Available online at www.sciencedirect.com
ScienceDirect
Int. J. Production Economics 113 (2008) 446–458
international journal of production economics
www.elsevier.com/locate/ijpe

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

Shunji Tanaka[a,*], Mituhiko Araki[b]

[a]*Graduate School of Electrical Engineering, Kyoto University, Kyotodaigaku-Katsura, Nishikyo-ku, Kyoto 615-8510, Japan*
[b]*Matsue National College of Technology, Japan*

Received 1 February 2006; accepted 24 October 2007
Available online 12 November 2007

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, the 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.
© 2007 Elsevier B.V. All rights reserved.

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. Szwarc et al. (1999) reported that their algorithm can handle instances with up to 300 jobs, and Tansel et al. (2001) 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 $P|p_j = d_j|\sum T_j$. However, these researches targeted special classes of $P\|\sum T_j$. To the authors' knowledge, Azizoglu and Kirca (1998) first treated the general problem $P\|\sum T_j$ explicitly. 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 three machines were optimally solved by their algorithm (they claims that it can handle instances with up to 15 jobs, but there were unsolved instances). Yalaoui and Chu (2002)

*Corresponding author. Tel.: +81 75 383 2204; fax: +81 75 383 2201.
E-mail address: tanaka@kuee.kyoto-u.ac.jp (S. Tanaka).

improved their algorithm by introducing some job dominance checks and problem instances with up to 15 jobs and three 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 four 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; 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.

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 $\mathscr{J} = \{\mathrm{J}_1, \ldots, \mathrm{J}_n\}$ are to be processed on m identical parallel machines $\mathrm{M}_1, \ldots, \mathrm{M}_m$. Each job J_j is given the integer processing time p_j and the integer duedate 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 here 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 non-decreasing 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 $\mathscr{D}(L)$ (L is a job priority list), and a set of all the schedules constructed by $\mathscr{D}(\bullet)$ is denoted by $\mathscr{S}_{\mathrm{D}}$. It is not difficult to see that job completion times in any schedule belonging to $\mathscr{S}_{\mathrm{D}}$ have the following property.

Corollary 1. *In any schedule belonging to $\mathscr{S}_{\mathrm{D}}$, the completion time C_j of J_j $(1 \leqslant j \leqslant n)$ satisfies*

$$C_j \leqslant \frac{1}{m}\sum_{i=1}^{n} p_i + \frac{m-1}{m} p_j. \tag{1}$$

Proof of Corollary 1. Consider a schedule belonging to $\mathscr{S}_{\mathrm{D}}$ and assume without loss of generality that J_j is assigned on M_1. Since $\mathscr{D}(\bullet)$ assigns jobs to earliest available machines, the starting time of J_j should be not more than total processing times on the other machines. Therefore,

$$C_j \leqslant P_k + p_j \quad (2 \leqslant k \leqslant m), \tag{2}$$

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

$$(m-1)C_j \leqslant \sum_{k=2}^{m} P_k + (m-1)p_j. \tag{3}$$

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

$$mC_j \leqslant \sum_{i=1}^{n} p_i + (m-1)p_j. \tag{4}$$

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 times 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 Section 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 $\mathcal{S}_{\mathrm{D}}$) 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) $d_j \leqslant \max(C_k - p_j, d_k)$ *if* $p_j < p_k$,
(d3) $d_j \leqslant d_k$ *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 $\mathcal{D}(L^{\mathrm{SPT}}(\mathcal{J}))$, where $L^{\mathrm{SPT}}(\mathcal{J})$ 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 $\mathcal{D}(L^{\mathrm{SPT}}(\mathcal{J}))$, 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 Section 4.3.

Lemma 1. *Assume that* M$_j$ $(1 \leqslant j \leqslant m)$ *is available from $R_j \geqslant 0$. If all the jobs are tardy in the schedule constructed by $\mathcal{D}(L^{\mathrm{SPT}}(\mathcal{J}))$, 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. This technique yields a tight lower bound for parallel machine scheduling problems. 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:

$$\text{(P):}\quad F = \min_x \sum_{j=1}^{n}\sum_{t=0}^{E_j} W_{jt}x_{jt}, \tag{5}$$

$$\text{s.t.}$$

$$x_{jt} \in \{0,1\} \quad (1 \leqslant j \leqslant n,\ 0 \leqslant t \leqslant E_j), \tag{6}$$

$$\sum_{t=0}^{E_j} x_{jt} = 1 \quad (1 \leqslant j \leqslant n), \tag{7}$$

$$\sum_{j=1}^{n}\sum_{s=\max(t-p_j+1,0)}^{\min(t,E_j)} x_{js} \leqslant Q_t \quad (0 \leqslant t \leqslant T_{\max}). \tag{8}$$

Here, x_{jt} are decision variables such that

$$x_{jt} = \begin{cases} 1 & \text{if J}_j \text{ is started at } t, \\ 0 & \text{otherwise,} \end{cases} \tag{9}$$

and the constants E_j, $T_{\max}$, W_{jt} and Q_t are given by

E_j	The latest possible starting time of J$_j$ $(E_j := (\sum_{i=1}^{n} p_i - p_j)/m)$,
$T_{\max}$	$T_{\max} = \max_{1 \leqslant j \leqslant n}(E_j + p_j - 1)$,
W_{jt}	The tardiness of J$_j$ when it is started from t $(W_{jt} = \max(t + p_j - d_j, 0))$,
Q_t	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 non-negative Lagrangian multipliers μ_t $(0 \leqslant t \leqslant T_{\max})$. Then, the relaxed problem becomes

$$\text{(LR):}\quad F^*(\mu) = \min_x \quad F(\mu), \tag{10}$$

$$\text{s.t.}\quad (6) \text{ and } (7), \tag{11}$$

where

$$\begin{aligned} F(\mu) &= \sum_{j=1}^{n}\sum_{t=0}^{E_j} W_{jt}x_{jt} \\ &\quad - \sum_{t=0}^{T_{\max}} \mu_t \left(Q_t - \sum_{j=1}^{n}\sum_{s=\max(t-p_j+1,0)}^{\min(t,E_j)} x_{js} \right) \\ &= -\sum_{t=0}^{T_{\max}} \mu_t Q_t + \sum_{j=1}^{n} F_j(\mu), \end{aligned}$$

$$F_j(\mu) = \sum_{t=0}^{E_j} W_{jt}x_{jt} + \sum_{t=0}^{T_{\max}} \mu_t \sum_{s=\max(t-p_j+1,0)}^{\min(t,E_j)} x_{js}$$
$$= \sum_{t=0}^{E_j}\left(W_{jt} + \sum_{s=t}^{t+p_j-1}\mu_s\right)x_{jt}. \tag{12}$$

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

$$(\mathrm{LR}_j):\quad F_j^*(\mu) = \min_{x_j} F_j(\mu) = \min_{0\leqslant t_j \leqslant E_j}\left(W_{jt_j} + \sum_{s=t_j}^{t_j+p_j-1}\mu_s\right). \tag{13}$$

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

$$(\mathrm{D}):\quad \underline{F} = \max_{\mu}\quad F^*(\mu)$$
$$= \max_{\mu}\quad \left(-\sum_{t=0}^{T_{\max}}\mu_t Q_t + \sum_{j=1}^{n} F_j^*(\mu)\right), \tag{14}$$
$$\text{s.t.}\quad (6)\text{ and }(7). \tag{15}$$

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

$$\mu^{k+1} = \mu^k + \alpha^k g(\mu^k), \tag{16}$$

where $g(\mu^k)$ is a subgradient vector of $F^*(\mu^k)$ with respect to μ^k and α^k is the kth step size. The step size is chosen as

$$\alpha^k = \lambda\frac{\overline{F} - F^*(\mu^k)}{g(\mu^k)^{\mathrm{T}}g(\mu^k)}. \tag{17}$$

Here, $\overline{F}$ is an upper bound of F and λ 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 $\mathscr{S}_{\mathrm{D}}$. 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
$$d_j \geqslant \frac{1}{m}\sum_{i=1}^{n}p_i + \frac{m-1}{m}p_j \tag{18}$$
is satisfied, J_j is always on time in any schedule belonging to $\mathscr{S}_{\mathrm{D}}$. Therefore, such on time jobs are removed in order to reduce the problem size (see Appendix B). Redefine $\mathscr{J}$ by the set of the remaining jobs and n by the number of the remaining jobs.

(2) Construct a schedule by $\mathscr{D}(L^{\mathrm{SPT}}(\mathscr{J}))$. 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 $\overline{F}^{\mathrm{SPT}}$.

(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 $\overline{F}^{\mathrm{KPM}}$.

(4) Let $\overline{F}:=\min(\overline{F}^{\mathrm{SPT}},\overline{F}^{\mathrm{KPM}})$. Apply Lagrangian relaxation and maximize $F^*(\mu)$ by subgradient optimization as explained in Section 3. The initial multipliers μ^0 are chosen as $\mu_t^0:=1.0$, and the step size parameter λ 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 $F^*(\mu^k)$ by resolving conflicts heuristically. $\overline{F}$ is updated if necessary. If $F^*(\mu^k)$ is not updated for 20 successive iterations, λ is scaled by $\lambda := 0.99\lambda$. The subgradient optimization is terminated if one of the following termination conditions is satisfied.

(1) $\overline{F} - F^*(\mu^k) < 1$ ($\overline{F}$ is optimal because the optimal objective value is integer).
(2) $F^*(\mu^k)$ is not updated for 600 successive iterations.
(3) $\lambda < 10^{-4}$.

Denote the obtained multipliers by μ^* and let $\underline{F} := F^*(\mu^*)$.

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; 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 $\mathscr{D}(L^{\mathrm{SPT}}(\mathscr{J}))$ 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 λ is reduced when the solution of (D) is not updated. The number of iterations for reducing λ and the number of iterations for terminating the subgradient optimization are determined by preliminary computational experiments.

At each step of the subgradient optimization, $F^*(\mu^k)$, a solution of (LR), is obtained. Since (LR) is a relaxation 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^k the starting time of J_j that minimizes $F_j(\mu^k)$ in the decomposed subproblem (LR$_j$). Jobs are scheduled on the earliest available machines one by one in the non-decreasing order of t_j^k if no conflict occurs (t_j^k 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 on time when scheduled on the earliest available machine, a job that can be completed as close as possible to its duedate is chosen.

Then, the upper bound $\overline{F}$ 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 $\mathscr{S}^l$ the partial schedule constructed by $\mathscr{D}(L^l)$. The set of the $(n-l)$ unscheduled jobs is denoted by $\mathscr{U}^l$. Let us denote the set of jobs assigned on M_k $(1\leqslant k\leqslant m)$ in the partial schedule $\mathscr{S}^l$ by $\mathscr{A}_k^l$, and the completion time of $\mathrm{J}_j \in \mathrm{M}_k$ $(1\leqslant k\leqslant m)$ in $\mathscr{S}^l$ simply by C_j. Then, the total processing time on M_k in $\mathscr{S}^l$, c_k^l, is defined by

$$c_k^l = \sum_{\mathrm{J}_j\in\mathscr{A}_k^l} p_j = \max_{\mathrm{J}_j\in\mathscr{A}_k^l} C_j. \tag{19}$$

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)–(d3) in Corollary 2 on M_{m^l}. Therefore, if an unscheduled job $\mathrm{J}_u \in \mathscr{U}^l$ 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 $J_u \in \mathscr{U}^l$ satisfies at least one of the following three conditions for some j ($J_j \in \mathscr{A}^l_{m^l}$), J_u cannot be a candidate for the $(l+1)$th job in the priority list.*

1. $p_u < p_j$ *and* $d_u \leqslant \max(C_j, d_j)$,
2. $p_u > p_j$ *and* $d_j > \max(c^l_{m^l} + p_u - p_j, d_u)$,
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 $J_u \in \mathscr{U}^l$ is chosen as a candidate for the $(l+1)$th job in the priority list. That is, the partial schedule $\mathscr{S}^{l+1}$ (the partial schedule obtained by adding J_u to $\mathscr{S}^l$) satisfies*

$$\begin{aligned} \mathscr{A}^{l+1}_k &= \mathscr{A}^l_k \cup \{J_u\} \quad \text{if } k = m^l, \\ \mathscr{A}^{l+1}_k &= \mathscr{A}^l_k \quad \text{otherwise,} \end{aligned} \tag{20}$$

and $C_u = c^l_{m^l} + p_u$. If there exists $J_w \in \mathscr{U}^l \backslash \{J_u\}$ such that for all k ($1 \leqslant k \leqslant m$) and for some j ($J_j \in \mathscr{A}^{l+1}_k$), 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* $d_w \leqslant \max(C_j, d_j)$.
2. $p_w > p_j$ *and* $d_j > \max(E_w + p_w - p_j, d_w)$.
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 $\mathscr{U}^l$ are tardy in the schedule constructed by applying $\mathscr{D}(L^{\mathrm{SPT}}(\mathscr{U}^l))$ to the partial schedule $\mathscr{S}^l$, it is optimal under the condition that $\mathscr{S}^l$ 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 $\mathscr{U}^l$ 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 $\mathscr{T}^l \subset \mathscr{U}^l$ is chosen so that all the jobs in $\mathscr{T}^l$ are tardy in the schedule constructed by applying $\mathscr{D}(L^{\mathrm{SPT}}(\mathscr{T}^l))$ to $\mathscr{S}^l$. Then, the total tardiness of the jobs in $\mathscr{T}^l$ in this schedule is used as a lower bound of the total tardiness of the jobs in $\mathscr{U}^l$. A procedure to compute this lower bound is given by the following.

```
(0) $c_i^B := c_i^l$ $(1 \leqslant i \leqslant m)$, $\mathscr{U}^B := \mathscr{U}^l$ and LB:=0.
(1) Remove a job with the smallest processing time from $\mathscr{U}^B$. Denote this job by $J_j$.
(2) $k := \arg\min_i c_i^B$. If $c_k^B + p_j < d_j$, go to (4).
(3) LB:=LB $+ c_k^B + p_j - d_j$, $c_k^B := c_k^B + p_j$.
(4) If $\mathscr{U}^B \neq \emptyset$, 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 $\mathscr{U}^l$ is formulated by

$$(\mathrm{SP}^l): \quad \min \sum_{J_j \in \mathscr{U}^l} \sum_{t=r_j^l}^{E_j} W_{jt} x_{jt}, \tag{21}$$

s.t.

$$x_{jt} \in \{0,1\} \quad (J_j \in \mathscr{U}^l,\ r_j^l \leqslant t \leqslant E_j), \tag{22}$$

$$\sum_{t=0}^{E_j} x_{jt} = 1 \quad (J_j \in \mathscr{U}^l), \tag{23}$$

$$\sum_{J_j \in \mathscr{U}^l} \sum_{s=\max(t-p_j+1, r_j^l)}^{\min(t,E_j)} x_{js} \leqslant Q_t^l \quad (r^l_{\min} \leqslant t \leqslant T_{\max}). \tag{24}$$

Here, Q_t^l denotes the number of machines available in the interval $[t, t+1)$. r_j^l denotes the earliest possible starting time of J_j corresponding to the partial schedule $\mathscr{S}^l$, and $r^l_{\min}$ is defined by

$r^l_{\min} := \min_{J_j \in \mathscr{U}^l} r^l_j$. A lower bound of the total tardiness of the jobs in $\mathscr{U}^l$ can be obtained by solving the Lagrangian dual corresponding to (SP^l). 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; 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^l), a lower bound is computed by fixing the multipliers to those obtained at the root node.

To improve this lower bound, r^l_u, the earliest possible starting time of $J_u \in \mathscr{U}^l$, 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 ($J_j \in \mathscr{A}^l_k$), J_u satisfies

$$p_u < p_j, \quad d_u \leqslant \max(C_j, d_j) \tag{25}$$

or

$$p_u = p_j, \quad d_u < d_j. \tag{26}$$

If we denote by $\mathscr{M}^l_u$ the set of the machines such that none of these conditions are satisfied, it follows that r^l_u is given by

$$r^l_u = \min_{M_k \in \mathscr{M}^l_u} c^l_k. \tag{27}$$

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

$$\begin{aligned} &p_u > p_j, \\ &d_j > \max\{(\text{The starting time of } J_u) + p_u - p_j, d_u\}. \end{aligned} \tag{28}$$

Therefore, for J_u to be assigned on M_k, it should satisfy

$$(\text{The starting time of } J_u) \geqslant d_j + p_j - p_u \tag{29}$$

for any job $J_j \in \mathscr{A}^l_k$ 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

$$\max\left\{ \max_{\substack{J_j \in \mathscr{A}^l_k \\ p_u > p_j,\, d_u < d_j}} (d_j + p_j - p_u),\ c^l_k \right\}. \tag{30}$$

Taking the minimum of (30) on $M_k \in \mathscr{M}^l_u$, we obtain

$$r^l_u = \min_{M_k \in \mathscr{M}^l_u} \max\left\{ \max_{\substack{J_j \in \mathscr{A}^l_k \\ p_u > p_j,\, d_u < d_j}} (d_j + p_j - p_u),\ c^l_k \right\}. \tag{31}$$

If none of the dominance conditions are taken into account, r^l_u is given simply by

$$r^l_u = \min_{1 \leqslant k \leqslant m} c^l_k. \tag{32}$$

Since (31) is not less than (32), r^l_u is restricted by the dominance conditions, and hence the lower bound is expected to be improved.

The time complexity to compute r^l_u for all the unscheduled jobs $J_u \in \mathscr{U}^l$ 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 $M_{m^{l-1}}$, 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^l_u by (32) (for all $J_u \in \mathscr{U}^l$) is $O(\log m)$.

To solve the Lagrangian relaxation of (SP^l) for a given set of r^l_u, the decomposed subproblem

$$\min_{r^l_u \leqslant t_j \leqslant E_j} \left(W_{jt_j} + \sum_{s=t_j}^{t_j+p_j-1} \mu_s \right) \tag{33}$$

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^l_u 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 2.4 GHz desktop computer.

Problem instances are generated by the standard method by Fisher (1976). First, the integer processing times p_j $(1 \leqslant j \leqslant n)$ are generated by the uniform distributions in [1, 100]. Then, let $P = \sum_{j=1}^{n} p_j$ and the integer duedates d_j $(1 \leqslant j \leqslant n)$ 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 τ and the range of duedates 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, τ 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:

- all the jobs are removed by the algorithm in Appendix B,
- all the jobs are tardy in the schedule constructed by $\mathscr{D}(L^{\mathrm{SPT}}(\mathscr{J}))$,
- all the jobs are on time in the schedule constructed by $\mathscr{D}(L^{\mathrm{SPT}}(\mathscr{J}))$ 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 s. 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 on time) for 12 instances, and $\mathscr{D}(L^{\mathrm{SPT}}(\mathscr{J}))$ or the KPM heuristics find on time solutions for five instances. On the other hand, in the case of $n = 20$ and $m = 10$, optimal objective values are always greater than zero, and five instances are solved optimally because all the jobs are tardy in the schedule constructed by $\mathscr{D}(L^{\mathrm{SPT}}(\mathscr{J}))$.

In the previous researches, the algorithm by Azizoglu and Kirca (1998) cannot solve all the instances with 15 jobs within 15 min on IBM 3090 Mainframe and the algorithm by Yalaoui and Chu (2002) can solve only half the instances with 20 jobs and two machines within 30 min on an HP workstation. The algorithm by Liaw et al. (2003) is for the more general problem $R\|\sum w_j T_j$, and can solve instances with up to 18 jobs and four jobs on a PentiumIII 600 MHz 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 2.1×10^6 nodes are generated even for optimally solved instances with $n = 15$ and $m = 2$, in the algorithm by Yalaoui and Chu (2002) more than 1.6×10^6 nodes are generated even for optimally solved instances with $n = 20$ and $m = 2$, and in the algorithm by Liaw et al. (2003) more than 6.4×10^7 nodes are generated for instances with $n = 18$ and $m = 2$. In contrast, at most 4.8×10^5 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, (OPT-LB)/OPT (%) and (OPT-LB), respectively, denote the relative and absolute duality gaps. Due to the nonsmoothness 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 τ and the range of duedates R are shown in Table 3, where average and maximum CPU times are shown in seconds.

Table 1
Performance of the proposed algorithm

		m								
		2	3	4	5	6	7	8	9	10
(a) $n = 20$										
Root time	Ave.	0.644	0.348	0.170	0.107	0.075	0.058	0.035	0.051	0.039
	Max.	2.282	1.291	0.882	0.659	0.540	0.495	0.389	0.423	0.476
Total time	Ave.	0.656	0.413	0.206	0.110	0.076	0.059	0.035	0.145	0.045
	Max.	2.282	4.467	4.416	0.662	0.565	0.503	0.393	10.886	0.698
Node	Ave.	18,447	97,315	53,884	3335	2473	1767	705	149,048	7539
	Max.	478,004	5,877,356	6,334,218	153,498	140,119	31,582	33,914	16,803,230	732,266
$N_1/N_2/N_3/N_4$		17/61/61/105	15/69/71/104	14/84/86/111	12/94/101/109	6/101/104/117	6/104/106/119	4/118/120/123	5/112/117/119	5/117/123/119
(b) $n = 25$										
Root time	Ave.	1.182	0.614	0.412	0.306	0.170	0.155	0.136	0.103	0.070
	Max.	3.224	1.851	1.320	1.095	0.849	0.698	0.783	0.821	0.640
Total time	Ave.	1.613	21.589	53.221	15.964	0.466	0.221	0.159	0.245	0.099
	Max.	14.954	885.132	4703.301	1719.265	31.901	3.238	1.184	16.302	3.838
Node	Ave.	542,301	27,123,265	71,048,747	19,342,607	352,492	74,581	24,122	199,340	40,288
	Max.	17,713,505	1,146,752,242	6,260,506,105	2,109,984,069	37,155,822	3,045,539	657,458	22,364,993	4,448,242
$N_1/N_2/N_3/N_4$		17/55/55/105	17/72/72/100	15/74/78/103	14/72/79/101	12/95/97/118	7/90/94/114	2/100/106/115	2/108/113/118	2/116/122/119

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

Number of instances		m								
		2	3	4	5	6	7	8	9	10
		64	56	41	31	24	21	7	13	8
(a) $n=20$										
(OPT-LB)/OPT (%)	Ave.	1.850	0.908	0.708	0.908	0.647	1.381	0.300	0.443	0.067
	Max.	29.269	5.485	3.500	7.682	2.386	6.279	0.598	1.423	0.356
(OPT-LB)	Ave.	8.806	4.632	3.350	2.763	3.185	2.297	2.139	1.279	0.305
	Max.	32.464	16.726	10.501	8.339	8.662	5.610	5.009	3.361	1.170
		70	53	51	53	30	35	25	17	9
(b) $n=25$										
(OPT-LB)/OPT (%)	Ave.	0.917	1.361	1.625	0.713	0.844	0.577	2.342	0.609	0.343
	Max.	8.535	8.744	40.971	3.909	8.333	2.923	37.811	5.070	1.468
(OPT-LB)	Ave.	7.000	7.246	3.937	3.303	2.880	2.429	2.176	1.063	0.518
	Max.	38.428	42.345	12.329	12.782	6.170	7.973	6.139	2.535	1.292

Table 3
Effect of τ and R on computational times

τ		R				
		0.2	0.4	0.6	0.8	1.0
(a) $n=20$						
0.2	Ave.	0.253	0.075	0.047	0.036	0.024
	Max.	4.467	1.864	0.445	0.376	0.276
0.4	Ave.	0.234	0.388	0.202	0.245	0.146
	Max.	1.629	4.416	0.921	1.927	1.171
0.6	Ave.	0.208	0.249	0.187	0.499	0.264
	Max.	2.138	1.955	1.385	10.886	1.421
0.8	Ave.	0.235	0.195	0.189	0.260	0.231
	Max.	1.781	1.972	1.739	2.166	1.243
1.0	Ave.	0.047	0.058	0.199	0.195	0.186
	Max.	0.787	1.640	2.282	1.952	1.699
(b) $n=25$						
0.2	Ave.	188.645	0.165	0.041	0.074	0.042
	Max.	4703.301	5.032	0.565	1.172	0.847
0.4	Ave.	5.217	36.921	20.579	0.525	0.317
	Max.	98.365	991.873	452.007	3.838	1.486
0.6	Ave.	0.806	0.406	0.649	0.630	0.880
	Max.	16.227	1.890	3.238	2.552	16.302
0.8	Ave.	0.338	0.410	0.538	0.578	0.564
	Max.	2.739	2.924	2.632	2.538	2.863
1.0	Ave.	0.128	0.172	0.360	0.401	0.550
	Max.	2.048	2.558	3.030	2.687	2.653

It is clear that the problem instances with smaller τ 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 τ and R seems easier to solve in Liaw et al. (2003). It is because they treated $R\|\sum w_jT_j$, 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 $R \leqslant 0.4$.

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 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 s. Thus in this case the average and maximum CPU times and

Table 4
Effect of the lower bound improvement on computational times

Number of instances			m								
			2	3	4	5	6	7	8	9	10
			64	56	41	31	24	21	7	13	8
(a) $n = 20$											
(A)	Time	Ave.	1.221	0.879	0.566	0.368	0.275	0.262	0.228	1.182	0.346
		Max.	2.282	4.467	4.416	0.662	0.565	0.503	0.393	10.886	0.698
	Node	Ave.	36,028	217,221	164,277	13,444	12,876	10,513	12,574	1,433,149	117,789
		Max.	478,004	5,877,356	6,334,218	153,498	140,119	31,582	33,914	16,803,230	732,266
(B)	Time	Ave.	1.337	1.194	0.702	0.373	0.278	0.260	0.228	1.159	0.344
		Max.	4.218	17.393	9.883	0.662	0.570	0.500	0.391	10.576	0.671
	Node	Ave.	175,712	639,669	379,965	23,078	13,828	10,853	12,794	1,447,177	132,038
		Max.	3,677,057	23,227,181	15,027,275	342,505	141,682	32,747	33,924	16,808,410	846,100
(C)	Time	Ave.	387.653	21.870	135.324	2.424	0.298	0.261	0.223	1.076	0.344
		Max.	9621.086	943.029	5505.958	62.598	0.753	0.499	0.400	9.472	0.707
	Node	Ave.	1,592,650,484	84,994,506	488,384,273	8,099,768	88,082	12,406	12,311	2,337,923	249,338
		Max.	40,615,449,451	3,798,862,538	19,936,585,429	245,482,207	1,069,544	35,779	37,780	26,917,324	1,544,446
			70	53	51	53	30	35	25	17	9
(b) $n = 25$											
(A)	Time	Ave.	2.801	50.797	130.333	37.600	1.769	0.711	0.628	1.550	0.790
		Max.	14.954	885.132	4703.301	1719.265	31.901	3.238	1.184	16.302	3.838
	Node	Ave.	968,394	63,969,962	174,139,084	45,619,355	1,468,712	266,359	120,606	1,465,728	559,539
		Max.	17,713,505	1,146,752,242	6,260,506,105	2,109,984,069	37,155,822	3,045,539	657,458	22,364,993	4,448,242
(B)	Time	Ave.	8.702	365.592	799.505	208.348	3.469	0.783	0.620	1.410	0.789
		Max.	259.715	11,768.664	36,689.496	10,631.435	75.994	3.190	1.167	14.078	3.831
	Node	Ave.	6,690,022	445,964,109	1,056,602,500	293,879,986	4,230,306	410,512	134,801	1,469,788	560,107
		Max.	246,345,216	13,775,008,021	48,243,633,225	15,050,139,269	109,705,840	3,541,277	708,900	22,397,383	4,448,242

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).

Appendix 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 $L^{\mathrm{S}}{:=}L^{\mathrm{SPT}}(\mathscr{J})$ and construct a schedule $\mathscr{S}^{\mathrm{S}}$ by applying $\mathscr{D}(L^{\mathrm{S}})$. Let $n^{\mathrm{S}}{:=}n$.
(1) Let

$$m^{\mathrm{S}}{:=}\arg\max_{1\leqslant j\leqslant m} R_j. \tag{34}$$

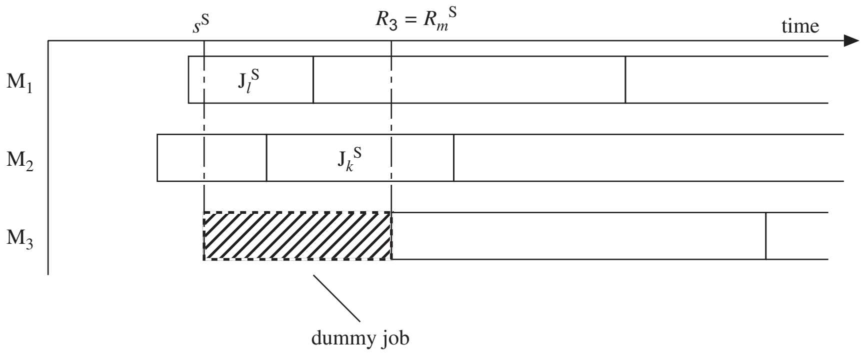

Fig. A1. Choice of s^S.

If $R_{m^S} = 0$, terminate.

(2) Let

$$k^S := \arg \min_{\substack{J_j \in L^S \\ R_{m^S} \leqslant C_j}} C_j. \tag{35}$$

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 Fig. A1). Here, it should be noted that $C_{l^S} - p_{l^S} \leqslant s^S \leqslant C_{k^S} - p_{k^S}$ and $p_{l^S} \leqslant R_{m^S} - s^S \leqslant p_{k^S}$ hold.

(4) $n^S := n^S + 1$. Generate a dummy job J_{n^S} such that $p_{n^S} = R_{m^S} - s^S$ and $C_{n^S} = d_{n^S} = R_{m^S}$. Insert J_{n^S} immediately before J_{k^S} in L^S. Let $R_{m^S} := s^S$ and go to (2).

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

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

Appendix B. A procedure to remove on time jobs

A procedure to remove on time jobs satisfying (18) is given as follows.

```
(0) $\mathscr{J}' := \mathscr{J}$, $L := \phi$.
(1) $j = 1$.
(2) If $J_j \notin \mathscr{J}'$, go to (5).
(3) If $J_j$ satisfies $d_j \geqslant \sum_{J_i \in \mathscr{J}'} p_i/m + (m-1)p_j/m$, go
    to (4). Otherwise, go to (5).
(4) Remove $J_j$ from $\mathscr{J}'$ and insert $J_j$ into the first
    position of $L$. Go to (1).
(5) $j := j + 1$. If $j \leqslant n$, go to (2).
```

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

References

Azizoglu, M., Kirca, O., 1998. Tardiness minimization on parallel machines. International Journal of Production Economics 55, 163–168.

Babu, P., Peridy, L., Pinson, E., 2004. A branch and bound algorithm to minimize total weighted tardiness on a single processor. Annals of Operations Research 129, 33–46.

Barnes, J.W., Brennan, J.J., 1977. An improved algorithm for scheduling jobs on identical machines. AIIE Transactions 9, 25–31.

Elmaghraby, S.E., Park, S.H., 1974. Scheduling jobs on a number of identical machines. AIIE Transactions 6, 1–13.

Emmons, H., 1969. One-machine sequencing to minimize certain functions of job tardiness. Operations Research 17, 701–715.

Fisher, M.L., 1976. A dual algorithm for the one-machine scheduling problem. Mathematical Programming 11, 229–251.

Fisher, M.L., 1981. The Lagrangian relaxation method for solving integer programming problems. Management Science 27, 1–18.

Ho, J.C., Chang, Y.-L., 1991. Heuristics for minimizing mean tardiness for m parallel machines. Naval Research Logistics 38, 367–381.

Koulamas, C., 1994. The total tardiness problem: Review and extensions. Operations Research 42, 1025–1041.

Koulamas, C., 1997. Polynomially solvable total tardiness problems: Review and extensions. Omega 25, 235–239.

Lawler, E.L., 1977. A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. Annals of Discrete Mathematics 1, 331–342.

Liaw, C.-F., Lin, Y.-K., Cheng, C.-Y., Chen, M., 2003. Scheduling unrelated parallel machines to minimize total weighted tardiness. Computers & Operations Research 30, 1777–1789.

Luh, P.B., Hoitomt, D.J., Max, E., Pattipati, K.R., 1990. Schedule generation and reconfiguration for parallel machines. IEEE Transactions on Robotics and Automation 6, 687–696.

Root, J.G., 1965. Scheduling with deadlines and loss functions on k parallel machines. Management Science 11, 460–475.

Szwarc, W., Croce, F.D., Grosso, A., 1999. Solution of the single machine total tardiness problem. Journal of Scheduling 2, 55–71.

Tansel, B.C., Kara, B.Y., Sabuncuoglu, I., 2001. An efficient algorithm for the single machine total tardiness problem. IIE Transactions 33, 661–674.

Wilkerson, L.J., Irwin, J.D., 1971. An improved method for scheduling independent tasks. AIIE Transactions 3, 239–245.

Yalaoui, F., Chu, C., 2002. Parallel machine scheduling to minimize total tardiness. International Journal of Production Economics 76, 265–279.