chang et al. (2003) considered SDS in scheduling in a real-world BOPP film factory, which was modeled as a two-stage flowshop, to minimize the sum of setup times, earliness, and tardiness. Mattfeld and Bierwirth (2004) adopted the operation-based representation and utilized the parametric active schedule builder to reduce the search space of GA. Their GA was tested in the job shop with release and due dates, and outperformed several existing approaches with respect to three objectives, including weighted number of tardy jobs. To exploit the advantages of rules and GA, Bertel and Billaut (2004) used dispatching rules to initialize the population of GA, and showed the efficiency of their approach in a hybrid flow shop in terms of weighted number of tardy jobs. An example of further integration of rules

and GA can be seen in Chiang et al. (2007), where they applied the GA to seek for the good linear combination of multiple dispatching rules. By using dispatching rules to guide the search process of GA (instead of direct sequencing of jobs), we can prevent the size of search space from growing exponentially with the increase of number of jobs. This is helpful for solving large-scale job shop scheduling problems.

The GA models the natural evolutionary process, and the algorithm itself is also evolving. Taillard et al. (2001) pointed out that one important element in the evolution is the incorporation of local search procedures. Franca et al. (2001) incorporated the hill climbing algorithm into GA to solve the total tardiness single machine scheduling problem with SDS. Artiques and Roubellat (2002) addressed the problem of minimizing the maximum lateness in the job shop, and proposed an efficient neighborhood function for doing local search. The genetic local search was used by Sevaux and Dauzere-Peres (2003) to minimize the weighted number of tardy jobs for the single machine scheduling problem. Essafi et al. (2007) also proposed a genetic local search and showed its good performance on minimizing total weighted tardiness in the job shop. In Essafi et al.'s work, they mentioned that "Despite the increasing importance of customer service in terms of meeting due dates, research works dealing with the minimization of due date related objectives in the job shops are very scarce." Moreover, the number of existing works on due date scheduling in the job shop with SDS is even much fewer. There are no more than ten papers on this topic in the recent survey reports by Allahverdi et al. (2006) and Zhu and Whilhem (2006), which covered more than 300 papers published in 1999–2006. To our best knowledge, our work is the first one on minimizing the number of tardy jobs in the job shop with SDS.

In this work, we propose a family of approaches to solve the target problem. The first member is a dispatching rule that prioritizes jobs based on the critical ratio-based indices. The rule, named enhanced critical ratio (ECR), is featured by considering the influence on the urgency of competing jobs caused by processing of the picked job. As the remaining processing time is an important factor in the ECR rule, the second member equips the first member with a schedule refining mechanism for adjusting the remaining processing times to obtain improved schedules. With several parameters in the ECR rule and refining mechanism, we need an automatic parameter optimizing algorithm. This is achieved by our third member, which is a GA that takes the ECR rule as the basis of genome encoding and the refining mechanism as the local search procedure.

The rest of this paper is organized as follows: Section 2 gives a review of the ECR rule. Then, the iterative schedule refining mechanism is presented in Section 3. In Section 4, we detail the GA for parameter optimization. A comprehensive study of performance of the proposed approaches is shown in Section 5. Conclusions and future research directions are given in Section 6.

2. The ECR dispatching rule

As mentioned, many dispatching rules have been proposed in the literature. In our observation, most of them share a common feature – when applying these rules to do job scheduling, only information concerning each individual job is used. It can be expected that they can not provide good decisions with this small amount of information. In our solution, we proposed a new dispatching rule, the ECR rule, which takes into account the "group information" based on the critical ratio (CR) rule. Although some other rules, like the cost over time (COVERT) and apparent tardiness cost (ATC) rules, also consider the information of other jobs or machines, the ECR rule is distinguished from them in that it evaluates a job by measuring the influence upon all competing jobs caused by processing of the selected job. The detailed steps to apply the ECR rule were presented in Chiang and Fu (2004). Here we only show its equation to calculate the index value of a job i. The job with the smallest index value is selected as the next processing target.

Let job b denote the last job processed on the released machine, t denote time to select the next job, and Q denote the queue containing waiting jobs. The index value Z_i of a job i is calculated by

$$Z_i = \sum_{k \in Q, i \neq k} \mathrm{urg}(r_k, d_k - t - s_{b^*i^*} - p_i - s_{i^*k^*}) + \mathrm{urg}(r_i - p_i, d_i - t - s_{b^*i^*} - p_i), \quad \text{where}$$

$$\mathrm{urg}(r, a) = \begin{cases} (r/a)^2, & a \geqslant r > 0, \\ B + D \cdot (r - a), & a < r, \\ 0, & r = 0. \end{cases} \tag{8}$$

The ECR rule intends to pick a job as the next processing target such that the total urgency of all jobs is kept minimal after that job is processed. The urgency of a job is measured by a function of the critical ratio (ratio of the remaining processing time to the allowance time) of the job. When the critical ratio is close to one, the urgency should increase very fast. In our current implementation, we use the function $f(x) = x^2$. Let us use Fig. 1 to demonstrate how the ECR rule works. There are two candidate jobs to be processed. If job J1 is processed first, the total urgency of both jobs is still low ($Z_1 = (0.5)^2 + (0.5)^2 = 0.5$). However, if job J2 is processed first, the urgency of job J1 will increase drastically, and the total urgency is higher ($Z_2 = (0.8)^2 + (0.27)^2 = 0.71 > 0.5$). Therefore, ECR picks job J1 first in this situation.

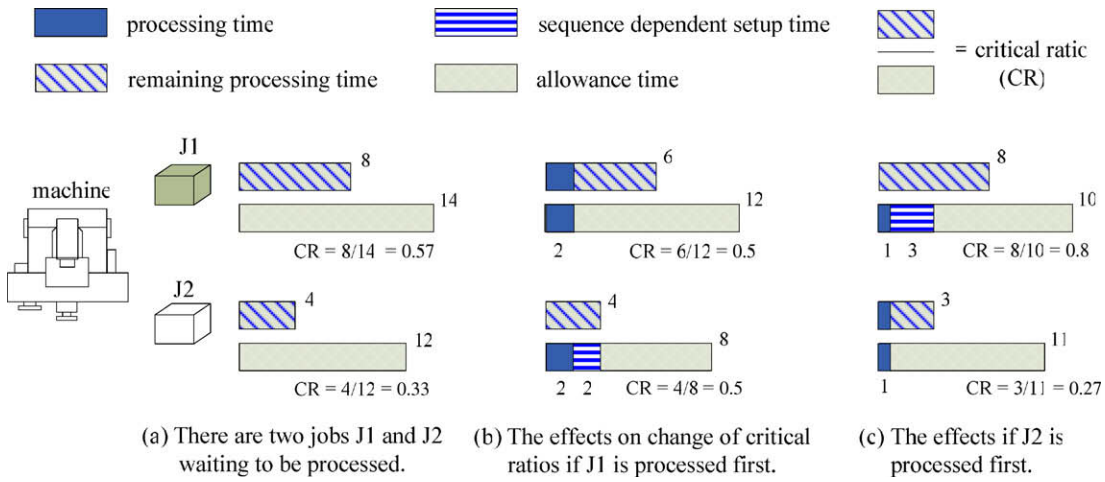

Fig. 1. Basic idea of the ECR rule.

The ECR rule performed well in preliminary experiments, and an improvement was proposed to raise both its solution quality and computation speed (Chiang and Fu, 2004). In our observations, the ECR rule might make an inappropriate decision when there are candidate jobs with very low or very high critical ratios. Thus, a filtering preprocess is added. Critical ratios of all candidate jobs are calculated, and only the jobs with critical ratios lying in a predefined interval $[L, U]$ are considered by ECR. This preprocess can eliminate the conditions in which the ECR might not behave well; meanwhile, the computation time is saved because the number of jobs to be prioritized is reduced. When there are no jobs with critical ratios falling into $[L, U]$, all jobs in the queue will be processed in the order following the shortest processing time first (SPT) rule. Two other parameters B and D are used in the ECR rule to make sure that the urgency of the tardy jobs will not decrease as the tardiness increases. To set the values of these parameters L, U, B, and D suitably, we develop a GA and present it in Section 4.

3. The iterative schedule refining mechanism

The remaining processing time is a factor commonly adopted in the design of dispatching rules. It is usually calculated as the sum of processing times of unfinished operations. However, this sum (hereafter we call it the minimal remaining processing time, $r^{\min}$) is only a lower bound and the actual remaining processing time (r^{act}) is usually longer than $r^{\min}$ because of resource contention in the shop. To cope with this problem, a common solution in the literature estimates the actual remaining processing time by multiplying an amplification ratio (A) to $r^{\min}$ excluding the processing time of current operation (p_c), i.e. $r^{\text{act}} = p_c + A \cdot (r^{\min} - p_c)$ where the value of A is determined empirically (Kim and Kim, 1994; Kim

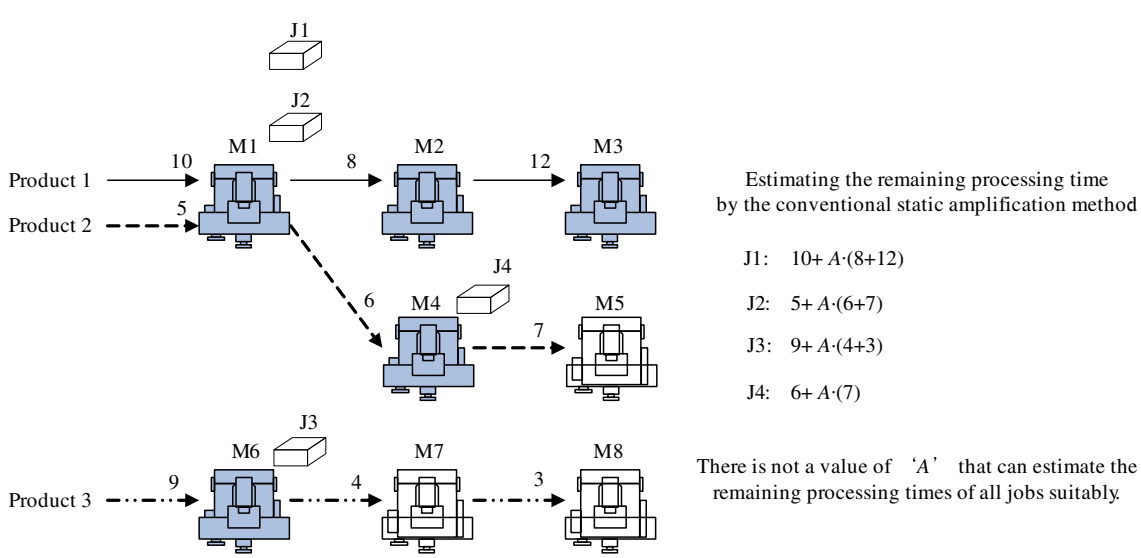

Fig. 2. An example for the drawbacks of the conventional method to estimate the remaining processing time.

et al., 2001). This kind of estimation is static since the amplification is made in a uniform way regardless of different degrees of contention encountered by different jobs at different stages.

To realize the drawbacks of the static amplification method, we take Fig. 2 as an example. In this shop, there are many kinds of products, and here only the routes of three of them are depicted. The number above the arrow is the processing time on the machine for each product. Severe resource contention takes place on the colored machines (M1–M4 and M6), and the degree of resource contention is assumed to be small for other machines. For the job of product 1, J1, the value of A should be large to reflect the fierce resource contention at machines M2 and M3. With a large amplification ratio and consequently longer estimated remaining processing time (than $r^{\min}$), the urgency of job J1 becomes higher, and that could instruct the ECR rule to sequence J1 in an earlier order. In other words, setting a large A value for J1 to reflect the long queueing time in its later stages intends to make it completed within due date with a greater chance. Nevertheless, for the job of product 3, J3, the value of A should be close to one due to little resource contention at machines M7 and M8. In this case, the static amplification method, which estimates r^{act} by amplifying $(r^{\min} - p_c)$ in a uniform way for all kinds of products, is believed to fail.

Even for jobs of the same product, the static amplification method could also have problems. Given two jobs of product 2, J2 at M1 and J4 at M4, the value of r^{act} of J2 should be significantly different from the value of $r^{\min}$, whereas the values of r^{act} and $r^{\min}$ of J4 should be almost the same. Again, a single value of the amplification ratio can not satisfy this situation. Besides, amplifying the processing time on M5 for J2 is unreasonable because there is little resource contention on M5.

After realizing the drawbacks of the static amplification method, we propose to use an iterative refining mechanism with two features: First, it considers the degree of contention of machines in a job-specific and operation-specific manner. The remaining processing time is estimated according to the actual queueing time of each job at each operation. Second, it refines the schedule iteratively based on the estimates of remaining processing times in each iteration.

The heuristic based on the ECR rule and the iterative refining mechanism is named I-ECR. Let q_{ij} denote the queueing time of operation o_{ij}, the entire procedure of I-ECR is given as follows:

Step 0. $t = 1, \quad r_{ij} = \sum_{k=j}^{n_i} p_{ik} = r_{ij}^{\min}.$ (9)

Step 1. Run a simulation and apply the ECR rule described in Section 2 to construct a schedule S_t.If $t \geqslant T$, the refining process ends, and the best schedule among all T schedules is reported.

Step 2. Calculate queueing times q_{ij} based on the schedule S_t obtained in Step 1.

Step 3. $r_{ij} = p_{ij} + \sum_{k=j+1}^{n_i} (p_{ik} + A \cdot q_{ik}) = r_{ij}^{\min} + A \cdot \sum_{k=j+1}^{n_i} q_{ik},$ (10)

$t = t + 1$. Goto Step 1.

In this approach, the increment of remaining processing time is determined according to the dynamic information obtained during the iterative refining process. The queueing time of each job at each operation is taken into account to adjust the remaining processing time. The original idea of this iterative refining mechanism was mentioned in Vepsalainen and Morton (1988). Taner et al. (2003) adopted this mechanism to minimize the maximum lateness in the job shop, and the authors revised the original mechanism to a parameterized version (Chiang and Fu, 2005). The iterative refining mechanism can be applied to any rule as long as the rule takes the remaining processing time to calculate the index values of jobs. For example, the COVERT and ATC rules are also good candidates. What we need to do is just to use the estimated remaining processing times when the remaining processing times are required by the cooperated dispatching rule. The values of parameters A and T certainly have influence on the performance of this mechanism. Related experiments and discussions are provided in Section 5.

4. Genetic algorithm with I-ECR

As mentioned in Section 2, there are four parameters L, U, B, and D in the ECR rule. Their default values are zero, infinity, two, and zero, respectively. The $[0, \infty]$ interval means that the ECR rule considers all jobs in the default condition. The default value of B, two, emphasizes the importance of avoiding tardy jobs (the maximal urgency of non-tardy job is one, see Eq. (8)), and the default value of D makes ECR concentrate on the jobs that are possible to be completed within due dates. Although the ECR rule with the default setting performs satisfactorily, the performance can be further improved by tuning these parameters to fit different problem instances.

In order to determine suitable values for these parameters, we resort to the GA, which is commonly adopted in parameter optimization (Goncalves et al., 2005; Chiang et al., 2007). To construct a GA, there are generally five primary components to be concerned – genome encoding, genome evaluation, genetic operators, initial population, and genetic parameters. Besides these five components, local search procedures are usually incorporated into modern GAs to improve the performance. The iterative schedule refining mechanism described in the previous section will serve this function. The

parameters A and T from the refining mechanism are then also taken as the tuning targets of the GA. Details of the proposed GA are given as below:

Genome encoding: Genomes represent the solutions to be sought, and our genome has six genes corresponding to parameters L, U, B, D, A, and T. The range of values of L, U, and D is $[0,1]$, and the range of value of B is $[1,2]$. The precision of these four parameters is up to the second digit. The range of value of A is $[0,1]$, with precision up to the first digit, and T is an integer in $[1,10]$.

Genome evaluation: To evaluate a genome, the I-ECR heuristic is conducted. Before applying the I-ECR heuristic, values of the parameters (L, U, B, and D) in the ECR rule and of the parameters (A and T) in the refining mechanism are set according to the values of corresponding genes. After the I-ECR stops, the number of tardy jobs of the best schedule among T generated schedules is recorded on the genome. During the evolutionary progress of GA, we observed that many genomes may have the same number of tardy jobs. In this condition, it is difficult to effectively qualify the genomes only by the number of tardy jobs. Therefore, we adopt the total tardiness, which is defined by $\sum_{i\in N}\max\{0, c_i - d_i\}$, as the secondary measure for qualification of genomes.

Genetic operators: To do mating selection, we use the 2-tournament selection operator. It randomly picks two genomes, and then selects the genome with a smaller number of tardy jobs as a parent. In case of a tie, the genome with a smaller total tardiness is selected. If there is still a tie, one genome is selected randomly. Two-point crossover and single-gene-substitution mutation are adopted for producing the offspring. Given two parents, the two-point crossover produces two offspring by randomly picking two points and then exchanging the sections enclosed by these two points between two selected parents. Then, the single-gene-substitution mutation is applied on each offspring with probability p_m (, which is a parameter of our GA). It randomly picks one gene and sets its value as a random value in the corresponding range. Each time when two offspring are produced, the best two genomes among the parents and offspring will replace the parents. In this way, the elitism strategy is realized implicitly. Besides, we introduce two population diversity control mechanisms in our GA. The first mechanism controls the diversity in the objective space. It requires that the two survivors from two parents and two offspring must have different number of tardy jobs or total tardiness, unless all four genomes have the same values. The second mechanism controls the diversity in the encoding space. This is achieved by replacing the worst $r_i\%$ genomes with random immigrants in each generation. (The values of genes of the immigrants are generated randomly within the corresponding ranges.) The idea of random immigration comes from the work by Goncalves et al. (2005), and its function is different from the mutation operator. The mutation operator makes a small modification to the genomes (that have already evolved during the GA process) and aims to produce better genomes by injecting some new material that is not easily obtained by doing crossover. On the other hand, the random immigration intends to do exploration on the search space by introducing totally new genomes into the population. In Section 5.4.1, we will provide experimental results about setting suitable values for the parameters p_m and r_i.

Initial population: Random creation is used, which means values of genes are generated randomly within their corresponding ranges.

Local search procedure: In contrast to the intention of GA to explore the solution space globally, the local search procedure usually aims at finding the optimal or near-optimal solution in a relatively smaller region, namely, the neighborhood of a given (base) solution. Searching in the neighborhood is usually achieved through generating similar solutions by making slight modifications to the base solution and then selecting the best one among them. The iterative refining mechanism is a good candidate to serve this function. It refines the schedule iteratively, which resembles searching a group of similar schedules. This local search process is conducted directly on the solution space, not on the coding space, and two consecutive search points are related by queueing times and estimated remaining processing times. Besides, different genomes can have different ways (parameters A and T) to do this local search. In other words, the local search procedure itself also evolves during the process. To give a global view of our proposed GA, we provide the flow chart in Fig. 3.

5. Experiments and results

5.1. Generation of problem instances

The problem instances used in our experiments are generated based on the public instances provided by Taillard (1993). We select 10 problem instances with 30 jobs and 15 machines (*ta*31–40) and 10 instances with 50 jobs and 15 machines (*ta*51–60). For each selected "*ta*" instance, we generate nine "*tan*" instances with different combinations of three levels of due date tightness factor (*FF*) and three levels of maximum ratio of setup time to mean processing time (*SS*). In total, there are $2 \cdot 9 = 18$ problem categories (defined by combinations of number of jobs, *FF*, and *SS*) and $2 \cdot 9 \cdot 10 = 180$ new instances. Parameter settings for these *tan* instances are summarized in Table 1. The instance *tan* $(10 \cdot k + i)$ is generated based on the instance *ta*$(30 + i)$, where $k = 0, \ldots, 8$, and $i = 1, \ldots, 10$. The instance *tan* $(10 \cdot l + i)$ is generated based on the

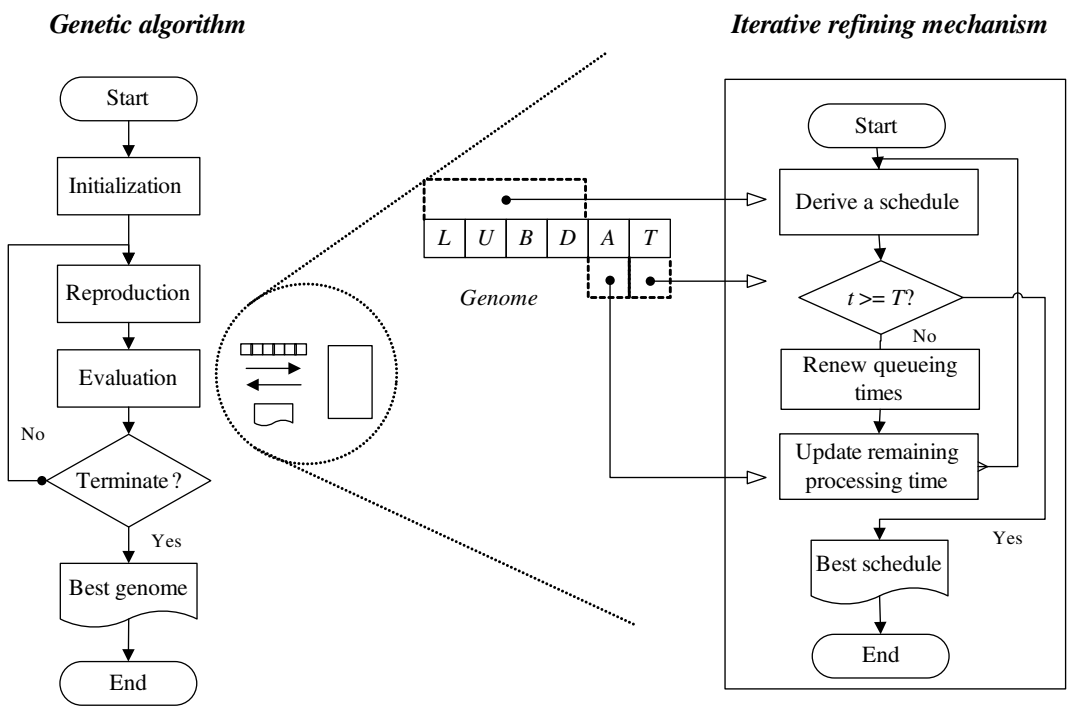

Fig. 3. The flow chart of proposed genetic algorithm.

Table 1
Parameter settings for generation of problem instances

	30 jobs, 15 machines			50 jobs, 15 machines		
	$FF = 1.7$	$FF = 1.9$	$FF = 2.1$	$FF = 2.6$	$FF = 2.8$	$FF = 3.0$
$SS = 0\%$	*tan* 1–10	*tan* 11–20	*tan* 21–30	*tan* 91–100	*tan* 101–110	*tan* 111–120
$SS = 25\%$	*tan* 31–40	*tan* 41–50	*tan* 51–60	*tan* 121–130	*tan* 131–140	*tan* 141–150
$SS = 50\%$	*tan* 61–70	*tan* 71–80	*tan* 81–90	*tan* 151–160	*tan* 161–170	*tan* 171–180

instance $ta(50 + i)$ where $l = 9, \ldots, 17$, and $i = 1, \ldots, 10$. The number of jobs, number of machines, required machines, and processing times in each *tan* instance are the same as they are in the based *ta* instance. The additional parts include due dates and SDS times. The SDS times in *tan* instances are generated by using the processing times in *ta* instances as the seeds. The due dates are generated by using the Total WorK content (TWK) method. The algorithm for generating the new instances is given in Table 2. Researchers who are interested in using these new problem instances can follow the algorithm to generate them. The electronic files of these instances are also available from the authors upon request.

In the experiments, all scheduling approaches are implemented in C++ language by Microsoft Visual C++ 6.0®. All experiments are conducted on personal computers with Intel 2 GHz CPU and 1 GB RAM.

5.2. Performance of ECR

5.2.1. Benchmarks

In order to test the performance of the ECR rule, we take 18 existing dispatching rules as benchmarks. Anderson and Nyirenda (1990) proposed rules CR + SPT and S/RPT + SPT, and showed that they are effective in minimizing the number of tardy jobs. Rules SPT, EDD, LTWK, SRPT, and SPT/TWK were the best five rules among 42 rules in terms of number of tardy jobs in Chang et al. (1996). Rules SLK/OPN, ATC, MDD, COVERT, and MOD were the best five rules for minimizing mean tardiness in Jeong and Kim's survey (1998). The abovementioned rules were also pervasively adopted as benchmarks in the literature (Jayamohan and Rajendran, 2000; Chiang and Fu, 2006). In addition to these well-known rules, four recently proposed rules are also taken into consideration in our experiments. The ATCS rule is an enhancement of ATC rule, and showed good performance to minimize the total weighted tardiness for single machine with SDS (Lee et al., 1997). Jayamohan and Rajendran (2000) proposed the PTPWODD rule, which performed well on minimizing the number of tardy jobs. The EADD rule was developed by Lodree et al. (2004) and showed good performance on minimizing the number of tardy jobs. Abu-suleiman et al. (2005) introduced the MCR rule, which yielded better performance than the

Table 2
Algorithm for generation of problem instances

```
N: number of jobs; M: number of machines
p_ij: processing time of operation j of job i in the ta instance
s[x]: variables for generating sequence dependent setup time
d_i: due date of job i in the tan instance
s_ijk: sequence dependent setup time from job i to job j on machine k in the tan instance
For i = 1,...,N Do
    For j = 1,...,M Do
      s[(i−1) · M + j] = p_ij
    End For
End For
s = 0
x = 1
For k = 1,...,M Do
    For i = 1,...,N Do
      For j = 1,...,N Do
        If i = j or SS = 0 Then
          s_ijk = 0
        Else
        s = s_ijk = (s[x] + s) Mod (SS · 100)
          x = x + 1
          If x > N · M Then
            x = 1
          End if
        End if
      End for
    End for
End for
For i = 1,...,N Do
    d_i = (FF + ((i − 1)Mod5) · 0.1) · (Σ_{j=1,...,M} p_ij + Σ_{k=1,...,M} (Σ_{j=1,...,N} s_jik/(N − 1)))
End for
```

Table 3a
Definitions of the benchmark rules

Rule	Index value $Z =$	Rule	Index value $Z =$
MCR	$(d-t)/r^z$	SPT	p
EDD	d	SRPT	r
MDD	$\max\{d,\ t+r\}$	LTWK	P
ODD	$d - c\cdot(r-p)$	SPT/TWK	p/P
MOD	$\max\{d - c\cdot(r-p),\ t+p\}$	SLACK	$d-r-t$
CR+SPT	$\max\{(d-t)/r\cdot p,\ p\}$	SLK/OPN	$(d-r-t)/o$
S/RPT+SPT	$\max\{(d-r-t)/r\cdot p,\ p\}$	COVERT	$(1/p)\cdot(1-(d-r-t)^+/(b\cdot k\cdot r))^+$
PTPWODD	$p+q+d-c\cdot(r-p)$	ATC	$(1/p)\cdot\exp(-(d-p-b\cdot(r-p)-t)^+/(k\cdot p_a))$
EADD	See Lodree Jr. *et al.* (2004)	ATCS	$(1/p)\cdot\exp(-(d-r-t)^+/(k_1\cdot p_a))\cdot\exp(-s/(k_2\cdot s_a))$
Notations			
p (P)	processing time of the imminent (all) operation(s)	p_a	average processing time of the competing operations
d	job due date	s	sequence dependent setup time
r	remaining processing time	t	system time
q	waiting time of the imminent operation	s_a	average sequence dependent setup time
o	number of unfinished operations	b, k	parameter of COVERT/ATC
c	parameter of ODD/MOD/PTPWODD	k_1, k_2	parameter of ATCS

For all rules except COVERT, ATC, and ATCS, the smaller the index value is, the higher the priority is.

Table 3b
Tested parameter values of the dispatching rules with parameters

Rule	Tested parameter values	Rule	Tested parameter values
MCR	$z = 0.2, 0.4, 0.6, \ldots, 2.0, 2.2$	COVERT	$b\cdot k = 1,2,3,\ldots,19,20$
ODD/MOD/PTPWODD	$c = 1,2,3,\ldots,9,10$	ATC	b and $k = 2,4,6,8,10$, respectively
		ATCS	k_1 and $k_2 = 1,2,\ldots,6$, respectively

original CR rule in terms of tardiness and earliness. For the rules with parameters, we test at least ten versions of each of them with different parameter values and then use the best results to do performance comparison. Here, the proposed ECR rule uses the default setting. The definitions of all tested rules are provided in Table 3a, and the tested parameter values are given in Table 3b.

5.2.2. Performance of each dispatching rule

For each of the 19 tested dispatching rules, the average numbers of tardy jobs over instances *tan* 1–90 (30 jobs and 15 machines) and *tan* 91–180 (50 jobs and 15 machines) are provided in Tables 4a and 4b, respectively. Besides, for each category of (10) instances with the same *FF* and *SS*, we assign a rank to each rule. The average rank of each rule over nine categories is shown in the parentheses. The rules are placed in increasing order of the average rank. (Due to the limitation of space, the average number of tardy jobs and rank of each rule in each category are omitted here.)

In Tables 4a and 4b, the proposed ECR rule and existing rules SRPT, MDD, LTWK, and EDD are always inside the group of best six rules. When the problem instances with 30 jobs are solved, some existing rules may outperform the ECR rule, especially when the job due dates are tight. When the instances with 50 jobs are solved, however, the ECR rule becomes the best performer, showing its advantage of dealing with large-scale problems.

In general, the rules perform stably, which means that for each rule its ranks in all problem categories are similar. Some special cases are reported as follows. The rules EADD and LTWK are particularly suitable for solving the instances with tight due dates. But they are not even in the group of best ten rules when solving the instances with loose due dates. On the contrary, the ATCS rule is ranked about the 10th place when solving the instances with tight or moderate due dates but can be the best or the second best rule when solving the instances with loose due dates. Besides, we also observe that the experience of using rules to minimize other due date-based performance measures like mean tardiness and maximum tardiness is not applicable to minimization of number of tardy jobs. Rules COVERT, ATC, and CR + SPT, which are recognized for minimizing mean tardiness, and SLACK, which is well known for minimizing maximum tardiness, provide bad performance. To choose one rule from the 19 tested rules, we recommend to use the proposed ECR rule when due dates are moderate or loose and to use the SRPT rule when due dates are tight.

5.3. Performance of I-ECR

In this subsection, we will examine the benefit of integrating the proposed ECR rule with the iterative refining mechanism. As mentioned, there are two parameters that control the execution of the refining process – A, which decides the percentage of queueing time being added to the predicted remaining processing time, and T, which decides the number of iterations to do the refining actions. In the experiments, we use $\{0.1, 0.2, \ldots, 1.0\}$ as the value of A and the value of T is 10. The minimal number of tardy jobs among the 100 ($10 \cdot 10$) schedules is recorded. To see how the iterative refining mechanism improves other rules, we also test all other rules except EDD and SPT since they do not consider the remaining processing time. Since the iterative refining mechanism is now responsible for estimating the remaining processing time, values of the parameters for estimating the remaining processing time in ODD/MOD/PTPWODD (c), COVERT (b and k), and ATC (b) are set as one in the experiment. A prefix "I-" is added before a rule name to denote the heuristic combining the rule and the iterative refining mechanism.

Table 4a
Average number of tardy jobs and average rank of each dispatching rule over problem instances *tan* 1–90

SRPT	MDD	EADD	ECR	EDD	LTWK	MCR	SPT	ATCS	ATC
14.1(3.6)	14.6(3.7)	13.9(4.1)	14.3(4.3)	14.9(4.9)	15.3(6.8)	16.0(7.7)	15.9(7.8)	15.7(8.4)	16.6(9.4)
COVERT	PTPW+	SPT/T	S/RPT+	MOD	ODD	SLACK	CR + SPT	SLK/OP	
16.8(10.4)	16.9(10.6)	17.2(12.1)	18.6(12.7)	17.9(13.9)	19.4(15.3)	20.3(16.3)	21.7(17.2)	25.2(19)	

Table 4b
Average number of tardy jobs average rank of each dispatching rule over problem instances *tan* 91–180

ECR	SRPT	MDD	MCR	LTWK	EDD	ATCS	EADD	ATC	SPT
18.4(2.3)	21.0(3)	21.8(3.7)	22.2(4.9)	22.5(5.3)	22.7(5.7)	21.4(8.7)	24.1(8.2)	24.5(8.7)	24.7(9)
PTPW+	COVERT	SPT/T	MOD	ODD	S/RPT+	SLACK	CR + SPT	SLK/OP	
25.8(11)	26.6(11.6)	27.1(12.6)	29.4(13.9)	29.8(14.9)	34.5(15.3)	31.8(16.4)	39.1(17.9)	44.3(19)	

Table 5a
Average number of tardy jobs average rank of each I-heuristic over problem instances *tan* 1–90

I-ECR	I-EADD	I-MDD	I-SRPT	I-LTWK	I-ATCS	I-MCR	I-SPT/T	I-PTPW+
7.7 (1.1)	9.5 (2.2)	10.7 (4.4)	10.7 (5.1)	11.2 (6.2)	12.2 (6.4)	12.1 (6.6)	12.3 (8)	13.2 (7.6)
I-ATC	I- S/RPT+	I-COVERT	I-CR + SPT	I-MOD	I-ODD	I-SLACK	I-SLK/OP	
13.4 (8.3)	13.7 (9.6)	16.5 (13.2)	16.5 (13.2)	17.1 (13.3)	18.0 (14.6)	18.7 (15.3)	22.2 (16.9)	

Table 5b
Average number of tardy jobs average rank of each I-heuristic over problem instances *tan* 91–180

I-ECR	I-EADD	I-MDD	I-SRPT	I-SPT/T	I-ATCS	I-LTWK	I-MCR	I-PTPW+
10.6 (1)	14.6 (2.6)	16.1 (3.3)	17.1 (4.4)	18.4 (6)	17.9 (6.2)	18.4 (6.4)	19.6 (8)	20.7 (8.7)
I-ATC	I- S/RPT+	I-MOD	I-ODD	I-SLACK	I-CR + SPT	I-COVERT	I-SLK/OP	
22.1 (9.4)	22.5 (9.6)	28.3 (12.4)	29.0 (13.6)	31.3 (14.7)	32.2 (14.7)	32.4 (14.7)	42.0 (17)	

5.3.1. Performance of each I-heuristic

Like the results summarized in Tables 4a and 4b, the average number of tardy jobs and average rank of each tested I-heuristic are provided in Tables 5a and 5b, respectively. The I-heuristics are placed in increasing order of the average rank. In both tables, the I-ECR heuristic is the best performer, followed by I-EADD, I-MDD, and I-SRPT. The result is not surprising since ECR, MDD, and SRPT themselves are good rules, which has been observed in Section 5.2.2. Applying the iterative refining mechanism to obtain more accurate remaining processing times makes their superiority more obvious. The improvement is particularly significant for ECR and EADD so that they become the best two among the 17 I-heuristics.

Generally, ranks of these I-heuristics are similar to ranks of their counterparts without applying the iterative refining mechanism. But now, the performance difference between good and bad approaches is larger. The ratios of average number of tardy jobs of the 1st performer to that of the 10th performer are 85% (14.1/16.9) and 74% (18.4/24.7) in Tables 4a and 4b, respectively. After applying the refining mechanism, the ratios of average number of tardy jobs of the 1st performer to that of the 9th performer are 58% (7.7/13.2) and 51% (10.6/20.7) in Tables 5a and 5b, respectively. It reveals that good rules are able to utilize the accurate information more effectively than the bad rules. It is worthy to note that the proposed ECR rule receives the largest performance improvement (more than 40%) among all tested rules.

Table 6a
Comparison of improvement by iterative refining and static amplification over problem instances*tan* 1–90

	SPT/TWK	LTWK	SRPT	MCR	ECR	PTPW+	EADD	S/RPT+	SLACK
IR	28.6% (89)	26.5% (86)	23.8% (88)	24.6% (89)	46.2% (90)	22.3% (80)	31.7% (84)	26.5% (79)	8.0% (55)
SA	0% (0)	0% (0)	13.0% (71)	15.1% (79)	37.1% (88)	14.5% (59)	24.9% (84)	21.9% (73)	4.4% (38)
	MDD	ODD	ATCS	ATC	COVERT	CR + SPT	SLK/OP	MOD	
IR	27.0% (87)	7.1% (56)	22.6% (84)	19.3% (79)	1.9% (45)	24.1% (89)	11.8% (81)	4.7% (44)	
SA	23.0% (86)	4.3% (38)	20.9% (74)	19.2% (84)	6.2% (54)	28.4% (90)	19.8% (85)	16.4% (78)	

Table 6b
Comparison of improvement by iterative refining and static amplification over problem instances *tan* 91–180

	SPT/TWK	LTWK	EADD	ECR	SRPT	PTPW+	MDD	ODD	SLACK
IR	31.9% (90)	18.2% (88)	39.5% (89)	42.2% (89)	18.4% (87)	19.8% (83)	26.3% (90)	2.8% (34)	1.8% (22)
SA	0% (0)	0% (0)	28.4% (90)	31.5% (87)	11.9% (80)	15.1% (73)	23.3% (90)	1.7% (17)	0.7% (13)
	S/RPT+	ATCS	SLK/OP	MCR	ATC	MOD	CR + SPT	COVERT	
IR	34.9% (90)	16.0% (73)	5.2% (72)	12.0% (82)	9.7% (64)	3.7% (40)	17.6% (86)	−21.6% (2)	
SA	35.4% (90)	18.6% (76)	8.8% (83)	16.1% (85)	16.0% (86)	14.2% (72)	34.6% (90)	−1.1% (39)	

5.3.2. Comparison of two refining mechanisms

In this section, we want to compare the iterative refining mechanism and the static amplification mechanism (mentioned in Section 3). The latter is applied on the same 17 rules in the previous subsection. The value of the amplification rate A ranges from 1 to 10 with granularity 0.1. The number of schedules generated by the static amplification mechanism, 100, is intentionally made the same as the number of schedules generated by the iterative refining mechanism in the previous experiment. Values of improvement percentage of each rule by both mechanisms are shown in Tables 6a and 6b. Besides, the number of instances whose solutions are improved is given in the parentheses. Here we let "IR" denote the iterative refining mechanism and "SA" denote the static amplification mechanism. The rules are placed in decreasing order of the improvement percentage by the IR mechanism minus that by the SA mechanism. A prefix "S-" is added before a rule name to denote the heuristic combining the rule and the static amplification mechanism.

In both tables, we first see that the SA mechanism does no improvement on the LTWK and SPT/TWK rules since static amplification of operation processing times does not change the processing sequences of jobs derived by these two rules. In contrast to the SA mechanism, the IR mechanism provides at least 18.2% and 28.6% improvement on average to LTWK and SPT/TWK, respectively. Besides LTWK and SPT/TWK, rules EADD, ECR, and SRPT also prefer the IR mechanism. Rules MOD, CR + SPT, and COVERT, on the other hand, prefer the SA mechanism. As for other rules, the IR and SA mechanisms produce approximately equal benefits. The possible reason for the preference of the SA mechanism is that it amplifies the remaining processing times substantially (up to 10 times), and in that condition the rules MOD, CR + SPT, and COVERT will behave like SPT (see Table 3a for their definitions), whose performance is much better than theirs. We also notice that rules which tend to favor the jobs with longer remaining processing times receive less improvement from the IR mechanism. The possible reason is as follows. During execution of the IR mechanism, a tardy job in the current iteration will have longer remaining processing time in the next iteration. For those rules which favor the jobs with longer remaining processing times, giving preference to these tardy jobs could lost more on-time jobs and then initiates a "vicious cycle." Note that the proposed ECR rule also favors the jobs with longer remaining processing times; however, the aforementioned problem is eliminated by careful consideration of the influence of processing of a job upon the total urgency of competing jobs.

Among the 34 heuristics formed by the combinations of 17 rules and two refining mechanisms, the best three heuristics are I-ECR, S-ECR, and I-EADD. Once again, the advantage of the proposed ECR rule and the integration with the iterative refining mechanism is verified.

5.4. Performance of GA with I-ECR

5.4.1. Benchmark algorithms and parameter settings

The GA is well known for its global exploration ability, but sometimes its performance is not satisfactory due to lack of ability to do local exploitation. Incorporating local search procedures and/or domain-specific knowledge to GA is becoming a recognized way to enhance its performance. In fact, the GA-I-ECR approach in this work is developed following this research trend. In this section, we will examine its performance by comparing with three recently proposed approaches, which are also realizations of integration of GA and domain-specific knowledge.

Mattfeld and Bierwirth (2004) used the GA to do job shop scheduling considering tardiness objectives. They found that the GA can not explore the large space of active schedules very well. Thus, they introduced a look-ahead parameter (δ) into the schedule builder in their GA so that the size of search space can be controlled by the value of δ. (The size of search space is between the number of non-delay schedules and the number of active schedules.) Their GA with tunable schedule builder showed good performance on several performance measures including the weighted number of tardy jobs. Later, Essafi et al. (2007) improved the GA by Mattfeld and Bierwirth by incorporating a local search procedure. In their local search procedure, the neighborhood structure is based on the disjunctive graph representation and the critical path. The neighboring solution is generated by swapping the operations belonging to the critical paths. There are two phases in their local search procedure. In the improving phase, the candidate operations to be swapped include only the operations of tardy jobs. In the perturbation phase, operations of all jobs are considered. In Yang et al. (2007), another form of combination of GA and domain knowledge was used. Like our approach, their GA did not search for a processing sequence of jobs directly. It aimed to choose a suitable dispatching rule for each machine in the shop, and the chosen dispatching rule is responsible for arranging the sequence of jobs requiring the machine By combining different dispatching rules, their approach showed up to 61% improvement percentage to the single rule. The above three approaches are taken into account as the benchmark algorithms in our experiment. In the following, we let MBGA, EMDGA, and YKCGA denote the GA proposed by Mattfeld and Bierwirth, Essafi et al., and Yang et al., respectively. To observe the effect of the iterative refining mechanism, we test two versions of our GA, with and without the iterative refining mechanism. They are denoted by GA-I-ECR and GA-ECR.

To conduct a fair comparison of these five GAs, we do parameter tuning for each of them. To do the tests, we randomly pick one instance from each problem category and form a small testing data set. Each GA is applied to each instance for ten times. Each run of GA-I-ECR and GA-ECR is allocated 100 seconds. As for the other three benchmark GAs, we

Table 7
Settings of five tested genetic algorithms

	GA-I-ECR		GA-ECR		YKCGA		MBGA		EMDGA	
	Components									
Encoding	I-ECR parameters (L, U, B, D, A, T)		ECR parameters (L, U, B, D)		Rule index		Preference list of operations		Preference list of operations	
Crossover	2-Point crossover		2-Point crossover		2-Point crossover		PPX		PMX	
Mutation	Single-gene-substitution		Single-gene-substitution		Swap		Insertion		–	
Domain knowledge	ECR rule iterative refining		ECR rule		Traditional rules		Parameterized schedule builder		Critical path-based local search	
	Best parameter values after tuning									
Instances(*tan*)	1–90	91–180	1–90	91–180	1–90	91–180	1–90	91–180	1–90	91–180
POP_SIZE	60	60	300	300	300	300	200	300	30	20
p_c	1	1	1	1	0.6	0.6	0.6	0.6	0.9	0.9
p_m	0.1	0.2	0.1	0.1	0.3	0.3	0.01	0.01	0	0
Self parameters	$r_i = 10$	$r_i = 10$	$r_i = 10$	$r_i = 10$	–	–	$\delta = 0.6$	$\delta = 0.8$	$n_{sd} = 20$	$n_{sd} = 5$

deliberately allocate four times running time, namely 400 seconds, so that the possible inefficiency due to our re-implementation of them can be compensated. With the limitation of space, here we only provide the best parameter values of the five GAs and the short summary of them in Table 7.

5.4.2. Performance of five GAs

After tuning, the GAs are used to solve the 180 *tan* instances, also for ten times. The computation time allocated for one run is the same as in the tuning phase. Let T_{ij} denote the number of tardy jobs of problem instance i in run j, we calculate $T^{\min} = \frac{1}{90}(\sum_{i=c+1}^{c+90} \min_{j=1,\ldots,10}\{T_{ij}\})$, $T^{\text{avg}} = \frac{1}{90\cdot 10}(\sum_{i=c+1}^{c+90}\sum_{j=1}^{10} T_{ij})$, and $T^{\max} = \frac{1}{90}(\sum_{i=c+1}^{c+90} \max_{j=1,\ldots,10}\{T_{ij}\})$ for each GA. The value of c is 0 and 90 for instances *tan* 1–90 and *tan* 91–180, respectively. We use these three measures to assess the performance of the tested GAs in the best, average, and worst cases. For each GA, we also count the number of instances for which it finds the best solution (n^{best}) and the number of instances for which only it finds the best solution (o^{best}). The experimental results are summarized in Table 8.

When instances *tan* 1–90 are solved, the proposed GA-I-ECR and the two benchmarks YKCGA and EMDGA, are the best three among the five tested GAs. They have close performance in the best case, and GA-I-ECR outperforms the other two in terms of the average-case and worst-case performance. In the best case, the GA can provide as large as 60% improvement percentage to the best dispatching rule. Each of the best three performers finds the best solution for at least one-third of the 90 instances, and each of them finds the best solution for at least 13 instances that no other can do it. This result reveals that significant performance improvement can be obtained by integrating GA with different forms of domain knowledge. By comparing the performance of GA-I-ECR and GA-ECR, the benefit of accurate estimation of remaining processing times by the iterative refining mechanism is easily verified.

With the increase of number of jobs and intuitively the problem complexity, the performance difference between tested GAs becomes larger when problem instances *tan* 91–180 are solved. The GA-I-ECR and YKCGA, which search for good schedules through manipulating dispatching rules, obviously defeat the EMDGA and MBGA, which search for good schedules through directly arranging the sequence of jobs. Although EMDGA and MBGA can provide at least 30% improvement to the best dispatching rule, they are still worse than the proposed I-ECR heuristic. This result shows the weakness of GAs that directly sequence the jobs when facing large-scale problems (and the accompanying huge search space). For the best two performers, the proposed GA-I-ECR is better than YKCGA in terms of all five measures ($T^{\min}$, T^{avg}, $T^{\max}$, o^{best}, and n^{best}). It demonstrates the potential of proposed ECR rule to beat a group of existing rules and also notifies us the importance of developing sophisticated rules besides manipulating the combinations of simple rules. The

Table 8
Performance of five genetic algorithms over problem instances *tan* 1–90 and *tan* 91–180

	GA-I-ECR	GA-ECR	YKCGA	MBGA	EMDGA
tan 1–90					
$T^{\min}\|T^{\text{avg}}\|T^{\max}$	5.13|5.58|5.94	9.40|9.42|9.44	5.07|5.76|6.47	7.02|8.33|9.53	5.76|6.59|7.52
$o^{\text{best}}/n^{\text{best}}$	16/51	0/3	15/56	1/12	13/39
tan 91–180					
$T^{\min}\|T^{\text{avg}}\|T^{\max}$	7.49|8.06|8.69	13.44|13.45|13.46	8.02|8.96|9.97	12.69|13.84|14.98	11.38|12.30|13.24
$o^{\text{best}}/n^{\text{best}}$	41/66	0/1	19/44	0/1	5/9

Table 9
Best known solutions of problem instances *tan* 1–180

tan 1–30			*tan* 31–60			*tan* 61–90			*tan* 91–120			*tan* 121–150			*tan* 151–180		
10^{3}	5^{13}	1^{35}	9^{1}	5^{13}	0^{35}	7^{5}	2^{5}	0^{135}	13^{1}	8^{1}	3^{3}	13^{1}	8^{1}	4^{1}	9^{1}	5^{1}	0^{13}
9^{3}	4^{35}	0^{135}	10^{13}	5^{15}	0^{35}	8^{1}	2^{5}	0^{1345}	14^{1}	11^{13}	5^{3}	14^{1}	10^{1}	5^{13}	11^{1}	7^{3}	0^{5}
11^{13}	6^{3}	2^{3}	10^{1}	6^{13}	1^{135}	9^{13}	4^{1}	0^{135}	12^{3}	8^{13}	3^{3}	11^{1}	7^{1}	3^{135}	9^{1}	3^{5}	0^{135}
11^{13}	6^{3}	3^{134}	10^{1}	5^{5}	0^{4}	9^{13}	2^{5}	0^{*}	13^{13}	9^{3}	5^{3}	12^{1}	9^{13}	5^{13}	9^{1}	4^{1}	0^{1}
10^{13}	6^{13}	2^{1345}	9^{3}	4^{5}	0^{35}	7^{5}	3^{5}	0^{135}	13^{1}	9^{13}	4^{3}	12^{1}	8^{1}	4^{13}	10^{1}	4^{1}	0^{5}
10^{1}	5^{35}	1^{3}	9^{13}	4^{5}	0^{5}	8^{1}	4^{14}	0^{1345}	14^{13}	9^{3}	5^{3}	13^{3}	9^{3}	5^{3}	10^{3}	6^{1}	1^{1}
9^{1}	4^{3}	0^{3}	9^{135}	3^{3}	0^{35}	7^{13}	1^{5}	0^{1345}	12^{3}	9^{13}	5^{13}	12^{3}	8^{1}	3^{1}	11^{13}	5^{1}	0^{3}
10^{1}	6^{3}	1^{3}	9^{1}	5^{3}	0^{3}	8^{1}	4^{13}	0^{1345}	13^{1}	9^{13}	5^{13}	13^{1}	8^{1}	4^{1}	10^{1}	6^{1}	1^{13}
10^{13}	6^{1}	3^{1}	9^{1}	6^{13}	2^{5}	8^{1}	5^{1}	0^{35}	13^{13}	8^{3}	4^{13}	12^{1}	7^{1}	3^{13}	9^{1}	1^{5}	0^{135}
9^{13}	4^{3}	0^{345}	9^{15}	4^{13}	0^{1235}	6^{5}	0^{45}	0^{*}	12^{1}	8^{1}	3^{3}	12^{13}	8^{13}	1^{5}	9^{1}	4^{1}	0^{*}

Meaning of subscript: 1 – GA-I-ECR, 2 – GA-ECR, 3 – YKCGA, 4 – MBGA, 5 – EMDGA, * – all five GAs.

best solutions found by the five tested GAs and the GAs that can find them are summarized in Table 9, which intends to identify the problem difficulty and to facilitate the convenience of performance comparison for other researchers.

5.4.3. Parameter settings of I-ECR and ECR

The mission of proposed GA is to find suitable parameter values for I-ECR and ECR. In this section, we analyze the best genomes found by the proposed GA and provide the results as a reference for the researchers who are interested in using our I-ECR and ECR. First, in each problem category, we collect the best 100 genomes obtained by GA-I-ECR and GA-ECR for 10 problem instances in 10 runs. Then, we calculate the mean value of each parameter (gene) over the collected genomes. Due to the limitation of space, we provide the range of mean value of each parameter over categories with different *FF* and *SS* in Table 10 and the mean values in each category are omitted here. The observations are presented in the following.

First, the value of L gets smaller and the value of U gets larger when the problem instances with larger number of jobs are solved. With the increase of number of jobs, it is expectable that the variation of critical ratios over jobs also becomes larger. To take account of enough jobs so that a good processing sequence can be determined, the ECR rule needs to use a larger $[L, U]$ interval, which implies the decrease of L and increase of U. Second, the value of U is smaller when ECR is applied without the iterative refining mechanism. Recall that in ECR the jobs whose critical ratios are greater than U are not considered, and this filtering procedure intends to disregard the jobs that are expected to be tardy. When the ECR rule is applied alone, the remaining processing time is a lower bound of the actual remaining processing time. In this condition, the critical ratios of jobs are smaller than what they are when the iterative refining mechanism is used. Therefore, the jobs that are likely to be tardy must be identified by a smaller U value. Third, setting the values of D, A, and T as 0, 0.9, and 10 seems reasonable based on the results in Table 10.

5.5. Use of ECR, I-ECR and GA-I-ECR

In Section 5.2, we compared the proposed ECR rule with 18 existing dispatching rules. In general, the ECR rule provides very good performance. Particularly, it is the best rule for solving the problem instances with moderate or loose due dates. In other words, if the initial allowance time is enough to complete the jobs within due dates, the ECR rule can utilize the time more effectively than the other rules. When the due date is tight, we recommend rules EADD and SRPT. (But note that EADD is very time-consuming due to the way in which it estimates the remaining processing time.) Enhancing ECR with the iterative refining mechanism, the I-ECR heuristic is the best performer in 17 of the 18 problem categories among all 34 tested heuristics, as we presented in Section 5.3. As for the exception (the category of instances *tan* 1–10), I-EADD is

Table 10
Ranges of mean values of parameters in ECR and I-ECR

	L	*U*	*B*	*D*		
ECR						
tan 1–90	[0.17, 0.42]	[0.44, 0.85]	[1.10, 1.69]	[0, 0.16]		
tan 91–180	[0.11, 0.26]	[0.53, 0.85]	[1.15, 1.53]	[0, 0.03]		
I-ECR						
	L	*U*	*B*	*D*	*A*	*T*
tan 1–90	[0.17, 0.37]	[0.81, 0.95]	[1.37, 1.97]	[0, 0.12]	[0.78, 0.99]	[9.18, 9.83]
tan 91–180	[0.09, 0.19]	[0.78, 0.96]	[1.18, 1.61]	[0, 0.02]	[0.80, 0.98]	[9.31, 9.52]

the only one that outperforms I-ECR. It shows that with the accurate information of remaining processing times, the ECR rule can beat most existing rules, no matter how tight the due dates are. In Section 5.4, the proposed GA-I-ECR is compared with three existing GAs. When the problem instances with 30 jobs are solved, two existing GAs are comparable to our GA-I-ECR. But when the problem instances with 50 jobs are solved, the GA-I-ECR outperforms all benchmark GAs. This result verifies the superiority of GA-I-ECR to solve large-scale problem instances.

To position each member in the proposed family of approaches, we have the following suggestions: Use the ECR rule when there is a request for a real-time or an interactive scheduler. The experimental results show that on average the ECR rule can reduce the number of tardy jobs by 15–25% when comparing with the 10th best rule among 19 tested rules. Another important advantage is that it requires no more information than the existing rules. In other words, when some rules are already built in the operation control system, ECR can also be implemented easily with little cost. If a simulation tool is available or the implementation cost of the simulation tool is affordable, the I-ECR is highly recommendable. By feeding the information of remaining processing times from the simulation tool to the ECR rule, the I-ECR is able to improve the performance of ECR by more than 40%. Finally, the performance of I-ECR can be further enhanced by optimizing its parameters through the proposed GA. The cost to implement the GA and the computation time to do optimization is justified by the 30% improvement to the performance of I-ECR.

6. Conclusions

In the literature, there are few research works on scheduling in the job shops in presence of SDS considering due date-based objectives. To our best knowledge, this is the first work on minimizing the number of tardy jobs in the job shops with SDS. The target problem is solved by a family of critical ratio-based approaches. Three main components include a dispatching rule, an iterative schedule refining mechanism, and a GA-based parameter tuning algorithm. Performance of the proposed approaches is shown to be superior to the existing ones through comprehensive experiments. The careful use of critical ratio values and its key factor, remaining processing time, contributes to the good performance of these approaches. In practical use, users can select one of the proposed approaches according to the cost of implementation, desired performance, allowable computation time, and so on. With the gradually-enhanced relationship, users can implement, test, and apply the proposed approach from the simplest form to the most sophisticated form. A considerable benefit from the proposed scheduler will be obtained in the early construction phase. With some moderate extension cost, the performance gain is significant.

We will continue the research in two directions. First, more objectives will be considered. In this work, we develop the approach specifically to minimize the number of tardy jobs. However, there are other important due date-based objectives such as mean tardiness. Some design of our approach, like filtering out the jobs with very high critical ratios before applying ECR, might not be suitable to those objectives. Thus, the performance of our proposed approach on these objectives should be examined, and that could inspire further improvement on our approach. With multiple objectives being regarded, developing a multi-objective GA will also be a natural extension. Second, the local search procedure in our GA could be executed more carefully. In our current implementation, the local search procedure is applied on each genome. Since in the proposed GA this procedure is the step that requires the largest computation effort, it may be a good idea to do local search only on a portion of the population. Striking a balance between global and local search will also be a topic of our future works.

Acknowledgment

This research is sponsored by the National Science Council under Research Grant No. NSC 96-2752-E-002-007-PAE.

References

Abu-suleiman, A., Pratt, D.B., Boardman, B., 2005. The modified critical ratio: Towards sequencing with a continuous decision domain. International Journal of Production Research 43, 3287–3296.

Allahverdi, A., Ng, C.T., Cheng, T.C.E., Kovalyov, M.Y., 2006. A survey of scheduling problems with setup times or costs. European Journal of Operational Research 187, 985–1032.

Anderson, E.J., Nyirenda, J.C., 1990. Two new rules to minimize tardiness in a job shop. International Journal of Production Research 28, 2277–2292.

Artiques, C., Roubellat, O., 2002. An efficient algorithm for operation insertion in a multi-resource job-shop schedule with sequence-dependent setup times. Production Planning and Control 13, 175–186.

Balogun, O.O., Popplewell, K., 1999. Towards the integration of flexible manufacturing system scheduling. International Journal of Production Research 37, 3399–3428.

Bertel, S., Billaut, J.-C., 2004. A genetic algorithm for an industrial multiprocessor flow shop scheduling problem with recirculation. European Journal of Operational Research 159, 651–662.

Chang, Y.-L., Sueyoshi, T., Sullivan, R.S., 1996. Ranking dispatching rules by data envelopment analysis in a job shop environment. IIE Transactions 28, 631–642.

Chang, P.-C., Hsieh, J.-C., Wang, Y.-W., 2003. Genetic algorithms applied in BOPP film scheduling problems: Minimizing total absolute deviation and setup times. Applied Soft Computing 3, 139–148.

Chern, C.-C., Liu, Y.-L., 2003. Family-based scheduling rules of a sequence-dependent wafer fabrication system. IEEE Transactions on Semiconductor Manufacturing 16, 15–25.

Chiang, T.-C., Fu, L.-C., 2004. Solving the FMS scheduling problem by critical ratio-based heuristics and the genetic algorithm. In: Proc. of IEEE International Conference on Robotics and Automation, pp. 3131–3136.

Chiang, T.-C., Fu, L-.C., 2005. An iterative refining mechanism for general job shop scheduling problems. In: Proc. of IEEE International Conference on Automation Science and Engineering, pp. 203–208.

Chiang, T.-C., Fu, L.-C., 2006. A simulation study on dispatching rules in semiconductor wafer fabrication facilities with due date-based objective. In: Proc. of IEEE International Conference on Systems, Man, and Cybernetics, pp. 4660–4665.

Chiang, T.-C., Shen, Y.-S., Fu, L.-C., 2007. A new paradigm for rule-based scheduling in the wafer probe center. International Journal of Production Research, in press, 10.1080/00207540601137199.

Dabbas, R.M., Fowler, J.W., 2003. A new scheduling approach using combined dispatching criteria in wafer fabs. IEEE Transactions on Semiconductor Manufacturing 16, 501–510.

Duwayri, Z., Mollaghasemi, M., Nazzal, D., Rabadi, G., 2006. Scheduling setup changes at bottleneck workstations in semiconductor manufacturing. Production Planning and Control 17, 717–727.

Essafi, I., Mati, Y., Dauzere-Peres, S., 2007. A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. Computers and Operations Research 35, 2599–2616.

Franca, P.M., Mendes, A., Moscato, P., 2001. A memetic algorithm for the total tardiness single machine scheduling problem. European Journal of Operational Research 132, 224–242.

Goncalves, J.F., Mendes, J.J. de M., Resende, M.G.C., 2005. A hybrid genetic algorithm for the job shop scheduling problem. European Journal of Operational Research 167, 77–95.

Jayamohan, M.S., Rajendran, C., 2000. New dispatching rules for shop scheduling: A step forward. International Journal of Production Research 38, 563–586.

Jeong, K.C., Kim, Y.-D., 1998. A real-time scheduling mechanism for a flexible manufacturing system: Using simulation and dispatching rules. International Journal of Production Research 36, 2609–2626.

Kim, M.H., Kim, Y.-D., 1994. Simulation-based real-time scheduling in a flexible manufacturing system. Journal of Manufacturing Systems 13, 85–93.

Kim, Y.-D., Kim, J.-G., Choi, B., Kim, H.-U., 2001. Production scheduling in a semiconductor wafer fabrication facility producing multiple product types with distinct due dates. IEEE Transactions on Robotics and Automation 17, 589–598.

Lee, Y.H., Bhaskaran, K., Pinedo, M., 1997. A heuristic to minimize the total weighted tardiness with sequence-dependent setups. IIE Transactions 29, 45–52.

Lodree Jr., E., Jang, W., Klein, C.M., 2004. A new rule for minimizing the number of tardy jobs in dynamic flow shops. European Journal of Operational Research 159, 258–263.

Mattfeld, D.C., Bierwirth, C., 2004. An efficient genetic algorithm for job shop scheduling with tardiness objectives. European Journal of Operational Research 155, 616–630.

Sevaux, M., Dauzere-Peres, S., 2003. Genetic algorithms to minimize the weighted number of late jobs on a single machine. European Journal of Operational Research 151, 296–306.

Taillard, E.D., 1993. Benchmarks for basic scheduling problems. European Journal of Operational Research 64, 278–285.

Taillard, E.D., Gambardella, L.M., Gendreau, M., Potvin, J.-Y., 2001. Adaptive memory programming: A united view of metaheuristics. European Journal of Operational Research 135, 1–16.

Taner, M.R., Hodgson, T.J., King, R.E., Thoney, K.A., 2003. Satisfying due-dates in a job shop with sequence-dependent family set-ups. International Journal of Production Research 41, 4153–4169.

Vepsalainen, A.P.J., Morton, T.E., 1988. Improving local priority rules with global lead-time estimates: A simulation study. Journal of Manufacturing and Operations Management 1, 102–118.

Yang, T., Kou, Y., Cho, C., 2007. A genetic algorithms simulation approach for the multi-attribute combinatorial dispatching decision problem. European Journal of Operational Research 176, 1859–1873.

Zhu, X., Whilhem, W.E., 2006. Scheduling and lot sizing with sequence-dependent setup: A literature review. IIE Transactions 38, 987–1007.