A branch-and-bound algorithm with Lagrangian relaxation to minimize total tardiness on identical parallel machines

Shunji Tanaka*
Graduate School of Electrical Engineering, Kyoto University
Mituhiko Araki
Matsue National College of Technology

Abstract
The purpose of this paper is to propose a new branch-and-bound algorithm for a class of scheduling problems to minimize total tardiness on identical parallel machines. In this algorithm, Lagrangian relaxation technique is applied to obtain a tight lower bound. In addition, the job dominance conditions for the single machine total tardiness problem are utilized for both restricting branches and improving the lower bound. As will be shown by computational experiments, instances with up to 25 jobs and with any number of machines are optimally solved by the proposed algorithm.

Keywords: Scheduling, total tardiness, identical parallel machines, branch-and-bound algorithm, Lagrangian relaxation.

1 Introduction

In this paper a class of scheduling problems to minimize total tardiness on identical parallel machines \((P||\sum T_j)\) is studied. The single machine total tardiness problem \((1||\sum T_j)\) has been extensively studied so far. Emmons (1969) first showed that some precedence relations of jobs hold in an optimal schedule for \(1||\sum T_j\), which are referred to as Emmons’ dominance conditions. Lawler (1977) proposed a pseudopolynomial algorithm to solve \(1||\sum T_j\) based on his theorem called Lawler’s decomposition theorem. After the Lawler’s research, solution algorithms have been improved by several researchers. In 1999, Szwarc et al. reported that their algorithm can handle instances with up to 300 jobs, and in 2001, Tansel et al. proposed another algorithm that can handle instances with 500 jobs.

As for \(P||\sum T_j\), exact solution algorithms have been proposed by several researchers. Root (1965) treated the common duedate problem \(P|d_j=d||\sum T_j\), and Elmaghraby and Park (1974), and Barnes and Brennan (1977) treated the problem \(P|p_{ij}||\sum T_j\). However, these researches targeted special classes of \(P||\sum T_j\). To the authors’ knowledge, Azizoglu and Kirca (1998) first treated the general problem \(P||\sum T_j\). They proposed a method to compute a lower bound by allowing job preemption and simultaneous job processing, and constructed a branch-and-bound algorithm. However, only instances with up to 12 jobs and 3 machines were optimally solved by their algorithm (they claims that it can with up to 15 jobs, but there were unsolved instances). Yalaoui and Chu (2002) improved their algorithm by introducing some job dominance checks and problem instances with up to 15 jobs and 3 machines were optimally solved. The most recent research by Liaw et al. (2003) targeted the more general problem \(R||\sum w_jT_j\), minimization of total weighted tardiness on unrelated parallel machines. In their algorithm a lower bound is computed based on the assignment problem approximation and it could optimally solve instances with up to 18 jobs and 4 machines.

In this study we propose a more efficient branch-and-bound algorithm for \(P||\sum T_j\). In our algorithm, problem decomposition by Lagrangian relaxation is applied to compute a lower bound because it gives tight lower bounds for scheduling problems (e.g. Fisher (1981) and Luh et al. (1990)). Another point of our algorithm is that the Emmons’ dominance conditions are utilized for both restricting branches and improving the lower bound. Computational experiments will show that our algorithm can handle instances with up to 25 jobs and with any number of machines.

*Corresponding author. Kyotodaigaku-Katsura, Nishikyo-ku, Kyoto 615–8510 JAPAN. Phone: +81-75-383-2204, Fax: +81-75-383-2201, E-mail: tanaka@kuee.kyoto-u.ac.jp
This paper is organized as follows. In Section 2, some properties of an optimal schedule for our problem are presented. In Section 3, problem decomposition by Lagrangian relaxation is introduced to compute a lower bound. Then, a branch-and-bound algorithm based on these properties and a lower bound computation method is proposed in Section 4. The effectiveness of the algorithm is examined in Section 5 by several computational experiments. Finally, our results are summarized in Section 6.

2 Total tardiness problem on identical parallel machines

In this section, we first give an explicit description of our problem, the total tardiness problem on identical parallel machines. Next, we present some properties of an optimal schedule for this problem.

Consider that a set of \( n \) jobs \( J = \{ J_1, \ldots, J_n \} \) are to be processed on \( m \) identical parallel machines \( M_1, \ldots, M_m \). Each job \( J_j \) is given the integer processing time \( p_j \) and the integer due date \( d_j \). All the jobs are available at time zero, and no job preemption is allowed. The tardiness \( T_{j} \) of \( J_j \) is given by \( T_{j} = \max(C_{j} - d_{j}, 0) \), where \( C_{j} \) is the completion time of \( J_j \). The objective is to search an optimal schedule that minimizes the total tardiness \( \sum_{j=1}^{n} T_{j} \). This problem is referred to as \( P|| \sum T_{j} \) according to the standard classification of scheduling problems.

Since total tardiness is a nondecreasing function of job completion times, there exists an optimal schedule where no idle times are inserted between jobs. Moreover, an optimal schedule can be constructed by assigning jobs on earliest available machines one by one according to an optimal job priority list. In the following, this construction procedure is denoted by \( D(L) \) (L is a job priority list), and a set of all the schedules constructed by \( D(\bullet) \) is denoted by \( S_D \). It is not difficult to see that job completion times in any schedule belonging to \( S_D \) have the following property.

**Corollary 1** In any schedule belonging to \( S_D \), the completion time \( C_j \) of \( J_j \) (1 \( \leq \) \( j \) \( \leq \) \( n \)) satisfies

\[
C_j \leq \frac{1}{m} \sum_{i=1}^{n} p_i + \frac{m-1}{m} p_j.
\]  

**Proof of Corollary 1.** Consider a schedule belonging to \( S_D \) and assume without loss of generality that \( J_1 \) is assigned on \( M_1 \). Since \( D(\bullet) \) assigns jobs to earliest available machines, the starting time of \( J_j \) should be not less than total processing times on the other machines. Therefore,

\[
C_j \leq P_k + p_j \quad (2 \leq k \leq m),
\]

should be satisfied where \( P_k \) denotes the total processing time of jobs assigned on \( M_k \). By summing up (2) for all \( k \) (2 \( \leq \) \( k \) \( \leq \) \( m \)), we obtain

\[
(m-1)C_j \leq \sum_{k=2}^{m} P_k + (m-1)p_j.
\]

If we note that \( C_j \leq P_1 \) and \( \sum_{k=1}^{m} P_k = \sum_{i=1}^{n} p_i \), (3) becomes

\[
mC_j \leq \sum_{i=1}^{n} p_i + (m-1)p_j.
\]

By dividing (4) by \( m \), we obtain (1).

It should be noted that similar results have been given by Azizoglu and Kirca (1998) and Liaw et al. (2003), but there is a slight difference between them and Corollary 1. The previous results concentrate on the sum of the processing time on each machine, and Corollary 1 on the completion time of each job. Corollary 1 is more appropriate for our purpose, which will be stated in 4.1 and Appendix B.

The following corollary states job dominance properties in an optimal schedule. It is a direct result of the Emmons’ dominance conditions, but plays an important role to restrict the search space in our branch-and-bound algorithm.

**Corollary 2** There exists an optimal schedule (belonging to \( S_D \)) for \( P|| \sum T_{j} \) such that for any pair of jobs \( J_j \) and \( J_k \) (\( C_j < C_k \)) assigned on a same machine,
\(d1\)  \(d_k > \max(C_j, d_j)\) if \(p_j > p_k\),
\(d2\)  \(d_j \leq \max(C_k - p_j, d_k)\) if \(p_j < p_k\),
\(d3\)  \(d_j \leq d_k\) if \(p_j = p_k\).

Although \(P\|\sum T_j\) itself is NP-hard, there are some instances of \(P\|\sum T_j\) that are known to be polynomially solvable. The following corollary gives the optimality of the schedule constructed by \(D(L^{SPT}(J))\), where \(L^{SPT}(J)\) denotes a job priority list sorted in SPT (Shortest Processing Time) order.

**Corollary 3** (Koulamas (1997)) *If all the jobs are tardy (including just-in-time) in the schedule constructed by \(D(L^{SPT}(J))\), it is optimal.***

This corollary can be extended to the case when not all the machines are available from time zero. It is used for a fathoming test in our branch-and-bound algorithm. The details will be stated in Subsection 4.3.

**Lemma 1** *Assume that \(M_j (1 \leq j \leq m)\) is available from \(R_j \geq 0\). If all the jobs are tardy in the schedule constructed by \(D(L^{SPT}(J))\), this schedule minimizes the total tardiness.***

The proof is given in Appendix A.

## 3 Lagrangian relaxation of \(P\|\sum T_j\)

In this section, Lagrangian relaxation is applied to \(P\|\sum T_j\) according to Luh *et al.* (1990). This technique enables us to decompose the original problem into relatively easy subproblems by relaxing “coupling” constraints via Lagrangian multipliers. It is known that the Lagrangian relaxation technique yields a tight lower bound for parallel machine scheduling problems (Luh *et al.* (1990)). Moreover, it enables us to construct good feasible schedules (upper bounds) from the solutions of the Lagrangian dual obtained in the course of subgradient optimization. These lower and upper bounds are utilized in the proposed branch-and-bound algorithm.

The problem \(P\|\sum T_j\) stated in Section 2 can be formulated by the following binary integer programming problem.

\[
(P) : \quad F = \min_x \sum_{j=1}^{n} \sum_{t=0}^{E_j} W_{jt} x_{jt},
\]

subject to
\[
x_{jt} \in \{0, 1\} \quad (1 \leq j \leq n, \ 0 \leq t \leq E_j),
\]
\[
\sum_{t=0}^{E_j} x_{jt} = 1 \quad (1 \leq j \leq n),
\]
\[
\sum_{j=1}^{n} \sum_{s=\max(t-p_j+1,0)}^{\min(t,E_j)} x_{js} \leq Q_t \quad (0 \leq t \leq T_{\text{max}}).
\]

Here, \(x_{jt}\) are decision variables such that
\[
x_{jt} = \begin{cases} 
1 & \text{if } J_j \text{ is started at } t, \\
0 & \text{otherwise},
\end{cases}
\]

and the constants \(E_j, T_{\text{max}}, W_{jt}\) and \(Q_t\) are given by
\[
E_j: \text{The latest possible starting time of } J_j \ (E_j := (\sum_{i=1}^{n} p_i - p_j - 1)/m),
\]
\[
T_{\text{max}}: T_{\text{max}} = \max_{1 \leq j \leq n} (E_j + p_j - 1),
\]
\[
W_{jt}: \text{The tardiness of } J_j \text{ when it is started from } t \ (W_{jt} = \max(t + p_j - d_j, 0)),
\]
\[
Q_t: \text{The number of machines available in the interval } [t, t+1) \ (Q_t = m).
\]

Now, the machine resource constraint (8) in (P) is relaxed by introducing nonnegative Lagrangian multipliers \(\mu_t\) \((0 \leq t \leq T_{\text{max}})\). Then, the relaxed problem becomes:

\[
(LR) : \quad F^*(\mu) = \min F(\mu),
\]

subject to (6) and (7),

\[\]

3
\[
F(\mu) = \sum_{j=1}^{n} \sum_{t=0}^{E_j} W_{jt} x_{jt} - \sum_{t=0}^{T_{\text{max}}} \mu_t \left( Q_t - \sum_{j=1}^{n} \sum_{s=\max(t-p_j+1,0)}^{\min(t,E_j)} x_{js} \right)
\]

\[
= - \sum_{t=0}^{T_{\text{max}}} \mu_t Q_t + \sum_{j=1}^{n} F_j(\mu),
\]

\[
F_j(\mu) = \sum_{t=0}^{E_j} W_{jt} x_{jt} + \sum_{t=0}^{T_{\text{max}}} \mu_t \sum_{s=\max(t-p_j+1,0)}^{\min(t,E_j)} x_{js}
\]

\[
= \sum_{t=0}^{E_j} (W_{jt} + \sum_{s=t}^{t+p_j-1} \mu_s)x_{jt}.
\]

(12)

The minimization in (10) can be performed separately with regard to \( j \). Moreover, the following trivial subproblems corresponding to \( J_j (1 \leq j \leq n) \) are obtained by noting the constraints (6) and (7).

\[
(LR_j): \quad F^*_j(\mu) = \min_{x_j} F_j(\mu) = \min_{0 \leq t_j \leq E_j} W_{jt} + \sum_{s=t_j}^{t_j+p_j-1} \mu_s.
\]

(13)

Therefore, \( F^*(\mu) \) for a fixed set of Lagrangian multipliers can be easily computed in \( O(nT) \) time.

The Lagrangian dual (D) of (LR) is given by

\[
(D): \quad \mathcal{F} = \max_{\mu} F^*(\mu) = \max_{\mu} \left( - \sum_{t=0}^{T_{\text{max}}} \mu_t Q_t + \sum_{j=1}^{n} F^*_j(\mu) \right),
\]

subject to (6) and (7).

(14)

(15)

To solve (D), subgradient optimization is applied as many other researches. More specifically, a set of the multipliers at \( (n+1) \)-th iteration, \( \mu^{n+1} \), is determined by \( \mu^n \) as follows:

\[
\mu^{n+1} = \mu^n + \alpha^n g(\mu^n),
\]

(16)

where \( g(\mu^n) \) is a subgradient of \( F^*(\mu^n) \) with respect to \( \mu^n \) and \( \alpha^n \) is the \( n \)-th step size. The step size is chosen as

\[
\alpha^n = \lambda \frac{\mathcal{F} - F^*(\mu^n)}{g(\mu^n)^T g(\mu^n)}.
\]

(17)

Here, \( \mathcal{F} \) is an upper bound of \( F \) and \( \lambda \) is the step size parameter.

### 4 A branch-and-bound algorithm

In this section, we state our branch-and-bound algorithm in details. As already mentioned in Section 2, we only need to consider the schedules belonging to \( \mathcal{S}_P \). Therefore, our algorithm searches an optimal priority list in the depth-first manner as in the previous researches by Azizoglu and Kirca (1998) and Yalaoui and Chu (2002). In the following, we first state how to obtain initial upper and lower bounds for the algorithm. Next, branch restriction based on Corollary 2 and a fathoming test based on Lemma 1 are presented. Then, a method to compute lower bounds for subproblems is explained.

#### 4.1 Initial upper and lower bounds

First, how to compute initial upper and lower bounds at the root node of the branch-and-bound algorithm is stated. As already explained in the preceding section, Lagrangian relaxation gives a tight lower bound for \( P||\sum T_j \). Indeed, there are no duality gaps for most problem instances as will be shown in Section 5. Therefore, a good feasible schedule (a tight upper bound) and a tight lower bound are searched at the root node so that an optimal schedule could be obtained without branching.

To search tight upper and lower bounds, the following procedure is applied.
1° From Corollary 1, if
\[ d_j \geq \frac{1}{m} \sum_{i=1}^{n} p_i + \frac{m-1}{m} p_j, \]
is satisfied, \( J_j \) is always non-tardy in any schedule belonging to \( S_D \). Therefore, such non-tardy jobs are removed in order to reduce the problem size (see Appendix B). Redefine \( J \) by the set of the remaining jobs and \( n \) by the number of the remaining jobs.

2° Construct a schedule by \( D(L^{SPT}(J)) \). If all the jobs are tardy in this schedule, the procedure is terminated because it is optimal from Corollary 3. Otherwise, a better schedule is searched from this schedule by a local search explained later. If all the jobs are on time, the schedule is optimal and the procedure is also terminated. Denote the total tardiness of the obtained schedule by \( \overline{T}^{SPT} \).

3° Construct a schedule by the KPM heuristics proposed by Koulamas (1994). Then, the local search is applied. If all the jobs are on time in the obtained schedule, the procedure is terminated. Otherwise, denote the total tardiness of the schedule by \( \overline{T}^{KPM} \).

4° Let \( \overline{F} := \min(\overline{T}^{SPT}, \overline{T}^{KPM}) \). Apply Lagrangian relaxation and maximize \( F^*(\mu) \) by subgradient optimization as explained in Section 3. The initial multipliers \( \mu^0 \) is chosen as \( \mu^0 := 1.0 \), and the step size parameter \( \lambda \) in (17) is set to be \( \lambda := 2.0 \) initially. At each step of the subgradient optimization, a schedule is constructed from a solution corresponding to \( F^*(\mu^*) \) by resolving conflicts heuristically. \( \overline{F} \) is updated if necessary.

If \( F^*(\mu^*) \) is not updated for 20 successive iterations, \( \lambda \) is scaled by \( \lambda := 0.99\lambda \). The subgradient optimization is terminated if one of the following termination conditions is satisfied.

1° \( \overline{F} - F^*(\mu^*) < 1 \) (\( \overline{F} \) is optimal because the optimal solution is integer).

2° \( F^*(\mu^*) \) is not updated for 600 successive iterations.

3° \( \lambda < 10^{-4} \).

Denote the obtained multipliers by \( \mu^* \) and let \( F := F^*(\mu^*) \).

Here, additional explanations are given on the above procedure. In 2° and 3°, feasible schedules are constructed heuristically. Although there are several types of simple heuristics proposed for \( P||\sum T_j \) (e.g. Wilkerson and Irwin (1971), Ho and Chang (1991) and Koulamas (1994)), the KPM heuristics seems to be the best among them according to Koulamas (1994). It motivates us to adopt the KPM heuristics to generate an initial upper bound. It can be also expected that \( D(L^{SPT}(J)) \) generates a good schedule since the schedule is optimal if all the jobs are tardy in that schedule. The time complexities of these two heuristics are \( O(n^2m) \) and \( O(n \log(nm)) \), respectively.

These two schedules themselves are not good enough to be adopted as an initial upper bound. Thus, a simple local search is applied to improve these schedules. The neighborhood structure of this search is given by

Insertion: A job is moved into another position, on the same machine or on another machine.

Exchange: Two jobs on the same machine or on different machines are exchanged.

The current schedule is updated by the best schedule in the neighborhood until no better schedule can be found. Next, by using the obtained upper bound \( \overline{F} \), subgradient optimization is applied to the Lagrangian dual (D) in 4°. As suggested by Fisher (1981), the step size parameter \( \lambda \) is reduced when the solution of (D) is not updated. The number of iterations for reducing \( \lambda \) and the number of iterations for terminating the subgradient optimization are determined by preliminary computational experiments.

At each step of the subgradient optimization, \( F^*(\mu^*) \), a solution of (LR), is obtained. Since (LR) is a relaxed problem of (P), it is in general not feasible for (P). However, it can be used for constructing a feasible solution of (P) heuristically. A heuristic algorithm in this study is a modified version of the one proposed by Luh et al. (1990). Let us denote by \( t_j^0 \) the starting time of \( J_j \) that minimizes \( F_j(\mu^*) \) in the decomposed subproblem (LRj). Jobs are scheduled on the earliest available machines one by one in the nondecreasing order of \( t_j^0 \) if no conflict occurs (\( t_j^0 \) is not less than the earliest machine release time). If some conflict occurs, a job is chosen from amongst the conflicting jobs by the following rule.

1° If there exists at least one conflicting job that is tardy when scheduled on the earliest available machine, a tardy job with the smallest processing time is chosen.
(2) If every conflicting job is non-tardy when scheduled on the earliest available machine, a job that can be completed as close as possible to its due date is chosen.

Then, the upper bound \( \bar{T} \) is updated if necessary.

4.2 Branching

Branching is performed by fixing the elements of the priority list from the first to the last. Thus, a subproblem corresponding to a node at depth \( l \) is to determine the last \( (n-l) \) elements of the priority list, and branching is performed at this node by fixing the \( (l+1) \)-th job in the priority list. These branches can be restricted by the dominance conditions in Corollary 2. Let us denote by \( L^l \) a partial priority list of length \( l \) corresponding to a node at depth \( l \), and by \( S^l \) the partial schedule constructed by \( D(L^l) \). The set of the \( (n-l) \) unscheduled jobs is denoted by \( U^l \). Let us denote the set of jobs assigned on \( M_k \) \( (1 \leq k \leq m) \) in the partial schedule \( S^l \) by \( A^l_k \), and the completion time of \( J_j \in M_k \) \( (1 \leq k \leq m) \) in \( S^l \) simply by \( C_j \). Then, the total processing time on \( M_k \) in \( S^l \), \( c_k^l \), is defined by

\[
c_k^l = \sum_{j \in A_k^l} p_j = \max_{j \in A_k^l} C_j. \tag{19}\]

Let us further define \( m^l = \arg \min_k c_k^l \). Since the \( (l+1) \)-th job in the priority list is scheduled on \( M_{m^l} \), it should satisfy the conditions (d1), (d2) and (d3) in Corollary 2 on \( M_{m^l} \). Therefore, if an unscheduled job \( J_u \in U^l \) breaks at least one of the dominance conditions on \( M_{m^l} \), it cannot be a candidate for the \( (l+1) \)-th job in the priority list. It is summarized as follows.

**Rule 1** If \( J_u \in U^l \) satisfies at least one of the following three conditions for some \( j \) \( (J_j \in A^l_{m^l}) \), \( J_u \) cannot be a candidate for the \( (l+1) \)-th job in the priority list.

1. \( p_u < p_j \) and \( d_u \leq \max(C_j, d_j) \).
2. \( p_u > p_j \) and \( d_j \geq \max(c_{m^l}^l + p_u - p_j, d_u) \).
3. \( p_u = p_j \) and \( d_u < d_j \).

Corollary 2 can derive another type of branch restriction. The basic idea is as follows. Since all the unscheduled jobs are to be scheduled on some machines after all, they should satisfy the conditions in Corollary 2 on the machines where they are scheduled. It follows that if an unscheduled job is known to break at least one of the conditions on all the machines, it cannot be scheduled to any machines. Therefore, the candidates for the \( (l+1) \)-th job in the priority list should be such that there is no unscheduled job that breaks the dominance conditions on all the machines. It is summarized as follows.

**Rule 2** Consider that \( J_u \in U^l \) is chosen as a candidate for the \( (l+1) \)-th job in the priority list. That is, the partial schedule \( S^{l+1} \) (the partial schedule obtained by adding \( J_u \) to \( S^l \)) satisfies

\[
A_k^{l+1} = A_k^l \cup \{J_u\} \quad \text{if } k = m^l, \\
A_k^{l+1} = A_k^l \quad \text{otherwise}, \tag{20}\]

and \( C_u = c_{m^l}^l + p_u \). If there exists \( J_w \in U^l \setminus \{J_u\} \) such that for all \( k \) \( (1 \leq k \leq m) \) and for some \( j \) \( (J_j \in A_k^{l+1}) \), at least one of the following conditions is satisfied, \( J_u \) cannot be a candidate for the \( (l+1) \)-th job in the priority list.

1. \( p_w < p_j \) and \( d_w \leq \max(C_j, d_j) \).
2. \( p_w > p_j \) and \( d_j \geq \max(E_w + p_w - p_j, d_w) \).
3. \( p_w = p_j \) and \( d_w < d_j \).

Here, \( E_w \) is the latest possible starting time of \( J_w \) (see Section 3).

4.3 Fathoming test by Lemma 1

From Lemma 1, we can see that if all the jobs belonging to \( U^l \) are tardy in the schedule constructed by applying \( D(U^{SPT}(U^l)) \) to the partial schedule \( S^l \), it is optimal under the condition that \( S^l \) is fixed. Therefore, in such a case the node is fathomed and the incumbent solution is updated if necessary.
4.4 Lower bound computation

In the proposed algorithm, two types of lower bounds for the total tardiness of the unscheduled jobs $U$ are considered. The one is based on Lemma 1. In this lower bound, only those unscheduled jobs that satisfy the condition of Lemma 1 are considered. More specifically, a set of jobs $T_i \subset U$ is chosen so that all the jobs in $T_i$ are tardy in the schedule constructed by applying $D(LSP^T(T_i))$ to $S_i$. Then, the total tardiness of the jobs in $T_i$ in this schedule is used as a lower bound of the total tardiness of the jobs in $U$. A procedure to compute this lower bound is given by the following.

1. Remove a job with the smallest processing time from $U^B$. Denote this job by $J_j$.
2. $k := \arg \min_i c^B_i$. If $c^B_k + p_j < d_j$, go to 4°.
3. $LB := LB + c^B_k + p_j$, $c^B_k := c^B_k + p_j$.
4. If $U^B \neq \emptyset$, go to 1°. Otherwise, output $LB$ as a lower bound and terminate.

The other lower bound is based on Lagrangian relaxation explained in Section 3. The subproblem to minimize the total tardiness of the unscheduled jobs $U$ is formulated by

$$\text{(SP'): \quad \min \sum_{j \in U^t} \sum_{t = r_j^i}^{E_j} W_j x_{jt},}$$

subject to

$$x_{jt} \in \{0, 1\} \quad (J_j \in U^t, \quad r_j^i \leq t \leq E_j),$$

$$\sum_{t=0}^{E_j} x_{jt} = 1 \quad (J_j \in U^t),$$

$$\sum_{J_j \in U^t \:\\ s = \max (t-p_j+1, r_j^i)} \min(t, E_j) \sum x_{js} \leq Q^l_t \quad (r^i_{\min} \leq t \leq T_{\max}).$$

Here, $Q^l_t$ denotes the number of machines available in the interval $[t, t + 1)$. $r_j^i$ denotes the earliest possible starting time of $J_j$ corresponding to the partial schedule $S_i$, and $r^i_{\min}$ is defined by $r^i_{\min} := \min_{J_j \in U^t} r^i_j$. A lower bound of the total tardiness of the jobs in $U$ can be obtained by solving the Lagrangian dual corresponding to (SP'). However, it takes a considerable number of iterations for the convergence of subgradient optimization. For this reason, in the previous researches that utilize Lagrangian relaxation in branch-and-bound algorithms (e.g. Fisher (1981) and Babu et al. (2004)) subgradient optimization is performed for a small number of iterations where multipliers are initialized by those obtained at the parent nodes. On the other hand, in this study subgradient optimization is performed only at the root node because in our algorithm multipliers are searched for a sufficient number of iterations at the root node and thus they are not expected to be improved much only by a small number of iterations. For every (SP'), a lower bound is computed by fixing the multipliers to those obtained at the root node.

To improve this lower bound, $r^i_u$, the earliest possible starting time of $J_u \in U^t$, is restricted by the dominance conditions. From the first and third conditions in Rule 1, it is not necessary to consider schedules such that $J_u$ is assigned on $M_k$, if for some $j$ ($J_j \in A^k_i$), $J_u$ satisfies

$$p_u < p_j, \quad d_u \leq \max(C_j, d_j)$$

or

$$p_u = p_j, \quad d_u < d_j.$$  

If we denote by $M_u^l$ the set of the machines such that none of these conditions are satisfied, it follows that $r^i_u$ is given by

$$r^i_u = \min_{M_k \in M_u^l} c^l_k.$$  

Next, let us consider the second condition in Rule 2. Since $c^l_k$ corresponds to the starting time of $J_u$ when it is scheduled on $M_k$, it can be interpreted as

$$p_u > p_j, \quad d_j > \max\{(The \ starting \ time \ of \ J_u) + p_u - p_j, d_u\}.  $$
Therefore, for \( J_u \) to be assigned on \( M_k \), it should satisfy
\[
\text{(The starting time of } J_u \text{) } \geq d_j + p_j - p_u
\]  
for any job \( J_j \in A_k \) with \( p_u > p_j \) and \( d_u < d_j \). Thus, the earliest possible starting time of \( J_u \) on \( M_k \) is given by
\[
\max \left\{ \max_{p_u > p_j, d_u < d_j} (d_j + p_j - p_u), c_k^l \right\}.
\]  
Taking the minimum of (30) on \( M_k \in M^l \), we obtain
\[
r_u^l = \min_{M_k \in M^l} \max \left\{ \max_{p_u > p_j, d_u < d_j} (d_j + p_j - p_u), c_k^l \right\}
\]  
If none of the dominance conditions are taken into account, \( r_u^l \) is given simply by
\[
r_u^l = \min_{1 \leq k \leq m} c_k^l.
\]  
Since (31) is not less than (32), \( r_u^l \) is restricted by the dominance conditions, and hence the lower bound is expected to be improved.

The time complexity to compute \( r_u^l \) for all the unscheduled jobs \( J_u \in U^l \) is \( O(n \log m) \) if the computation results at the parent node can be utilized. It is because the value of (30) only varies on \( M_{m+1} \), the machine on which the \((l-1)\)-th job in the priority list is assigned, from that at the parent node. On the other hand, the time complexity to compute \( r_u^l \) by (32) (for all \( J_u \in U^l \)) is \( O(\log m) \).

To solve the Lagrangian relaxation of (\( SP^l \)) for a given set of \( r_u^l \), the decomposed subproblem
\[
\min_{r_u^l, t_j \leq t_j \leq E_j, j} \left( W_{f_j t_j} + \sum_{s=t_j}^{t_j+p_j-1} \mu_s \right)
\]  
for every job \( J_j \) should be solved. It can be done in \( O(n) \) time if the value of (33) for every possible \( r_u^l \) is computed and stored in advance (at the root node). Therefore, the overall time complexities of lower bound computation with (31) and (32) are given by \( O(n \log m) \) and \( O(n + \log m) \), respectively. It follows that the time complexity for lower bound computation slightly increases if (31) is used. However, as will be shown in Section 5, the effect of the improvement of the lower bound dominates the increase of the time complexity.

5 Computational experiments

The efficiency of the proposed algorithm is examined by computational experiments. Computation is performed on a Pentium4 2.4GHz desktop computer.

Problem instances are generated by the standard method by Fisher (1976). First, the integer processing times \( p_j \) (1 \( \leq j \leq n \)) are generated by the uniform distributions in \([1, 100]\). Then, let \( P = \sum_{j=1}^{n} p_j \) and the integer due dates \( d_j \) (1 \( \leq j \leq n \)) are generated by the uniform distributions in \([P(1 - \tau - R/2)/m, P(1 - \tau + R/2)/m]\). The number of jobs \( n \), the number of machines \( m \), the tardiness factor \( \tau \) and the range of due dates \( R \) are changed by \( n = 20, 25, m = 2, 3, 4, 5, 6, 7, 8, 9, 10, \tau = 0.2, 0.4, 0.6, 0.8, 1.0, \) and \( R = 0.2, 0.4, 0.6, 0.8, 1.0. \) For every combination of \( n, m, \tau \) and \( R \), five problem instances are generated. Thus, for each combination of \( n \) and \( m \), 125 problem instances are generated.

The results are shown in Table 1. Here, “root time”, “total time” and “node” respectively denote the average and maximum of CPU times at the root node, the average and maximum of total CPU times in seconds, and the average and maximum numbers of generated nodes. \( N_1, N_2, N_3 \) and \( N_4 \) denote the numbers of instances such that
\( N_1 \): the problem instance is solved without lower bound computation,
\( N_2 \): the problem instance is solved at the root node (including \( N_1 \)),
\( N_3 \): the gap between the initial lower bound and the optimal solution is less than 1 (including \( N_1 \)),
\( N_4 \): the initial upper bound and the optimal solution are identical (including \( N_1 \)),
where \( N_1 \) includes the following three cases:
Table 1: Performance of the proposed algorithm

(a) \( n = 20 \)

<table>
<thead>
<tr>
<th>( m )</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>root time</td>
<td>ave.</td>
<td>0.644</td>
<td>0.348</td>
<td>0.170</td>
</tr>
<tr>
<td>max.</td>
<td>2.282</td>
<td>1.291</td>
<td>0.882</td>
<td>0.659</td>
</tr>
<tr>
<td>total time</td>
<td>ave.</td>
<td>0.656</td>
<td>0.413</td>
<td>0.206</td>
</tr>
<tr>
<td>max.</td>
<td>2.582</td>
<td>4.467</td>
<td>4.416</td>
<td>0.662</td>
</tr>
<tr>
<td>node</td>
<td>max.</td>
<td>487004</td>
<td>5877366</td>
<td>6334218</td>
</tr>
<tr>
<td>( N_1/N_2/N_3/N_4 )</td>
<td>17/61/61/105</td>
<td>15/69/71/104</td>
<td>14/84/86/111</td>
<td>12/94/101/109</td>
</tr>
</tbody>
</table>

(b) \( n = 25 \)

<table>
<thead>
<tr>
<th>( m )</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>root time</td>
<td>ave.</td>
<td>0.075</td>
<td>0.058</td>
<td>0.035</td>
<td>0.051</td>
</tr>
<tr>
<td>max.</td>
<td>0.540</td>
<td>0.495</td>
<td>0.389</td>
<td>0.423</td>
<td>0.476</td>
</tr>
<tr>
<td>total time</td>
<td>ave.</td>
<td>0.076</td>
<td>0.059</td>
<td>0.035</td>
<td>0.145</td>
</tr>
<tr>
<td>max.</td>
<td>0.565</td>
<td>0.503</td>
<td>0.393</td>
<td>10.886</td>
<td>6.986</td>
</tr>
<tr>
<td>node</td>
<td>max.</td>
<td>2473</td>
<td>1767</td>
<td>705</td>
<td>149048</td>
</tr>
</tbody>
</table>

\( N_1 \): solved without lower bound computation, \( N_2 \): solved at the root node, \( N_3 \): OPT-LB<1, \( N_4 \): UB=OPT
Table 2: Gaps between optimal solutions and proposed lower bounds

(a) \(n = 20\)

<table>
<thead>
<tr>
<th>(m)</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of instances</td>
<td>64</td>
<td>56</td>
<td>41</td>
<td>31</td>
</tr>
<tr>
<td>((\text{OPT-LB})/\text{OPT})</td>
<td>ave. 1.850</td>
<td>0.908</td>
<td>0.708</td>
<td>0.908</td>
</tr>
<tr>
<td></td>
<td>max. 29.269</td>
<td>5.485</td>
<td>3.500</td>
<td>7.682</td>
</tr>
<tr>
<td>((\text{OPT-LB}))</td>
<td>ave. 8.806</td>
<td>4.632</td>
<td>3.350</td>
<td>2.763</td>
</tr>
<tr>
<td></td>
<td>max. 32.464</td>
<td>16.726</td>
<td>10.501</td>
<td>8.339</td>
</tr>
</tbody>
</table>

(b) \(n = 25\)

<table>
<thead>
<tr>
<th>(m)</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of instances</td>
<td>24</td>
<td>21</td>
<td>7</td>
<td>13</td>
<td>8</td>
</tr>
<tr>
<td>((\text{OPT-LB})/\text{OPT})</td>
<td>ave. 0.647</td>
<td>1.381</td>
<td>0.300</td>
<td>0.433</td>
<td>0.067</td>
</tr>
<tr>
<td></td>
<td>max. 2.386</td>
<td>6.279</td>
<td>0.598</td>
<td>1.423</td>
<td>0.356</td>
</tr>
<tr>
<td>((\text{OPT-LB}))</td>
<td>ave. 3.185</td>
<td>2.297</td>
<td>2.139</td>
<td>1.279</td>
<td>0.305</td>
</tr>
<tr>
<td></td>
<td>max. 8.662</td>
<td>5.610</td>
<td>5.009</td>
<td>3.361</td>
<td>1.170</td>
</tr>
</tbody>
</table>

- all the jobs are removed by the algorithm in Appendix B,
- all the jobs are tardy in the schedule constructed by \(D(L^{\text{SPT}}(J))\),
- all the jobs are on-time in the schedule constructed by \(D(L^{\text{SPT}}(J))\) or the KPM heuristics.

From Table 1, we can see that all the instances with 25 jobs are solved by the proposed algorithm, and all the instances with 20 jobs are solved within 11 seconds. Instances with a larger number of machines seem easier to solve because almost all the instances have no duality gaps (to be more precise, the duality gaps are less than 1; \(N_3\) is large) and are solved at the root node (\(N_2\) is large). In addition, we can see that the initial upper bounds are so good that they are almost optimal (\(N_4\) is large). When \(m\) is small, more than 10% of the instances can be solved optimally by simple heuristics (\(N_1\) is large). In the case of \(n = 20\) and \(m = 2\), the algorithm in Appendix B removes all the jobs (and thus all the jobs are non-tardy) for 12 instances, and \(D(L^{\text{SPT}}(J))\) or the KPM heuristics find non-tardy solutions for 5 instances. On the other hand, in the case of \(n = 20\) and \(m = 10\), optimal objective values are always greater than zero, and 5 instances are solved optimally because all the jobs are tardy in the schedule constructed by \(D(L^{\text{SPT}}(J))\).

In the previous researches, the algorithm by Azizoglu and Kirca (1998) cannot solve all the instances with 15 jobs within 15 minutes on IBM 3090 Mainframe and the algorithm by Yalaoui and Chu (2002) can solve only half the instances with 20 jobs and 2 machines within 30 minutes on an HP workstation. The algorithm by Liaw et al. (2003) is for the more general problem \(R||\sum w_i T_j\), and can solve instances with up to 18 jobs and 4 jobs on a PentiumIII 600MHz computer. Our algorithm seems faster than the previous algorithms, although direct comparison of computational times cannot be done because of the differences in computer speed and the targeted problem class. It can be verified by the fact that the nodes generated in our algorithm are much less than those in the previous algorithms. Indeed, in the algorithm by Azizoglu and Kirca more than \(2.1 \times 10^6\) nodes are generated even for optimally solved instances with \(n = 15\) and \(m = 2\), in the algorithm by Yalaoui and Chu (2002) more than \(1.6 \times 10^6\) nodes are generated even for
Table 3: Effect of $\tau$ and $R$ on computational times

(a) $n = 20$

<table>
<thead>
<tr>
<th>$\tau$</th>
<th>0.2</th>
<th>0.4</th>
<th>0.6</th>
<th>0.8</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ave.</td>
<td>0.253</td>
<td>0.075</td>
<td>0.047</td>
<td>0.036</td>
<td>0.024</td>
</tr>
<tr>
<td>max.</td>
<td>4.467</td>
<td>1.864</td>
<td>0.445</td>
<td>0.376</td>
<td>0.276</td>
</tr>
<tr>
<td>ave.</td>
<td>0.234</td>
<td>0.388</td>
<td>0.202</td>
<td>0.245</td>
<td>0.146</td>
</tr>
<tr>
<td>max.</td>
<td>2.138</td>
<td>1.955</td>
<td>1.385</td>
<td>10.886</td>
<td>1.421</td>
</tr>
<tr>
<td>ave.</td>
<td>0.208</td>
<td>0.249</td>
<td>0.187</td>
<td>0.260</td>
<td>0.231</td>
</tr>
<tr>
<td>max.</td>
<td>2.138</td>
<td>1.955</td>
<td>1.385</td>
<td>10.886</td>
<td>1.421</td>
</tr>
<tr>
<td>ave.</td>
<td>0.235</td>
<td>0.195</td>
<td>0.189</td>
<td>0.260</td>
<td>0.186</td>
</tr>
<tr>
<td>max.</td>
<td>1.781</td>
<td>1.927</td>
<td>1.739</td>
<td>2.166</td>
<td>1.243</td>
</tr>
</tbody>
</table>

(b) $n = 25$

<table>
<thead>
<tr>
<th>$\tau$</th>
<th>0.2</th>
<th>0.4</th>
<th>0.6</th>
<th>0.8</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ave.</td>
<td>188.645</td>
<td>0.165</td>
<td>0.041</td>
<td>0.074</td>
<td>0.042</td>
</tr>
<tr>
<td>max.</td>
<td>470.301</td>
<td>5.032</td>
<td>0.565</td>
<td>1.172</td>
<td>0.847</td>
</tr>
<tr>
<td>ave.</td>
<td>5.217</td>
<td>36.921</td>
<td>20.579</td>
<td>0.525</td>
<td>0.317</td>
</tr>
<tr>
<td>max.</td>
<td>98.365</td>
<td>991.873</td>
<td>452.007</td>
<td>3.838</td>
<td>1.486</td>
</tr>
<tr>
<td>ave.</td>
<td>0.806</td>
<td>0.406</td>
<td>0.649</td>
<td>0.630</td>
<td>0.880</td>
</tr>
<tr>
<td>max.</td>
<td>10.227</td>
<td>3.238</td>
<td>2.552</td>
<td>16.302</td>
<td></td>
</tr>
<tr>
<td>ave.</td>
<td>0.338</td>
<td>0.410</td>
<td>0.538</td>
<td>0.578</td>
<td>0.564</td>
</tr>
<tr>
<td>max.</td>
<td>2.739</td>
<td>2.632</td>
<td>2.508</td>
<td>2.863</td>
<td></td>
</tr>
<tr>
<td>ave.</td>
<td>0.128</td>
<td>0.172</td>
<td>0.360</td>
<td>0.401</td>
<td>0.550</td>
</tr>
<tr>
<td>max.</td>
<td>2.048</td>
<td>2.558</td>
<td>3.030</td>
<td>2.687</td>
<td>2.653</td>
</tr>
</tbody>
</table>

optimally solved instances with $n = 20$ and $m = 2$, and in the algorithm by Liaw et al. (2003) more than $6.4 \times 10^7$ nodes are generated for instances with $n = 18$ and $m = 2$. In contrast, at most $4.8 \times 10^5$ nodes are generated from Table 1 in our algorithm for the 125 instances with $n = 20$ and $m = 2$.

Next, to investigate the effectiveness of the initial lower bounds, the average and maximum duality gaps for those instances that cannot be solved at the root node (125 $- N_2$ instances for each setting) are shown in Table 2. In Table 2, $\text{(OPT-LB)}/\text{OPT} (%)$ and $\text{(OPT-LB)}$ respectively denote the relative and absolute duality gaps. Due to the nonlinearity of the Lagrangian dual, subgradient optimization fails to find good multipliers and the gap becomes relatively large for some instances (e.g. $n = 20$ and $m = 2$). Nevertheless, the gap is at most 2.4% on average and the effectiveness of our lower bounding scheme is confirmed.

The effects of the tardiness factor $\tau$ and the range of duedates $R$ are shown in Table 3, where average and maximum CPU times are shown in seconds. It is clear that the problem instances with smaller $\tau$ and $R$ are harder to solve. Indeed, it takes much time for the problem instances with $\tau = 0.2$ and $R = 0.2$ when $n = 25$. This tendency is also observed in the previous researches by Azizoglu and Kirca (1998) and Yalaoui and Chu (2002). On the other hand, problem instances with smaller $\tau$ and $R$ seems easier to solve in Liaw et al. (2003). It is because they treated $R||\sum w_j T_j$, not $P||\sum T_j$ and there is a difference between the methods to generate problem instances. In their computational experiments, optimal objective values were always zero when $\tau = 0.2$, regardless of $R$. However, optimal objective values are always greater than zero in our experiments when $\tau = 0.2$ and $R \leq 0.4$.

Finally, to evaluate the impact of the dominance conditions on the efficiency of our algorithm, we tested the following three algorithms:

(A) the proposed algorithm,

(B) the proposed algorithm without the lower bound improvement by dominance conditions: Lower bounds are computed by using (32) instead of (31),

(C) the proposed algorithm without the dominance conditions (without Rules 1 nor 2 and without the lower bound improvement by the dominance conditions)

The results are shown in Table 4, where the average and maximum CPU times over the instances that cannot be solved at the root node (125 $- N_2$ instances) are given in seconds. With regard to (C), only
Table 4: Effect of the lower bound improvement on computational times

(a) \( n = 20 \)

<table>
<thead>
<tr>
<th>( m )</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of instances</td>
<td>64</td>
<td>56</td>
<td>41</td>
<td>31</td>
</tr>
<tr>
<td>(A) time</td>
<td>ave.</td>
<td>1.221</td>
<td>0.879</td>
<td>0.566</td>
</tr>
<tr>
<td></td>
<td>max.</td>
<td>2.282</td>
<td>4.467</td>
<td>4.416</td>
</tr>
<tr>
<td>(A) node</td>
<td>ave.</td>
<td>36028</td>
<td>217221</td>
<td>164277</td>
</tr>
<tr>
<td></td>
<td>max.</td>
<td>478004</td>
<td>5877356</td>
<td>6334218</td>
</tr>
<tr>
<td>(B) time</td>
<td>ave.</td>
<td>1.337</td>
<td>1.194</td>
<td>0.702</td>
</tr>
<tr>
<td></td>
<td>max.</td>
<td>4.218</td>
<td>17.393</td>
<td>9.883</td>
</tr>
<tr>
<td>(B) node</td>
<td>ave.</td>
<td>175712</td>
<td>639669</td>
<td>379965</td>
</tr>
<tr>
<td></td>
<td>max.</td>
<td>3677057</td>
<td>2322718</td>
<td>15027275</td>
</tr>
<tr>
<td>(C) time</td>
<td>ave.</td>
<td>387.633</td>
<td>21.870</td>
<td>135.324</td>
</tr>
<tr>
<td></td>
<td>max.</td>
<td>96241.086</td>
<td>943.029</td>
<td>5565.958</td>
</tr>
<tr>
<td>(C) node</td>
<td>ave.</td>
<td>1592650484</td>
<td>849994506</td>
<td>488834273</td>
</tr>
<tr>
<td></td>
<td>max.</td>
<td>40615449451</td>
<td>3798862538</td>
<td>19936585429</td>
</tr>
</tbody>
</table>

(b) \( n = 25 \)

<table>
<thead>
<tr>
<th>( m )</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of instances</td>
<td>24</td>
<td>21</td>
<td>7</td>
<td>13</td>
<td>8</td>
</tr>
<tr>
<td>(A) time</td>
<td>ave.</td>
<td>0.275</td>
<td>0.262</td>
<td>0.228</td>
<td>1.182</td>
</tr>
<tr>
<td></td>
<td>max.</td>
<td>0.565</td>
<td>0.503</td>
<td>0.393</td>
<td>10.886</td>
</tr>
<tr>
<td>(A) node</td>
<td>ave.</td>
<td>12876</td>
<td>10313</td>
<td>12574</td>
<td>1433149</td>
</tr>
<tr>
<td></td>
<td>max.</td>
<td>140119</td>
<td>31582</td>
<td>33914</td>
<td>16803230</td>
</tr>
<tr>
<td>(B) time</td>
<td>ave.</td>
<td>0.278</td>
<td>0.260</td>
<td>0.228</td>
<td>1.159</td>
</tr>
<tr>
<td></td>
<td>max.</td>
<td>0.570</td>
<td>0.500</td>
<td>0.391</td>
<td>10.576</td>
</tr>
<tr>
<td>(B) node</td>
<td>ave.</td>
<td>143828</td>
<td>10653</td>
<td>12974</td>
<td>1447177</td>
</tr>
<tr>
<td></td>
<td>max.</td>
<td>141682</td>
<td>32747</td>
<td>33924</td>
<td>16808410</td>
</tr>
<tr>
<td>(C) time</td>
<td>ave.</td>
<td>0.298</td>
<td>0.261</td>
<td>0.223</td>
<td>1.076</td>
</tr>
<tr>
<td></td>
<td>max.</td>
<td>0.753</td>
<td>0.499</td>
<td>0.400</td>
<td>9.472</td>
</tr>
<tr>
<td>(C) node</td>
<td>ave.</td>
<td>88082</td>
<td>12406</td>
<td>12311</td>
<td>2337923</td>
</tr>
<tr>
<td></td>
<td>max.</td>
<td>1069544</td>
<td>35779</td>
<td>37780</td>
<td>26917324</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( m )</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of instances</td>
<td>70</td>
<td>53</td>
<td>51</td>
<td>53</td>
</tr>
<tr>
<td>(A) time</td>
<td>ave.</td>
<td>2.801</td>
<td>50.797</td>
<td>130.333</td>
</tr>
<tr>
<td></td>
<td>max.</td>
<td>14.954</td>
<td>885.132</td>
<td>4703.303</td>
</tr>
<tr>
<td>(A) node</td>
<td>ave.</td>
<td>968394</td>
<td>63969906</td>
<td>174139084</td>
</tr>
<tr>
<td></td>
<td>max.</td>
<td>17713505</td>
<td>1146752242</td>
<td>6260506105</td>
</tr>
<tr>
<td>(B) time</td>
<td>ave.</td>
<td>8.702</td>
<td>365.592</td>
<td>799.505</td>
</tr>
<tr>
<td></td>
<td>max.</td>
<td>259.715</td>
<td>11768.664</td>
<td>36689.496</td>
</tr>
<tr>
<td>(B) node</td>
<td>ave.</td>
<td>6690022</td>
<td>445964109</td>
<td>1056602500</td>
</tr>
<tr>
<td></td>
<td>max.</td>
<td>246345216</td>
<td>13775008021</td>
<td>48243633225</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( m )</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of instances</td>
<td>30</td>
<td>35</td>
<td>25</td>
<td>17</td>
<td>9</td>
</tr>
<tr>
<td>(A) time</td>
<td>ave.</td>
<td>1.769</td>
<td>0.711</td>
<td>0.628</td>
<td>1.550</td>
</tr>
<tr>
<td></td>
<td>max.</td>
<td>31.901</td>
<td>3.238</td>
<td>1.184</td>
<td>16.302</td>
</tr>
<tr>
<td>(A) node</td>
<td>ave.</td>
<td>1468712</td>
<td>266359</td>
<td>120066</td>
<td>1465728</td>
</tr>
<tr>
<td></td>
<td>max.</td>
<td>37155822</td>
<td>3045539</td>
<td>657458</td>
<td>22364993</td>
</tr>
<tr>
<td>(B) time</td>
<td>ave.</td>
<td>3.469</td>
<td>0.783</td>
<td>0.620</td>
<td>1.410</td>
</tr>
<tr>
<td></td>
<td>max.</td>
<td>75.994</td>
<td>3.190</td>
<td>1.167</td>
<td>14.078</td>
</tr>
<tr>
<td>(B) node</td>
<td>ave.</td>
<td>4230306</td>
<td>410512</td>
<td>134801</td>
<td>1469788</td>
</tr>
<tr>
<td></td>
<td>max.</td>
<td>109705840</td>
<td>3541277</td>
<td>708900</td>
<td>22397383</td>
</tr>
</tbody>
</table>
the results for 20 jobs instances are presented because not all the instances with \( n = 20 \) and \( m = 2 \) can be solved optimally within 50,000 seconds. Thus in this case the average and maximum CPU times and nodes are over optimally solved instances (58 out of 64 instances). By comparing (A) with (B) or with (C), we can see that the efficiency of the algorithm improves by taking into account the dominance conditions especially for smaller \( m \). It is because Rule 2 works effectively when \( m \) is small, which requires that one of the dominance conditions are satisfied on all the machines. The effect of the dominance conditions reduces as \( m \) becomes large, and average CPU times of (A) are longer than those of (B) or (C) in some problem instances (e.g. \( m = 9 \)), although the number of nodes is smaller. It is because it takes a little more time to compute a lower bound itself and to check the dominance conditions.

6 Conclusion

In this study a new branch-and-bound algorithm is proposed for a class of scheduling problems to minimize total tardiness on identical parallel machines. In our algorithm, the Lagrangian relaxation technique is applied for lower bound computation. In addition, job dominance conditions are utilized for both branch restriction and lower bound improvement. Computational experiments showed that most problem instances have no duality gaps and can be solved at the root node without branching. Even when an optimal solution is not found at the root node, the proposed algorithm can find an optimal solution efficiently. Indeed, it can handle instances with up to 25 jobs and any number of machines. To improve its efficiency, it would be necessary to consider better choices of the step size parameter for the subgradient optimization. It would be also necessary to apply other types of Lagrangian relaxation such as the one proposed by Babu et al. (2004).

A Proof of Lemma 1

The assertion of the lemma is proved by dividing into dummy jobs the periods that the machines are unavailable. It is done by the following procedure.

0° Let \( L^S := L^{SPT}(J) \) and construct a schedule \( S^S \) by applying \( D(L^S) \). Let \( n^S := n \).

1° Let \( m^S := \arg \max_{1 \leq j \leq m} R_j \).

If \( R_{m^S} = 0 \), terminate.

2° Let \( k^S := \arg \min_{J \in L^S} C_j \) \( R_{m^S} \leq C_j \).

If \( J_{k^S} \) is the first element of \( L^S \), let \( s^S := \max(R_{m^S} - p_{k^S}, 0) \) and go to 4°.

3° Denote by \( J_{l^S} \) the immediate predecessor of \( J_{k^S} \) in \( L^S \). Let \( s^S := \max(R_{m^S} - p_{k^S}, C_{l^S} - p_{l^S}) \) (see Figure A). Here, it should be noted that \( C_{l^S} - p_{l^S} \leq s^S \leq C_{k^S} - p_{k^S} \) and \( p_{l^S} \leq R_{m^S} - s^S \leq p_{k^S} \) hold.
4° $n^S := n^S + 1$. Generate a dummy job $J_{n^S}$ such that $p_{n^S} = R_{n^S} - s^S$ and $C_{n^S} = d_{n^S} = R_{n^S}$. Insert $J_{n^S}$ immediately before $J_{n^S}$ in $L^S$. Let $R_{n^S} := s^S$ and go to 2°.

By using $L^S$ obtained by this procedure, let us construct a schedule $S^S$ by $D(L^S)$, where all the machines are assumed to be available from time zero. Denote by $C^S_j$ the completion time of $J_j$ ($1 \leq j \leq n^S$) in this schedule. Then, $C^S_j = C_j$ ($1 \leq j \leq n$) and $C^S_{n^S} = d_{n^S} = d_j$ ($n + 1 \leq j \leq n^S$) hold. Moreover, the problem to minimize the total tardiness of $J_j$ ($1 \leq j \leq n^S$) with the dummy jobs fixed to their positions in $S^S$ is equivalent to the problem to minimize the total tardiness of $J_j$ ($1 \leq j \leq n$) under the condition that $M_j$ is available only from $R_j$. Therefore, if $C_j = C^S_j \geq d_j$ ($1 \leq j \leq n$), $S^S$ minimizes the total tardiness of $J_j$ ($1 \leq j \leq n^S$) from Corollary 3 and hence $S^S$ also minimizes the total tardiness of $J_j$ ($1 \leq j \leq n$).

B A procedure to remove non-tardy jobs

A procedure to remove non-tardy jobs satisfying (18) is given as follows.

0° $J' := J$, $L := \phi$.

1° $j = 1$.

2° If $J_j \notin J'$, go to 5°.

3° If $J_j$ satisfies $d_j \geq \sum_{i \in J'} p_i / m + (m - 1)p_J / m$, go to 4°. Otherwise, go to 5°.

4° Remove $J_j$ from $J'$ and insert $J_j$ into the first position of $L$. Go to 1°.

5° $j := j + 1$. If $j \leq n$, go to 2°.

Its time complexity is $O(n^2)$.

References


1If a tie occurs between some pair of jobs while applying $D(L^S)$, it can happen that $C^S_j \neq d_j$ ($1 \leq j \leq n$) or $C^S_{n^S} \neq d_{n^S}$ ($n + 1 \leq j \leq n^S$) depending on how to break ties. However, it does not make any problems and we can choose $S^S$ so that $C^S_j = C_j$ ($1 \leq j \leq n$) and $C^S_{n^S} = d_{n^S}$ ($n + 1 \leq j \leq n^S$), because how to break ties does not affect SPT optimality.


