A dynamic-programming-based exact algorithm for general single-machine scheduling with machine idle time

Tanaka, Shunji; Fujikuma, Shuji

Journal of Scheduling manuscript No.
(will be inserted by the editor)

A Dynamic-Programming-Based Exact Algorithm for General Single-Machine Scheduling with Machine Idle Time

Shunji Tanaka · **Shuji Fujikuma**

Received: date / Accepted: date

Abstract This paper proposes an efficient exact algorithm for the general single-machine scheduling problem where machine idle time is permitted. The algorithm is an extension of the authors' previous algorithm for the problem without machine idle time, which is based on the SSDP (Successive Sublimation Dynamic Programming) method. We first extend our previous algorithm to the problem with machine idle time and next propose several improvements. Then, the proposed algorithm is applied to four types of single-machine scheduling problems: the total weighted earliness-tardiness problem with equal (zero) release dates, that with distinct release dates, the total weighted completion time problem with distinct release dates, and the total weighted tardiness problem with distinct release dates. Computational experiments demonstrate that our algorithm outperforms existing exact algorithms and can solve instances of the first three problems with up to 200 jobs and those of the last problem with up to 80 jobs.

Keywords Single-machine scheduling · Machine idle time · Exact algorithm · Lagrangian relaxation · Dynamic programming

1 Introduction

In this study we consider the general single-machine scheduling problem to minimize total job completion cost where machine idle time is permitted. Assume that n jobs (job 1, job 2, ..., job n) are to be processed on a single machine that can process at most one job at a time. Each job $i \in \mathcal{N} = \{1, 2, \ldots, n\}$ is given an integer processing time $p_i > 0$ and an integer release date $r_i \geq 0$. A cost function $f_i(t)$ ($t \geq r_i + p_i$) is also given for each job i and the cost $f(t)$ is incurred when job i is completed at t. We assume that the completion time C_i of job i is integral and that $f_i(t)$ is an integer-valued function for integer t. No preemption is allowed and all the jobs should be started and completed in the interval $[T_{\mathrm{S}}, T_{\mathrm{E}}]$, where $T_{\mathrm{S}} = \min_{i \in \mathcal{N}} r_i$. The machine can be idle even when there remain unprocessed jobs. The objective is to minimize the total job completion cost $\sum_{i \in \mathcal{N}} f_i(C_i)$.

For a special class of this problem, the single-machine scheduling problem without machine idle time (and with equal release dates), the authors (Tanaka et al. 2009) already proposed an efficient exact algorithm. This algorithm is an improvement of the algorithm proposed by Ibaraki and Nakamura (1994) that is based on the SSDP (Successive Sublimation Dynamic Programming) method (Ibaraki 1987). In this algorithm a lower bound is computed by solving a Lagrangian relaxation of the original problem via dynamic programming. Then, it is improved by successively adding constraints (cuts) to the relaxation until the gap between the lower and upper bounds disappears. One of the important features of the algorithm is that unnecessary dynamic programming states are eliminated in the course of the algorithm to suppress the increase of states caused by the addition of the constraints. This state elimination is also effective for reduction of computational efforts. In our previous study (Tanaka et al. 2009), it is shown that the algorithm can handle 300 jobs instances when it is applied to the total weighted tardiness problem ($1||\sum w_i T_i$, according to the standard classification of scheduling problems in Graham et al. (1979)) and the total weighted earliness-tardiness problem ($1||\sum(\alpha_i E_i + \beta_i T_i)$) without machine idle time. In this study we will extend this algorithm to a more general single-machine problem where machine idle time is permitted. We will also propose several improvements:

S. Tanaka and S. Fujikuma
Department of Electrical Engineering, Kyoto University
Kyotodaigaku-Katsura, Nishikyo-ku, Kyoto 615-8510, Japan
Tel.: +81-75-383-2204
Fax: +81-75-383-2201
E-mail: tanaka@kuee.kyoto-u.ac.jp

- the conjugate subgradient algorithm for adjustment of Lagrangian multipliers instead of the ordinary subgradient algorithm,
- state elimination (network reduction) by the constraint propagation technique,
- network reduction by node compression,
- strict check of dominance of successive jobs,
- introduction of a tentative upper bound.

It should be noted that a similar lower bounding scheme as in our previous algorithm and in the proposed algorithm is used in the branch-and-bound algorithm for the single-machine total weighted earliness-tardiness problem ($1||\sum(\alpha_i E_i+\beta_i T_i)$ and $1|r_i|\sum(\alpha_i E_i+\beta_i T_i)$) by Sourd (2009). Furthermore, he has already proposed to utilize the constraint propagation technique in their algorithm, which we will also introduce in this paper. As pointed out in Tanaka et al. (2009), one of the primary differences is that our algorithm is based fully on dynamic programming, while his one is a branch-and-bound algorithm. This difference makes our algorithm much faster than the Sourd's algorithm, which will be shown by numerical results for benchmark instances of $1||\sum(\alpha_i E_i+\beta_i T_i)$ and $1|r_i|\sum(\alpha_i E_i+\beta_i T_i)$. We will also show that our algorithm outperforms the current best algorithm by Pan and Shi (2008) for the single-machine total weighted completion time problem with distinct release dates ($1|r_i|\sum w_i C_i$), and that by Jouglet et al. (2004) for the single-machine total weighted tardiness time problem with distinct release dates ($1|r_i|\sum w_i T_i$).

This paper is organized as follows. First, in Section 2, our problem is converted to a constrained shortest path problem. Some notation and definitions are also introduced there. Next, in Section 3, the Lagrangian relaxation technique is applied to compute lower bounds by dynamic programming. Then, Section 4 describes several methods for the elimination of unnecessary dynamic programming states in terms of the reduction of the network. An outline of the proposed algorithm is shown in Section 5, and the upper bound computation scheme used therein is given in Section 6. The algorithm is further improved in Section 7, and the final version of the algorithm is summarized in Section 8. In Section 9, the algorithm is applied to the benchmark instances and its effectiveness is confirmed. Finally, in Section 10, our contributions and future research directions are stated.

2 Network Representation

In the proposed algorithm, as in our previous algorithm, the Lagrangian relaxation technique is applied to obtain a tight lower bound. In Tanaka et al. (2009), it was explained in terms of the time-indexed formulation (Pritsker et al. 1969, Dyer and Wolsey 1990, Sousa and Wolsey 1992, van den Akker et al. 1999) of the problem, but here we start from a network representation to make the description as concise as possible. The method to compute lower bounds will be explained in the next section.

Unlike the problem in Tanaka et al. (2009), machine idle time should be considered in our problem. Fortunately, the extension is not difficult: Our problem can be converted into a problem without machine idle time by introducing zero cost dummy jobs with a unit processing time that correspond to unit idle times. However, this direct extension is not efficient because the total number of jobs to be considered increases from n to $n+n_\mathrm{d}$, where $n_\mathrm{d}=T_\mathrm{E}-T_\mathrm{S}-\sum_{i\in\mathcal{N}} p_i$ is the number of such dummy jobs. To get around this, only one dummy job is introduced instead of distinct n_d dummy jobs and it is assumed to be processed n_d times. Hereafter, the dummy job is referred to as "idle job" and is denoted by job 0. The processing time, release date and cost function of job 0 are defined by $p_0=1$, $r_0=T_\mathrm{S}$, $f_0(t)=0$ ($T_\mathrm{S}+1\le t\le T_\mathrm{E}$), respectively. In addition, the set of all jobs including the idle job is defined by $\mathcal{N}_0=\mathcal{N}\cup\{0\}$.

By noting the assumption that job completion times are integral, we allocate one node to every possible completion of jobs (including the idle job), and construct an acyclic weighted directed graph $G=(V,A)$. Thus, the node set V is defined by

$$V=\{v_{n+1,T_\mathrm{S}}\}\cup V_\mathrm{O}\cup\{v_{n+1,T_\mathrm{E}+1}\}, \tag{1}$$
$$V_\mathrm{O}=\{v_{it}\,|\,i\in\mathcal{N}_0,\ r_i+p_i\le t\le T_\mathrm{E}\}. \tag{2}$$

Here, another dummy job $n+1$ with $p_{n+1}=1$, $r_{n+1}=T_\mathrm{S}-1$ and $f_{n+1}(t)=0$ is introduced. It is always completed at T_S and $T_\mathrm{E}+1$, and v_{n+1,T_S} and $v_{n+1,T_\mathrm{E}+1}$ denote the source and sink nodes, respectively. The arc set A is defined accordingly by

$$A=\{(v_{j,t-p_i},v_{it})\,|\,v_{j,t-p_i},v_{it}\in V\}. \tag{3}$$

Each arc has its length (weight): The length of an arc $(v_{j,t-p_i},v_{it})\in A$ is given by $f_i(t)$. Then, our problem, which is referred to as (P), becomes equivalent to the shortest path problem from v_{n+1,T_S} to $v_{n+1,T_\mathrm{E}+1}$ on G under the constraints that v_{it} ($r_i+p_i\le t\le T_\mathrm{E}$) should be visited exactly once for any $i\in\mathcal{N}$. More specifically, the optimal objective value, i.e., the minimum of the total job completion cost is identical to the shortest path length, and v_{it} visited on the shortest path corresponds to the completion of job i at t in an optimal solution.

Here, we introduce some notation and definitions. Let us define by $\mathcal{P}$ a set of nodes visited on a path from v_{n+1,T_S} to $v_{n+1,T_\mathrm{E}+1}$ on G. The path corresponding to $\mathcal{P}$ is referred to as "path $\mathcal{P}$" if there is no ambiguity. Let $L(\mathcal{P})$ be the length of a path $\mathcal{P}$ defined by

$$L(\mathcal{P})=\sum_{\substack{v_{it}\in\mathcal{P}\\ i\in\mathcal{N}_0}} f_i(t)=\sum_{\substack{v_{it}\in\mathcal{P}\\ i\in\mathcal{N}}} f_i(t). \tag{4}$$

The constraints in our problem that v_{it} $(r_i + p_i \leq t \leq T_E)$ should be visited exactly once for any $i \in \mathscr{N}$ on a path $\mathscr{P}$ are written by

$$\mathscr{V}_i(\mathscr{P}) = |\{v_{it} \,|\, v_{it} \in \mathscr{P}\}| = 1, \quad \forall i \in \mathscr{N}, \tag{5}$$

where $\mathscr{V}_i(\mathscr{P})$ denotes the number of occurrences of v_{it} $(r_i + p_i \leq t \leq T_E)$ in $\mathscr{P}$. Accordingly, define a set of all the feasible paths by

$$\mathscr{Q} = \{\mathscr{P} \,|\, \mathscr{V}_i(\mathscr{P}) = 1 \ (\forall i \in \mathscr{N})\}. \tag{6}$$

Then, the problem (P) can be described by

$$(\mathrm{P}): \ \min_{\mathscr{P}} L(\mathscr{P}) \quad \text{s.t. } \mathscr{P} \in \mathscr{Q}. \tag{7}$$

Please note that the constraint that the idle job should be processed n_d times is not imposed on the problem. It is because it is automatically satisfied when the constraints (5) are satisfied.

3 Lower bound computation

To obtain a tight lower bound of the constrained shortest path problem (P), the Lagrangian relaxation technique is applied. This relaxation is also referred to as state-space relaxation, which was originally proposed by Christofides et al. (1981) for routing problems. Then it was applied to single-machine scheduling by Abdul-Razaq and Potts (1988). Following their results, Ibaraki and Nakamura (1994) proposed an exact algorithm based on the SSDP (Successive Sublimation Dynamic Programming) method (Ibaraki 1987), and it was improved in our previous study (Tanaka et al. 2009). This type of relaxation also appears in the context of column generation for the time-indexed formulation (van den Akker et al. 2000), or in the context of branch-and-bound algorithms for single-machine scheduling problems (Péridy et al. 2003, Sourd 2009). Especially in Sourd (Sourd 2009), a similar improvement to that in Tanaka et al. (2009) was proposed. It will be explained in 3.2.

Let us penalize the violation of the constraints (5) by Lagrangian multipliers μ_i $(i \in \mathscr{N})$. Then, the objective function of (P) becomes

$$\begin{aligned} &L(\mathscr{P}) + \sum_{i \in \mathscr{N}} \mu_i (1 - \mathscr{V}_i(\mathscr{P})) \\ &\quad = \sum_{\substack{v_{it} \in \mathscr{P} \\ i \in \mathscr{N}}} f_i(t) + \sum_{i \in \mathscr{N}} \mu_i - \sum_{i \in \mathscr{N}} \mu_i |\{v_{it} \,|\, v_{it} \in \mathscr{P}\}| \\ &\quad = \sum_{\substack{v_{it} \in \mathscr{P} \\ i \in \mathscr{N}}} (f_i(t) - \mu_i) + \sum_{i \in \mathscr{N}} \mu_i \\ &\quad = L_R(\mathscr{P};\, \boldsymbol{\mu}) + \sum_{i \in \mathscr{N}} \mu_i, \end{aligned} \tag{8}$$

where $L_R(\mathscr{P};\, \boldsymbol{\mu})$ is defined by

$$L_R(\mathscr{P};\, \boldsymbol{\mu}) = \sum_{\substack{v_{it} \in \mathscr{P} \\ i \in \mathscr{N}}} (f_i(t) - \mu_i). \tag{9}$$

Equations (8) and (9) imply that the relaxation for a fixed set of multipliers is equivalent to the problem to find a shortest *unconstrained* path from v_{n+1,T_S} to v_{n+1,T_E+1} on G where the length of an arc $(v_{j,t-p_i}, v_{it}) \in A$ is given not by $f_i(t)$ but by $f_i(t) - \mu_i$ (we assume that $\mu_0 = \mu_{n+1} = 0$). This relaxation is denoted by (LR_0), i.e.,

$$(\mathrm{LR}_0): \ \min_{\mathscr{P}} L_R(\mathscr{P};\, \boldsymbol{\mu}) + \sum_{i \in \mathscr{N}} \mu_i. \tag{10}$$

It is easy to see that (LR_0) is solvable in $O(n(T_E - T_S))$ time by dynamic programming as shown by Abdul-Razaq and Potts (1988).

To improve the lower bound more, the following three types of constraints are imposed on this relaxation. The first and the third were proposed by Christofides et al. (1981) and were applied in Abdul-Razaq and Potts (1988) and Ibaraki and Nakamura (1994). The second constraints were proposed by Sourd (2009) and Tanaka et al. (2009), and were applied in our previous algorithm together with the other two.

3.1 Constraints on successive jobs

The first constraints are to forbid job duplication in successive jobs of a solution. In the network representation, these are interpreted as constraints on successively visited nodes on a path. More specifically, they are described as follows.

> For any $i \in \mathscr{N}$, nodes corresponding to job i, i.e., v_{it} $(r_i + p_i \leq t \leq T_E)$ should not be visited more than once in any $\lambda + 1 > 0$ successive nodes on a path.

Note that these constraints are not imposed on the idle job. It corresponds to the fact that the constraints (5) are not imposed on the idle job. A subset of paths satisfying these constraints on successive nodes is denoted by $\mathscr{Q}_\lambda$ $(\mathscr{Q} \subseteq \cdots \subseteq \mathscr{Q}_2 \subseteq \mathscr{Q}_1)$, and the relaxation with the constraints is denoted by

$$(\mathrm{LR}_\lambda): \ \min_{\mathscr{P}} L_R(\mathscr{P};\, \boldsymbol{\mu}) + \sum_{i \in \mathscr{N}} \mu_i \quad \text{s.t. } \mathscr{P} \in \mathscr{Q}_\lambda. \tag{11}$$

In our algorithm, the cases when $\lambda = 1, 2$ are considered. For $\lambda = 1$, (LR_1) becomes more tractable if we introduce a weighted directed graph $G_S = (V, A_S)$, where A_S is defined by

$$A_S = A \backslash \{(v_{i,t-p_i}, v_{it}) \,|\, i \in \mathscr{N},\ v_{i,t-p_i}, v_{it} \in V_O\}. \tag{12}$$

Indeed, (LR_1) is equivalent to the unconstrained shortest path problem on G_S. On the other hand, (LR_2) is equivalent to the shortest path problem on G_S under the constraints on three successive nodes. These relaxations can be solved by dynamic programming and their time complexities are $O(n(T_E - T_S))$ and $O(n^2(T_E - T_S))$, respectively (Abdul-Razaq and Potts 1988, Péridy et al. 2003).

3.2 Constraints on adjacent pairs of jobs

The second constraints are derived from the dominance theorem of dynamic programming (Potts and Van Wassenhove 1985) for adjacent pairs of jobs. For example, consider that two jobs i and j ($i, j \in \mathscr{N}_0$, $i \neq j$) are successively processed and completed at t ($\max\{r_i, r_j\} + p_i + p_j \leq t \leq T_E$). The total completion cost of the two jobs is $f_i(t - p_j) + f_j(t)$ when they are sequenced as $i \to j$, and $f_j(t - p_i) + f_i(t)$ when sequenced as $j \to i$. It follows that $i \to j$ never occurs at t in an optimal solution if $f_i(t - p_j) + f_j(t) > f_j(t - p_i) + f_i(t)$ because interchanging these jobs decreases the objective value without affecting the other jobs. On the other hand, $j \to i$ never occurs at t if $f_i(t - p_j) + f_j(t) < f_j(t - p_i) + f_i(t)$. Therefore, the processing order of jobs i and j at t can be restricted by checking the total cost of the two. This also holds even if $f_i(t - p_j) + f_j(t) = f_j(t - p_i) + f_i(t)$, and either (but not arbitrary) processing order can be forbidden without loss of optimality (Tanaka et al. 2009). To summarize, the processing order of adjacent pairs of jobs can be restricted and it is imposed on the relaxation as constraints. Since we should take the idle job into account in this study, the processing order of an ordinary job (job $i \in \mathscr{N}$) and the idle job (job 0) is also restricted.

In the network representation, these adjacency constraints eliminate from G_S, those arcs corresponding to the forbidden processing orders. Thus, we define $\widehat{G}_S = (V, \widehat{A}_S)$, where

$$\widehat{A}_S = A_S \setminus \{(v_{j,t-p_i}, v_{it}) \mid j \to i \text{ is forbidden at } t\}. \tag{13}$$

(LR_1) and (LR_2) with the adjacency constraints are equivalent to the unconstrained and constrained shortest path problems on $\widehat{G}_S$, respectively. Since the time complexities of (LR_1) and (LR_2) with the adjacency constraints are both $O(n^2(T_E - T_S))$ (Sourd 2009, Tanaka et al. 2009), only (LR_2) with the adjacency constraints, which is denoted by ($\widehat{LR}_2$), is used as in our previous algorithm. Let $\widehat{\mathscr{Q}}_2$ denote a subset of $\mathscr{Q}_2$ composed of paths on $\widehat{G}_S$ that satisfy the constraints on three successive nodes.

3.3 Constraints on state-space modifiers

The last constraints are described in terms of state-space modifiers: Each job i ($i \in \mathscr{N}$) is given a value $q_i \geq 0$ called state-space modifier and we impose the constraint that the total modifier in a solution should be $\sum_{i \in \mathscr{N}} q_i$. In Tanaka et al. (2009), modifiers are chosen so that $q_i = 1$ for some i and $q_j = 0$ for $j \in \mathscr{N} \setminus \{i\}$. In this case the constraint simply requires that job i should be processed exactly once and thus is equivalent to $\mathscr{V}_i(\mathscr{P}) = 1$, i.e. the constraint (5) for job i. It follows that all the constraints (5) are once relaxed, but one of them is recovered to improve the lower bound.

In our algorithm, as in Tanaka et al. (2009), not the constraint (5) for a single job i but those for a set of jobs $\mathscr{M} \subseteq \mathscr{N}$ are recovered to ($\widehat{LR}_2$). Hereafter, ($\widehat{LR}_2$) with the constraints

$$\mathscr{V}_i(\mathscr{P}) = 1, \quad \forall i \in \mathscr{M} \tag{14}$$

is denoted by ($\widehat{LR}_2^m$), where $m = |\mathscr{M}|$. Clearly, an optimal solution of ($\widehat{LR}_2^m$) is also optimal for the original problem (P) if $\mathscr{M} = \mathscr{N}$. It is also clear from (5) that it is not necessary to impose these constraints on the idle job.

The network representation of ($\widehat{LR}_2^m$) is a little complicated. Let us define an m-dimensional vector $\boldsymbol{q}_i^m$ of state-space modifiers for job i by $\boldsymbol{q}_i^m = (q_{i1}, \ldots, q_{im})$, where

$$q_{ij} = \begin{cases} 1, & \text{if the } j\text{th element of } \mathscr{M} \text{ is } i, \\ 0, & \text{otherwise.} \end{cases} \tag{15}$$

Let us also define m-dimensional vectors $\boldsymbol{q}_0^m$ and $\boldsymbol{q}_{n+1}^m$ by $\boldsymbol{q}_0^m = \boldsymbol{q}_{n+1}^m = (0, \ldots, 0)$. Next, we introduce a weighted directed graph $\widehat{G}_S^m = (V^m, \widehat{A}_S^m)$. The node set V^m is defined by

$$V^m = \{v_{n+1,T_S}^{\boldsymbol{0}_m}\} \cup V_O^m \cup \{v_{n+1,T_E+1}^{\boldsymbol{1}_m}\}, \tag{16}$$

$$V_O^m = \{v_{it}^{\boldsymbol{b}} \mid v_{it} \in V_O,\ \boldsymbol{q}_i^m \leq \boldsymbol{b} \leq \boldsymbol{1}_m\}, \tag{17}$$

where $\boldsymbol{0}_m$ and $\boldsymbol{1}_m$ denote m-dimensional vectors whose elements are all zero and all one, respectively. The arc set $\widehat{A}_S^m$ is defined by

$$\widehat{A}_S^m = \{(v_{j,t-p_i}^{\boldsymbol{b}-\boldsymbol{q}_i^m}, v_{it}^{\boldsymbol{b}}) \mid (v_{j,t-p_i}, v_{it}) \in \widehat{A}_S,\ \boldsymbol{q}_i^m + \boldsymbol{q}_j^m \leq \boldsymbol{b} \leq \boldsymbol{1}_m\}, \tag{18}$$

and the length of an arc $(v_{j,t-p_i}^{\boldsymbol{b}-\boldsymbol{q}_i^m}, v_{it}^{\boldsymbol{b}})$ is given by $f_i(t) - \mu_i$. Then, ($\widehat{LR}_2^m$) is equivalent to the shortest path problem from $v_{n+1,T_S}^{\boldsymbol{0}_m}$ to $v_{n+1,T_E+1}^{\boldsymbol{1}_m}$ on $\widehat{G}_S^m$ under the constraints on three successive nodes. Hereafter, a set of paths from $v_{n+1,T_S}^{\boldsymbol{0}_m}$ to $v_{n+1,T_E+1}^{\boldsymbol{1}_m}$ on $\widehat{G}_S^m$ that satisfy the constraints on three successive nodes is denoted by $\widehat{\mathscr{Q}}_2^m$. ($\widehat{LR}_2^m$) is solvable by dynamic programming in $O(n^2 2^m (T_E - T_S))$ time.

4 Network reduction

As explained in the preceding section, (LR_1), ($\widehat{LR}_2$) and ($\widehat{LR}_2^m$) are solvable by dynamic programming. To reduce memory usage and improve its efficiency, unnecessary dynamic programming states are eliminated, which is a key technique of the SSDP method. It is interpreted in the network representation as the network reduction via the elimination of unnecessary nodes and arcs. In Tanaka et al. (2009), the following two types of reductions were applied.

4.1 Network reduction by upper bound

The first network reduction that was proposed by Ibaraki and Nakamura (1994) utilizes an upper bound and is applied to all the relaxations (LR_1), ($\widehat{\mathrm{LR}}_2$) and ($\widehat{\mathrm{LR}}_2^m$). Here, only an explanation for ($\widehat{\mathrm{LR}}_2$) is given here because it does not differ much from those for (LR_1) and ($\widehat{\mathrm{LR}}_2^m$).

Let us define $h_2((v_{k,t-p_j}, v_{jt});\, \boldsymbol{\mu})$ and $H_2(v_{k,t-p_j}, v_{jt};\, \boldsymbol{\mu})$ ($(v_{k,t-p_j}, v_{jt}) \in \widehat{A}_{\mathrm{S}}$) by

$$h_2((v_{k,t-p_j}, v_{jt});\, \boldsymbol{\mu}) = \min_{\substack{\mathscr{P} \in \widehat{\mathscr{Q}}_2 \\ v_{k,t-p_j}, v_{jt} \in \mathscr{P}}} L_{\mathrm{R}}(\{v_{is} \,|\, v_{is} \in \mathscr{P},\ s \le t\}), \tag{19}$$

$$H_2((v_{k,t-p_j}, v_{jt});\, \boldsymbol{\mu}) = \min_{\substack{\mathscr{P} \in \widehat{\mathscr{Q}}_2 \\ v_{k,t-p_j}, v_{jt} \in \mathscr{P}}} L_{\mathrm{R}}(\{v_{is} \,|\, v_{is} \in \mathscr{P},\ s \ge t\}). \tag{20}$$

More specifically, $h_2((v_{k,t-p_j}, v_{jt});\, \boldsymbol{\mu})$ denotes the shortest path length from $v_{n+1,T_{\mathrm{S}}}$ to v_{jt} that passes through $(v_{k,t-p_j}, v_{jt})$ and that satisfies the constraints on three successive nodes. Similarly, $H_2((v_{k,t-p_j}, v_{jt});\, \boldsymbol{\mu})$ denotes the shortest path length from $v_{k,t-p_j}$ to $v_{n+1,T_{\mathrm{E}}+1}$ that passes through $(v_{k,t-p_j}, v_{jt})$ and that satisfies the constraints.

The summation of (19) and (20) yields

$$\begin{aligned} &h_2((v_{k,t-p_j}, v_{jt});\, \boldsymbol{\mu}) + H_2((v_{k,t-p_j}, v_{jt});\, \boldsymbol{\mu}) \\ &\quad = \min_{\substack{\mathscr{P} \in \widehat{\mathscr{Q}}_2 \\ v_{k,t-p_j}, v_{jt} \in \mathscr{P}}} L_{\mathrm{R}}(\mathscr{P};\, \boldsymbol{\mu}) + (f_j(t) - \mu_j). \end{aligned} \tag{21}$$

Since the first term of the righthand side of (21) gives the shortest path length over $\widehat{\mathscr{Q}}_2$ under the constraint that $(v_{k,t-p_j}, v_{jt})$ should be passed through, it can be said from (8) that the shortest path over $\widehat{\mathscr{Q}}_2$ never passes through $(v_{k,t-p_j}, v_{jt})$ if an upper bound UB of $L(\mathscr{P})$ satisfies

$$\begin{aligned} \mathrm{UB} < {} & h_2((v_{k,t-p_j}, v_{jt});\, \boldsymbol{\mu}) + H_2((v_{k,t-p_j}, v_{jt});\, \boldsymbol{\mu}) \\ & - (f_j(t) - \mu_j) + \sum_{i \in \mathscr{N}} \mu_i. \end{aligned} \tag{22}$$

In this case, the arc $(v_{k,t-p_j}, v_{jt})$ can be eliminated from $\widehat{G}_{\mathrm{S}}$.

$h_2((v_{k,t-p_j}, v_{jt});\, \boldsymbol{\mu})$ and $H_2((v_{k,t-p_j}, v_{jt});\, \boldsymbol{\mu})$ for every arc are recursively computed in the dynamic programming for ($\widehat{\mathrm{LR}}_2$): $h_2((v_{k,t-p_j}, v_{jt});\, \boldsymbol{\mu})$ in the forward dynamic programming and $H_2((v_{k,t-p_j}, v_{jt});\, \boldsymbol{\mu})$ in the backward dynamic programming. Therefore, this reduction can be performed by applying the dynamic programming in both the directions. If arcs are eliminated, the graph size reduces and, as a result, both memory usage and computational efforts for the dynamic programming reduce.

In practice, $(\mathrm{UB} - 1)$ instead of UB is used in the left-hand side of (22) to eliminate the arc $(v_{k,t-p_j}, v_{jt})$. It is because the cost function is assumed to be integer-valued.

4.2 Network reduction by dominance of four successive jobs

To reduce memory usage more, network reduction by dominance of four successive jobs (Tanaka et al. 2009) is applied to ($\widehat{\mathrm{LR}}_2^m$). Let us define a set of paths $\mathscr{Q}^m$ by

$$\mathscr{Q}^m = \{\mathscr{P} \,|\, \mathscr{P} \in \widehat{\mathscr{Q}}_2^m,\ \mathscr{V}_i(\mathscr{P}) = 1\ (\forall i \in \mathscr{N})\}. \tag{23}$$

More specifically, $\mathscr{Q}^m$ is a set of paths on $\widehat{G}_{\mathrm{S}}^m$ that correspond to the paths belonging to $\mathscr{Q}$ on G or, equivalently, feasible solutions of (P). Let us also define a path $\mathscr{P}_{\mathrm{opt}}^m$ corresponding to an optimal solution of (P) by

$$\mathscr{P}_{\mathrm{opt}}^m = \arg \min_{\mathscr{P} \in \mathscr{Q}^m} L(\mathscr{P}). \tag{24}$$

If, for every $\mathscr{P} \in \mathscr{Q}^m$ passing through the arc $(v_{j,t-p_i}^{\boldsymbol{b}-\boldsymbol{q}_i^m}, v_{it}^{\boldsymbol{b}})$, there exists a dominating path $\mathscr{P}' \in \mathscr{Q}^m$ such that

$$L(\mathscr{P}') < L(\mathscr{P}), \tag{25}$$

$\mathscr{P}_{\mathrm{opt}}^m$ cannot pass through the arc and hence it can be eliminated. To check this, only four nodes $v_{l,t-p_i-p_j-p_k}^{\boldsymbol{b}-\boldsymbol{q}_i^m-\boldsymbol{q}_j^m-\boldsymbol{q}_k^m}$, $v_{k,t-p_i-p_j}^{\boldsymbol{b}-\boldsymbol{q}_i^m-\boldsymbol{q}_j^m}$, $v_{j,t-p_i}^{\boldsymbol{b}-\boldsymbol{q}_i^m}$ and $v_{it}^{\boldsymbol{b}}$ in $\mathscr{P}$ are considered in the forward dynamic programming. That is, $4! - 1$ paths are checked for one $\mathscr{P}$ as a candidate for a dominating path $\mathscr{P}'$, where the visiting orders of the four nodes are interchanged. Similarly, in the backward dynamic programming $v_{j,t-p_i}^{\boldsymbol{b}-\boldsymbol{q}_i^m}$, $v_{it}^{\boldsymbol{b}}$, $v_{k,t+p_k}^{\boldsymbol{b}+\boldsymbol{q}_k^m}$ and $v_{l,t+p_k+p_l}^{\boldsymbol{b}+\boldsymbol{q}_k^m+\boldsymbol{q}_l^m}$ are considered to eliminate $(v_{j,t-p_i}^{\boldsymbol{b}-\boldsymbol{q}_i^m}, v_{it}^{\boldsymbol{b}})$.

5 Outline of the proposed algorithm

Our previous algorithm in Tanaka et al. (2009) has three stages. When it is directly extended to our problem, the algorithm becomes as follows. In the first stage the subgradient algorithm is applied to the Lagrangian dual corresponding to (LR_1), and next in the second stage to the dual corresponding to ($\widehat{\mathrm{LR}}_2$), to adjust Lagrangian multipliers. Then, in the third stage ($\widehat{\mathrm{LR}}_2^m$) is successively solved by adding jobs from $\mathscr{N} \setminus \mathscr{M}$ to $\mathscr{M}$, until the gap between lower and upper bounds becomes less than one. The upper bound is computed and updated in the course of the algorithm by a combination of Lagrangian heuristics and a local search, and it is utilized in the network reduction in 4.1. The reduction in 4.2 is also performed.

It follows that now only the extension of upper bound computation is not finished yet, but in this study we propose several improvements together with the extensions. The extensions and improvements are summarized as:

Extensions:
- lower bound computation (Section 3),
- network reduction (Section 4),

– upper bound computation (Section 6).

Improvements:

– the conjugate subgradient algorithm instead of the ordinary subgradient algorithm,
– network reduction by the constraint propagation technique,
– network reduction by node compression,
– strict check of dominance of successive jobs in 4.2,
– introduction of a tentative upper bound.

The extension of the upper bound computation method will be described in the next section, and the improvements will be given in Section 7.

6 Upper bound computation

An upper bound is computed from a solution of $(\widehat{\mathrm{LR}}_2)$ or $(\widehat{\mathrm{LR}}_2^m)$. As in Tanaka et al. (2009), the computation consists of two parts: The solution is first converted to a feasible solution of (P) by some Lagrangian heuristics, and then it is improved by a neighborhood search. Here, the Lagrangian heuristics are first presented and then the neighborhood search is explained.

6.1 Lagrangian heuristics

In Tanaka et al. (2009), two types of heuristics are switched. The one is a slightly modified version of the heuristic proposed by Ibaraki and Nakamura (1994) and the other is a simple heuristic to "detour" a path on the network so that job duplication does not occur. On the other hand, in our algorithm we employ only an extension of the former heuristic, but optimal and greedy versions of it are switched.

The heuristic by Ibaraki and Nakamura (1994) is:

(1) A partial job sequence is generated from a solution of a relaxation by removing duplicated jobs. The number of jobs in the partial sequence is denoted by n_1.
(2) The other $n_2(= n - n_1)$ jobs are inserted optimally into the partial sequence without changing the precedence relations of the n_1 jobs. An optimal sequence can be obtained by dynamic programming in $O(n_2(n_1+1)2^{n_2})$ time.

In our algorithm it is extended to the problem with machine idle time as follows:

(1') A partial job sequence is generated from a solution of a relaxation by removing duplicated jobs and *the idle job*. The number of jobs in the partial sequence is denoted by n_1.
(2') The other $n_2(= n - n_1)$ ordinary jobs are inserted optimally into the partial sequence without changing the precedence relations of the n_1 jobs, where idle times are taken into account. In other words, when finding optimal job positions, the objective value is evaluated after idle times are optimally inserted. An optimal sequence can be obtained by dynamic programming in $O(n_2(n_1+1)2^{n_2}(T_\mathrm{E}-T_\mathrm{S}))$ time.

The dynamic programming in (2') of this heuristic is time- (and space-) consuming because its time complexity is multiplied by the length of the scheduling horizon $(T_\mathrm{E}-T_\mathrm{S})$. Therefore, we adopt the method in Sourd (2005), which was originally proposed to improve the efficiency of dynamic programming for optimal idle time insertion. In this method, the objective function is assumed to be piecewise linear and it is evaluated only at the endpoints of linear segments. If the cost function $f_i(t)$ is piecewise linear with few segments, it is much helpful to reduce computational efforts.

Nevertheless, it is hard to apply this heuristic when the number of jobs to be inserted, n_2 is large because the time complexity also depends on n_2 exponentially. Hence, we apply a greedy version of the heuristic when $n_2 \geq 9$. In this case, unscheduled n_2 jobs are inserted one by one according to the SPT (shortest processing time) order into their optimal positions. That is, the following procedure is used in place of (2') in the optimal version of the heuristic:

(2") The other n_2 ordinary jobs are inserted one by one according to the SPT order into their optimal positions of the partial sequence. Here, the precedence relations of the n_1 jobs are kept unchanged and idle times are taken into account.

6.2 Improvement by neighborhood search

To improve a solution obtained by the heuristics in the preceding subsection, the dynasearch is applied. It is a powerful neighborhood search proposed by Congram et al. (2002) for the single-machine scheduling problem without machine idle time. Grosso et al. (2004) proposed the enhanced dynasearch that improves the search ability of the dynasearch by enlarging the neighborhood, which was employed in Tanaka et al. (2009). Another extension of the (enhanced) dynasearch was done by Sourd (2006) based on the results in Sourd (2005), and it enables us to apply the dynasearch to the problem with machine idle time. In our proposed algorithm, both the extended enhanced dynasearch and the extended dynasearch are applied, but the latter is mainly employed because the former is a little time-consuming.

6.3 Initial upper bound

To obtain the initial upper bound, we first construct solutions by the greedy version of the heuristic stated in 6.1. In this case, all the jobs are assumed to be unscheduled ($n_1 = 0$), and not only the SPT order but also the LPT (longest processing time), EDD (earliest duedate) and LDD (latest duedate) orders are used (EDD and LDD are only for those problems with duedates). Then, the extended dynasearch is applied to the best of the four solutions.

7 Further Improvements

7.1 Conjugate subgradient algorithm

In the first two stages of our previous algorithm, the subgradient algorithm is employed to adjust Lagrangian multipliers. More specifically, the vector of Lagrangian multipliers $\boldsymbol{\mu}^{(k)}$ at the kth iteration is updated by the following equation.

$$\boldsymbol{\mu}^{(k+1)} = \boldsymbol{\mu}^{(k)} + \gamma^{(k)} \frac{\mathrm{UB} - \mathrm{LB}^{(k)}}{\|\boldsymbol{g}^{(k)}\|^2} \boldsymbol{g}^{(k)}. \tag{26}$$

Here, UB denotes the current upper bound and $\mathrm{LB}^{(k)}$ the optimal objective value of (LR_1) or ($\widehat{\mathrm{LR}}_2$) for $\boldsymbol{\mu}^{(k)}$. That is,

$$\mathrm{LB}^{(k)} = \min_{\mathscr{P}} L_{\mathrm{R}}(\mathscr{P};\, \boldsymbol{\mu}^{(k)}) + \sum_{i \in \mathscr{N}} \mu_i^{(k)}. \tag{27}$$

The subgradient vector $\boldsymbol{g}^{(k)}$ is chosen as

$$g_i^{(k)} = 1 - \mathscr{V}_i(\mathscr{P}^{(k)}), \tag{28}$$

where

$$\mathscr{P}^{(k)} = \arg\min_{\mathscr{P}} L_{\mathrm{R}}(\mathscr{P};\, \boldsymbol{\mu}^{(k)}). \tag{29}$$

To cope with the poor convergence of the subgradient algorithm, our previous algorithm controlled the step size parameter $\gamma^{(k)}$ in a more sophisticated way than in the ordinary version of the algorithm (Fisher 1985). Nonetheless, a considerable part of the total computational time was consumed by the subgradient algorithm especially when the problem size is large. Therefore, the proposed algorithm employs the conjugate subgradient algorithm in Sherali and Ulular (1989), Sherali and Lim (2007) instead to improve the convergence, and Lagrangian multipliers are updated by

$$\boldsymbol{d}^{(k)} = \boldsymbol{g}^{(k)} + \xi^{(k)} \boldsymbol{d}^{(k-1)}, \tag{30}$$

$$\boldsymbol{\mu}^{(k+1)} = \boldsymbol{\mu}^{(k)} + \gamma^{(k)} \frac{\mathrm{UB} - \mathrm{LB}^{(k)}}{\|\boldsymbol{d}^{(k)}\|^2} \boldsymbol{d}^{(k)}. \tag{31}$$

Following Sherali and Ulular (1989), Sherali and Lim (2007), we choose the parameter $\xi^{(k)}$ as $\xi^{(k)} = \|\boldsymbol{g}^{(k)}\| / \|\boldsymbol{d}^{(k-1)}\|$. The step size parameter $\gamma^{(k)}$ is controlled in a similar way to our previous algorithm. More specifically,

- It is initialized by $\gamma^{(0)} = \gamma^{\mathrm{ini}}$.
- It is decreased by $\gamma^{(k)} = \kappa_{\mathrm{S}} \gamma^{(k-1)}$ if the best lower bound is not updated for δ_{S} successive iterations.
- It is increased by $\gamma^{(k)} = \kappa_{\mathrm{E}} \gamma^{(k-1)}$ if the best lower bound is updated, i.e., $\mathrm{LB}^{(k)} > \max_{i \le k-1} \mathrm{LB}^{(i)}$.

The algorithm is terminated if the best lower bound does not increase by $100\varepsilon/(1-\varepsilon)\%$ and the gap between the best lower and upper bounds does not decrease by $100\varepsilon\%$ in δ_{T} successive iterations.

7.2 Network reduction by constraint propagation

To reduce the network more, the constraint propagation technique is utilized as in Sourd (2009) for ($\widehat{\mathrm{LR}}_2$) and ($\widehat{\mathrm{LR}}_2^m$). Constraint propagation for scheduling problems has been well studied (e.g. Baptiste et al. (2001)) especially in the context of shop scheduling problem from the pioneering works by Carlier and Pinson (1989, 1990). This technique enables us to restrict job time windows within which jobs can be processed by checking their consistency, and it has been utilized for the reduction of the search space. In our algorithm, the time window $[\underline{r}_i, \overline{d}_i]$ of each job i is computed by

$$\underline{r}_i = \min_{v_{it} \in V} t - p_i, \quad \overline{d}_i = \max_{v_{it} \in V} t \tag{32}$$

for ($\widehat{\mathrm{LR}}_2$) (for ($\widehat{\mathrm{LR}}_2^m$), V is replaced by V^m in the above equation). Then, the constraint propagation technique is applied, and if the time window reduces to $[\underline{r}_i', \overline{d}_i']$, the nodes outside that, i.e.

$$\{v_{it} \mid \underline{r}_i + p_i \le t < \underline{r}_i' + p_i\} \cup \{v_{it} \mid \overline{d}_i' < t \le \overline{d}_i\} \tag{33}$$

are eliminated from V.

Among several consistency tests proposed so far, three types are utilized in our algorithm; immediate selection (Carlier and Pinson 1989, Brucker et al. 1994), edge-finding (Carlier and Pinson 1990, Carlier and Pinson 1994, Baptiste et al. 2001), and not-first/not-last (Carlier and Pinson 1989, Baptiste et al. 2001). In addition to these, we apply the following simple test that directly eliminates nodes.

Direct elimination

It is clear that the completion time C_i of job i satisfies $C_i \le \overline{d}_j - p_j$ if job i precedes job j, and $C_i \ge \underline{r}_j + p_i + p_j$ if job i is preceded by job j. Therefore, job i cannot be completed in the interval $[\overline{d}_j - p_j + 1, \underline{r}_j + p_i + p_j - 1]$ if $\overline{d}_j - p_j + 1 \le \underline{r}_j + p_i + p_j - 1$. In this case, the corresponding nodes are eliminated.

7.3 Network reduction by node compression

To reduce memory usage for storing the network structure, successive idle jobs are compressed into one long idle job. Moreover, successive nodes are compressed into one supernode. These compressions are performed for the nodes of

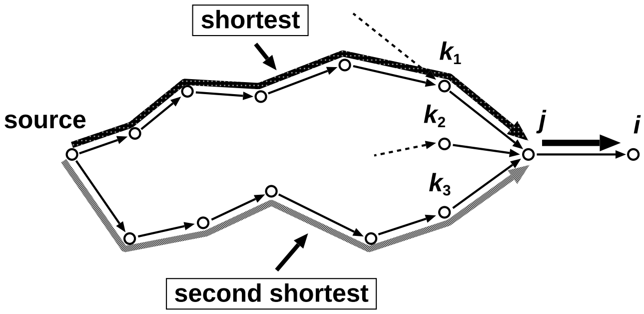


Fig. 1 Shortest and second shortest constrained paths stored in dynamic programming (nodes are denoted by their corresponding jobs)

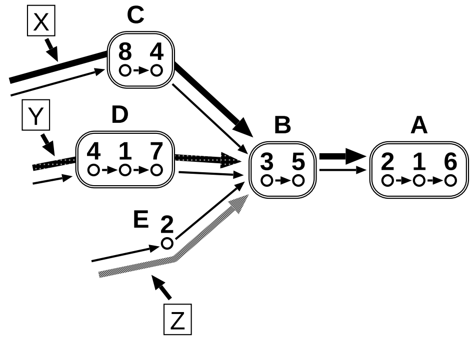


Fig. 2 An example of the network when nodes are compressed

$\widehat{G}^m_{\mathrm{S}}$ whose in-degrees are one. Up to $\max_{i\in\mathscr{N}} p_i$ unit idle jobs are compressed int one long idle job and up to three successive nodes are compressed into one super-node.

These compressions also contribute to forbidding job duplication. For example, job 1 $\rightarrow$ job 2 $\rightarrow$ job 3 $\rightarrow$ job 4 $\rightarrow$ job 5 $\rightarrow$ job 1 is feasible in the original expression of the network, while it becomes infeasible when (job 1, job 2, job 3) and (job 4, job 5, job 1) are compressed into two super-nodes, respectively, because we can remove the arc connecting these super-nodes by checking whether they have the same job (in this case, job 1).

Since job duplication in three successive (ordinary) nodes is forbidden in $(\widehat{\mathrm{LR}}^m_2)$, it is natural to expect that it can also be done for super-nodes. Unfortunately, this is not true because of the dynamic programming algorithm for $(\widehat{\mathrm{LR}}^m_2)$. The forward (backward) dynamic programming for $(\widehat{\mathrm{LR}}^m_2)$ recursively computes the shortest path from $v^{\mathbf{0}_m}_{n+1,T_{\mathrm{S}}}$ to $v^{\boldsymbol{b}}_{it}$ (from $v^{\boldsymbol{b}-\boldsymbol{q}^m_j}_{k,t-p_j}$ to $v^{\mathbf{1}_m}_{n+1,T_{\mathrm{E}}+1}$) through $(v^{\boldsymbol{b}-\boldsymbol{q}^m_j}_{k,t-p_j}, v^{\boldsymbol{b}}_{jt})$ that satisfies the constraints on three successive nodes. To achieve this, the shortest and the second shortest constrained paths to (from) each node are stored (Abdul-Razaq and Potts 1988, Péridy et al. 2003, Tanaka et al. 2009). For example, let us consider that the forward dynamic programming is applied to $\widehat{G}^m_{\mathrm{S}}$ in Fig. 1, where nodes are denoted by the corresponding jobs to simplify the notation. Let us also assume that the shortest and the second shortest constrained paths from the source node are already obtained for j in the figure. Then, the shortest constrained path from the source node to $i \in \mathscr{N}$ through (j,i) is given by

- (the shortest constrained path to j) + (j,i) if $k_1 \neq i$,
- (the second shortest constrained path to j) + (j,i) if $k_1 = i$.

Therefore, the shortest constrained path from the source node to i through (j,i) can be computed in $O(1)$ time. Since at most n nodes are connected to i, time complexity of the shortest and the second shortest constrained paths from the source node to i is $O(n)$. The overall time complexity $O(n^2 2^m (T_{\mathrm{E}} - T_{\mathrm{S}}))$ is obtainable by multiplying it by $O(n2^m(T_{\mathrm{E}} - T_{\mathrm{S}}))$, the number of nodes.

Next, let us assume that nodes are compressed as in Fig. 2, where A, B, C and D denote super-nodes. If path X is the shortest constrained path, we can safely adopt it to construct the shortest constrained path to super-node A through super-node B, because super-nodes C and A have no common job. A problem arises when paths Y and Z are the shortest and the second shortest, respectively. Since the super-nodes D and A have job 1, path Y, which passes through super-node D, cannot be adopted and path Z should be adopted instead. However, it passes through node E although super-node A has job 2. Therefore, the method explained for Fig. 1 is not applicable and we should check all the paths from the source node to super-node B if we try to forbid job duplication in three successive super-nodes. This, of course, leads to the increase of computational efforts required for solving $(\widehat{\mathrm{LR}}^m_2)$.

To avoid this difficulty, we ignore job duplication in the second shortest constrained path. If paths Y and Z are the shortest and the second shortest in Fig. 2, respectively, we adopt path Z to construct the shortest constrained path to super-node A through super-node B by ignoring the job duplication in node E and super-node A. It follows that $(\widehat{\mathrm{LR}}^m_2)$ is solved only heuristically and job duplication may occur even in three successive ordinary nodes. However, an optimal solution of (P) is obtained even if we solve $(\widehat{\mathrm{LR}}^n_2)$ $(\mathscr{M} = \mathscr{N})$ in this manner because the constraints (14) are not ignored and hence are always satisfied. Therefore, the framework of the proposed exact algorithm remains valid.

7.4 Strict check of dominance of successive jobs

As already described in 4.2, our previous algorithm utilizes dominance of four successive jobs for network reduction. In this reduction, an arc is eliminated if a strictly dominating path is found for every possible path passing through that arc. To make it work more effective, the proposed algorithm eliminate an arc even when only a path yielding the equal cost is found for some path passing through that arc, if the former dominates the latter under an appropriate tie-breaking rule. This tie-breaking rule should be such that

– a path corresponding to an optimal solution of (P) dominates any other paths,
– it is consistent with the tie-breaking rule used in the adjacency constraints explained in 3.2.

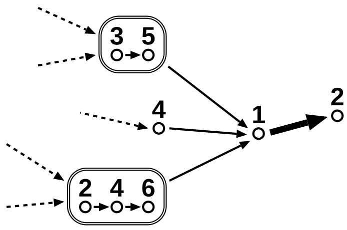
It is shown in Tanaka et al. (2009) that a path corresponding to an optimal solution of (P) is not forbidden by the adjacency constraints under a mild assumption that ties are broken independently of t. To ensure the consistency between the tie-breaking rule in the adjacency constraints and that in the network reduction here, we should put a slightly stronger assumption on them, but it is still mild in practice.

Let us assign a number π_i to every job i ($i \in \mathcal{N}_0$), where $\pi_i \in \mathcal{N}$ ($i \in \mathcal{N}$), $\pi_i \neq \pi_j$ ($i \neq j$) and $\pi_0 = n+1$. Next, a total order on a set of job sequences with the same length is introduced by the lexicographical order of corresponding sequences of π_i, and ties are broken according to this order. When, for example, $(\pi_1, \pi_2, \pi_3, \pi_4) = (2, 4, 1, 3)$, the job sequence $1 \to 2$ dominates $2 \to 1$ and $3 \to 1 \to 4 \to 2$ dominates $1 \to 2 \to 3 \to 4$ under this tie-breaking rule, if their objective values are identical. Clearly, this rule does not eliminate the optimal solution that dominates all the other optimal solutions in the order. How to determine π_i will be explained in Section 9.

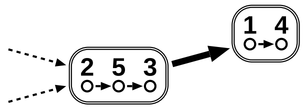
Instead of considering permutations of four successive nodes as in 4.2, the variable number of nodes is considered in the proposed algorithm to eliminate an arc in the above manner. More specifically, three successive (super-)nodes are considered if the target arc connects two ordinary nodes. Otherwise, only the two (super-)nodes connected by the target arc is considered. It follows that permutations of at least three and up to six jobs are to be considered depending on the situation. For example, to eliminate the arc $(1, 2)$ in Fig. 3(a) existence of dominating permutations is checked for $3 \to 5 \to 1 \to 2$ and $4 \to 1 \to 2$ ($2 \to 4 \to 6 \to 1 \to 2$ is ignored because job 2 is duplicated and hence it is infeasible). On the other hand, to eliminate the arc $(\{2 \to 5 \to 3\}, \{1 \to 4\})$ in Fig. 3(b), a dominating permutation is checked only for $2 \to 5 \to 3 \to 1 \to 4$.

7.5 Introduction of a tentative upper bound

Due to the existence of machine idle time, it is not easy to obtain a tight upper bound for our problem compared to the problem without machine idle time even by the algorithm stated in Section 6. Because the efficiency of the network reduction in 4.1 depends much on the tightness of an upper bound, not only dynamic programming requires considerable computational efforts, but also memory space is sometimes exhausted when solving $(\widehat{\mathrm{LR}}_2^m)$ in the third stage, if a tight upper bound is unavailable. To reduce the dependence on the tightness of an upper bound, a tentative upper bound $\mathrm{UB}^{\mathrm{tent}}$ instead of the current upper bound UB is utilized for



(a) elimination of the arc $(1, 2)$



(b) elimination of the arc $(\{2 \to 5 \to 3\}, \{1 \to 4\})$

Fig. 3 Network reduction by dominance of successive jobs in the forward dynamic programming

the network reduction in 4.1 for $(\widehat{\mathrm{LR}}_2^m)$, where $\mathrm{UB}^{\mathrm{tent}}$ is chosen as $\mathrm{UB}^{\mathrm{tent}} \leq \mathrm{UB}$. If $\mathrm{UB}^{\mathrm{tent}}$ is greater than the optimal objective value OPT, it is ensured that the algorithm can find an optimal solution because $\mathrm{UB}^{\mathrm{tent}}$ is a valid upper bound. In this case $\mathrm{UB}^{\mathrm{tent}} = \mathrm{UB} = \mathrm{OPT}$ holds when the algorithm is terminated because UB and $\mathrm{UB}^{\mathrm{tent}}$ are updated if a better upper bound is found in the course of the algorithm. On the other hand, if $\mathrm{UB}^{\mathrm{tent}}$ is less than or equal to OPT, the optimality of the solution yielding UB is not ensured even if the algorithm is terminated. However, we can at least say that OPT is not less than $\mathrm{UB}^{\mathrm{tent}}$. Therefore, the solution yielding UB is optimal also in this case if $\mathrm{UB}^{\mathrm{tent}} = \mathrm{UB}$ holds when the algorithm is terminated. Only when $\mathrm{UB}^{\mathrm{tent}} < \mathrm{UB}$, we should increase $\mathrm{UB}^{\mathrm{tent}}$ and solve $(\widehat{\mathrm{LR}}_2^m)$ again from the start ($m = |\mathcal{M}| = 0$).

Here, $\mathrm{UB}^{\mathrm{tent}}$ is chosen as follows.

– If more than 1/64 of the total memory is occupied at the beginning of the third stage, calculate Δ by $\Delta = (\mathrm{UB} - \mathrm{LB})/4$, where LB denotes the current best lower bound. Then, $\mathrm{UB}^{\mathrm{tent}}$ is increased from $(\mathrm{UB} + \mathrm{LB})/2$ by Δ at one iteration. Hence, the maximal number of iterations is three. $\mathrm{UB}^{\mathrm{tent}}$ is rounded to the nearest integer, if necessary.
– Otherwise, let $\mathrm{UB}^{\mathrm{tent}} = \mathrm{UB}$.

8 Proposed Algorithm

This section summarizes the proposed exact algorithm.

8.1 Stage 1

The initial upper bound UB is computed by the algorithm in 6.3. Then, the conjugate subgradient algorithm in 7.1 is applied to the following Lagrangian dual corresponding to (LR_1):

$$\max_{\boldsymbol{\mu}} \left(\min_{\mathscr{P} \in \mathscr{Q}_1} L_{\mathrm{R}}(\mathscr{P};\, \boldsymbol{\mu}) + \sum_{i \in \mathscr{N}} \mu_i \right), \tag{34}$$

where the multipliers are initialized by $\boldsymbol{\mu} = \mathbf{0}_n$. After the conjugate subgradient algorithm is terminated, (LR_1) for the obtained best multipliers $\boldsymbol{\mu}^{\mathrm{stage1}}$ is solved in both forward and backward directions and the network reduction in 4.1 is performed. The algorithm is terminated without entering the next stage if the gap between the best lower bound and UB becomes less than one.

8.2 Stage 2

The conjugate subgradient algorithm in 7.1 is applied to the Lagrangian dual corresponding to ($\widehat{\mathrm{LR}}_2$), where the multipliers are initialized by $\boldsymbol{\mu} = \boldsymbol{\mu}^{\mathrm{stage1}}$. An upper bound is computed by a combination of the Lagrangian heuristics in 6.1 and the extended dynasearch every 50 iterations, and UB is updated if necessary. The backward dynamic programming and the network reductions in 4.1 and 7.2 are applied every time when the best lower bound or UB is updated. The multipliers obtained in this stage are denoted by $\boldsymbol{\mu}^{\mathrm{stage2}}$. The algorithm is terminated without entering the next stage if the gap between the best lower bound and UB becomes less than one. Otherwise, the current best solution yielding UB is further improved by the extended enhanced dynasearch.

8.3 Stage 3

Solve ($\widehat{\mathrm{LR}}_2$) for $\boldsymbol{\mu}^{\mathrm{stage2}}$ with the network reduction in 7.4 applied. Let the current best lower bound LB be

$$\mathrm{LB} = \min_{\mathscr{P} \in \widehat{\mathscr{Q}}_2} L_{\mathrm{R}}(\mathscr{P};\, \boldsymbol{\mu}^{\mathrm{stage2}}) + \sum_{i \in \mathscr{N}} \mu_i^{\mathrm{stage2}}. \tag{35}$$

Then, the subprocedure is repeated by increasing the tentative upper bound $\mathrm{UB}^{\mathrm{tent}}$ as explained in 7.5. It is terminated if $\mathrm{UB}^{\mathrm{tent}} = \mathrm{UB}$ at the end of the subprocedure.

Subprocedure

Let $\mathrm{LB}^{\mathrm{sub}} = \mathrm{LB}$ and $m = |\mathscr{M}| = 0$. With starting from $\widehat{G}_{\mathrm{S}}^0 = \widehat{G}_{\mathrm{S}}$, the relaxation ($\widehat{\mathrm{LR}}_2^m$) for $\boldsymbol{\mu}^{\mathrm{stage2}}$ is solved with $\mathscr{M}$ increased. To solve ($\widehat{\mathrm{LR}}_2^m$), the forward or backward dynamic programming is applied in turns, where the network reductions in 7.2, 7.3 and 7.4 are performed. The network reduction in 4.1 is also performed by $\mathrm{UB}^{\mathrm{tent}}$ instead of UB. Update $\mathrm{LB}^{\mathrm{sub}}$ by

$$\mathrm{LB}^{\mathrm{sub}} = \min_{\mathscr{P} \in \widehat{\mathscr{Q}}_2^m} L_{\mathrm{R}}(\mathscr{P};\, \boldsymbol{\mu}^{\mathrm{stage2}}) + \sum_{i \in \mathscr{N}} \mu_i^{\mathrm{stage2}}. \tag{36}$$

An upper bound is computed by a combination of the Lagrangian heuristics in 6.1 and the extended dynasearch if $\mathrm{LB}^{\mathrm{sub}}$ is improved from its previous value. $\mathrm{UB}^{\mathrm{tent}}$ and UB are updated if necessary. This subprocedure is terminated if the gap between $\mathrm{LB}^{\mathrm{sub}}$ and $\mathrm{UB}^{\mathrm{tent}}$ becomes less than one.

The choice of $\mathscr{M}$ in the subprocedure of Stage 3 follows Tanaka et al. (2009). Although two methods are switched in our previous algorithm, we apply only one of them for simplicity. The job whose corresponding nodes appear less frequently in $\widehat{G}_{\mathrm{S}}^m$ is chosen first from $\mathscr{N} \setminus \mathscr{M}$, and up to three are added to $\mathscr{M}$ at one iteration in the subprocedure, depending on the current memory usage.

9 Numerical results

The proposed algorithm is applied to benchmark instances of the single-machine total weighted earliness-tardiness problem ($1||\sum(\alpha_i E_i + \beta_i T_i)$ and $1|r_i|\sum(\alpha_i E_i + \beta_i T_i)$), the single-machine total weighted completion time problem with distinct release dates ($1|r_i|\sum w_i C_i$), and the single-machine total weighted tardiness problem with distinct release dates ($1|r_i|\sum w_i T_i$). The algorithm is coded in C (gcc) and we run it on a 3.4GHz Pentium4 desktop computer with 1GB RAM. The maximum memory size for storing the network structure (dynamic programming states) is restricted to 384MB. As the parameters $(\gamma^{\mathrm{ini}}, \delta_{\mathrm{T}}, \delta_{\mathrm{S}}, \varepsilon, \kappa_{\mathrm{S}}, \kappa_{\mathrm{E}})$ in the conjugate subgradient algorithm, we choose $(1.2, n, 2, 0.02, 0.95, 1.1)$ in stage 1 and $(1.0, n, 2, 0.002, 0.95, 1.18)$ in stage 2 by preliminary experiments. As the job sequence π_i for the tie-breaking in 7.4 the EDD sequence is used for $1||\sum(\alpha_i E_i + \beta_i T_i)$, $1|r_i|\sum(\alpha_i E_i + \beta_i T_i)$ and $1|r_i|\sum w_i T_i$ as in Tanaka et al. (2009). For $1|r_i|\sum w_i C_i$, the problem without duedates, the solution yielding the initial upper bound is used.

9.1 Total weighted earliness-tardiness problem ($1||\sum(\alpha_i E_i + \beta_i T_i)$ and $1|r_i|\sum(\alpha_i E_i + \beta_i T_i)$)

In this problem each job i is given a release date r_i, a duedate d_i, an earliness weight α_i and a tardiness weight β_i. All these values are assumed to be integral and the cost function $f_i(t)$ is expressed by

$$f_i(t) = \max\{\alpha_i(d_i - t), \beta_i(t - d_i)\}. \tag{37}$$

The end of the scheduling horizon T_{E} is chosen simply as

$$T_{\mathrm{E}} = \max\left\{ \max_{i \in \mathscr{N}} r_i,\ \max_{i \in \mathscr{N}} d_i \right\} + \sum_{i \in \mathscr{N}} p_i. \tag{38}$$

The proposed algorithm is applied to two sets of benchmark instances: the instance set with equal (zero) release dates ($1||\sum(\alpha_i E_i + \beta_i T_i)$) in Sourd and Kedad-Sidhoum (2008)

Table 1 Computational results for the Sourd's benchmark set of $1||\sum(\alpha_i E_i + \beta_i T_i)$

n	optimally solved instances				CPU time (s)	
	Stage1	Stage2	Stage3	total	ave.	max.
20	190	998	86	1274	0.07	0.19
30	46	1078	150	1274	0.29	0.86
40	31	1036	207	1274	0.78	2.51
50	12	913	349	1274	1.71	4.65
60	0	30	15	45	3.66	8.39
90	0	18	27	45	17.46	45.99

Table 2 Computational results for the Bülbül's benchmark set of $1|r_i|\sum(\alpha_i E_i + \beta_i T_i)$

n	optimally solved instances				CPU time (s)	
	Stage1	Stage2	Stage3	total	ave.	max.
20	42	232	26	300	0.02	0.07
40	1	224	75	300	0.25	0.85
60	0	125	175	300	1.23	3.92
80	0	79	221	300	3.68	10.84
100	0	25	275	300	19.57	43.03
130	0	11	289	300	53.59	141.60
170	0	2	298	300	164.43	443.99
200	0	4	296	300	318.43	679.88

and Sourd (2009), and that with distinct release dates ($1|r_i|\sum(\alpha_i E_i + \beta_i T_i)$) in Bülbül et al. (2007). We refer to these as the Sourd's benchmark set and the Bülbül's benchmark set, respectively. Their generation schemes are summarized in A.1 and A.2, respectively.

The results are shown in Tables 1 and 2. We can see that all the instances are optimally solved. For $1||\sum(\alpha_i E_i + \beta_i T_i)$ and $1|r_i|\sum(\alpha_i E_i + \beta_i T_i)$, various exact algorithms have been proposed so far by several researchers (Yano and Kim 1991, Davis and Kanet 1993, Kim and Yano 1994, Fry et al. 1996, Hoogeveen and van de Velde 1996, Chang 1999, Sourd and Kedad-Sidhoum 2003, Sourd and Kedad-Sidhoum 2008, Yau et al. 2008, Sourd 2009, Detienne et al. 2010). To the best of our knowledge, the most efficient algorithm for the problem with general earliness and tardiness weights is the branch-and-bound algorithm proposed by Sourd (2009). He reported that his algorithm succeeded in solving all the 50 jobs instances in the Sourd's benchmark set within 1,000 seconds, and all the 60 jobs instances in the Bülbül's benchmark set within 500 seconds on a 3.2GHz Pentium4 desktop computer. On the other hand, our algorithm only takes at most 5 and 4 seconds for these instances, respectively. It is much faster even if the difference of the processors is taken into account.

9.2 Total weighted completion time problem with distinct release dates ($1|r_i|\sum w_i C_i$)

In this problem each job i is given an integer release date r_i and an integer weight w_i. The cost function $f_i(t)$ is expressed

Table 3 Computational results for the Pan's benchmark set of $1|r_i|\sum w_i C_i$

(a) Results with 384MB memory space

n	optimally solved instances				CPU time (s)	
	Stage1	Stage2	Stage3	total	ave.	max.
20	29	68	3	100	0.04	0.12
30	14	71	15	100	0.21	0.43
40	9	81	10	100	0.53	1.38
50	6	75	19	100	1.15	2.42
60	3	67	30	100	2.45	10.71
70	1	60	39	100	4.55	16.37
80	4	49	47	100	7.49	23.35
90	2	47	51	100	12.18	40.22
100	1	48	51	100	17.89	49.96
110	0	46	54	100	25.19	77.98
120	0	46	54	100	34.89	102.94
130	0	41	59	100	50.26	371.45
140	0	44	56	100	60.48	424.35
150	0	46	54	100	86.99	743.05
160	0	39	61	100	109.92	504.30
170	0	41	59	100	134.32	529.98
180	0	40	60	100	152.52	802.49
190	0	38	61	99	*196.67*	*1119.87*
200	0	39	60	99	*251.12*	*1376.49*

(b) Result with 768MB memory space

n	optimally solved instances				CPU time (s)	
	Stage1	Stage2	Stage3	total	ave.	max.
190	0	0	1	1	1976.11	1976.11
200	0	0	1	1	2883.25	2883.25

by

$$f_i(t) = w_i C_i, \tag{39}$$

and T_{E} is chosen as

$$T_{\mathrm{E}} = \max_{i \in \mathcal{N}} r_i + \sum_{i \in \mathcal{N}} p_i. \tag{40}$$

This problem can be treated as a special class of $1|r_i|\sum(\alpha_i E_i + \beta_i T_i)$ where $d_i = 0$, $\alpha_i = 0$, $\beta_i = w_i$ ($i \in \mathcal{N}$).

The proposed algorithm is applied to the set of benchmark instances in Pan and Shi (2008), which is referred to as the Pan's benchmark set. The generation scheme is given in A.3. The results are summarized in Table 3. The algorithm failed to solve one instance with 190 jobs and one instance with 200 jobs due to memory shortage. Therefore, the average and maximum CPU times in Table 3(a) are shown over optimally solved instances. The unsolved instances can be solved optimally when we increase the limit of memory space from 384MB to 768MB. These results are shown in Table 3(b).

Several types of exact algorithms for this type of problem have been proposed so far (Bianco and Ricciardelli 1982, Hariri and Potts 1983, Belouadah et al. 1992, Gélinas and Soumis 1997, Jouglet et al. 2004, Pan and Shi 2005, Pan and Shi 2008). To be more precise, the problem treated in Gélinas and Soumis (1997) and Pan and Shi (2005) is the total weighted completion time problem with distinct release dates and *deadlines*, but the algorithms can be applied to $1|r_i|\sum w_iC_i$ by choosing the job deadlines as T_{E}. Among these exact algorithms, the most efficient for $1|r_i|\sum w_iC_i$ seems the hybrid dynamic programming/branch-and-bound algorithm proposed by Pan and Shi (2008). Indeed, they claim that their algorithm could solve all the instances in the Pan's benchmark set. However, we found that their optimal objective values or lower bounds (they offer only lower bounds for some instances) are incorrect for 22 instances. Anyway, our algorithm is much faster than their algorithm for larger instances because their algorithm took nearly four days for the hardest instance on a 2.8GHz Pentium4 computer, while our algorithm can solve all the instances within 1 hour on a 3.4GHz Pentium4 computer.

9.3 Total weighted tardiness problem with distinct release dates ($1|r_i|\sum w_iT_i$)

In this problem each job i is given an integer release date r_i, an integer duedate d_i and an integer tardiness weight w_i. The cost function $f_i(t)$ is expressed by

$$f_i(t) = w_i \max\{t - d_i, 0\}, \tag{41}$$

and T_{E} is chosen as (38). This problem can be treated as a special class of $1|r_i|\sum(\alpha_iE_i + \beta_iT_i)$ where $\alpha_i = 0$, $\beta_i = w_i$ ($i \in \mathcal{N}$).

The generation scheme of the benchmark set is summarized in A.4. It is almost the same with those in the previous researches on exact algorithms for this problem (Akturk and Ozdemir 2000, Jouglet et al. 2004) except that the maximum processing time is not 10 but 100. The results are summarized in Table 4. The algorithm failed to solve two instances with 90 jobs even with 768MB memory space. Nevertheless, it outperforms the best exact algorithm by Jouglet et al. (2004) that can solve instances with up to 30 jobs. It is worth noting that our algorithm is less efficient for $1|r_i|\sum w_iT_i$ than for $1||\sum(\alpha_iE_i + \beta_iT_i)$, $1|r_i|\sum(\alpha_iE_i + \beta_iT_i)$ and $1|r_i|\sum w_iC_i$. The reason will be that completion cost of an on-time job is zero. Because of this, there exist many optimal and near-optimal solutions in $1|r_i|\sum w_iT_i$. As seen in 3.3, the number of dynamic programming states increases exponentially as the relaxed constraints are recovered. To suppress it, the state elimination techniques should work effectively. However, they do not when there exist many (near-)optimal solutions. This makes our algorithm terminate due to shortage of memory space. Introduction of problem specific dominance properties might be some help, but it is beyond this study because our primary objective here is to construct an exact algorithm for the *general* single-machine scheduling problem.

It should be also noted that our framework can solve instances with 300 jobs of the single-machine total weighted tardiness problem ($1||\sum w_iT_i$). This fact implies that whether release dates are equal (zero) or distinct affects the problem solvability much at least for our algorithm. It would be necessary to investigate the impact of distinct release dates on the problem structure.

Table 4 Computational results for $1|r_i|\sum w_iT_i$

(a) Results with 384MB memory space

n	optimally solved instances				CPU time (s)	
	Stage1	Stage2	Stage3	total	ave.	max.
10	67	53	0	120	0.00	0.01
20	41	76	3	120	0.04	0.11
30	32	84	4	120	0.18	0.59
40	29	80	11	120	0.52	3.17
50	26	75	19	120	1.20	8.15
60	31	71	18	120	2.20	12.68
70	30	60	30	120	4.29	34.07
80	27	67	24	118	*6.95*	*134.20*
90	29	60	28	117	*10.10*	*104.08*
100	30	56	29	115	*15.26*	*187.32*

(b) Results with 768MB memory space

n	optimally solved instances				CPU time (s)	
	Stage1	Stage2	Stage3	total	ave.	max.
80	0	0	2	2	266.51	273.11
90	0	0	1	1	*316.63*	*316.63*
100	0	0	2	2	*424.57*	*489.73*

9.4 Comparison with the direct extension

To examine the effectiveness of the improvements in Section 7, the algorithm without the improvements is applied to the Bülbül's benchmark set and Pan's benchmark set. The results are shown in Tables 5 and 6. Although the algorithm without the improvements is still faster than the existing algorithms, the improvements make the algorithm about twice faster for larger instances. The number of instances that the algorithm fails to solve optimally decreases much owing to the reduction of memory usage. Moreover, the algorithm without the improvements failed to solve some instances in the Pan's benchmark set optimally even with 768MB memory space.

Detailed comparison between the algorithms with and without the improvements for the Bülbül's benchmark set are shown in Table 7. From this table, we can verify that the proposed improvements reduce computational time in all stages. In Stages 1 and 2, it is achieved by applying the conjugate subgradient algorithm in 7.1 instead of the ordinary subgradient algorithm. On the other hand, in Stage 3, the constraint propagation in 7.2 and the tentative upper bound in 7.5 reduce computational time. Node compression in 7.3 and strict check of dominance in 7.4 do not affect the computational time in Stage 3 much, but they are effective for the reduction of memory usage.

Table 5 Computational results for the Bülbül's benchmark set of $1|r_i|\sum(\alpha_i E_i + \beta_i T_i)$ (without the improvements)

n	optimally solved instances				CPU time (s)	
	Stage1	Stage2	Stage3	total	ave.	max.
20	37	221	42	300	0.03	0.09
40	1	153	146	300	0.32	1.09
60	0	64	236	300	1.89	6.88
80	0	38	262	300	6.53	18.73
100	0	10	290	300	36.58	92.34
130	0	2	298	300	104.76	307.48
170	0	1	299	300	305.86	1085.64
200	0	0	300	300	608.74	1618.53

Table 6 Computational results for the Pan's benchmark set of $1|r_i|\sum w_i C_i$ (without the improvements)

(a) Results with 384MB memory space

n	optimally solved instances				CPU time (s)	
	Stage1	Stage2	Stage3	total	ave.	max.
20	28	64	8	100	0.06	0.15
30	14	68	18	100	0.26	0.62
40	9	61	30	100	0.69	1.83
50	7	59	34	100	1.53	3.60
60	2	56	42	100	3.71	12.01
70	1	48	51	100	7.23	25.03
80	4	41	55	100	12.47	44.07
90	2	39	59	100	21.70	107.54
100	1	38	61	100	31.32	95.48
110	0	36	64	100	46.10	166.27
120	0	39	61	100	66.28	243.77
130	0	36	62	98	*83.97*	*381.91*
140	0	35	64	99	*109.25*	*507.35*
150	0	37	62	99	*162.99*	*1011.72*
160	0	34	66	100	225.57	1272.50
170	0	33	65	98	*255.63*	*1164.78*
180	0	31	67	98	*296.74*	*1265.00*
190	0	29	65	94	*343.88*	*1980.38*
200	0	28	64	92	*396.85*	*1790.85*

(b) Results with 768MB memory space

n	optimally solved instances				CPU time (s)	
	Stage1	Stage2	Stage3	total	ave.	max.
130	0	0	1	1	*539.86*	*539.86*
140	0	0	1	1	643.69	643.69
150	0	0	1	1	1340.49	1340.49
170	0	0	1	1	*1396.38*	*1396.38*
180	0	0	2	2	1484.83	1601.97
190	0	0	3	3	*2340.35*	*3028.77*
200	0	0	6	6	*2873.47*	*4370.77*

10 Conclusion

In this study we proposed a dynamic-programming-based exact algorithm for general single-machine scheduling with machine idle time by extending and improving our previous algorithm for the problem without machine idle time. Computational experiments showed that our algorithm can optimally solve 200 jobs instances of the single-machine total weighted earliness-tardiness problem and the single-machine total weighted completion time problem with distinct release dates, and 80 jobs instances of the single-machine total weighted tardiness problem with distinct release dates. It is much faster than the current best algorithms for these problems, but our algorithm could be further improved by introducing better constraints (cuts), dominance properties, better choices of parameters, and so on. These are left for future research. Another direction of research will be to extend our algorithm so that it is applicable to a wider class of problems such as the problem with precedence constraints and/or setup times, the parallel-machine problem, and so on.

Table 7 Detailed comparison between the algorithms with and without the improvements for the Bülbül's benchmark set of $1|r_i|\sum(\alpha_i E_i + \beta_i T_i)$

(a) without the improvements

n	Stage 1 CPU time (s)		Stage 2 CPU time (s)		Stage 3 CPU time (s)	
	ave	max	ave	max	ave	max
20	0.02	0.06	0.01	0.05	0.00	0.02
40	0.16	0.49	0.15	0.86	0.03	0.10
60	0.64	1.90	1.15	5.29	0.13	0.73
80	1.92	5.11	4.19	13.95	0.48	2.63
100	11.70	29.02	22.48	73.21	2.48	13.18
130	28.83	59.47	65.55	208.86	10.45	66.94
170	67.29	127.92	188.95	644.77	49.79	483.81
200	109.39	227.84	359.38	1082.12	139.97	778.45

(b) with the improvements

n	Stage 1 CPU time (s)		Stage 2 CPU time (s)		Stage 3 CPU time (s)	
	ave	max	ave	max	ave	max
20	0.01	0.04	0.00	0.03	0.00	0.01
40	0.16	0.37	0.08	0.52	0.02	0.07
60	0.56	1.41	0.59	2.58	0.13	0.67
80	1.44	3.49	1.91	7.43	0.45	1.88
100	7.12	14.62	10.79	29.91	1.82	7.82
130	15.90	36.06	30.75	84.36	7.21	49.85
170	34.36	70.28	95.06	350.07	35.24	139.53
200	54.29	133.66	179.26	468.58	86.03	333.40

Acknowledgements This work is partially supported by Grant-in-Aid for Young Scientists (B) 19760273, from the Ministry of Education, Culture, Sports, Science and Technology (MEXT) Japan.

References

Abdul-Razaq, T. S., & Potts, C. N. (1988). Dynamic programming state-space relaxation for single-machine scheduling. *Journal of the Operational Research Society, 39*, 141–152.

Akturk, M. S., & Ozdemir, D. (2000). An exact approach to minimizing total weighted tardiness with release dates, *IIE Transactions, 32*, 1091–1101.

Baptiste, P., Le Pape, C., & Nuijten, W. (2001). *Constraint-based scheduling: Applying constraint programming to scheduling problems*. Kluwer Academic Publishers.

Belouadah, H., Posner, M. E., & Potts, C. N. (1992). Scheduling with release dates on a single machine to minimize total weighted completion time. *Discrete Applied Mathematics, 36*, 213–231.

Bianco, L, & Ricciardelli, S. (1982). Scheduling of a single machine to minimize total weighted completion time subject to release dates. *Naval Research Logistics Quarterly, 29*, 151–167.

Brucker, P., Jurisch, B., & Krarner, A. (1994). The job-shop problem and immediate selection. *Annals of Operations Research, 50*, 73–114.

Bülbül, K., Kaminsky, P., & Yano, C. (2007). Preemption in single machine earliness/tardiness scheduling. *Journal of Scheduling, 10*, 271–292.

Carlier, J., & Pinson, E.. (1989). An algorithm for solving the job-shop problem. *Management Science, 35*, 164–176.

Carlier, J., & Pinson, E. (1990). A practical use of Jackson's preemptive scheduling for solving the job-shop problem. *Annals of Operations Research, 26*, 268–287.

Carlier, J., & Pinson, E. (1994). Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research, 78*, 146–161.

Chang, P. C. (1999). A branch and bound approach for single machine scheduling with earliness and tardiness penalties. *Computers and Mathematics with Applications, 37*, 133–144.

Christofides, N., Mingozzi, A., & Toth, P. (1981). State-space relaxation procedures for the computation of bounds to routing problems. *Networks, 11*, 145–164.

Congram, R. K., Potts, C. N., & van de Velde, S. L. (2002). An iterated dynasearch algorithm for the single machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing, 14*, 52–67.

Davis, J. S., & Kanet, J. J. (1993). Single-machine scheduling with early and tardy completion costs. *Naval Research Logistics, 40*, 85–101.

Detienne, B., Pinson, É., & Rivreau, D. (2010) Lagrangian domain reductions for the single machine earliness-tardiness problem with release dates, *European Journal of Operational Research, 201*, 45–54.

Dyer, M. E., & Wolsey, L.A. (1990). Formulating the single-machine sequencing problem with release dates as a mixed integer problem. *Discrete Applied Mathematics, 26*, 255–270.

Fisher, M. L. (1985). An applications oriented guide to Lagrangian relaxation. *Interfaces, 15*, 10–21.

Fry, T. D., Armstrong, R. D., Darby-Dowman, K., & Philipoom, P. R. (1996). A branch and bound procedure to minimize mean absolute lateness on a single processor. *Computers & Operations Research, 23*, 171–182.

Gélinas, S., & Soumis, F. (1997). A dynamic programming algorithm for single machine scheduling with ready times. *Annals of Operations Research, 69*, 135–156.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Rinnooy Kan, A. H. G. (1979) Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics, 5*, 287–326.

Grosso, A., Della Croce, F., & Tadei, R. (2004). An enhanced dynasearch neighborhood for the single machine total weighted tardiness scheduling problem. *Operations Research Letters, 32*, 68–72.

Hariri, A. M. A., & Potts, C. N. (1983). An algorithm for single machine sequencing with release dates to minimize total weighted completion time. *Discrete Applied Mathematics, 5*, 99–109.

Hoogeveen, J. A., & van de Velde, S. L. (1996). A branch-and-bound algorithm for single-machine earliness-tardiness scheduling with idle time. *INFORMS Journal on Computing, 8*, 402–412.

Ibaraki, T. (1987). Enumerative approaches to combinatorial optimization. *Annals of Operations Research, 10 and 11*.

Ibaraki, T., & Nakamura, Y. (1994). A dynamic programming method for single machine scheduling, *European Journal of Operational Research, 76*, 72–82.

Jouglet, A, Baptiste, P., & Carlier, J. (2004). *Branch-and-bound algorithms for total weighted tardiness*. In Leung, J. Y-T. (Ed.), Handbook of Scheduling, Algorithms, Models, and Performance Analysis (Chapter 13). CRC Press.

Kim, Y. -D., & Yano, C. A. (1994). Minimizing mean tardiness and earliness in single-machine scheduling problems with unequal due dates. *Naval Research Logistics, 41*, 913–933.

Pan, Y., & Shi, L. (2005). Dual constrained single machine sequencing to minimize total weighted completion time. *IEEE Transactions on Automation Science and Engineering, 2*, 344-358.

Pan, Y., & Shi, L. (2008). New hybrid optimization algorithms for machine scheduling problems. *IEEE Transactions on Automation Science and Engineering, 5*, 337–348.

Péridy, L., Pinson, É., & Rivreau, D. (2003). Using short-term memory to minimize the weighted number of late jobs on a single machine. *European Journal of Operational Research, 148*, 591–603.

Potts, C. N., & Van Wassenhove, L. N., (1985). A branch and bound algorithm for the total weighted tardiness problem. *Operations Research, 33*, 363–377.

Pritsker, A. A. B., Watters, L. J., & Wolfe, P. M. (1969). Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science, 16*, 93–108.

Sherali, H. D., & Ulular, O. (1989). A primal-dual conjugate subgradient algorithm for specially structured linear and convex programming problems. *Applied Mathematics and Optimization, 20*, 193–221.

Sherali, H. D., & Lim, C. (2007). Enhancing Lagrangian dual optimization for linear programs by obviating nondifferentiability. *INFORMS Journal on Computing, 19*, 3–13.

Sourd, F., Kedad-Sidhoum, S. (2003). The one-machine problem with earliness and tardiness penalties. *Journal of Scheduling, 6*, 533–549.

Sourd, F. (2005). Optimal timing of a sequence of tasks with general completion cost. *European Journal of Operational Research, 165*, 82–96.

Sourd, F. (2006). Dynasearch for the earliness-tardiness scheduling problem with release dates and setup constraints. *Operations Research Letters, 34*, 591–598.

Sourd, F., & Kedad-Sidhoum, S. (2008). A faster branch-and-bound algorithm for the earliness-tardiness scheduling problem. *Journal of Scheduling, 11*, 49–58.

Sourd, F. (2009). New exact algorithms for one-machine earliness-tardiness scheduling. *INFORMS Journal on Computing, 21*, 167–175.

Sousa, J. P., & Wolsey, L. A. (1992). A time indexed formulation of non-preemptive single machine scheduling problems. *Mathematical Programming, 54*, 353–367.

Tanaka, S., Fujikuma, S., & Araki, M. (2009). An exact algorithm for single-machine scheduling without machine idle time. *Journal of Scheduling, 12*, 575–593.

van den Akker, J. M, van Hoesel, C. P. M., & Savelsbergh, M. W. P. (1999). A polyhedral approach to single-machine scheduling problems. *Mathematical Programming, 85*, 541–572.

van den Akker, J. M., Hurkens, C. A. J., & Savelsbergh, M. W. P. (2000). Time-indexed formulations for machine scheduling problems: column generation. *INFORMS Journal on Computing, 12*, 111–124.

Yano, C. A., & Kim, Y.-D. (1991). Algorithms for a class of single-machine weighted tardiness and earliness problems. *European Journal of Operational Research, 52*, 167–178.

Yau, H., Pan, Y., & Shi, L. (2008). New solution approaches to the general single machine earliness-tardiness problem. *IEEE Transactions on Automation Science and Engineering, 5*, 349–360.

A Details on Benchmark Sets

A.1 Sourd's benchmark set

1. Processing times p_i $(1 \leq i \leq n)$ are generated from the uniform distribution $U[10,100)$. Let $P = \sum_{i=1}^{n} p_i$.

2. Duedates d_i are generated from $U[d_{\min}, d_{\max}]$, where

$$d_{\min} = \max(p_i, \lfloor P(\tau - \rho/2) \rfloor), \quad d_{\max} = d_{\min} + \lfloor \rho P \rfloor. \tag{42}$$

3. Both tardiness weights α_i and earliness weights β_i are generated from $U[1,5]$.
4. For each combination of (n, τ, ρ), 26 instances are generated.
5. $n \in \{20,30,40,50\}$, $\tau \in \{0.2,0.3,0.4,0.5,0.6,0.7,0.8\}$, and $\rho \in \{0.2,0.3,0.4,0.5,0.6,0.7,0.8\}$.
6. For $n \in \{60,90\}$, only 5 instances are generated for each combination of (n, τ, ρ), where $\tau \in \{0.2,0.5,0.8\}$ and $\rho \in \{0.2,0.5,0.8\}$.
7. Available from http://www-poleia.lip6.fr/˜sourd/project/et.

A.2 Bülbül's benchmark set

1. Processing times p_i $(1 \le i \le n)$ are generated from $U[1, p_{\max}]$. Let $P = \sum_{i=1}^{n} p_i$.
2. Release dates are generated from $U[0,P]$.
3. Duedates d_i are generated from $U[d_{\min}, d_{\max}]$, where

$$d_{\min} = \max(0, \lceil (1 - \tau - \rho/2)P \rceil), \quad d_{\max} = \lceil (1 - \tau + \rho/2)P \rceil. \tag{43}$$

4. Both earliness weights α_i and tardiness weights β_i are generated from $U[0,100]$.
5. For each combination of (n, $p_{\max}$, τ, ρ), 5 instances are generated.
6. $\tau \in \{0.2,0.4,0.5,0.6,0.8\}$ and $\rho \in \{0.4,0.7,1.0,1.3\}$. For $n \in \{20,40,60,80\}$, $p_{\max} \in \{10,30,50\}$ and for $n \in \{100,130,170,200\}$, $p_{\max} \in \{50,75,100\}$.
7. Available from http://www-poleia.lip6.fr/˜sourd/project/et.

A.3 Pan's benchmark set

1. Processing times p_i $(1 \le i \le n)$ are generated from $U[1,100]$.
2. Release dates r_i are generated from $U[0, \lfloor 50.5 n\tau \rfloor]$.
3. Weights w_i are generated from $U[1,10]$.
4. For each combination of (n, τ), 10 instances are generated.
5. $n \in \{20,30,\ldots,200\}$ and $\tau \in \{0.2,0.4,0.6,0.8,1.0,1.25,1.5,1.75,2.0,3.0\}$.
6. Available from http://pages.cs.wisc.edu/˜yunpeng/test/sm/dwct/instances.htm.

A.4 Total weighted tardiness benchmark set

1. Processing times p_i $(1 \le i \le n)$ are generated from $U[1,100]$. Let $P = \sum_{i=1}^{n} p_i$.
2. Release dates r_i are generated from $U[0, \tau P]$.
3. Duedates d_i are generated from $U[r_i + p_i, r_i + p_i + \rho P]$.
4. Weights w_i are generated from $U[1,10]$.
5. For each combination of (n, τ, ρ), 10 instances are generated.
6. $n \in \{10,20,\ldots,100\}$, $\tau \in \{0.0,0.5,1.0,1.5\}$ and $\rho \in \{0.05,0.25,0.5\}$.