Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2013, Article ID 537127, 11 pages
http://dx.doi.org/10.1155/2013/537127

Research Article

A Hybrid Genetic Algorithm to Minimize Total Tardiness for Unrelated Parallel Machine Scheduling with Precedence Constraints

Chunfeng Liu

School of Management, Hangzhou Dianzi University, Hangzhou 310018, China

Correspondence should be addressed to Chunfeng Liu; lcf_spring@163.com

Received 9 March 2013; Revised 7 June 2013; Accepted 19 June 2013

Academic Editor: Jyh-Horng Chou

The paper presents a novel hybrid genetic algorithm (HGA) for a deterministic scheduling problem where multiple jobs with arbitrary precedence constraints are processed on multiple unrelated parallel machines. The objective is to minimize total tardiness, since delays of the jobs may lead to punishment cost or cancellation of orders by the clients in many situations. A priority rule-based heuristic algorithm, which schedules a prior job on a prior machine according to the priority rule at each iteration, is suggested and embedded to the HGA for initial feasible schedules that can be improved in further stages. Computational experiments are conducted to show that the proposed HGA performs well with respect to accuracy and efficiency of solution for small-sized problems and gets better results than the conventional genetic algorithm within the same runtime for large-sized problems.

1. Introduction

We consider an unrelated parallel machine scheduling problem with arbitrary precedence constraints to minimize total tardiness. Such a problem typically occurs in an office or project management environment, where unrelated machines represent workers who have different skills in office scheduling problem, or represent various types of resources which are allocated to activities in multimode project scheduling problem. Moreover, a task (or project) can be separated into several subtasks (or activities) with precedence constraints between them (e.g., one subtask may have to be finished before another subtask can be started). In many situations, delays of the subtasks (or activities) may lead to punishment cost or cancellation of orders by the clients. Mangers should adopt some methods to select the suitable workers (or resources) to undertake each subtask (or activity) separately, in order to maximize utilization of these workers (or resources), improve productivity, and reduce overall cost.

The remainder is organized as follows. A comprehensive review of the related literature is presented in Section 2. The problem is formulated as an integer programming model in Section 3. In Section 4, a priority rule-based heuristic algorithm (PRHA) is suggested for the feasible schedules. In Section 5, a hybrid genetic algorithm (HGA), taking the solutions of the PRHA as a part of initial population, is proposed for the final solution. In Section 6, two categories of numerical experiments are conducted to evaluate the performance of the proposed HGA. Finally, the paper closes with a general discussion of the proposed approach as well as a few remarks on research perspectives in Section 7.

2. The Literature Review

There are some studies about multiplemachines and precedence constraints in the literature. The machines may be identical, that is, they have equal speeds, or uniform, that is, each machine has a constant speed, independent of the tasks, or they may be unrelated if the speed of each machine depends on the task it processes. Precedence constraints include chains, out-tree, in-tree, forest, special precedence constraints, and arbitrary precedence constraints.

For identical machine scheduling with precedence-related jobs, Ramachandra and Elmaghraby [1] offered a binary integer program (BIP) and a dynamic program (DP) to solve two-machine problem with arbitrary precedence constraints to minimize total weighted completion time. They have also

introduced a genetic algorithm (GA) procedure that is capable of solving any problem size. Queyranne and Schulz [2] presented a 4-approximation algorithm for the identical machine problem with precedence delays to minimize total weighted completion time. In that problem each precedence constraint is associated with a certain amount of time that must elapse between the completion and start times of the corresponding jobs. Kim et al. [3] considered an identical machine problem with s-precedence constraints to minimize total completion time and formulated it as an LP problem with preemption allowed. To solve the LP problem efficiently, they developed a cutting plane approach in which a pseudopolynomial dynamic programming algorithm was derived to solve the involved separation problem. Gacias et al. [4] proposed an exact branch-and-bound procedure and a climbing discrepancy search (CDS) heuristic for the identical machine scheduling problem with precedence constraints and setup times between the jobs. Driessel and Mönch [5] suggested several variants of variable neighborhood search (VNS) schemes for scheduling jobs on identical machines with sequence-dependent setup times, precedence constraints, and ready times. Yuan et al. [6] considered the online scheduling on two identical machines with chain precedence constraints to minimize makespan, where jobs arrive over time and have identical processing times,and provided a best possible online algorithm of competitive ratio $(\sqrt{13}-1)/2$.

For uniform machine scheduling with precedence-related jobs, Brucker et al. [7] considered a problem of scheduling identical jobs with chain precedence constraints on two uniform machines. It was shown that the corresponding makespan problem could be solved in linear time. Woeginger [8] proposed a 2-approximation algorithm for uniform machine problem subject to chain precedence constraints to minimize makespan. Kim [9] derived an LP-based heuristic procedure for scheduling s-precedence constrained jobs on uniform machines with different speeds to minimize the weighted total completion time. van Zuylen [10] proved a randomized $O(\log m)$-approximation algorithm that is monotone in expectation for scheduling uniform machines with precedence constraints to minimize makespan.

For unrelated machine scheduling with precedence-related jobs, Herrmann et al. [11] considered chain precedence constraints between the tasks and proposed a number of heuristics to minimize makespan. Kumar et al. [12] presented polylogarithmic approximations to minimize makespan and total weighted completion time when the precedence constraints forming a forest. Nouri and Ghodsi [13] studied a scheduling problem where some related tasks with exponential duration were processed by some unrelated workers so that the total expected time to execute the tasks is minimum and gave a polynomial time algorithm for the problem in the restricted form.

In addition, for unrelated machine scheduling with tardiness related criteria, Chen and Chen [14] considered a flexible flow line scheduling problem with unrelated parallel machines at each stage and with a bottleneck stage on the line and proposed the bottleneck-based heuristics to minimize total tardiness. Zhang et al. [15] addressed a dynamic unrelated machine scheduling problem where the arrival time and the due date of each job are stochastic and applied a reinforcement learning method to minimize mean weighted tardiness. Liaw et al. [16] examined an unrelated machine scheduling problem to minimize the total weighted tardiness and presented a branch-and-bound algorithm incorporating various dominance rules. Kim et al. [17] presented an unrelated machine scheduling problem with sequence dependent setup times and suggested four search heuristics to minimize total weighted tardiness: the earliest weighted due date, the shortest weighted processing time, the two-level batch scheduling heuristic, and the simulated annealing method.

To the best of our knowledge, there is no work on unrelated machine scheduling problem with total tardiness criteria and precedence constraints of the jobs. Motivated from that fact, a hybrid genetic algorithm (HGA) is proposed for the practical scheduling problem.

3. Problem Formulation

A set $J = \{1, \ldots, n\}$ of n jobs has to be processed on m unrelated parallel machines $M = \{1, \ldots, m\}$. Each machine is capable of processing these jobs at different speeds and can process at most one job at a time. Each job is ready at the beginning of the scheduling horizon, processed on only one machine, and nonpreemptive during the processing period. Each job j has an integer processing time p_{jv} on machine v and a distinct due date d_j. There are arbitrary precedence constraints between the jobs. The constraints force a job not to be started before all its predecessors are finished. The objective is to find a feasible schedule that minimizes total tardiness $\mathrm{tt} = \sum_{j=1}^{n} \max(\mathrm{FT}_j - d_j, 0)$, where tardiness of job j is the amount of time its finish time FT_j exceeds its due date d_j. In standard scheduling notation [18], this problem can be denoted as $Rm|\mathrm{prec}|\mathrm{tt}$, where R denotes unrelated parallel machines, and prec denotes arbitrary precedence constraints. The problem is NP-hard even for a single machine [19].

The problem can be represented as a mathematical formulation as follows:

$$\text{minimize} \quad \mathrm{tt} = \sum_{j=1}^{n} \max\left(\mathrm{FT}_j - d_j, 0\right) \tag{1}$$

$$\text{subject to} \quad \sum_{v=1}^{m} \sum_{r=1}^{\phi} x_{jvr} = 1, \quad \forall j \in J, \tag{2}$$

$$\sum_{j=1}^{n} x_{jvr} \leqslant 1, \quad \forall r \in R, \forall v \in M, \tag{3}$$

$$\sum_{i=1}^{n} x_{ivr} - \sum_{j=1}^{n} x_{j,v,r-1} \leqslant 0, \quad \forall v \in M, \forall r \in \{2, \ldots, \phi\}, \tag{4}$$

$$\mathrm{FT}_j - \mathrm{FT}_i + L\left(2 - x_{jvr} - x_{i,v,r-1}\right) \geqslant p_{jv}, \quad \forall i, j \in J, \quad i \neq j, \quad \forall v \in M, \quad \forall r \in \{2, \ldots, \phi\}, \tag{5}$$

$$\mathrm{FT}_j \geqslant \sum_{r=1}^{\phi} p_{jv} x_{jvr}, \quad \forall j \in J,\ \forall v \in M, \tag{6}$$

$$\mathrm{FT}_j - \mathrm{FT}_i \geqslant \sum_{v=1}^{m} \sum_{r=1}^{\phi} p_{jv} x_{jvr}, \quad \forall i \in P_j, \tag{7}$$

$$\begin{aligned} &x_{jvr} \in \{0,1\}, \quad \mathrm{FT}_j \geqslant 0, \\ &\forall j \in J, \quad \forall v \in M, \quad \forall r \in R. \end{aligned} \tag{8}$$

The input parameters of the model include

J: the job set, $J = \{1, \ldots, n\}$;

M: the machine set, $M = \{1, \ldots, m\}$;

ϕ: the maximum number of positions on each machine that jobs are placed on them. It is computed as follows: $\phi = n - m + 1$ (i.e., the maximum machine utilization is met, so that all machines are used);

R: the position set, $R = \{1, \ldots, \phi\}$;

p_{jv}: the processing time of job j on machine v;

d_j: the due date of task j, $j \in J$;

P_j: the set of immediate predecessors of job j;

L: a large positive number.

The decision variables of the model include

x_{jvr}: equals 1 if job j is processed in the position r on machine v and 0 otherwise;

FT_j: the finish time of job j.

The objective function (1) minimizes the total tardiness. Constraints (2) ensure that each job is assigned to one of the existing positions on the machines. Constraints (3) guarantee that at most one job can be assigned to each position. Constraints (4) ensure that until one position on a machine is occupied, jobs are not assigned to subsequent positions. Constraints (5) ensure that the finish time of a job in sequence on a machine is at least equal to the sum of the finish time of the preceding job and the processing time of the present job. Constraints (6) ensure that the finish time of each job is not less than its processing time. Constraints (7) observe precedence relationships. Constraints (8) define the type of decision variables.

4. Priority Rule-Based Heuristic Algorithm

Heuristics have taken an important position in the research of solutions in many combinatory problems, as it is simple, easy to implement, and can be embedded in more sophisticated heuristics or metaheuristics for determining initial feasible schedules that can be improved in further stages. We develop a priority rule-based heuristic algorithm (PRHA) consisting of several iterations. At each iteration, a prior job is selected according to EDD (earliest due date first) rule and inserted inside a partial schedule on a selected prior machine (respecting the precedence constraints), while keeping the start times of the already scheduled jobs unchanged. The prior machine can be selected according to different conditions.

Then, three job-sets associated with each iteration are defined. Jobs which have been finished up to the schedule time point are in the completion job-set C_u. Jobs which have been scheduled are in the total scheduled job-set H. Jobs which are available for scheduling with respect to precedence constraints but yet unscheduled are in the decision job-set D_u.

The variables used for the PRHA are summarized as follows:

u: the counter of iteration;

H: the total scheduled job-set;

C_u: the completion job-set at the uth iteration;

D_u: the decision job-set at the uth iteration, $D_u = \{j \mid j \notin H, P_j \subseteq C_u\}$;

I_v: the release time of machine v (i.e., the machine is not occupied after the time point);

(j^*, v^*): the prior job-machine pair (j^* and v^* indicate the selected prior job and prior machine, resp.);

F_{jv}: the hypothetical finish time of job j when processed on machine v;

t: the scheduling time point.

We give a pseudocode description of the priority rule-based heuristic algorithm which consists of n iterations (cf. Algorithm 1). Step 1 initializes some variables u, H, tt, and I_v. In step 2 each job is scheduled at each iteration until $u \leqslant n$ does not hold. In step 2.1 T is denoted as a set containing the release times of all machines. In step 2.2 the scheduling time point t is determined by the minimum release time of machines. The completion job-set C_u and the decision job-set D_u at the time point t are computed in steps 2.3 and 2.4. In step 2.5 the schedule time point t is postponed to the next minimum release time until D_u is not empty. In step 2.6 an EDD rule is used to determine a prior job that is, the job with minimum due date in D_u is selected as the prior job j^*. Each hypothetical finish time F_{j^*v} of job j^* on machine v of the set M is first computed in step 2.7, and then the prior machine v^* is selected in step 2.8. If all hypothetical finish times of job j^* are not less than its due date, the machine with minimum hypothetical finish time is selected as the prior machine v^*; otherwise, v^* is randomly selected from these machines which process job j^* with the associated hypothetical finish times less than its due date. In step 2.9 the start and finish times of job j^* and the release time of machine v^* are saved. The total scheduled job-set H, total tardiness tt, and counter of iteration u are updated in steps 2.10–2.12.

To better illustrate the proposed PRHA, let us consider a simple instance. Nine jobs have to be processed on two unrelated parallel machines. The processing times and due dates of all jobs are given in Table 1, and the precedence constraints are displayed in Figure 1. The solution is reached after nine iterations using the PRHA. Table 2 shows the computational process of each iteration.

When $u = 1$, the decision job-set D_u is computed in step 2.5. The selected prior job $j^* = 1$ is processed on the selected prior machine $v^* = 1$, started at time $\mathrm{ST}_{j^*} = 0$, and completed

1. Initialize: $u = 1, H = \emptyset,\ \mathrm{tt} = 0;$
   $I_v = 0,\ \forall v \in M$
2. **WHILE** $(u \leqslant n)$ **DO**
   2.1. $T = \{I_v \mid v \in M\}$
   2.2. $t = \min\{\tau \mid \tau \in T\}$
   2.3. $C_u = \{j \mid FT_j \leqslant t,\ \forall j \in H\}$
   2.4. compute D_u
   2.5. **WHILE** $(D_u = \emptyset)$ **DO**
      2.5.1. $T := T \setminus \{t\}$
      2.5.2. $t = \min\{\tau \mid \tau \in T\}$
      2.5.3. $C_u = \{j \mid \mathrm{FT}_j \leqslant t,\ \forall j \in H\}$
      2.5.4. compute D_u
   2.6. $j^* : d_{j^*} = \min\{d_j \mid j \in D_u\}$
   2.7. **FOR** $(v = 1 : m)$
      IF $(I_v \geqslant t)$
         $F_{j^*v} = I_v + p_{j^*v}$
      ELSE
         $F_{j^*v} = t + p_{j^*v}$
   2.8. **IF** $(F_{j^*v} \geqslant d_{j^*},\ \forall v \in M)$
      $v^* : F_{j^*v^*} = \min\{F_{j^*v} \mid v \in M\}$
   ELSE
      randomly select v^* from $\{v \mid F_{j^*v} < d_{j^*}, v \in M\}$
   2.9. $\mathrm{ST}_{j^*} = F_{j^*v^*} - p_{j^*v^*}, \mathrm{FT}_{j^*} = F_{j^*v^*}, I_{v^*} = \mathrm{FT}_{j^*}$
   2.10. $H := H \cup \{j^*\}$
   2.11. $\mathrm{tt} := \mathrm{tt} + \max(\mathrm{FT}_{j^*} - d_{j^*}, 0)$
   2.12. $u := u + 1$

Algorithm 1: Priority rule-based heuristic algorithm.

Table 1: Problem data of the instance.

	job1	job2	job3	job4	job5	job6	job7	job8	job9
p_{j1}	3	4	8	2	5	9	3	5	8
p_{j2}	9	5	2	6	9	4	8	7	5
d_j	3	4	5	7	9	8	11	13	12

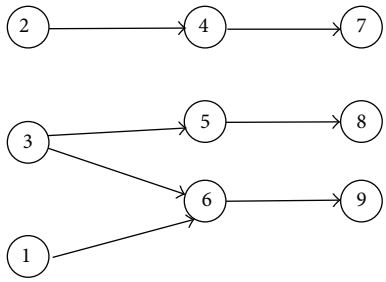

Figure 1: Project precedence constraints graph of the instance.

at time $\mathrm{FT}_{j^*} = 3$, which are calculated in steps 2.6, 2.8, and 2.9. The total tardiness $\mathrm{tt} = 0$ is computed in step 2.11. The following iterations are similar to the first iteration. The final solution is presented by the Gantt chart in Figure 2.

5. Proposed HGA and Its Implementation

Genetic algorithm (GA) is a powerful and broadly applicable stochastic search and optimization technique based on principles of evolution theory. In the past few years, GA has received considerable attention for solving difficult combinatorial optimization problems.

Table 2: Summary of iterations.

u	D_u	j^*	v^*	ST_{j^*}	FT_{j^*}	tt
1	{1, 2, 3}	1	1	0	3	0
2	{2, 3}	2	2	0	5	1
3	{3}	3	2	5	7	3
4	{4, 5, 6}	4	1	7	9	5
5	{5, 6}	6	2	7	11	8
6	{5, 7}	5	1	9	14	13
7	{7, 9}	7	1	14	17	19
8	{9}	9	2	11	16	23
9	{8}	8	1	17	22	32

We propose a hybrid genetic algorithm (HGA) which combines the PRHA approach with conventional genetic algorithm. In HGA, the PRHA plays an important role in generating good initial solutions to avoid the blind search of GA at the beginning while exploring the solution space. The HGA explores the solution spaces with three genetic operators, including the patching crossover, swap mutation, and "roulette wheel" selection. The principle of HGA is illustrated in Figure 3, and the detailed description is as follows.

5.1. Coding and Fitness Function. The first step in the proposed HGA is to consider a chromosome representation or solution structure. Suppose that there are n jobs to be assigned to m machines. A chromosome is modeled by a string of $n + m - 1$ distinct genes, composed of n job genes, numbered

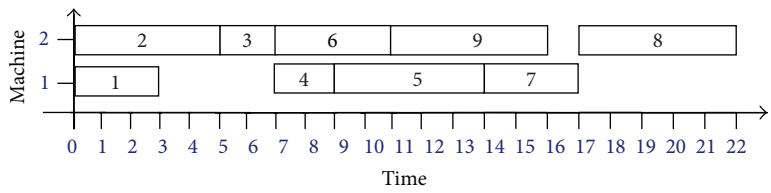

Figure 2: Gantt chart using the PRHA.

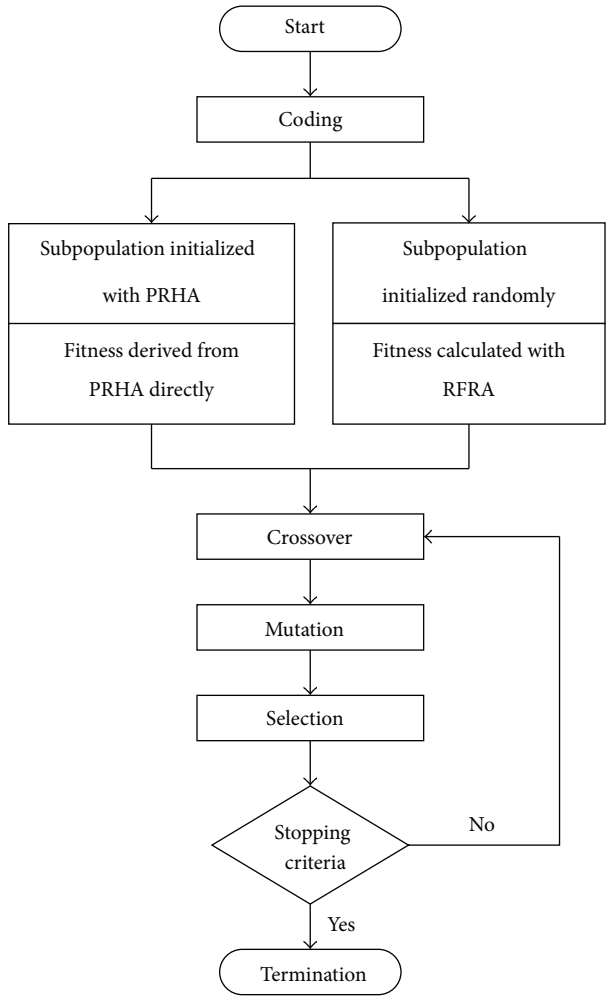

Figure 3: Program flow chart of the HGA.

from 1 to n, and $m - 1$ partitioning genes "$*$" with distinct subscripts to separate the machines [20]. Meanwhile, the chromosome may be decomposed as m subchromosomes, and genes of each subchromosome represent unordered jobs on a machine. An example is demonstrated in Figure 4, where the chromosome would assign jobs 9, 1, 5 to machine 1, jobs 8, 6, 2 to machine 2, job 7 to machine 3, and jobs 3, 4 to machine 4. When there are no job genes between two consecutive partitioning genes, the corresponding machine of the second partitioning gene is not occupied in the schedule. Similarly, when there are no job genes after the last partitioning gene, the rest machine is not occupied in the schedule.

The HGA manipulates solutions to propagate similarities among the high performance chromosomes to the next population based on fitness values. The fitness function corresponds to the objective function under consideration. Since each job has a constant processing time in its subchromosome, the start and finish times of all jobs can be computed by applying a revised forward recursion algorithm (RFRA) according to the precedence constraints (cf. Algorithm 2). The fitness value of each chromosome can thus be obtained.

We use the instance in Section 4 to illustrate the RFRA. Suppose that jobs 1, 2, 3, 5, 6, and 7 are assigned to machine 1 and jobs 4, 8, and 9 are assigned to machine 2. Figure 5 shows the computational process of the RFRA. At first, the job 1 is randomly selected from the unscheduled job-set $U = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, and the finish time $FT_1 = 3$ is computed in step 2.2.2. Then, the job 6 is randomly selected

Chromosome:	7	^{*3}	9	1	5	^{*1}	8	6	2	^{*2}	3	4
Subchromosome1:	9	1	5									
Subchromosome2:	8	6	2									
Subchromosome3:	7											
Subchromosome4:	3	4										

Figure 4: Chromosome encode.

1. Initialize unscheduled job-set $U = \{1, \ldots, n\}$ and the release times of all machines $I_v = 0$, $\forall v \in M$.
2. Repeat the following steps until U is empty:
   2.1. randomly select a job j from U.
   2.2. Assign job j to machine k (k is known for job j according to the chromosome).
      2.2.1. If the finish time of predecessor i of job j is not known, execute step 2.2 for job i recursively.
      2.2.2. Let ξ_j denote the maximum finish time of the predecessors of job j. Compute the start time and finish time of job j: $\mathrm{ST}_j = \max(\xi_j, I_k)$, $\mathrm{FT}_j = \mathrm{ST}_j + p_{jk}$.
      2.2.3. Update the unscheduled job-set and the release time of machine k: $U := U \setminus \{j\}$, $I_k = \mathrm{FT}_j$.
3. $\mathrm{tt} = \sum_{j=1}^{n} \max(\mathrm{FT}_j - d_j, 0)$.

Algorithm 2: Revised forward recursion algorithm.

from the set $U = \{2, 3, 4, 5, 6, 7, 8, 9\}$. Because the finish time of predecessor 3 of job 6 is not known yet, step 2.2 is executed for job 3 recursively, and the finish time $\mathrm{FT}_3 = 11$ can be calculated. The following iterations are similar to the previous iterations, and the final total tardiness is 157.

5.2. Initial Population. PopSize is the population size or the number of chromosomes at each population that is known in advance. In the HGA, the population is initialized from two subpopulations with identical number of chromosomes: one subpopulation comes from the solutions of the PRHA, so that it can enhance population optimization; the other is generated by randomly assigning all jobs to the machines, so that it can enhance population diversity.

5.3. Crossover. Crossover is the kernel operation in genetic algorithm, which combines two chromosomes to generate next-generation chromosomes preserving their characteristics. In our crossover, all chromosomes in the parent generation are mutually crossed over according to a given crossover probability P_c. The mechanism is accomplished through the following steps, and Figure 6 demonstrates an example.

(i) Choose two chromosomes, named Parent1 and Parent2, from the parent population.

(ii) Randomly produce a string of $n + m - 1$ flags, each with value of either "0" or "1".

(iii) The flags are first matched with Parent1 such that those genes with flag "1" in Parent1 are selected and saved in Selected1 with their original positions unchanged.

(iv) Cross out the same genes as Selected1 from Parent2, and the remainder genes are saved in Selected2.

(v) The new offspring is obtained through filling out the remaining empty gene locations of Selected1 with the genes of Selected2 by preserving their gene sequence in Selected2.

(vi) The genes of offspring are assigned to the subchromosomes according to the subindexed "$*$".

(vii) The fitness value of offspring is computed by applying the RFRA algorithm.

An offspring acceptance method is employed to accept the offspring generated by crossover operator [21]. If the fitness value of offspring is not greater than the average fitness value of its parent generation, the offspring will be accepted for the new generation and will be thrown otherwise. This method reduces the computational time of the algorithm and leads to convergence toward the optimum solution neighborhood.

5.4. Mutation. Mutation reorganizes the structure of genes in a chromosome randomly so that a new combination of genes may appear in the next generation. It serves the search by jumping out of local optimal solutions. The swap mutation is used as mutation operator according to a given mutation probability P_m, and all offsprings from the mutation are accepted for the new generation. The mechanism is accomplished through the following steps, and Figure 7 demonstrates an example.

(i) Randomly select two genes from two different subchromosomes in a selected chromosome and swap their places.

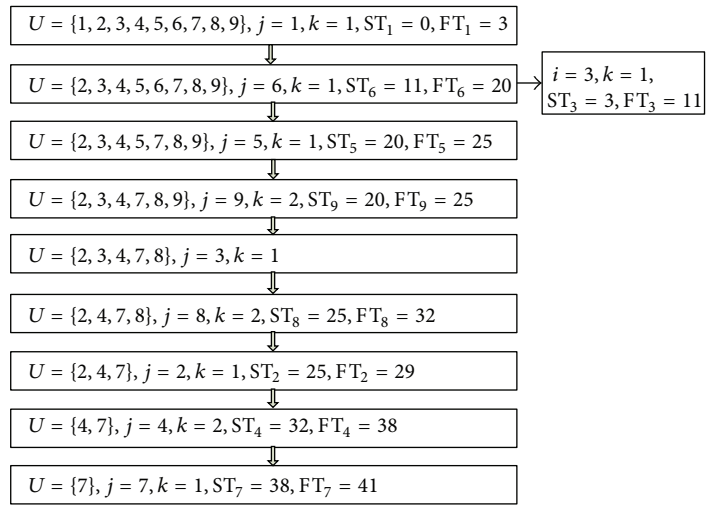

Figure 5: The computational process of the RFRA.

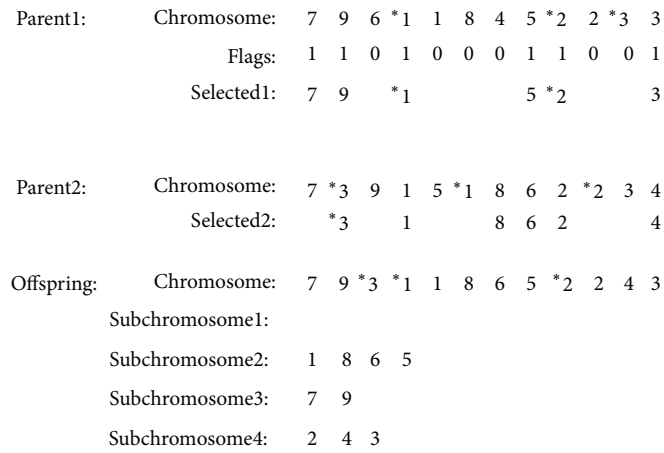

Figure 6: Example of our crossover mechanism.

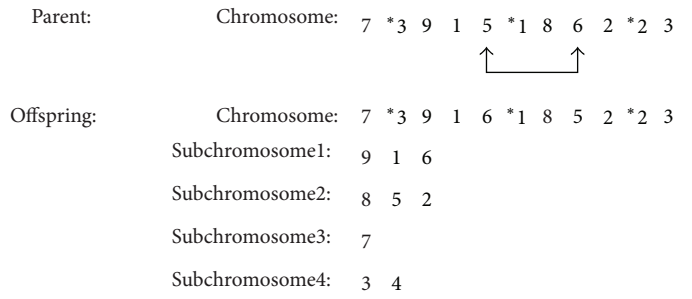

Figure 7: Example of our mutation mechanism.

(ii) The genes of new offspring are assigned to the subchromosomes according to the subindexed "$*$".

(iii) The fitness value of offspring is computed by applying the RFRA algorithm.

5.5. Selection. Selection is an operation to choose good chromosomes for the next generation. It is important in regulating the bias in the reproduction process. Let α_g^k denote the selection value of the kth chromosome in generation g before selection. It is computed as

$$\alpha_g^k = s + \max_{i \in \{1,\ldots,e\}} F_g^i - F_g^k, \tag{9}$$

where F_g^k is the fitness value of the kth chromosome in generation g, e is the number of chromosomes in generation g before selection, and s is a small constant (say 3). Obviously,

Table 3: Parameters of the HGA.

Parameter	Value
Population size (PopSize)	20
Crossover probability (P_c)	0.9
Mutation probability (P_m)	0.05
Arbitrary constant for stopping criterion (ε)	0.0001
Maximum number of generation ($G_{\max}$)	200

the less the fitness value of chromosome, the greater its selection value (i.e., the selection value and fitness value of the chromosome have an inverse relationship). Generally, it is better that the solution with minimum fitness value (i.e., maximum selection value) in the current generation has more chance to be selected as parent in order to create offspring. The most common method for the selection mechanism is the "roulette wheel" sampling. Each chromosome is assigned a slice of the circular roulette wheel and the size of the slice is proportional to the selection value of chromosome. The wheel is spun PopSize times. On each spin, the chromosome under the wheel's marker is selected to be in the pool of parents for the next generation.

5.6. Stopping Rules. The HGA is stopped with satisfying either of the following conditions: (1) the number of current generation (g) is greater than the maximum number of generation ($G_{\max}$), and (2) the standard deviation of the fitness values of chromosomes in the current generation (σ_g) is not greater than an arbitrary constant (ε) [22]. σ_g implies a degree of diversity or similarity in the current population in terms of the fitness value. It is computed as

$$\sigma_g = \left[\left(\frac{1}{\text{PopSize}}\right)\sum_{k=1}^{\text{PopSize}}\left(F_g^k - \overline{F}_g\right)^2\right]^{1/2}, \tag{10}$$

where $\overline{F}_g$ is the mean fitness value of all chromosomes in generation g that is computed as $\overline{F}_g = (1/\text{PopSize})\sum_{k=1}^{\text{PopSize}} F_g^k$.

6. Computational Experiments

To evaluate the performance of the proposed HGA, the following two categories of numerical experiments for small and large-sized problems are conducted. The small-sized problems are solved by the branch-and-bound approach (B&B) under the CPLEX software and the proposed HGA. The large-sized problems are solved by the HGA and the conventional genetic algorithm (CGA) which is not combined with the priority rule-based heuristic algorithm, since they cannot be optimally solved by the CPLEX in a reasonable CPU time. Through referring to [23] and preliminary tests, the appropriate parameters of the HGA are determined and listed in Table 3. The CGA is stopped when its runtime reaches the HGA's.

The processing times of the jobs are randomly generated in DU$[a, b]$ which represents a discrete uniform distribution with a range from a to b. The due dates are randomly obtained by [24]

$$d_i = \left[\left(\frac{\theta}{2m}\right)\left(1 - \rho - \frac{\text{RD}}{2}\right), \left(\frac{\theta}{2m}\right)\left(1 - \rho + \frac{\text{RD}}{2}\right)\right], \tag{11}$$

$$\theta = \sum_{j=1}^{n}\sum_{v=1}^{m} p_{jv}, \tag{12}$$

where ρ is the delay coefficient and RD is the relative range of due dates. To produce the precedence relations, let D = density of precedence constraints = Pr{arc(i, j) exists in precedence constraints}, and let P_{ij} = Pr{arc(i, j) exists in the immediate precedence graph}, for $1 \leqslant i < j \leqslant n$, where i, j are different jobs and Pr is abbreviation of probability. Hall and Posner [25] have proved that

$$P_{ij} = \frac{D(1-D)^{j-i-1}}{1 - D\left(1 - (1-D)^{j-i-1}\right)}. \tag{13}$$

For a given D, each precedence relation of jobs i and j can be determined by P_{ij}.

The experiments have been performed on a Pentium-based Dell-compatible personal computer with 2.30 GHz clock-pulse and 2.00 GB RAM. The HGA and CGA algorithms have been coded in C++, compiled with the Microsoft Visual C++ 6 compiler, and tested under Microsoft Windows 7 (64-bit) operating system.

The first category of experiments is conducted for the small-sized problems. We generate nine groups of problem instances (each group including 10 instances), given the density of precedence constraints $D = 0.5$, the delay coefficient $\rho = 0.5$, the relative range of due dates RD $= 0.1$, and the processing times $p_{jv} \sim [1, 10]$. Each instance is solved by the HGA; meanwhile, its exact solution is achieved by the B&B under the CPLEX 12.2 software. Let f_{BB} and f_{HGA} denote the mean objective function values (i.e., total tardiness) of the problem group using the B&B and HGA, respectively, and let Gap denote the relative distance between f_{BB} and f_{HGA}; that is, Gap $= (f_{\text{HGA}} - f_{\text{BB}})/f_{\text{BB}} \times 100\%$. The computational results are shown in Table 4. It can be observed that for the small-sized problems, the HGA is able to get good results (the average Gap is only 0.3%) and to obtain exact solutions (i.e., Gap = 0) for most of the problems. In addition, it also can be seen from the average of mean CPU time of the B&B and HGA (24.5 s and 3.1 s) that the runtime of the B&B is not obviously comparable with the one of HGA. Figure 8 shows a typical convergence of the HGA during 27 successive generations related to a single run.

The second category of experiments is conducted for the large-sized problems. The performance of the HGA is compared with the CGA by use of six impact factors including number of jobs (n), number of machines (m), density of precedence constraints (D), delay coefficient (ρ), relative range of due dates (RD), and processing times (p_{jv}). Here, six sets of subexperiments are conducted. In the first set displayed in Table 5, n is allowed to vary to test its impact effect, given $m = 10$, $D = 0.6$, $\rho = 0.5$, RD $= 0.1$, and $p_{jv} \sim$ DU$[1, 10]$. The other five sets displayed in Tables 6–10 test

Table 4: Performance of the HGA for small-sized instances.

No. of group	$D = 0.5$, $\rho = 0.5$, RD = 0.1, $p_{jv} \sim [1, 10]$		B&B		HGA		
	n	m	f_{BB}	Mean CPU time (s)	f_{HGA}	Mean CPU time (s)	*Gap* (%)
1	9	4	39	1.5	39	0.4	0
2	9	3	72	5.5	72	0.6	0
3	9	2	61	72.7	61	0.7	0
4	10	4	68	9.3	68	7.1	0
5	10	3	63	2.9	63	5.7	0
6	10	2	64	18.7	65	8.4	1.6
7	11	4	32	86.2	32	2,3	0
8	11	3	123	2.8	124	0.6	0.8
9	11	2	129	20.5	129	1.9	0
Average				24.5		3.1	0.3

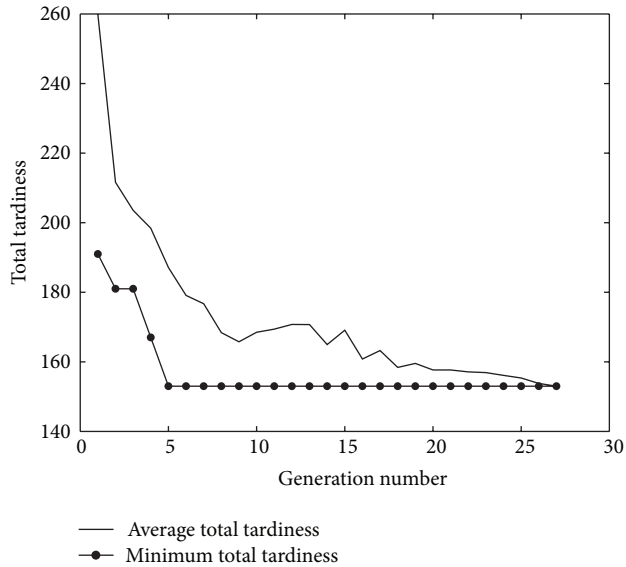

Figure 8: A typical convergence of the HGA during 27 successive generations related to a single run.

Table 5: Comparison between HGA and CGA for large-sized instances with different number of jobs (n).

	$m = 10$, $D = 0.6$, $\rho = 0.5$, RD = 0.1, $p_{jv} \sim$ DU[1, 10]	f_{HGA}	f_{CGA}	Mean CPU time (s)	Δf_{GA} (%)
n	30	129.6	216.3	59.1	40.1
	60	311.9	807.4	182.9	61.4
	90	914.1	2882.5	360.8	68.3
	120	855.5	5500.5	603.9	84.4

Table 6: Comparison between HGA and CGA for large-sized instances with different number of machines (m).

	$n = 60$, $D = 0.6$, $\rho = 0.5$, RD = 0.1, $p_{jv} \sim$ DU[1, 10]	f_{HGA}	f_{CGA}	Mean CPU time (s)	Δf_{GA} (%)
m	5	773.6	1252.2	163.6	38.2
	10	316.0	794.7	184.2	60.2
	15	204.2	717.2	205.9	71.5
	20	281.8	698.7	228.7	59.7

the effects of varying m, D, ρ, RD, and p_{jv}, respectively. Each table entry represents 10 randomly generated instances. Let f_{HGA} and f_{CGA} denote the mean objective function values using the HGA and CGA, respectively, and let Δf_{GA} denote the declining percentage of f_{HGA} lower over f_{CGA}, that is, $\Delta f_{GA} = (f_{CGA} - f_{HGA})/f_{CGA} \times 100\%$.

From Tables 5, 6, 7, 8, 9, and 10 we can see that, within the same runtime Δf_{GA} can reach 14.7%–84.4%, and it increases

Table 7: Comparison between HGA and CGA for large-sized instances with different density of precedence constraints (D).

	$n = 60$, $m = 10$, $\rho = 0.5$, RD = 0.1, $p_{jv} \sim$ DU[1, 10]	f_{HGA}	f_{CGA}	Mean CPU time (s)	Δf_{GA} (%)
D	0.2	100.8	364.0	183.2	72.3
	0.4	135.5	481.1	182.8	71.8
	0.6	375.5	935.6	183.2	59.9
	0.8	782.0	1733.3	184.3	54.9

Table 8: Comparison between HGA and CGA for large-sized instances with different delay coefficients (ρ).

	$n = 30$, $m = 5$, $D = 0.6$, RD = 0.1, $p_{jv} \sim$ DU[1, 10]	f_{HGA}	f_{CGA}	Mean CPU time (s)	Δf_{GA} (%)
ρ	0.3	248.7	313.0	105.2	20.5
	0.5	226.7	311.2	112.1	27.2
	0.7	297.9	377	112.3	21.0
	0.9	497.4	583.0	113.0	14.7

Table 9: Comparison between HGA and CGA for large-sized instances with different relative range of due dates (RD).

	$n = 30$, $m = 5$, $D = 0.6$, $\rho = 0.5$, $p_{jv} \sim$ DU[1, 10]	f_{HGA}	f_{CGA}	Mean CPU time (s)	Δf_{GA} (%)
RD	0.1	265.0	370.2	97.3	28.4
	0.3	311.2	411.4	112.5	24.4
	0.5	293.6	436.4	114.6	32.7
	0.7	226.8	323.8	77.8	30.0

Table 10: Comparison between HGA and CGA for large-sized instances with different processing times (p_{jv}).

	$n = 60$, $m = 10$, $D = 0.6$, $\rho = 0.5$, RD = 0.1	f_{HGA}	f_{CGA}	Mean CPU time (s)	Δf_{GA} (%)
p_{jv}	DU[1, 5]	263.1	531.1	183.7	50.5
	DU[1, 10]	304.1	843.5	183.3	63.9
	DU[1, 15]	290.2	1047.3	183.1	72.3
	DU[1, 20]	598.8	1631.0	183.7	63.3

markedly as the number of jobs (n) increases. This also shows the optimization advantage of the HGA regardless of different number of machines, different density of precedence constraints, different delay coefficient, different relative range of due dates, and different processing times, because the HGA takes the solutions of heuristic algorithm as a part of initial population, which avoids blind search of the CGA. It is worth mentioning that the total tardiness will decrease generally as m increases; however, each instance is randomly generated under given conditions in Table 6, so f_{CGA} may show an obvious down trend for the solutions using the CGA are not exact results, and f_{HGA} may show an approximate down trend for the solutions using the HGA are very near to exact results. Table 10 shows the same situation for the reason of randomly generated instances. Moreover, Δf_{GA} decreases as the density of precedence constraints (D) increases in Table 7, because as D increases, the precedence relations of all jobs become tighter, the schedule of these jobs tends to be more deterministic, and the advantage of the HGA over CGA will decrease.

7. Conclusions

In this paper, an intuitive priority rule-based heuristic algorithm (PRHA) is first developed for the unrelated machine scheduling problem to construct some feasible schedules. The PRHA can schedule a prior job on a prior machine according to the priority rule at each iteration. Then, a hybrid genetic algorithm (HGA) is proposed to improve the final solution. We design subindexed genes, genetic operators, and add the standard deviation of the fitness values as stopping criterion to the traditional genetic algorithm. The fitness values can be computed through the revised forward recursion algorithm.

In order to evaluate the effectiveness and efficiency of the proposed HGA, two categories of numerical experiments are conducted. The obtained results show that the HGA performs accurately and efficiently for small-sized problems and can obtain exact solutions for most cases. Moreover, it gets better results than the conventional genetic algorithm within the same runtime for large-sized problems. However, the advantage of the HGA over CGA will decrease if the due dates of all jobs are identical or large enough, because the selection method for the prior job and machine in the PRHA algorithm may be ineffective.

The HGA can be applied to the just-in-time production systems considering penalties for tardy jobs. An important research direction that might be pursued in the future is extension of the developed priority rule in this work. The other potential interest would be to consider lower bound and exact algorithms for the complex problem.

Acknowledgments

This research was supported by the Humanities and Social Sciences Base of Zhejiang (no. RWSKZD02-201211) and the National Natural Science Foundation of China (no. 71101040). The author is grateful for the financial supports.

References

[1] G. Ramachandra and S. E. Elmaghraby, "Sequencing precedence-related jobs on two machines to minimize the weighted

completion time," *International Journal of Production Economics*, vol. 100, no. 1, pp. 44–58, 2006.

[2] M. Queyranne and A. S. Schulz, "Approximation bounds for a general class of precedence constrained parallel machine scheduling problems," *SIAM Journal on Computing*, vol. 35, no. 5, pp. 1241–1253, 2006.

[3] E.-S. Kim, C.-S. Sung, and I.-S. Lee, "Scheduling of parallel machines to minimize total completion time subject to s-precedence constraints," *Computers & Operations Research*, vol. 36, no. 3, pp. 698–710, 2009.

[4] B. Gacias, C. Artigues, and P. Lopez, "Parallel machine scheduling with precedence constraints and setup times," *Computers & Operations Research*, vol. 37, no. 12, pp. 2141–2151, 2010.

[5] R. Driessel and L. Mönch, "Variable neighborhood search approaches for scheduling jobs on parallel machines with sequence-dependent setup times, precedence constraints, and ready times," *Computers & Industrial Engineering*, vol. 61, no. 2, pp. 336–345, 2011.

[6] J. Yuan, W. Li, and J. Yuan, "A best possible online algorithm for scheduling equal-length jobs on two machines with chain precedence constraints," *Theoretical Computer Science*, vol. 457, pp. 174–180, 2012.

[7] P. Brucker, J. Hurink, and W. Kubiak, "Scheduling identical jobs with chain precedence constraints on two uniform machines," *Mathematical Methods of Operations Research*, vol. 49, no. 2, pp. 211–219, 1999.

[8] G. J. Woeginger, "A comment on scheduling on uniform machines under chain-type precedence constraints," *Operations Research Letters*, vol. 26, no. 3, pp. 107–109, 2000.

[9] E.-S. Kim, "Scheduling of uniform parallel machines with s-precedence constraints," *Mathematical and Computer Modelling*, vol. 54, no. 1-2, pp. 576–583, 2011.

[10] A. van Zuylen, "An improved monotone algorithm for scheduling related machines with precedence constraints," *Operations Research Letters*, vol. 39, no. 6, pp. 423–427, 2011.

[11] J. Herrmann, J.-M. Proth, and N. Sauer, "Heuristics for unrelated machine scheduling with precedence constraints," *European Journal of Operational Research*, vol. 102, no. 3, pp. 528–537, 1997.

[12] V. S. A. Kumar, M. V. Marathe, S. Parthasarathy, and A. Srinivasan, "Scheduling on unrelated machines under tree-like precedence constraints," *Algorithmica*, vol. 55, no. 1, pp. 205–226, 2009.

[13] M. Nouri and M. Ghodsi, "Scheduling tasks with exponential duration on unrelated parallel machines," *Discrete Applied Mathematics*, vol. 160, no. 16-17, pp. 2462–2473, 2012.

[14] C.-L. Chen and C.-L. Chen, "Bottleneck-based heuristics to minimize total tardiness for the flexible flow line with unrelated parallel machines," *Computers & Industrial Engineering*, vol. 56, no. 4, pp. 1393–1401, 2009.

[15] Z. Zhang, L. Zheng, N. Li, W. Wang, S. Zhong, and K. Hu, "Minimizing mean weighted tardiness in unrelated parallel machine scheduling with reinforcement learning," *Computers & Operations Research*, vol. 39, no. 7, pp. 1315–1324, 2012.

[16] C.-F. Liaw, Y.-K. Lin, C.-Y. Cheng, and M. Chen, "Scheduling unrelated parallel machines to minimize total weighted tardiness," *Computers & Operations Research*, vol. 30, no. 12, pp. 1777–1789, 2003.

[17] D.-W. Kim, D.-G. Na, and F. F. Chen, "Unrelated parallel machine scheduling with setup times and a total weighted tardiness objective," *Robotics and Computer-Integrated Manufacturing*, vol. 19, no. 1-2, pp. 173–181, 2003.

[18] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of Discrete Mathematics*, vol. 5, pp. 287–326, 1979.

[19] J. Du and J. Y.-T. Leung, "Minimizing total tardiness on one machine is NP-hard," *Mathematics of Operations Research*, vol. 15, no. 3, pp. 483–495, 1990.

[20] C. Jou, "A genetic algorithm with sub-indexed partitioning genes and its application to production scheduling of parallel machines," *Computers & Industrial Engineering*, vol. 48, no. 1, pp. 39–54, 2005.

[21] R. Tavakkoli-Moghaddam, F. Taheri, M. Bazzazi, M. Izadi, and F. Sassani, "Design of a genetic algorithm for bi-objective unrelated parallel machines scheduling with sequence-dependent setup times and precedence constraints," *Computers and Operations Research*, vol. 36, no. 12, pp. 3224–3230, 2009.

[22] R. Tavakkoli-Moghaddam, N. Safaei, and F. Sassani, "A memetic algorithm for the flexible flow line scheduling problem with processor blocking," *Computers & Operations Research*, vol. 36, no. 2, pp. 402–414, 2009.

[23] C. Liu and S. Yang, "A hybrid genetic algorithm for integrated project task and multi-skilled workforce scheduling," *Journal of Computational Information Systems*, vol. 7, no. 6, pp. 2187–2194, 2011.

[24] N. Balakrishnan, J. J. Kanet, and V. Sridharan, "Early/tardy scheduling with sequence dependent setups on uniform parallel machines," *Computers & Operations Research*, vol. 26, no. 2, pp. 127–141, 1999.

[25] N. G. Hall and M. E. Posner, "Generating experimental data for computational testing with machine scheduling applications," *Operations Research*, vol. 49, no. 6, pp. 854–865, 2001.