A simulated annealing approach to minimize the maximum lateness on uniform parallel machines

Kai Li*, Shan-Lin Yang, Hua-Wei Ma
School of Management, Hefei University of Technology, Hefei 230009, PR China
Key Laboratory of Process Optimization and Intelligent Decision-making, Ministry of Education, Hefei 230009, PR China

ABSTRACT

This paper considers the uniform parallel machine scheduling problem which is to minimize the maximum lateness. This problem is equivalent to the uniform parallel machine scheduling problem, which is to minimize the maximal completion time of \( n \) jobs whose release times are zero, processing times depend on the speed of the machine to which they are assigned, and their delivery times are different. This problem is NP-hard, even if the machines' speeds are identical and all the delivery times equal to zero. We propose a simulated annealing algorithm, named LPDT-SA, to obtain solutions with high quality for large-sized problems. A heuristic algorithm LPDT is built to generate initial solutions. An effective method for solution representation is designed, which is efficient to realize the swap and insertion neighborhood, and simultaneously avoid some obvious inferior solutions, therefore the efficiency of the proposed simulated annealing algorithm is improved. A large set of instances are generated randomly to test the solution quality of LPDT-SA and assess its runtime. The results and analysis of experiments are reported and discussed.

1. Introduction

We consider the uniform parallel machine scheduling problem to minimize the maximal lateness. It is described generally as \( Q_{m} || L_{\text{max}} \) according to the standard three parameter notation proposed by Graham et al. [1]. Each scheduling problem is denoted by the standard three-field notation \( \alpha|\beta|\gamma \). The first field \( \alpha \) describes the scheduling type, the second field \( \beta \) is reserved for the information and conditions of scheduling, while the third field \( \gamma \) contains the performance criteria. This problem is NP-hard, even if its simpler \( P_{2} || C_{\text{max}} \) case, in which there are only 2 machines and all of the jobs have the same due date [2]. This problem is equivalent to the uniform parallel machine problem with delivery times to minimize the maximal completion time, i.e. \( Q_{m} |d_j| C_{\text{max}} \) problem. In many practical scheduling problems, the delivery times must elapse after the jobs are processed on the machine. The delivery times may be brought by the transportation times or the need of some special products, such as the course of cooling for steel and iron products or the course of drying for painted products.

The uniform parallel machine scheduling problems are fundamental to numerous complex real-world applications. Although much research has been devoted to the parallel machine scheduling problems with identical machine speeds, little research has been done on uniform parallel machines. Koumas and Kyparisis [3] showed that an extension of the EDD (Earliest Due Date first) rule to the \( Q_{m} || L_{\text{max}} \) problem yields a maximum lateness value does not exceed the optimal value by more than \( p_{\text{max}} \), where \( p_{\text{max}} \) is the maximum job processing time. They also showed that the LDT (Largest Delivery Time first) heuristic is \( (m-1)s_1/\sum_{i=1}^{m} s_i + 1 \) approximation algorithm for \( Q_{m} |d_j| C_{\text{max}} \) problem, where \( m \) is the number of

* Corresponding author at: School of Management, Hefei University of Technology, 230009 Hefei, PR China. Tel.: +86 551 2919156.
E-mail addresses: hfutlk@gmail.com (K. Li), hgdysl@gmail.com (S.-L. Yang), colt_mhw@126.com (H.-W. Ma).

0895-7177/$ – see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.mcm.2010.10.022
the machines, $s_i$ is the speed of the $i$-th machine, and $s_1$ is the maximum speed. Dessouky [4] considered $Q_n|r_j, p_j = 1|L_{\text{max}}$ problem, in which the job processing times are identical. He proposed six simple heuristics, and then developed a branch-and-bound procedure which could solve the problems within 5 machines and 80 jobs within a reasonable time.

Some other papers considered the corresponding identical parallel machine scheduling problem in which the machines have the same speed. The literature in recent years mainly focused on the problem with unequal release dates, i.e. $P_m|r_j|L_{\text{max}}$ problem. Carlier [5] and Néron et al. [6] developed some exact branch-and-bound algorithms for the problem. Vakhania [7] and Gharbi and Haouari [8] considered the development of heuristics. Mastroiliar [9] and Carlier and Pinson [10] proposed some approximation algorithms. Eren [11] considered the m-identical parallel machine scheduling problem with a learning effect to minimize the maximum lateness. He proposed a model, which can optimally solve the problems with 18 jobs and 4 machines within 7000 s on a personal computer with Pentium IV/2 512 Ram.

For the single machine with minimizing the maximum lateness, recent literature mainly considered some extensive problems. For example, Wu et al. [12] considered the single-machine maximum lateness minimization problem with a learning effect. The simulated annealing algorithm they proposed outperforms the traditional heuristic algorithm in terms of quality and execution time for a large number of jobs. It can solve the problems with 200 jobs within 21.24 s on a Pentium IV personal computer. Uzsoy and Velásquez [13] addressed the problem of scheduling a single machine subject to family-dependent set-up times in order to minimize maximum lateness. The incomplete dynamic programming heuristic they developed can solve the problems with 50 jobs within 334.21 s on a Pentium II, 266 MHz notebook with 64 MB of RAM.

Considering the NP-hardness of our scheduling problem, we introduce a simulated annealing approach to generate the near-optimal solutions with high quality. The sizes of the problems considered in this paper are more than 100 jobs, while the existing exact algorithms are merely effective within a reasonable time for the small-sized problems generally in which the job number is less than 100. Simulated annealing (SA) algorithm was proposed by Metropolis et al. [14], and it has been successfully applied to combinatorial problems by Kirkpatrick et al. [15]. Many researchers have used SA to scheduling problems, but mainly focused on single machine problems. For example, Potts and Wassenaar [16] and Ben-Daya and Al-Fawzan [17] developed SA algorithms for the single machine scheduling problem so as to minimize total tardiness. The total tardiness single machine problem that Tan and Narasimhan [18] considered assumed that the jobs are under the constraint of setup times. Crauwels et al. [19] introduced SA to solve a kind of single machine scheduling problem to minimize the sum of weighted completion times. Only a few papers have focused on parallel machine scheduling problems, Lee et al. [20] developed a SA algorithm for the identical parallel machine problem to minimize the makespan.

The remainder of this paper is organized as follows. In the next section, we describe $Q_n|L_{\text{max}}$ problem and $Q_n|q_j|C_{\text{max}}$ problem and analyze them. In the third section, we consider a simulated annealing (SA) approach to solve our problem. The accuracy and efficiency of the proposed algorithm are tested over 2400 random instances in the fourth section. Finally, conclusions are discussed in the Section 5.

2. Problem description and analysis

The problem under consideration is the problem of scheduling uniform parallel machines so as to minimize the maximal lateness. We are given a set of $n$ jobs, $J_1, \ldots, J_n$, each of them has to be scheduled without interruption on one of $m$ machines, $M_1, \ldots, M_m$. Machine $M_i$ ($i = 1, \ldots, m$) has a speed $s_i$ ($s_1 > 0$). Without loss of generality, we assume that $s_1 \geq s_2 \geq \cdots \geq s_m$. A machine can process at most one job at a time, and a job can run on only one machine at a time. All jobs and machines are available at time zero. If a job $J_j$ is processed on a machine $M_i$, it will take a positive processing time $p_{ij}$ and $p_{ij} = p_i/s_i$, here $p_i$ is the length of job $J_j$. Each job has a distinct due date $d_j$ for $j = 1, \ldots, n$. The objective is to determine a schedule so that the maximum lateness $L_{\text{max}} = \max_{1 \leq j \leq n} L_j$ is minimized, where $L_j = c_j - d_j$ is the lateness and $c_j$ is the completion time of job $J_j$.

Because the objective function $L_{\text{max}}$ is not always positive in $Q_n|L_{\text{max}}$ problems, the equivalent form $Q_n|q_j|C_{\text{max}}$ is frequently considered in the literature. Let $q_j = d_{\text{max}} - d_j$ for all $j$ and $d_{\text{max}} = \max_{1 \leq j \leq n} d_j$ is the maximum due date [21]. In this paper, we mainly focus on the $Q_n|q_j|C_{\text{max}}$ problem too. Here each job $J_j$ has a distinct positive delivery time $q_j$, that must elapse between its completion on the machine and its exit from the system. Let $c_j$ denote the completion time of job $J_j$ on a machine, then for the consumers the effective completion time $C_j = c_j + q_j$. The objective is to minimize the largest completion time $C_{\text{max}} = \max_{1 \leq j \leq n} C_j = \max_{1 \leq j \leq n} (c_j + q_j)$. Note that the makespan corresponds to $C_{\text{max}} = \max_{1 \leq j \leq n} c_j$, and it is different from the maximum completion time here.

Let “$J_j \rightarrow M_i$” denote that the job $J_j$ is processed on the machine $M_i$. “$C_{\text{max}}(J_j \rightarrow M_i, J_j \rightarrow M_j)$” is the larger completion time between job $J_j$ and $J_j$, when they are all scheduled on machine $M_i$, and the job $J_j$ is processed before job $J_j$. “$C_{\text{max}}(J_j \rightarrow M_i, J_j \rightarrow M_j)$” denotes the larger completion time between job $J_j$ and $J_j$, when they are proposed on the machine $M_i$ and $M_j$ respectively. The sequence of the jobs on the same machine forms a sub-schedule.

Theorem 1 indicates that the jobs with different delivery times should be sequenced by the LDT rule on each machine.

Theorem 1. There exists an optimal sequence where on each machine jobs are scheduled in non-increasing order of $q_j$ ($j = 1, 2, \ldots, n$).

According to Theorem 1, Koumas and Kyparisis [3] built an approximation algorithm LDT for $Q_n|q_j|C_{\text{max}}$ problem. However, they did not consider the influence of the length of jobs on a schedule. Theoretically, when the delivery times
are all identical, the LDT algorithm will degenerate to sequencing randomly, and the quality of the solutions will not be guaranteed.

**Theorem 2.** Only consider the two jobs $J_j$ and $J_{j'}$, $p_j + q_j \geq p_{j'} + q_{j'}$, the two machines $M_i$ and $M_{i'}$ are both free, let $s_i = k \geq 1$. Then $C_{\text{max}}(J_j \rightarrow M_i, J_{j'} \rightarrow M_{i'}) \leq C_{\text{max}}(J_j \rightarrow M_{i'}, J_{j'} \rightarrow M_i)$.

**Proof.**
\[
C_{\text{max}}(J_j \rightarrow M_i, J_{j'} \rightarrow M_{i'}) = \max\left(p_j/k + q_j, p_{j'} + q_{j'}\right)
\]
\[
C_{\text{max}}(J_j \rightarrow M_{i'}, J_{j'} \rightarrow M_i) = \max\left(p_j + q_j, p_{j'}/k + q_{j'}\right).
\]
Because $p_j/k + q_j \leq p_j + q_j$ and $p_{j'}/k + q_{j'} \leq p_{j'} + q_{j'}$, therefore, the conclusion holds. $\square$

We name Theorem 2 as the LPDT (Largest the sum of Processing time and Delivery Time first) rule. The LPDT rule considers the influence of the length and the delivery time of the jobs on a schedule simultaneously, and shows that the job with larger sum of length and delivery time should be scheduled to the faster machine. In the next section, we will develop a heuristic algorithm LPDT for $Q_m|q_j|C_{\text{max}}$ problem according to Theorem 2, and use it to generate initial solutions for our simulated annealing algorithm.

### 3. Development of the simulated annealing approach

#### 3.1. Initial solution

We develop the following heuristic algorithm LPDT for $Q_m|q_j|C_{\text{max}}$ problem according to Theorem 2.

**Algorithm LPDT for $Q_m|q_j|C_{\text{max}}$ problem:**

Step 1. Sequence the whole jobs according to the LPDT rule, and obtain the job list $\pi$. Set the sub-schedules on each machines to null.

Step 2. Pre-insert the head job in $\pi$ to each sub-schedule according to the LDT rule. Choose the machine with the least $C_{\text{max}}$ to process the head job (If the number of the machines that the least $C_{\text{max}}$ corresponds to is greater than 1, then we choose the slowest one). Delete the head job from the job list $\pi$.

Step 3. If the job list $\pi$ is null, then stop; otherwise, go to Step 2.

In the first step, the jobs are ordered by LPDT rule, therefore we can consider the influence of the length and the delivery time of the jobs simultaneously when we assign a job to a certain machine. However, the LPDT order cannot ensure that a sub-schedule on a machine is optimal. So we pre-sequence a job to a sub-schedule according to the LDT rule in Step 2.

#### 3.2. Solution representation

Obviously, the schedule for each individual machine in an optimal solution of $Q_m|q_j|C_{\text{max}}$ problem would make each sub-schedule optimized. Moreover, we can use the LPDT rule to obtain the optimal solution of $1|q_j|C_{\text{max}}$ problem. Therefore, we renumber the jobs in LDT order, and use a $n$-digit string to represent a solution, where the elements correspond to jobs 1, \ldots, n, respectively. A solution $\sigma$ is formulated as $\sigma = (m_1, \ldots, m_n)$, where $m_j$ ($j = 1, \ldots, n$) is the machine number that job $J_j$ is scheduled on. The cost $F(\sigma)$ represents the objective function value $C_{\text{max}}$, and it can be calculated out by calculating the maximum completion time for each sub-schedule in which no idle time is inserted between two adjacent jobs. Fig. 1 gives the corresponding relation between a string and a schedule for a problem with 2 machines and 5 jobs.

**3.3. Neighborhood generation**

In this paper, we define two rules, swap and insertion, to generate neighborhoods of the current solution. Based on our solution representation method proposed in Section 3.2, the two neighbor generation rules are easy to realize, and many obvious inferior solutions can be avoided, therefore the efficiency of the proposed SA algorithm can be improved.
Swap neighborhood \((N_1)\): Exchange the machines of a pair of selected jobs originally assigned to different machines. Note that it means to exchange the values of the corresponding two positions in a solution \(\sigma\). Fig. 2 describes the process to generate a swap neighbor on a solution \(\sigma\), and the corresponding change of a schedule. From Fig. 2, we can find that the swap neighborhood generation rule on a string can avoid the inferior schedules during the cause. For example, the sub-schedule of machine \(M_1\) is re-optimized by adjusting the order of jobs \(J_3\) and \(J_2\).

The limitation of swap neighborhood generation rule is that it cannot change the job number in a sub-schedule on a certain machine, i.e. the initial solution determines the job number of the sub-schedules. Therefore, the final solution of SA would not be optimal if the numbers of the sub-schedules in the initial solution are not exactly equal to the optimal solution. Here, we introduce the insertion neighborhood generation rule to change the job number of the sub-schedules.

Insertion neighborhood \((N_2)\): Select a job and insert it on a different machine. Obviously, it is unnecessary to select a job and insert it to the same sub-schedule, because the LDT rule can make the corresponding \(1\mid q_j\mid C_{\text{max}}\) problem optimized. An insertion neighbor means to change the value of the corresponding position to a job in a string \(\sigma\). Fig. 3 represents the process of the generation of an insertion neighbor on a string and the schedule.

### 3.4. Simulated annealing algorithm

The developed simulated annealing algorithm LPDT-SA is based on the algorithm LPDT. The parameters in the algorithm LPDT-SA are illuminated as follows.

1. The lower bound \(LB\) and \(K\) in Step 2. Koumas and Kyparisis [3] relaxed \(Q_m\mid q_j\mid C_{\text{max}}\) problem to the corresponding single machine problem, in which the speed of the single machine \(s = \sum_{i=1}^{m} s_i (i = 1, 2, \ldots, m)\), therefore it can obtain the optimal solution by the LDT rule, and notes its optimal value as \(LB_1\). It is obvious that for each job \(f_j\) \((j = 1, \ldots, n)\), its real
completion time $C_j = p_j/s_j + q_j$, and then $LB_2 = \max_{j=1}^n (p_j/s_j + q_j)$ is another lower bound for our $Q_m|q_j|C_{\text{max}}$ problem. Here, we define the lower bound $LB = \max(LB_1, LB_2)$. The value of $K$ affects the initial temperature, and it should be a large enough positive number to make the initial accepted probability be close to 1, here we set $K = 200$.

(2) The two positive numbers $\varepsilon_1$ and $\varepsilon_2$ in Step 3 are random small numbers near to zero. The lower value of $\varepsilon_1$ means that the current solution $\sigma$ is closer to the lower bound. Here we let $\varepsilon_1 = 0.00001$, but it can be changed in the real-world scheduling problems according to the degree of the importance of the problems. The value of the last temperature $\varepsilon_2$ would not reduce down to zero, because it is obtained by an annealing coefficient $\alpha$ to multiply the current temperature step by step. Here we set $\varepsilon_2 = 0.001$.

(3) $L$ is the iterative time under the same temperature. Here, we set $L = n^2/2$.

(4) The random number $r_1$ in Step 5 is used to control the selection between the swap and insertion neighborhood. If $r_1 < 0.5$, we adopt swap neighborhood, otherwise select an insertion neighbor.

(5) The random number $r_6$ in Step 10 decides if the current inferior solution is accepted.

(6) The annealing coefficient $\alpha$ in Step 11 should be controlled within (0, 1). For the balance between the accuracy and efficiency of LPDT-SA, we set an experimental value 0.6 to $\alpha$.

The steps of the proposed LPDT-SA algorithm are as follows.

**Algorithm LPDT-SA for $Q_m|q_j|C_{\text{max}}$ problem:**

**Step 1.** Construct an initial schedule by the LPDT heuristic. Describe it as $\sigma$ and compute the value of $F(\sigma)$.

**Step 2.** Compute the lower bound $LB$. Set the initial temperature $T := K \times (F(\sigma) - LB)$.

**Step 3.** If $(F(\sigma) - LB)/LB < \varepsilon_1$ or $T < \varepsilon_2$, then return $\sigma$ and $F(\sigma)$, stop; otherwise, go to Step 4.

**Step 4.** Set $L := n^2/2$.

**Step 5.** Generate a random number $r_1 \sim U(0, 1)$.

**Step 6.** If $r_1 < 0.5$, then go to Step 7; otherwise go to Step 8.

**Step 7.** Generate two different random numbers $r_2, r_3 \sim U(1, n)$. If the jobs $r_2$ and $r_3$ are not scheduled on the same machine in the current solution, then swap the corresponding two machine numbers. i.e. select a neighbor $\sigma'$ from $N_1(\sigma)$. Go to Step 9.

**Step 8.** Generate a random number $r_4 \sim U(1, n)$. Generate a random number $r_5 \sim U(1, m)$, which is not equal to the machine number that the jobs $r_4$ corresponds to. Change the machine number of job $r_4$ to $r_5$, i.e. select a neighbor $\sigma'$ from $N_2(\sigma)$.

**Step 9.** Set $\Delta F(\sigma) := F(\sigma') - F(\sigma)$, if $\Delta F(\sigma) < 0$, then $\sigma := \sigma'$ and go to Step 11.

**Step 10.** Generate a random number $r_6 \sim U(0, 1)$, if $\text{Exp}(-\Delta F(\sigma)/T) > r_6$, then $\sigma := \sigma'$.

**Step 11.** Set $L := L - 1$; If $L = 0$, then $T := \alpha \times T$, and go to Step 3; otherwise go to Step 5.

4. Computational experiments

To analyze and test the accuracy and efficiency of our algorithms, and compare them with the LDT heuristic and the lower bound in Section 3.4, we considered $Q_m|q_j|C_{\text{max}}$ problems with 100, 150, 200, 250, 300, 350 jobs, and 2, 4, 6, 8 machines. All algorithms mentioned above were implemented in C++ code using the Dev-C++ 4.9.9.0 compile environment. The environment of our experiments is CPU: Pentium IV 2.93 GHz, memory: 480 MB, operating system: Microsoft Windows XP SP1. All the experimental data were generated randomly by the computer according to the following settings.

For each job $j$, an integer processing time $p_j$ from the uniform distribution [1,100] was generated. For each machine $M_i$, an integer speed $s_i$ from the uniform distribution [1,10] was generated. Since the delivery times are likely to influence the efficiency of the algorithms and the lower bounds, the delivery times were generated from the uniform distribution $[0, \lambda \cdot (\sum_{j=1}^n p_j)/\sum_{j=1}^n s_j]$. Ten $\lambda$ values, $0.2, 0.4, 0.6, 0.8, 1.0, 1.25, 1.5, 1.75, 2.0, 3.0$, were considered. For each problem size and each value of $\lambda$, we did the experiment 10 times and observed the average, the best and the worst value of objective functions. Therefore, the algorithms were used to solve $4 \times 6 \times 10 \times 10 = 2400$ problems.

Let $C_{\text{max}}^H$ represent the value of the objective function of $S^H$, which is generated by the heuristic algorithm $H$, and $\text{Gap}(H) = \frac{C_{\text{max}}^H - LB}{LB} \cdot 100$ represents the gap between $C_{\text{max}}^H$ and the lower bound LB. For a certain problem, LB is lower than the optimal value. Therefore the value of Gap (H) is always lower than the relative error between $C_{\text{max}}^H$ and the optimal value too. For the simulated annealing algorithm LPDT-SA, the notion “Time (LPDT-SA)” denotes the CPU times for the corresponding problem size, and it is denoted as second. The notation “-” represents the CPU time is near to zero. LDT and LPDT are all polynomial, therefore the CPU times that they used are close to zero and can be ignored.

Table 1 provides details of the test results of the $Q_m|q_j|C_{\text{max}}$ problems with different $\lambda$ values. The data in Table 1 are taken over all 24 $(4 \times 6)$ combinations of $m$ and $n$, where the value of $m$ is in $\{2, 4, 6, 8\}$ and the value of $n$ is in $\{100, 150, 200, 250, 300, 350\}$. It is obvious that LPDT is more accurate than LDT. The average error between $C_{\text{max}}^{LPDT}$ and the corresponding lower bounds is 1.128%, while LPDT is 0.619%. For all of the ten $\lambda$ values, the average qualities of the LPDT solutions are always better than that of LDT solutions. One exceptional case is that $\lambda = 3$, the worst solution that LPDT generated is inferior to LDT, it shows the superiority of LDT when the delivery times have a distributed tendency. However, the LPDT algorithm...


Table 1
Computational results of algorithms with different values of $\lambda$.

<table>
<thead>
<tr>
<th>$\lambda$</th>
<th>$\text{Gap (LDT)}$</th>
<th>$\text{Gap (LPDT)}$</th>
<th>$\text{Gap (LPDT-SA)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average</td>
<td>Minimum</td>
<td>Maximum</td>
</tr>
<tr>
<td>0.2</td>
<td>1.652</td>
<td>0.003</td>
<td>7.587</td>
</tr>
<tr>
<td>0.4</td>
<td>1.721</td>
<td>0.004</td>
<td>7.529</td>
</tr>
<tr>
<td>0.6</td>
<td>1.944</td>
<td>0.040</td>
<td>9.238</td>
</tr>
<tr>
<td>0.8</td>
<td>1.980</td>
<td>0.000</td>
<td>7.885</td>
</tr>
<tr>
<td>1</td>
<td>1.922</td>
<td>0.000</td>
<td>9.831</td>
</tr>
<tr>
<td>1.25</td>
<td>0.891</td>
<td>0.000</td>
<td>5.636</td>
</tr>
<tr>
<td>1.5</td>
<td>0.519</td>
<td>0.000</td>
<td>3.410</td>
</tr>
<tr>
<td>1.75</td>
<td>0.348</td>
<td>0.000</td>
<td>3.346</td>
</tr>
<tr>
<td>2</td>
<td>0.226</td>
<td>0.000</td>
<td>2.482</td>
</tr>
<tr>
<td>3</td>
<td>0.074</td>
<td>0.000</td>
<td>1.668</td>
</tr>
</tbody>
</table>

Table 2
Computational results of algorithms with identical problem size.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$m$</th>
<th>$\text{Gap (LDT)}$</th>
<th>$\text{Gap (LPDT)}$</th>
<th>$\text{Gap (LPDT-SA)}$</th>
<th>$\text{Time (LPDT-SA)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Average</td>
<td>Minimum</td>
<td>Maximum</td>
<td>Average</td>
</tr>
<tr>
<td>100</td>
<td>2</td>
<td>0.580</td>
<td>0.000</td>
<td>1.535</td>
<td>0.393</td>
</tr>
<tr>
<td>150</td>
<td>2</td>
<td>0.364</td>
<td>0.000</td>
<td>1.179</td>
<td>0.196</td>
</tr>
<tr>
<td>200</td>
<td>2</td>
<td>0.411</td>
<td>0.000</td>
<td>0.897</td>
<td>0.141</td>
</tr>
<tr>
<td>250</td>
<td>2</td>
<td>0.190</td>
<td>0.000</td>
<td>0.724</td>
<td>0.114</td>
</tr>
<tr>
<td>300</td>
<td>2</td>
<td>0.148</td>
<td>0.000</td>
<td>0.597</td>
<td>0.095</td>
</tr>
<tr>
<td>350</td>
<td>2</td>
<td>0.125</td>
<td>0.000</td>
<td>0.473</td>
<td>0.081</td>
</tr>
<tr>
<td>400</td>
<td>2</td>
<td>0.104</td>
<td>0.000</td>
<td>0.396</td>
<td>0.067</td>
</tr>
</tbody>
</table>

considers the influence of the length and the delivery time of the jobs on the solutions simultaneously, therefore for most instances LPDT is obviously superior to the LDT algorithm. The limitation of the LDT algorithm becomes more apparent when the delivery times are more concentrated. For example, when $\lambda = 0.2$, the average error of the LDT algorithm is 1.652%, it is larger than that of LPDT 0.376% distinctly. Considering an extreme situation in which the jobs have the same delivery time (equal to 0), the $Q_m || q_j || C_{\text{max}}$ problem degenerates to the corresponding $Q_m || || C_{\text{max}}$ problem, LDT cannot guarantee the quality of the solutions by sequencing the jobs at random, while LPDT is equivalent to LPT for special problems.

Table 2 gives the computational results of the LDT, LPDT and LPDT-SA algorithms with identical problem size. For each problem size, we consider respectively ten values of $\lambda$. The efficiency of the LPDT-SA algorithm is strongly dependent on the problem size. Simulated annealing can improve the quality of the initial solutions that LPDT generates. Compared with the lower bound, LPDT-SA has an average gap 0.339% for all instances. The average CPU time that LPDT-SA used for all of the 2400 instances is only 17.763 s. For the 350-job problems, the worst CPU time of LPDT-SA is only 90 s. Therefore the efficiency of LPDT-SA is acceptable in lots of our practical production problems.

To solve the $Q_m || q_j || C_{\text{max}}$ problems in the real world, we give the following suggestions: Case 1, the size of the problem is less than 100 jobs, the exact solutions can be obtained by the traditional optimized method such as branch-and-bound and dynamic programming; Case 2, the size of the problem is large ($n > 100$) and the requirement of the solution quality is rigorous, it can be given near-optimal solutions by our LPDT-SA algorithm; Case 3, the real-time problems with a large size can be solved by the polynomial LPDT algorithm.
5. Conclusions

In this paper, we have developed a LPDT-SA algorithm for the uniform parallel machine scheduling problem with delivery times. The criterion is to minimize the largest completion time. The problem is equivalent to the uniform parallel machine scheduling problem to minimize the maximum lateness. A heuristic algorithm LPDT is proposed to generate initial solutions for the simulated annealing algorithm. A simple method is proposed to represent the solutions, and it can realize the swap and insertion neighborhood easily. The accuracy of the existing LDT algorithm, LPDT and LPDT-SA, and the efficiency of LPDT-SA are tested by numerous different instances. Heuristic LPDT is more accurate than LDT. LPDT-SA can solve the 350-job problems within 90 s, and the average error between the lower bounds for all 2400 random instances is 0.339%. Therefore, for large-scale problems, the LPDT-SA algorithm turned out to be very efficient and accurate.

Acknowledgements

The authors would like thank the anonymous referees for their valuable comments and suggestions that helped improve the paper. This work was supported by the fund from National Natural Science Foundation of China under grant number 70631003, 90924201, and 70971035 and by the fund from National 863 High-Tech Research and Development Program of China through approved number 2008AA042901.

References