

Chapter 2

An Exact Algorithm for the Single-Machine Earliness–Tardiness Scheduling Problem

Shunji Tanaka

Abstract This paper introduces our exact algorithm for the single-machine total weighted earliness–tardiness scheduling problem, which is based on the Successive Sublimation Dynamic Programming (SSDP) method. This algorithm starts from a Lagrangian relaxation of the original problem and then constraints are successively added to it until the gap between lower and upper bounds becomes zero. The relaxations are solved by dynamic programming, and unnecessary dynamic programming states are eliminated in the course of the algorithm to suppress the increase of states caused by the addition of constraints. This paper explains the methods employed in our algorithm to construct the Lagrangian relaxations, to eliminate states and to compute an upper bound together with some other improvements. Then, numerical results for known benchmark instances are given to show the effectiveness of our algorithm.

2.1 Introduction

This paper introduces our exact algorithm [29] for the single-machine total weighted earliness–tardiness scheduling problem. Let us consider that n jobs (job 1, ..., job n) are to be processed on a single machine that can process at most one job at a time. No preemption is allowed and once the machine starts processing a job, it cannot be interrupted. After the machine finishes processing a job, it can be idle even when there exist unprocessed jobs. Each job i is given a processing time p_i , due date d_i and release date r_i , where $d_i \geq r_i + p_i$. It is also given a earliness weight α_i and a tardiness weight β_i . The earliness E_i and the tardiness T_i are defined by

$$E_i = \max(d_i - C_i, 0), \quad T_i = \max(C_i - d_i, 0), \quad (2.1)$$

Shunji Tanaka

Department of Electrical Engineering, Kyoto University, Kyotodaigaku-Katsura,
Nishikyo-ku, Kyoto 615-8510, Japan, e-mail: tanaka@kuee.kyoto-u.ac.jp

where C_i denotes the completion time of job i . Our objective is to find a schedule that minimizes $\sum_{1 \leq i \leq n} (\alpha_i E_i + \beta_i T_i)$.

This problem, the single-machine total weighted earliness–tardiness problem ($1||\sum(\alpha_i E_i + \beta_i T_i)$ and $1|r_i|\sum(\alpha_i E_i + \beta_i T_i)$ according to the standard classification of scheduling problems [11]) includes the single-machine total weighted tardiness problem ($1||\sum w_i T_i$) as a special class¹ that appears frequently in the literature as a typical strongly NP-hard scheduling problem [17, 18]. Therefore, it is also strongly NP-hard. Nonetheless, many researchers have tackled this problem and constructed exact algorithms [3, 6, 7, 10, 13, 16, 22, 25, 26, 33, 34] because of its importance in JIT scheduling. Almost all the existing algorithms are branch-and-bound algorithms. To the best of the author’s knowledge, the best algorithm so far is that by Sourd [26] and it can solve instances with up to 50 or 60 jobs within 1,000 s.

On the other hand, our exact algorithm proposed in [29] is based on the Successive Sublimation Dynamic Programming (SSDP) method [14]. The SSDP method is a dynamic-programming-based exact algorithm that starts from a relaxation of the original problem and then repeats the following procedure until the gap between lower and upper bounds becomes zero:

1. The relaxation is solved by dynamic programming and a lower bound is computed.
2. Unnecessary states are eliminated.
3. A relaxation with more detailed information of the original problem, which is referred to as sublimation, is constructed.

In [15], the SSDP method was applied to $1||\sum(\alpha_i E_i + \beta_i T_i)$ without idle time by utilizing the relaxations proposed in [1] as sublimations. Their algorithm could solve instances with up to 35 jobs and hence was better than the branch-and-bound algorithm in [1] that could solve only those with up to 25 jobs. However, they concluded that it is not easy to apply the algorithm to larger instances because of its heavy memory usage for storing dynamic programming states.

Things have changed much from those days and more powerful computers with several gigabytes memory are now readily accessible. It motivated us to improve and extend their algorithm for the total weighted earliness–tardiness problem. First, we showed in [28] that their algorithm with several significant improvements can solve instances with up to 300 jobs of $1||\sum w_i T_i$ and $1||\sum(\alpha_i E_i + \beta_i T_i)$ without idle time. Next, in [29], we extended the algorithm to the problem with idle time and showed that it can solve benchmark instances of $1||\sum(\alpha_i E_i + \beta_i T_i)$ and $1|r_i|\sum(\alpha_i E_i + \beta_i T_i)$ with even 200 jobs. Moreover, the framework is not restricted to these problems, but is applicable to a wide class of single-machine scheduling problems.

The purpose of this paper is to introduce this algorithm, but only its essential part will be given because of limited space. For details, please refer to our papers [28, 29]. Up-to-date information will be available on the web page:

<http://turbine.kuee.kyoto-u.ac.jp/~tanaka/SiPS/>.

¹ When α_i are chosen as zero, the total weighted earliness–tardiness problem is reduced to the total weighted tardiness problem.

This paper is organized as follows. In Sect. 2.2, our problem will be given as so-called a time-indexed formulation and then its network representation will be introduced. In Sect. 2.3, Lagrangian relaxations of the problem, which are utilized as the relaxation and subproblems in our algorithm (3. in the SSDP method), will be expressed in terms of the network representation. Dynamic programming recursions to solve them (1. in the SSDP method) will also be explained. Next, Sect. 2.4 will state how to reduce the networks corresponding to the Lagrangian relaxations (2. in the SSDP method). Then, Sect. 2.5 will summarize our algorithm. Section 2.6 will describe how to obtain a tight upper bound from a solution of the Lagrangian relaxation in this algorithm. Section 2.7 will present results of numerical experiments and show the effectiveness of our algorithm. Finally, in Sect. 2.8, the contents of this paper and future research directions will be summarized.

2.2 Problem Formulation

This section will give a time-indexed formulation [8, 21, 27, 31] of our problem and its network representation. It is well-known that a tight lower bound can be obtained by a linear programming (LP) relaxation of a time-indexed formulation. However, there is a major drawback that the number of decision variables is very large. In our algorithm, an alternative way to apply the Lagrangian relaxation technique is taken to obtain a lower bound, which will be explained in the next section by the network representation.

2.2.1 Time-Indexed Formulation

Let us assume that the processing times p_i , due dates d_i and release dates r_i are all integral. Then, we can assume without loss of optimality that the job completion times C_i are also integral. Thus, we introduce binary decision variables x_{it} ($1 \leq i \leq n$, $1 \leq t \leq T_E$) such that

$$x_{it} = \begin{cases} 1 & \text{if } C_i = t, \\ 0 & \text{otherwise,} \end{cases} \quad (2.2)$$

where T_E is a scheduling horizon and can be chosen, e.g., as

$$T_E = \max_{1 \leq i \leq n} d_i + \sum_{1 \leq i \leq n} p_i. \quad (2.3)$$

We also introduce a cost function $f_i(t)$ ($1 \leq i \leq n$, $1 \leq t \leq T_E$) defined by

$$f_i(t) = \begin{cases} \infty & t < r_i + p_i, \\ \max(\alpha_i(d_i - t), \beta_i(t - d_i)) & t \geq r_i + p_i. \end{cases} \quad (2.4)$$

Then, the problem can be formulated as follows:

$$(P) : \min_{\mathbf{x}} \sum_{\substack{1 \leq i \leq n \\ 1 \leq t \leq T_E}} f_i(t) x_{it}, \quad (2.5)$$

$$\text{s.t.} \quad \sum_{\substack{1 \leq i \leq n \\ t \leq s \leq \min(t+p_i-1, T_E)}} x_{is} \leq 1, \quad 1 \leq t \leq T_E, \quad (2.6)$$

$$\sum_{1 \leq t \leq T_E} x_{it} = 1, \quad 1 \leq i \leq n, \quad (2.7)$$

$$x_{it} \in \{0, 1\}, \quad 1 \leq i \leq n, 1 \leq t \leq T_E. \quad (2.8)$$

In (P), the constraints (2.6) require that at most one job can be processed in the interval $[t-1, t)$, and the constraints (2.7) ensure that each job is processed exactly once. This type of formulation is referred to as time-indexed formulation.

For convenience hereafter, let us assume that a dummy job is processed when no (ordinary) job is processed and the machine is idle. For this purpose, job 0 with $p_0 = 1$ and $f_0(t) = 0$ ($1 \leq t \leq T_E$) is introduced. This job, which is referred to as idle job, is processed $(T_E - \sum_{1 \leq i \leq n} p_i)$ times because it is the total length of idle time. By using the idle job, the problem (P) can be reformulated as follows.

$$(P') : \min_{\mathbf{x}} \sum_{\substack{0 \leq i \leq n \\ 1 \leq t \leq T_E}} f_i(t) x_{it}, \quad (2.9)$$

$$\text{s.t.} \quad \sum_{\substack{0 \leq i \leq n \\ t \leq s \leq \min(t+p_i-1, T_E)}} x_{is} = 1, \quad 1 \leq t \leq T_E, \quad (2.10)$$

$$\sum_{1 \leq t \leq T_E} x_{it} = 1, \quad 1 \leq i \leq n, \quad (2.11)$$

$$x_{it} \in \{0, 1\}, \quad 0 \leq i \leq n, 1 \leq t \leq T_E. \quad (2.12)$$

In this formulation, the constraint on the number of occurrences is not imposed on the idle job because it is automatically satisfied. Please also note that the inequality constraints (2.6) in (P) become the equality constraints (2.10) in (P').

2.2.2 Network Representation

The problem (P') formulated in the preceding subsection can be converted to a constrained shortest path problem on a network. In this subsection, this network representation will be introduced as a preparation for the next section.

To construct the network, that is, an acyclic weighted directed graph $G = (V, A)$, we allocate a node to every decision variable x_{it} . More specifically, the node set V is defined by

$$V = \{v_{n+1,0}\} \cup V_O \cup \{v_{n+1,T+1}\}, \quad (2.13)$$

$$V_O = \{v_{it} \mid 0 \leq i \leq n, 1 \leq t \leq T_E\}. \quad (2.14)$$

Here, another dummy job $n+1$ with $p_{n+1} = 1$, $f_{n+1}(t) = 0$ is introduced. This job is assumed to be completed at 0 and $T_E + 1$, and $v_{n+1,0}$ and $v_{n+1,T+1}$ denote the source and sink nodes, respectively. The arc set A is defined by

$$A = \{(v_{j,t-p_i}, v_{it}) \mid v_{j,t-p_i}, v_{it} \in V\}. \quad (2.15)$$

In addition, the length (weight) of an arc $(v_{j,t-p_i}, v_{it}) \in A$ is given by $f_i(t)$.

Let us consider a path from $v_{n+1,0}$ to $v_{n+1,T+1}$ on this network. Then, its length is equal to the objective value (2.9) of (P') if we choose $x_{it} = 1$ when v_{it} ($0 \leq i \leq n$) is on the path. These decision variables satisfy the resource constraints (2.10) but do not always satisfy the constraints (2.11). To satisfy (2.11), v_{it} ($1 \leq t \leq T_E$) should be visited exactly once for any i ($1 \leq i \leq n$). Therefore, (P') is equivalent to the problem to find a shortest path from $v_{n+1,0}$ to $v_{n+1,T+1}$ on G under the constraints that v_{it} ($1 \leq t \leq T_E$) should be visited exactly once for any i ($1 \leq i \leq n$).

Here, some notation and definitions are introduced. Let us denote by \mathcal{P} a set of nodes visited by a path from $v_{n+1,0}$ to $v_{n+1,T+1}$ on G . A node set \mathcal{P} and the corresponding path are both referred to as “path \mathcal{P} ” unless there is ambiguity. Let $L(\mathcal{P})$ be the length of a path \mathcal{P} defined by

$$L(\mathcal{P}) = \sum_{\substack{v_{it} \in \mathcal{P} \\ 0 \leq i \leq n}} f_i(t) = \sum_{\substack{v_{it} \in \mathcal{P} \\ 1 \leq i \leq n}} f_i(t). \quad (2.16)$$

We also define by $\mathcal{V}_i(\mathcal{P})$ ($1 \leq i \leq n$) the number of occurrences of v_{it} ($1 \leq t \leq T_E$) in \mathcal{P} . That is,

$$\mathcal{V}_i(\mathcal{P}) = |\{v_{it} \mid v_{it} \in \mathcal{P}\}|. \quad (2.17)$$

Then, the constraints that v_{it} ($1 \leq t \leq T_E$) should be visited exactly once on a path \mathcal{P} for any i ($1 \leq i \leq n$) can be written by

$$\mathcal{V}_i(\mathcal{P}) = 1, \quad 1 \leq i \leq n. \quad (2.18)$$

Accordingly, define a set of all the feasible paths by

$$\mathcal{Q} = \{\mathcal{P} \mid \mathcal{V}_i(\mathcal{P}) = 1, 1 \leq i \leq n\}. \quad (2.19)$$

Then, our problem on the network, which is referred to as (N), can be described simply by

$$(N) : \min_{\mathcal{P}} L(\mathcal{P}) \quad \text{s.t. } \mathcal{P} \in \mathcal{Q}. \quad (2.20)$$

2.3 Lagrangian Relaxation

As already mentioned in the preceding section, the LP relaxation of (P') obtained by removing the integrity constraints (2.12) yields a tight lower bound. However, it is not easy to solve when the number of jobs n becomes large because (P') has $O(nT_E)$ decision variables. To avoid this difficulty, the Lagrangian relaxation technique is employed instead. There are two types of relaxations for (P'), that is, the relaxations of (2.10) and (2.11), respectively. One of the advantages of the former relaxation is that (P') can be decomposed into trivial n subproblems corresponding to the n jobs. Therefore, it is sometimes referred to as Lagrangian decomposition. There is an early attempt by Fisher [9] to apply this relaxation for lower bound computation in a branch-and-bound algorithm. On the other hand, the primary advantage of the latter relaxation is that it gives an easy way to obtain a tighter lower bound than that by the LP relaxation. It is also referred to as state-space relaxation, which originates in the study by Christofides et al. [4] for routing problems. It was first applied to single-machine scheduling by Abdul-Razaq and Potts [1] and following their study, Ibaraki and Nakamura [15] proposed an exact algorithm based on the SSDP method [14]. Our exact algorithm also utilizes this type of relaxation. It also appears in the context of the column generation approach [32], or branch-and-bound algorithms [19, 26] for single-machine scheduling problems.

In the following subsections, the Lagrangian relaxation of (2.11) in (P') will be explained for its counterpart (N) in the network representation. Then, three types of constraints will be introduced and imposed on it to improve the lower bound.

2.3.1 Lagrangian Relaxation of the Number of Job Occurrences

To begin with, the violation of the constraints on the number of job occurrences (2.11) in (P') are penalized by Lagrangian multipliers μ_i ($1 \leq i \leq n$). It corresponds to the relaxation of the constraints (2.18) in (N), and its objective function $L(\mathcal{P})$ becomes

$$\begin{aligned}
 L(\mathcal{P}) &+ \sum_{1 \leq i \leq n} \mu_i (1 - \mathcal{V}_i(\mathcal{P})) \\
 &= \sum_{\substack{v_{it} \in \mathcal{P} \\ 1 \leq i \leq n}} f_i(t) + \sum_{1 \leq i \leq n} \mu_i - \sum_{1 \leq i \leq n} \mu_i |\{v_{it} \mid v_{it} \in \mathcal{P}\}| \\
 &= \sum_{\substack{v_{it} \in \mathcal{P} \\ 1 \leq i \leq n}} (f_i(t) - \mu_i) + \sum_{1 \leq i \leq n} \mu_i \\
 &= L_R(\mathcal{P}; \mu) + \sum_{1 \leq i \leq n} \mu_i,
 \end{aligned} \tag{2.21}$$

where $L_R(\mathcal{P}; \mu)$ is defined by

$$L_R(\mathcal{P}; \mu) = \sum_{\substack{v_{it} \in \mathcal{P} \\ 1 \leq i \leq n}} (f_i(t) - \mu_i). \quad (2.22)$$

Equations (2.21) and (2.22) imply that the relaxation for a fixed set of multipliers is equivalent to the problem to find a shortest unconstrained path from $v_{n+1,0}$ to $v_{n+1,T+1}$ 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) , that is,

$$(LR_0) : \min_{\mathcal{P}} L_R(\mathcal{P}; \mu) + \sum_{1 \leq i \leq n} \mu_i. \quad (2.23)$$

Clearly,

$$\min_{\mathcal{P}} L_R(\mathcal{P}; \mu) + \sum_{1 \leq i \leq n} \mu_i \leq \min_{\mathcal{P} \in \mathcal{Q}} L(\mathcal{P}) \quad (2.24)$$

holds and (LR_0) gives a lower bound of the original problem (N).

The relaxation (LR_0) is easy to solve in $O(nT_E)$ time by dynamic programming [1]. If we denote the partial path from $v_{n+1,0}$ to v_{it} by $\mathcal{P}_P(v_{n+1,0}, v_{it})$, the forward dynamic programming recursion is expressed by

$$\min_{\mathcal{P}} L_R(\mathcal{P}; \mu) = h_0(T_E + 1; \mu), \quad (2.25)$$

$$\begin{aligned} h_0(t; \mu) &= \min_{v_{it} \in V} L_R(\mathcal{P}_P(v_{n+1,0}, v_{it}); \mu) \\ &= \min_{v_{it} \in V} (h_0(t - p_i; \mu) + f_i(t) - \mu_i). \end{aligned} \quad (2.26)$$

We can also formulate the backward dynamic programming recursion in a similar manner.

To improve the lower bound more, the following three types of constraints are imposed on this relaxation. The first and the third were proposed in [4] and were applied in [1, 15]. The second constraints were proposed in [26, 28], and were applied in our previous algorithm [28] together with the other two.

2.3.2 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 ($1 \leq i \leq n$), nodes corresponding to job i , that is, v_{it} ($1 \leq t \leq T_E$) should not be visited more than once in any $\lambda + 1 > 0$ successive nodes on a path.

Therefore, these constraints forbid $v_{i,t-p_i} \rightarrow v_{it}$ when $\lambda = 1$ and $v_{i,t-p_i-p_j} \rightarrow v_{j,t-p_j} \rightarrow v_{it}$ when $\lambda = 2$, and so on. A subset of paths satisfying these constraints is denoted by \mathcal{Q}_λ ($\mathcal{Q} \subseteq \dots \subseteq \mathcal{Q}_2 \subseteq \mathcal{Q}_1$), and the relaxation with the constraints is denoted by

$$(\text{LR}_\lambda) : \min_{\mathcal{P}} L_{\text{R}}(\mathcal{P}; \mu) + \sum_{1 \leq i \leq n} \mu_i \quad \text{s.t. } \mathcal{P} \in \mathcal{Q}_\lambda. \quad (2.27)$$

Clearly, (LR_λ) gives a better (or at least not worse) lower bound as λ increases. However, the time complexity also increases because it is given by $O(n^\lambda T_E)$ [1, 19].

It might not be intuitive that the time complexity of (LR_1) is $O(nT_E)$ because that of (LR_0) is also $O(nT_E)$, but the following recursion confirms this fact:

$$\min_{\mathcal{P} \in \mathcal{Q}_1} L_{\text{R}}(\mathcal{P}; \mu) = y_1(T_E + 1; \mu), \quad (2.28)$$

$$y_1(t; \mu) = \min_{v_{it} \in V} h_1(v_{it}; \mu), \quad (2.29)$$

$$m_1(t; \mu) = \arg \min_i h_1(v_{it}; \mu), \quad (2.30)$$

$$z_1(t; \mu) = \min_{\substack{v_{it} \in V \\ i \neq m_1(t; \mu)}} h_1(v_{it}; \mu), \quad (2.31)$$

$$h_1(v_{it}; \mu) = \begin{cases} y_1(t - p_i; \mu) + f_i(t) - \mu_i & \text{if } i = 0 \text{ or } i \neq m_1(t - p_i; \mu), \\ z_1(t - p_i; \mu) + f_i(t) - \mu_i & \text{otherwise,} \end{cases} \quad (2.32)$$

$$y_1(0; \mu) = 0, \quad z_1(0; \mu) = +\infty, \quad m_1(0; \mu) = n + 1. \quad (2.33)$$

Roughly speaking, the shortest path $y_1(t; \mu)$ and the second shortest path $z_1(t; \mu)$ among those from $v_{n+1,0}$ to v_{it} are stored in the above recursion. When the path is expanded ($h_1(v_{it}; \mu)$), the shortest path is used if no successive job duplication occurs and otherwise the second shortest is used. It can be shown in a similar way that the time complexity of (LR_λ) for $\lambda \geq 2$ is $O(n^\lambda T_E)$.

In our algorithm, only (LR_1) and (LR_2) are considered. The relaxation (LR_1) becomes more tractable if we introduce a subnetwork $G_S = (V, A_S)$, where A_S is defined by

$$A_S = A \setminus \{(v_{i,t-p_i}, v_{it}) \mid v_{i,t-p_i}, v_{it} \in V_O, 1 \leq i \leq n\}, \quad (2.34)$$

and the length of an arc $(v_{j,t-p_j}, v_{it}) \in A_S$ is given by $f_i(t) - \mu_i$. Indeed, (LR_1) is equivalent to the unconstrained shortest path problem on G_S . On the other hand, (LR_2) is equivalent to the constrained shortest path problem even on G_S , under the constraints on three successive nodes.

2.3.3 Constraints on Adjacent Pairs of Jobs

The second constraints derive from the dominance theorem of dynamic programming [20] for adjacent pairs of jobs. For example, consider that two jobs i and j ($0 \leq i, j \leq n$, $i \neq j$) are successively processed and completed at t . The total completion cost of the two jobs is $f_i(t - p_j) + f_j(t)$ when they are sequenced as $i \rightarrow j$, and $f_j(t - p_i) + f_i(t)$ when $j \rightarrow i$. It follows that $i \rightarrow j$ 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, $j \rightarrow i$ 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 any 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 [28]. To summarize, the processing order of adjacent pairs of jobs can be restricted and it is imposed on the relaxation as constraints. Please note that the processing order of an ordinary job and the idle job can also be restricted.

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

$$\widehat{A}_S = A_S \setminus \{(v_{j,t-p_i}, v_{it}) \mid j \rightarrow i \text{ is forbidden at } t\}. \quad (2.35)$$

The relaxations (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 $O(n^2 T_E)$ [26, 28], only (LR_2) with the adjacency constraints, which is, denoted by (\widehat{LR}_2) , is used as in our previous algorithm. Let $\widehat{\mathcal{Q}}_2$ denote a subset of \mathcal{Q}_2 composed of paths on \widehat{G}_S that satisfy the constraints on three successive nodes.

2.3.4 Constraints on State-Space Modifiers

The last constraints are described in terms of state-space modifiers: Each ordinary job i ($1 \leq i \leq n$) is given a value $q_i \geq 0$ called state-space modifier and the constraint that the total modifier in a solution should be $\sum_{1 \leq i \leq n} q_i$ is imposed on (\widehat{LR}_2) . In our algorithm, the modifiers are chosen so that $q_i = 1$ for some i and $q_j = 0$ for $j \neq i$ ($1 \leq j \leq n$). In this case, the constraint simply requires that job i should be processed exactly once and hence is equivalent to $\mathcal{V}_i(\mathcal{P}) = 1$, that is, the constraint (2.18) for job i . It follows that all the constraints (2.18) are once relaxed, but one of them is recovered to improve the lower bound.

Let us consider that not the constraint (2.18) for a single job i but those for a subset of jobs \mathcal{M} are recovered to (\widehat{LR}_2) . Hereafter, (\widehat{LR}_2) with the constraints

$$\mathcal{V}_i(\mathcal{P}) = 1, \quad \forall i \in \mathcal{M} \quad (2.36)$$

is denoted by $(\widehat{\text{LR}}_2^m)$, where $m = |\mathcal{M}|$. Clearly, an optimal solution of $(\widehat{\text{LR}}_2^m)$ is also optimal for the original problem (N) when $m = n$.

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

$$q_{ij} = \begin{cases} 1, & \text{if the } j\text{th element of } \mathcal{M} \text{ is } i, \\ 0, & \text{otherwise.} \end{cases} \quad (2.37)$$

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

$$V^m = \{v_{n+1,0}^{\mathbf{0}_m}\} \cup V_O^m \cup \{v_{n+1,T+1}^{\mathbf{1}_m}\}, \quad (2.38)$$

$$V_O^m = \{v_{it}^{\mathbf{b}} \mid v_{it} \in V_O, \mathbf{q}_i^m \leq \mathbf{b} \leq \mathbf{1}_m\}, \quad (2.39)$$

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

$$\widehat{A}_S^m = \{(v_{j,t-p_i}^{\mathbf{b}-\mathbf{q}_i^m}, v_{it}^{\mathbf{b}}) \mid (v_{j,t-p_i}, v_{it}) \in \widehat{A}_S, \mathbf{q}_i^m + \mathbf{q}_j^m \leq \mathbf{b} \leq \mathbf{1}_m\}, \quad (2.40)$$

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

2.4 Network Reduction

Our exact algorithm utilizes $(\widehat{\text{LR}}_2)$ and $(\widehat{\text{LR}}_2^m)$ as sublimations of (LR_1) . More specifically, it first solves (LR_1) , next $(\widehat{\text{LR}}_2)$ and then $(\widehat{\text{LR}}_2^m)$ with jobs added to \mathcal{M} (with m increased). As already explained in the preceding section, all these are solvable by dynamic programming. In the SSDP method, unnecessary dynamic programming states are eliminated in the course of the algorithm. The efficiency of this state elimination determines the efficiency of the SSDP method and hence is very important because it enables us to reduce both computational efforts and memory usage.

The state elimination is interpreted as the removal of unnecessary nodes and arcs from G_S , \widehat{G}_S and \widehat{G}_S^m in the network representation. This section will give two types of network reductions utilized in our algorithm.

2.4.1 Network Reduction by Upper Bound

The first network reduction [15] utilizes an upper bound and is applied to all the relaxations (LR_1) , $(\widehat{\text{LR}}_2)$ and $(\widehat{\text{LR}}_2^m)$. Here, only the reduction for (LR_1) will be described because it does not differ much from those for $(\widehat{\text{LR}}_2)$ and $(\widehat{\text{LR}}_2^m)$.

Let us denote by $h_1(v_{it}; \mu)$ the shortest path length from $v_{n+1,0}$ to v_{it} on G_S (see (2.32)). Let us also denote by $H_1(v_{it}; \mu)$ the shortest path length from v_{it} to $v_{n+1,T+1}$ on G_S . Clearly,

$$h_1(v_{it}; \mu) + H_1(v_{it}; \mu) = \min_{\substack{\mathcal{P} \in \mathcal{Q}_1 \\ v_{it} \in \mathcal{P}}} L_R(\mathcal{P}; \mu) \quad (2.41)$$

holds. In other words, the lefthand side of (2.41) gives the shortest path length from $v_{n+1,0}$ to $v_{n+1,T+1}$ on G_S under the additional constraint that v_{it} should be passed through. Therefore, it can be said from (2.21) that if an upper bound UB of (N) satisfies

$$\text{UB} < h_1(v_{it}; \mu) + H_1(v_{it}; \mu) + \sum_{1 \leq i \leq n} \mu_i, \quad (2.42)$$

any optimal path for (N) never passes through v_{it} . Hence, v_{it} can be eliminated from G_S .

$h_1(v_{it}; \mu)$ appears in the forward recursion of dynamic programming, while $H_1(v_{it}; \mu)$ in the backward recursion. Therefore, this reduction can be performed by applying dynamic programming in both the directions.

2.4.2 Network Reduction by Dominance of Successive Jobs

To reduce the size of \widehat{G}_S^m for $(\widehat{\text{LR}}_2^m)$ more, dominance of n_D ($n_D \geq 3$) successive jobs is utilized to eliminate unnecessary arcs [28]. Let us define a set of paths \mathcal{Q}^m by

$$\mathcal{Q}^m = \{ \mathcal{P} \mid \mathcal{P} \in \widehat{\mathcal{Q}}_2^m, \mathcal{V}_i(\mathcal{P}) = 1 \ (1 \leq i \leq n) \}. \quad (2.43)$$

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

$$\mathcal{P}_{\text{opt}}^m = \arg \min_{\mathcal{P} \in \mathcal{Q}^m} L(\mathcal{P}). \quad (2.44)$$

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

$$L(\mathcal{P}') < L(\mathcal{P}), \quad (2.45)$$

$\mathcal{P}_{\text{opt}}^m$ never passes through the arc and hence it can be eliminated. To check this, only n_D nodes visited just before v_{it}^b (including v_{it}^b) in \mathcal{P} are considered in forward dynamic programming. That is, $(n_D! - 1)$ paths are checked for one \mathcal{P} as a candidate for a dominating path \mathcal{P}' , where the visiting order of the n_D nodes is interchanged. Similarly, in backward dynamic programming n_D nodes visited just after $v_{j,t-p_i}^{b-q_i^m}$ (including $v_{j,t-p_i}^{b-q_i^m}$) are considered to eliminate $(v_{j,t-p_i}^{b-q_i^m}, v_{it}^b)$.

For example, suppose that part of \widehat{G}_S^1 ($\mathcal{M} = \{3\}$) is given by Fig. 2.1, where $p_1 = 1$, $p_2 = 3$ and $p_3 = 2$. Let us check whether the arc $(v_{3,21}^1, v_{2,24}^1)$ can be eliminated in forward dynamic programming by setting $n_D = 3$. In this case, the following three types of paths passing through the arc $(v_{3,21}^1, v_{2,24}^1)$ should be considered:

$$\mathcal{P}_A = (\dots, v_{0,19}^0, v_{3,21}^1, v_{2,24}^1, \dots), \quad (2.46)$$

$$\mathcal{P}_B = (\dots, v_{1,19}^0, v_{3,21}^1, v_{2,24}^1, \dots), \quad (2.47)$$

$$\mathcal{P}_C = (\dots, v_{2,19}^0, v_{3,21}^1, v_{2,24}^1, \dots). \quad (2.48)$$

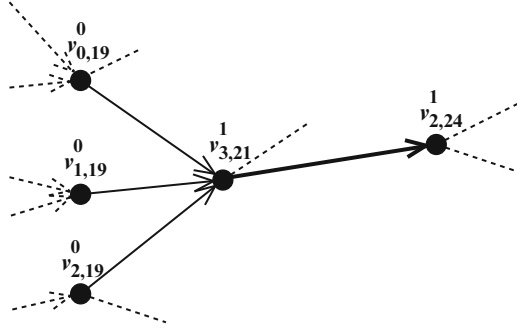


Fig. 2.1: An example of network reduction by dominance of three successive jobs ($n_D = 3$)

Since nodes corresponding to job 2 appear twice in \mathcal{P}_C , only \mathcal{P}_A and \mathcal{P}_B belong to \mathcal{Q}^m . Here, we are to check whether there exist dominating paths for these two types of paths. First, \mathcal{P}_A is considered and the following five candidates are generated by interchanging the visiting order of $v_{0,19}^0$, $v_{3,21}^1$ and $v_{2,24}^1$:

$$\mathcal{P}'_{A1} = (\dots, v_{0,19}^0, v_{2,22}^0, v_{3,24}^1, \dots), \quad (2.49)$$

$$\mathcal{P}'_{A2} = (\dots, v_{3,20}^1, v_{0,21}^0, v_{2,24}^1, \dots), \quad (2.50)$$

$$\mathcal{P}'_{A3} = (\dots, v_{3,20}^1, v_{2,23}^1, v_{0,24}^0, \dots), \quad (2.51)$$

$$\mathcal{P}'_{A4} = (\dots, v_{2,21}^0, v_{0,22}^0, v_{3,24}^1, \dots), \quad (2.52)$$

$$\mathcal{P}'_{A5} = (\dots, v_{2,21}^0, v_{3,23}^1, v_{0,24}^0, \dots). \quad (2.53)$$

Then, we search for k ($1 \leq k \leq 5$) satisfying $L(\mathcal{P}'_{Ak}) < L(\mathcal{P}_A)$. This is not difficult because $L(\mathcal{P}_A) - L(\mathcal{P}'_{Ak})$ depends only on the interchanged three nodes (eg. $L(\mathcal{P}_A) - L(\mathcal{P}'_{A3}) = f_3(21) + f_2(24) - f_3(20) - f_2(23)$). In addition, we need not consider $k = 1, 2$ because $L(\mathcal{P}'_{A1}) \geq L(\mathcal{P}_A)$ and $L(\mathcal{P}'_{A2}) \geq L(\mathcal{P}_A)$ hold from the constraints on adjacent pairs of jobs in Sect. 2.3.3.

Here, assume that $L(\mathcal{P}'_{A3}) < L(\mathcal{P}_A)$ holds. Then, \mathcal{P}_B is checked next and if there also exists a dominating path, the arc $(v_{3,21}^1, v_{2,24}^1)$ can be eliminated.

This reduction becomes more effective as n_D becomes larger, but both the number of paths passing through the target arc (cf. (2.46)–(2.48)) and the number of permutations of nodes that should be checked (cf. (2.49)–(2.53)) increase exponentially. Therefore, n_D is chosen as $n_D = 4$ in our algorithm.

2.5 Algorithm Overview

To summarize, our exact algorithm is composed of the following three stages that correspond to (LR_1) , (\widehat{LR}_2) and (\widehat{LR}_2^m) , respectively. If the gap between lower and upper bounds becomes zero in Stages 1 and 2, the algorithm is terminated² and the solution yielding the current upper bound UB is outputted as an optimal solution.

Stage 1 An initial upper bound UB is computed by the algorithm in Sect. 2.6.3. Lagrangian multipliers μ are adjusted by applying the subgradient algorithm to the following Lagrangian dual corresponding to (LR_1) :

$$\max_{\mu} \left(\min_{\mathcal{P} \in \mathcal{Q}_1} L_R(\mathcal{P}; \mu) + \sum_{1 \leq i \leq n} \mu_i \right). \quad (2.54)$$

Then, the network reduction in Sect. 2.4.1 is performed.

Stage 2 Multipliers μ are re-adjusted by applying the subgradient algorithm to the Lagrangian dual corresponding to (\widehat{LR}_2) . In the course of the algorithm, an upper bound is computed by the method in Sect. 2.6 and UB is updated if necessary. The network reduction in Sect. 2.4.1 is applied every time when the best lower bound or UB is updated.

Stage 3 Let the current best lower bound LB be $LB = \min_{\mathcal{P} \in \mathcal{Q}_2} L_R(\mathcal{P}; \mu) + \sum_{1 \leq i \leq n} \mu_i$. Then, the subprocedure is repeated by increasing the tentative upper bound UB^{tent} . It is terminated if $UB^{\text{tent}} = UB$ at the end of the subprocedure.

Subprocedure: Let $LB^{\text{sub}} = LB$ and $m = |\mathcal{M}| = 0$. Starting from $\widehat{G}_S^0 = \widehat{G}_S$, the relaxation (\widehat{LR}_2^m) for μ is solved with \mathcal{M} increased. When solving (\widehat{LR}_2^m) , all the network reductions in Sect. 2.4 are performed. In the course of the algorithm, an upper bound is computed by the method in Sect. 2.6, and UB^{tent} and UB are updated if necessary.

² To be more precise, the algorithm can be terminated when the gap becomes less than one because the objective function is integral.

In Stage 3 of the algorithm, a tentative upper bound UB^{tent} is introduced and used for the network reduction in Sect. 2.4.1 in place of the current upper bound UB . Since the effectiveness of the network reduction in Sect. 2.4.1 depends highly on the tightness of an upper bound, shortage of memory space for storing the network structure may occur in Stage 3 unless a tight upper bound is obtained. To reduce this dependence on the tightness, UB^{tent} is chosen as $UB^{\text{tent}} \leq UB$, so that the network reduction is performed by UB^{tent} . If UB^{tent} is a valid upper bound, it does not cause any problems and we need not repeat the subprocedure. If otherwise, UB^{tent} is increased and the subprocedure is applied again. It can be easily verified that UB^{tent} is proved to be a valid upper bound if $UB^{\text{tent}} = UB$ holds at the end of the subprocedure.

It is also important in Stage 3 which jobs should be added to \mathcal{M} . To suppress the increase of memory usage as much as possible, the job whose corresponding nodes appear less frequently in \widehat{G}_S^m is chosen first from $\mathcal{N} \setminus \mathcal{M}$ [28].

2.6 Upper Bound Computation

This section will describe the upper bound computation method in the algorithm of Sect. 2.5. It is composed of two parts: A solution of a relaxation is first converted to a feasible solution of the original problem by some heuristic algorithm, and then it is improved by a neighborhood search. In the following, they will be explained one by one. A tight lower bound is important also for the network reduction in Sect. 2.4.1.

2.6.1 Lagrangian Heuristic

The first part is an extension of the heuristic proposed in [15]. Since it exploits dynamic programming and thus is time-consuming, its greedy version is also applied.

Let us assume that some solution of a relaxation ((\widehat{LR}_2) or (\widehat{LR}_2^m)) is given as a sequence of jobs that possibly includes duplicated ordinary jobs. Our heuristic converts it to a feasible solution of the original problem by the following two steps:

1. A partial job sequence is generated from the solution by removing the idle job and duplicated jobs (only one job is kept in the sequence for every duplicated ordinary jobs). The number of jobs in the partial sequence is denoted by n_1 .
2. The unscheduled $n_2 (= n - n_1)$ 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 time is optimally inserted into the sequence. An optimal sequence can be obtained by dynamic programming in $O(n_2(n_1 + 1)2^{n_2}T_E)$ time.

The dynamic programming in 2. is time- (and space-) consuming because its time complexity is multiplied by T_E . Therefore, the method in [23] is adopted, 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 it exponentially. Hence, a greedy version of the heuristic is applied when n_2 is large. In this case, unscheduled n_2 jobs are inserted one by one according to the shortest processing time (SPT) order into their optimal positions. That is, the following procedure is used in place of 2.:

- 2'. The unscheduled 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 time is taken into account.

2.6.2 Improvement by Neighborhood Search

To improve a solution obtained by the heuristics, the dynasearch is applied. The dynasearch is a powerful neighborhood search and was first proposed for the single-machine scheduling problem without idle time [5]. Then, the enhanced dynasearch was proposed in [12] to improve its search ability by enlarging the neighborhood. Another extension [24] was for the problem with idle time and based on the results in [23]. In our algorithm this extended dynasearch is employed.

2.6.3 Initial Upper Bound

To obtain an initial upper bound, the greedy version of the heuristic in Sect. 2.6.1 is first employed, where all the jobs are assumed to be unscheduled ($n_1 = 0$ and $n_2 = n$), and not only the SPT order but also the longest processing time (LPT), earliest due date (EDD) and latest due date (LDD) orders are considered when jobs are inserted. Then, the extended dynasearch is applied to the best of the four solutions.

2.7 Numerical Experiments

Our 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 [25, 26], and that with distinct release dates ($1 |r_i| \sum (\alpha_i E_i + \beta_i T_i)$) in [2]. These are referred to as the Sourd's

benchmark set and the Bülbül's benchmark set, respectively. The Sourd's benchmark set is generated in the following way:

1. Processing times p_i are generated from the uniform distribution $U[10, 100]$. Let $P = \sum_{i=1}^n p_i$.
2. Due dates d_i are generated from $U[d_{\min}, d_{\max}]$, where $d_{\min} = \max(p_i, \lfloor P(\tau - \rho/2) \rfloor)$, $d_{\max} = d_{\min} + \lfloor \rho P \rfloor$.
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 five instances are generated for each combination of (n, τ, ρ) , where $\tau \in \{0.2, 0.5, 0.8\}$ and $\rho \in \{0.2, 0.5, 0.8\}$.

On the other hand, the generation scheme of the Bülbül's benchmark set is as follows:

1. Processing times p_i 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. Due dates d_i are generated from $U[d_{\min}, d_{\max}]$, where $d_{\min} = \max(0, \lceil (1 - \tau - \rho/2)P \rceil)$ and $d_{\max} = \lceil (1 - \tau + \rho/2)P \rceil$.
4. Both earliness weights α_i and tardiness weights β_i are generated from $U[0, 100]$.
5. For each combination of $(n, p_{\max}, \tau, \rho)$, five 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 $n \in \{100, 130, 170, 200\}$, $p_{\max} \in \{50, 75, 100\}$.

Both the benchmark sets are available from the aforementioned web page.

The algorithm is coded in C (gcc) and we run it on a 3 GHz Intel® Core2 Duo E6850 desktop computer with 4 GB RAM. The maximum memory size for storing the network structure is restricted to 384 MB.

Benchmark results are summarized in Tables 2.1 and 2.2, where the number of optimally solved instances in each stage, the average and maximum CPU times in seconds are shown. We can verify that all the instances are optimally solved. The most efficient algorithm for the problem so far except ours is the branch-and-bound algorithm by Sourd [26]. He reported that he succeeded in solving all the 50 jobs instances in the Sourd's benchmark set within 1,000 s, and all the 60 jobs instances in the Bülbül's benchmark set within 500 s on a 3.2 GHz Intel® Pentium4 desktop computer. On the other hand, our algorithm only takes at most 3.5 and 3 s for these instances, respectively. It is true that our CPU is about twice as fast as Sourd's CPU, but our algorithm is much faster even if this difference is taken into account.

The detailed results are given in Tables 2.3 and 2.4. In these tables, average and maximum CPU times in each stage are given separately. The average and maximum gaps between lower and upper bounds are also shown in percent, where the gap is calculated by $100(UB - LB)/UB$. Please note that it is not easy to examine memory usage of our algorithm because the number of jobs added to \mathcal{M} at one iteration of the subprocedure in Stage 3 is changed adaptively. Therefore, it is not shown here. It can be observed from the tables that the gap becomes small in Stage 2. This implies that (\widehat{LR}_2) yields a tight lower bound.

Table 2.1: Computational results for the Sourd’s benchmark set of $1||\sum(\alpha_iE_i+\beta_iT_i)$

<i>n</i>	Instances	Optimally solved instances				CPU time (s)	
		Stage1	Stage2	Stage3	Total	Ave.	Max.
20	1,274	133	952	189	1,274	0.05	0.16
30	1,274	22	902	350	1,274	0.22	0.55
40	1,274	13	757	504	1,274	0.58	1.77
50	1,274	2	631	641	1,274	1.33	3.44
60	45	0	22	23	45	2.61	6.09
90	45	0	10	35	45	13.22	30.25

Table 2.2: Computational results for the Bülbül’s benchmark set of $1|r_i|\sum(\alpha_iE_i+\beta_iT_i)$

<i>n</i>	Instances	Optimally solved instances				CPU time (s)	
		Stage1	Stage2	Stage3	Total	Ave.	Max.
20	300	37	221	42	300	0.01	0.05
40	300	1	155	144	300	0.18	0.58
60	300	0	63	237	300	1.02	3.05
80	300	0	35	265	300	3.16	11.63
100	300	0	6	294	300	17.83	44.92
130	300	0	2	298	300	51.62	140.26
170	300	0	0	300	300	155.27	379.97
200	300	0	0	300	300	303.27	771.52

Table 2.3: Detailed results for the Sourd’s benchmark set of $1||\sum(\alpha_iE_i+\beta_iT_i)$

n	Stage 1					Solved	Stage 2					Solved	Stage 3			
	Time (s)		Gap (%)				Time (s)		Gap (%)				Time (s)		Solved	
	Ave	Max	Ave	Max			Ave	Max	Ave	Max			Ave	Max		
20	0.04	0.15	6.17	62.88		133	0.01	0.08	0.61	17.01		952	0.01	0.04		189
30	0.15	0.34	6.45	54.38		22	0.07	0.42	0.49	11.01		902	0.01	0.10		350
40	0.36	1.00	6.14	59.91		13	0.21	1.15	0.39	9.30		757	0.03	0.23		504
50	0.73	1.56	5.59	49.95		2	0.57	2.71	0.37	5.62		631	0.08	0.46		641
60	1.44	2.38	5.12	27.67		0	1.09	4.67	0.35	3.96		22	0.17	0.66		23
90	5.80	9.98	5.77	30.59		0	6.79	19.43	0.33	1.89		10	0.81	3.73		35

Table 2.4: Detailed results for the Bülbül’s benchmark set of $1|r_i|\sum(\alpha_iE_i+\beta_iT_i)$

<i>n</i>	Stage 1						Stage 2						Stage 3		
	Time (s)		Gap (%)		Solved		Time (s)		Gap (%)		Solved		Time (s)		Solved
	Ave	Max	Ave	Max			Ave	Max	Ave	Max			Ave	Max	
20	0.01	0.04	3.02	48.09	37		0.00	0.03	0.23	8.29	221		0.00	0.01	42
40	0.09	0.28	3.58	32.19	1		0.08	0.45	0.17	1.79	155		0.01	0.07	144
60	0.36	0.93	3.82	14.77	0		0.61	2.11	0.18	1.71	63		0.06	0.43	237
80	0.90	2.35	3.41	13.77	0		2.07	9.46	0.17	0.93	35		0.22	1.06	265
100	4.56	9.79	3.35	20.70	0		12.09	35.11	0.17	1.20	6		1.21	7.90	294
130	11.28	24.69	3.10	15.02	0		34.81	112.59	0.16	0.75	2		5.57	51.13	298
170	26.41	53.50	3.07	11.65	0		100.01	277.55	0.16	0.59	0		28.85	174.55	300
200	43.48	91.59	2.96	11.95	0		185.44	427.72	0.15	0.48	0		74.35	405.55	300

2.8 Conclusion

This paper introduced our exact algorithm [29] for the single-machine total weighted earliness–tardiness problem. Due to the tightness of lower and upper bounds, this dynamic-programming-based exact algorithm is so efficient that it outperformed the existing best algorithm and could optimally solve 200 jobs instances. In our most recent paper [30] it is shown that a new algorithm after several improvements is also effective for other types of single-machine problems such as $1|r_i|\sum_i w_iC_i$ and $1|r_i|\sum_i w_iT_i$.

The algorithm is based on dynamic programming and hence reduction of memory usage is crucial for further improving the algorithm. Some additional constraints (cuts) to obtain a better lower bound and/or new network reduction methods to reduce memory usage directly would be necessary for this purpose, which we should consider in future research. Another direction of research will be to extend the algorithm to a wider class of problems such as the problems with precedence constraints, setup times and so on.

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

- 1. Abdul-Razaq, T.S., Potts, C.N.: Dynamic programming state-space relaxation for single-machine scheduling. *Journal of the Operational Research Society* **39**, 141–152 (1988)
- 2. Bülbül, K., Kaminsky, P., Yano, C.: Preemption in single machine earliness/tardiness scheduling. *Journal of Scheduling* **10**, 271–292 (2007)
- 3. Chang, P.C.: A branch and bound approach for single machine scheduling with earliness and tardiness penalties. *Computers and Mathematics with Applications* **37**, 133–144 (1999)

4. Christofides, N., Mingozzi A, Toth P.: State-space relaxation procedures for the computation of bounds to routing problems. *Networks* **11**, 145–164 (1981)
5. Congram, R.K., Potts, C.N., van de Velde, S.L.: An iterated dynasearch algorithm for the single machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing* **14**, 52–67 (2002)
6. Davis, J.S., Kanet, J.J.: Single-machine scheduling with early and tardy completion costs. *Naval Research Logistics* **40**, 85–101 (1993)
7. Detienne, B., Pinson, É., Rivreau, D.: Lagrangian domain reductions for the single machine earliness-tardiness problem with release dates, *European Journal of Operational Research* **201**, 45–54 (2010)
8. Dyer, M.E, Wolsey, L.A.: Formulating the single-machine sequencing problem with release dates as a mixed integer problem. *Discrete Applied Mathematics* **26**, 255–270 (1990)
9. Fisher, M.L.: Optimal solution of scheduling problems using Lagrange multipliers: Part I. *Operations Research* **21** 1114–27 (1973)
10. Fry, T.D., Armstrong, R.D., Darby-Dowman, K., Philipoom, P.R.: A branch and bound procedure to minimize mean absolute lateness on a single processor. *Computers & Operations Research* **23**, 171–182 (1996)
11. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics* **5**, 287–326 (1979)
12. Grosso, A., Della Croce, F., Tadei, R.: An enhanced dynasearch neighborhood for the single machine total weighted tardiness scheduling problem. *Operations Research Letters* **32**, 68–72 (2004)
13. Hoogeveen, J.A., van de Velde, S.L.: A branch-and-bound algorithm for single-machine earliness-tardiness scheduling with idle time. *INFORMS Journal on Computing* **8**, 402–412 (1996)
14. Ibaraki, T.: Enumerative approaches to combinatorial optimization. *Annals of Operations Research* **10** and **11** (1987)
15. Ibaraki, T., Nakamura, Y.: A dynamic programming method for single machine scheduling, *European Journal of Operational Research* **76**, 72–82 (1994)
16. Kim, Y.-D., Yano, C.A.: Minimizing mean tardiness and earliness in single-machine scheduling problems with unequal due dates. *Naval Research Logistics* **41**, 913–933 (1994)
17. Lawler, E.L.: A “pseudopolynomial” algorithm for sequencing jobs to minimize total tardiness, *Annals of Discrete Mathematics* **1**, 331–342 (1977)
18. Lenstra, J.K, Rinnooy Kan, A.H.G. and Brucker, P.: Complexity of machine scheduling problems. *Annals of Discrete Mathematics* **1**, 343–362 (1977)
19. Péridy, L., Pinson, É., Rivreau, D.: Using short-term memory to minimize the weighted number of late jobs on a single machine. *European Journal of Operational Research* **148**, 591–603 (2003)
20. Potts, C.N., Van Wassenhove, L.N.: A branch and bound algorithm for the total weighted tardiness problem. *Operations Research* **33**, 363–377 (1985)
21. Pritsker, A.A.B., Watters, L.J., Wolfe, P.M.: Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science* **16**, 93–108 (1969)
22. Sourd, F., Kedad-Sidhoum, S.: The one-machine problem with earliness and tardiness penalties. *Journal of Scheduling* **6** 533–549 (2003)
23. Sourd, F.: Optimal timing of a sequence of tasks with general completion cost. *European Journal of Operational Research* **165**, 82–96 (2005)
24. Sourd, F.: Dynasearch for the earliness-tardiness scheduling problem with release dates and setup constraints. *Operations Research Letters* **34**, 591–598 (2006)
25. Sourd, F., Kedad-Sidhoum, S.: A faster branch-and-bound algorithm for the earliness-tardiness scheduling problem. *Journal of Scheduling* **11**, 49–58 (2008)
26. Sourd, F.: New exact algorithms for one-machine earliness-tardiness scheduling. *INFORMS Journal on Computing* **21**, 167–175 (2009)
27. Sousa, J.P., Wolsey, L.A.: A time indexed formulation of non-preemptive single machine scheduling problems. *Mathematical Programming* **54**, 353–367 (1992)

28. Tanaka, S., Fujikuma, S., Araki, M.: An exact algorithm for single-machine scheduling without machine idle time. *Journal of Scheduling* **12**, 575–593 (2009)
29. Tanaka, S., Fujikuma, S.: An efficient exact algorithm for general single-machine scheduling with machine idle time. 4th IEEE Conference on Automation Science and Engineering (IEEE CASE 2008), 371–376 (2008)
30. Tanaka, S., Fujikuma, S.: A dynamic-programming-based exact algorithm for single-machine scheduling with machine idle time. *Journal of Scheduling*, available online. DOI: 10.1007/s10951-011-0242-0
31. van den Akker, J.M., van Hoesel, C.P.M., Savelsbergh, M.W.P.: A polyhedral approach to single-machine scheduling problems. *Mathematical Programming* **85** 541–572 (1999)
32. van den Akker, J.M., Hurkens, C.A.J., Savelsbergh, M.W.P.: Time-indexed formulations for machine scheduling problems: column generation. *INFORMS Journal on Computing* **12**, 111–124 (2000)
33. Yano, C.A., Kim, Y.-D.: Algorithms for a class of single-machine weighted tardiness and earliness problems. *European Journal of Operational Research* **52**, 167–178 (1991)
34. Yau, H., Pan, Y., Shi, L.: New solution approaches to the general single machine earliness-tardiness problem. *IEEE Transactions on Automation Science and Engineering* **5**, 349–360 (2008)



<http://www.springer.com/978-1-4614-1122-2>

Just-in-Time Systems

Rios, R.; Ríos-Solís, Y.A. (Eds.)

2012, XII, 308 p., Hardcover

ISBN: 978-1-4614-1122-2