An exact algorithm for the single-machine total weighted tardiness problem with sequence-dependent setup times

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

[a]*Department of Electrical Engineering, Kyoto University*
Kyotodaigaku-Katsura, Nishikyo-ku, Kyoto 615-8510, Japan
[b]*Matsue College of Technology,*
14–4 Nishiikuma-Cho, Matsue City, Shimane 690-8518, Japan

Abstract

This study proposes an exact algorithm for the single-machine total weighted tardiness problem with sequence-dependent setup times. The algorithm is an extension of the authors' previous algorithm for the single-machine scheduling problem without setup times, which is based on the SSDP (Successive Sublimation Dynamic Programming) method. In the first stage of the algorithm, the conjugate subgradient algorithm or the column generation algorithm is applied to a Lagrangian relaxation of the original problem to adjust multipliers. Then, in the second stage, constraints are successively added to the relaxation until the gap between lower and upper bounds becomes zero. The relaxation is solved by dynamic programming and unnecessary dynamic programming states are eliminated to suppress the increase of computation time and memory space. In this study a branching scheme is integrated into the algorithm to manage to solve hard instances. The proposed algorithm is applied to benchmark instances in the literature and almost all of them are optimally solved.

Keywords:
single-machine total weighted tardiness problem, sequence-dependent setup times, exact algorithm, Lagrangian relaxation, dynamic programming

*Tel.: +81-75-383-2204, Fax: +81-75-383-2201, E-mail: tanaka@kuee.kyoto-u.ac.jp

1. Introduction

This study is devoted to the single-machine total weighted tardiness problem with sequence-dependent setup times. In practical production systems it is often the case that setup times are incurred when a machine starts processing a job. It is also the case that they differ in accordance with the job finished just before it. Such setup times are called sequence-dependent [1]. Our problem is to minimize total weighted tardiness on a single machine under the existence of sequence-dependent setup times. According to the standard classification of scheduling problems, this problem is denoted by $1|s_{ij}|\sum w_iT_i$.

For $1|s_{ij}|\sum w_iT_i$, or, its unweighted version $1|s_{ij}|\sum T_i$, there are many studies on metaheuristics such as GA (Genetic Algorithm), SA (Simulated Annealing), TS (Tabu Search), ACO (Ant Colony Optimization), PSO (Particle Swarm Optimization), DE (Differential Evolution Algorithm), GRASP (Greedy Randomized Adaptive Search Procedure), VNS (Variable Neighborhood Search) and so on [2–28]. In recent studies, effectiveness of algorithms for $1|s_{ij}|\sum w_iT_i$ is evaluated by 120 benchmark instances with 60 jobs, which were generated by Cicirello [8, 16]. Best solutions of the instances have been updated by several researchers and the latest are summarized in [28]. For $1|s_{ij}|\sum T_i$, the benchmark instances with 15, 25, 35 and 45 jobs by Rubin and Ragatz [2] and those with 55, 65, 75 and 85 jobs by Gagné et al. [7] play the same role.

There are also some studies on exact algorithms for $1|s_{ij}|\sum T_i$ [29–31], for the unweighted earliness-tardiness problem with a common duedate ($1|s_{ij},d_i = d|\sum(E_i+T_i)$) [32], for the weighted earliness-tardiness problem with distinct duedates ($1|s_{ij}|\sum(E_i+T_i)$) [33], and for the maximum tardiness problem [34]. Among them, Bigras et al. [31] succeeded in solving instances of $1|s_{ij}|\sum T_i$ with up to 45 jobs, but it took more than 7 days for the hardest instance. Moreover, there has been no attempts so far to solve $1|s_{ij}|\sum w_iT_i$ optimally. Therefore, it is quite a challenging theme to solve this NP-hard combinatorial optimization problem.

This study proposes an exact algorithm for $1|s_{ij}|\sum w_iT_i$. It is derived from the authors' previous ones [35, 36] for single-machine scheduling without setup times. These algorithms are based on the SSDP (Successive Sublimation Dynamic Programming) method [37, 38]. In [35, 36] it was shown that this framework is so effective that instances with 300 jobs of the total weighted tardiness problem ($1||\sum w_iT_i$) can be solved optimally. However, the framework is not so efficient for $1|s_{ij}|\sum w_iT_i$ as for the problems in [35, 36] because a tight lower bound is harder to obtain due to the existence of sequence-dependent setup times. Therefore, a

branching scheme is integrated into it to suppress memory usage, which is the bottleneck of the framework. It will be demonstrated that this algorithm can solve all the $1|s_{ij}|\sum w_iT_i$ instances by Cicirello, all the $1|s_{ij}|\sum T_i$ instances by Rubin and Ragatz, and almost all of the $1|s_{ij}|\sum T_i$ instances by Gagné et al. Although it is still time-consuming to solve some instances, the optimal objective values enable us an absolute (and not relative) performance evaluation of the existing and developing metaheuristic approaches.

The remainder of this paper is organized as follows. In Section 2, the problem is formulated as a constrained shortest path problem. Next, in Section 3, an exact algorithm is constructed based on our previous ones, and in Section 4, it is applied to the $1|s_{ij}|\sum w_iT_i$ instances. Then, in Section 5, the algorithm is improved to solve the unsolved instances, and in Section 6, the improved algorithm is applied to the remaining instances. Finally, Section 7 summarizes this study.

2. Problem description and formulation

Consider that a given set of n jobs $\mathcal{N} = \{1, \ldots, n\}$ are to be processed without preemption on a machine that can process at most one job at a time. Job i ($i \in \mathcal{N}$) has a processing time $p_i > 0$, a duedate $d_i \geq 0$ and a weight for tardiness $w_i \geq 0$. A setup time $s_{ij} \geq 0$ is given for every pair of jobs i and j, and s_{ij} is required before processing job j if job i is an immediate predecessor of job j. A setup time $s_{0j} \geq 0$ is required even when job j is the first job on the machine. All the processing times, duedates, weights and setup times are assumed to be integers.

Let us define the cost function $f_i(t)$ of job i by

$$f_i(t) = w_i \max(t - d_i, 0). \tag{1}$$

Then, the objective of our problem is to find an optimal schedule that minimizes the total cost $\sum_{i \in \mathcal{N}} f_i(C_i)$, where C_i denotes the completion time of job i. Since $f_i(t)$ is a nondecreasing function of t, we only need to consider schedules without idle time. In other words, a job should be started immediately when it can be started, after the preceding job is finished and then the required setup is finished. Therefore, job completion times can be assumed to be integral without loss of optimality because all the processing times and the setup times are integral. In addition, the optimal objective value can also be assumed to be integral.

Next, to make our problem more tractable, it is converted to a constrained shortest path problem on an acyclic directed graph $G = (V, A, W)$. Roughly speaking, this is done by assigning one node v_{it} to the completion of job $i \in \mathcal{N}$ at

$t = p_i, p_i + 1, \ldots, T_{\max}$, where $T_{\max}$ denotes the maximum makespan over all feasible schedules. Let us define a node set V by

$$V = \{v_{00}\} \cup V_{\mathrm{O}} \cup \{v_{n+1,T_{\max}}\}, \tag{2}$$

$$V_{\mathrm{O}} = \{v_{it} \,|\, i \in \mathscr{N},\ p_i \leq t \leq T_{\max}\}. \tag{3}$$

Here, v_{00} and $v_{n+1,T_{\max}}$ denote the source and sink nodes, respectively. We also define an arc set A by

$$A = A_{\mathrm{A}} \cup A_{\mathrm{B}} \cup A_{\mathrm{C}}, \tag{4}$$

$$A_{\mathrm{A}} = \{(v_{00}, v_{i,s_{0i}+p_i}) \,|\, v_{i,s_{0i}+p_i} \in V_{\mathrm{O}}\}, \tag{5}$$

$$A_{\mathrm{B}} = \{(v_{j,t-s_{ji}-p_i}, v_{it}) \,|\, v_{j,t-s_{ji}-p_i}, v_{it} \in V_{\mathrm{O}},\ i \neq j\}, \tag{6}$$

$$A_{\mathrm{C}} = \{(v_{jt}, v_{n+1,T_{\max}}) \,|\, v_{jt} \in V_{\mathrm{O}},\ T_{\min} \leq t \leq T_{\max}\}, \tag{7}$$

where $T_{\min}$ is the minimum makespan over all feasible schedules. The length $W(e)$ of an arc $e \in A$ is given by

$$W(e) = \begin{cases} f_i(t), & \text{if } e = (v_{j,t-s_{ji}-p_i}, v_{it}) \in A_{\mathrm{A}} \cup A_{\mathrm{B}}, \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

Then, our problem, which is referred to as (P), is equivalent to the shortest path problem from v_{00} to $v_{n+1,T_{\max}}$ on G under the constraint that v_{it} should be visited exactly once for any $i \in \mathscr{N}$. Indeed, the length of a shortest path is identical to the minimum total weighted tardiness and v_{it} visited in the path corresponds to the completion of job i at t in an optimal solution.

The problem to obtain the exact values of $T_{\min}$ and $T_{\max}$ is equivalent to the asymmetric traveling salesman problem [1] and hence is not easy to solve optimally. Therefore, we compute a lower bound of $T_{\min}$ and an upper bound of $T_{\max}$ by solving the following assignment relaxation problems and regard them as $T_{\min}$

and $T_{\max}$, respectively.

$$T_{\min} = \min_{x} \sum_{\substack{0\le i\le n\\ 1\le j\le n\\ i\neq j}} s_{ij}x_{ij} + \sum_{i=1}^{n} p_i \tag{9}$$

$$\text{s.t.} \sum_{\substack{0\le i\le n\\ i\neq j}} x_{ij} = 1, \qquad 0 \le j \le n, \tag{10}$$

$$\sum_{\substack{0\le j\le n\\ j\neq i}} x_{ij} = 1, \qquad 0 \le i \le n, \tag{11}$$

$$x_{ij} \in \{0,1\}, \qquad 0 \le i,j \le n,\ i \neq j, \tag{12}$$

$$T_{\max} = \max_{x} \sum_{\substack{0\le i\le n\\ 1\le j\le n\\ i\neq j}} s_{ij}x_{ij} + \sum_{i=1}^{n} p_i, \tag{13}$$

$$\text{s.t. } (10),\ (11),\ (12).$$

These problems can be polynomially solved by, for example, the Hungarian method [39].

Let $\mathscr{P}$ denote a set of nodes visited in a path from v_{00} to $v_{n+1,T_{\max}}$ on the network G. In the following, the path corresponding to $\mathscr{P}$ is referred to as "path $\mathscr{P}$" for simplicity. Let $L(\mathscr{P})$ denote the length of a path $\mathscr{P}$, which is defined by

$$L(\mathscr{P}) = \sum_{\substack{v_{it}\in\mathscr{P}\\ i\in\mathscr{N}}} f_i(t). \tag{14}$$

Then, the constraints in our problem that v_{it} should be visited exactly once in a path $\mathscr{P}$ for any $i \in \mathscr{N}$ can be written as

$$\mathscr{V}_i(\mathscr{P}) = |\{v_{it} \,|\, v_{it} \in \mathscr{P}\}| = 1 \quad (i \in \mathscr{N}). \tag{15}$$

If we denote by $\mathscr{Q}$, the set of all feasible paths satisfying (15), our problem (P) can be formulated as follows.

$$(\mathrm{P}) : \min_{\mathscr{P}} L(\mathscr{P}) \quad \text{s.t. } \mathscr{P} \in \mathscr{Q}. \tag{16}$$

3. Proposed Algorithm

In our previous studies [35, 36], exact algorithms for the single-machine problem without setup times were proposed based on the SSDP method [37, 38]. In these algorithms, we start from a Lagrangian relaxation of the original problem and then constraints are successively added to it until the gap between lower and upper bounds vanishes. Since the relaxation is solved by dynamic programming, it is inevitable that the number of dynamic programming states increases as the number of added constraints increases. To suppress it, unnecessary dynamic programming states are eliminated in the course of the algorithm.

One of the primary differences between the problem in this study and the problems in [35, 36] is that constraints derived from the dominance of adjacent pairs of jobs [40, 35] cannot be imposed on the relaxations. These constraints are to restrict the processing order of adjacent pairs of jobs i and j, by checking their costs when sequenced as $i \to j$ and as $j \to i$, respectively. However, it relies on the fact that interchanging jobs i and j does not affect the other jobs, which is not true for $1|s_{ij}|\sum w_i T_i$ because of the existence of sequence-dependent setup times. Since these constraints are very much effective for both improving the lower bound and reducing computation time, we cannot expect that the framework in [35, 36] works for $1|s_{ij}|\sum w_i T_i$ as efficiently as for the problems without setup times. Nonetheless, a simple extension of our algorithm can solve most of the benchmark instances, although further improvements are necessary to solve the remaining instances. In the following, we will first give the Lagrangian relaxation and the constraints to be added. Next, how to reduce the size of the network, which corresponds to the elimination of dynamic programming states, will be stated briefly. Then, a heuristic algorithm to obtain a tight upper bound will be constructed. Finally, we will explain our algorithm.

3.1. Lagrangian Relaxation

To obtain a lower bound of (P), the violation of the constraints (15) is penalized by Lagrangian multipliers μ_i ($i \in \mathcal{N}$). This relaxation is denoted by (LR_1). In

(LR_1), the length of a path $\mathscr{P}$ is given by

$$\begin{aligned} L(\mathscr{P}) + \sum_{i\in\mathscr{N}} \mu_i(1-\mathscr{V}_i(\mathscr{P})) &= \sum_{\substack{v_{it}\in\mathscr{P}\\ i\in\mathscr{N}}} f_i(t) + \sum_{i\in\mathscr{N}} \mu_i - \sum_{i\in\mathscr{N}} \mu_i \left|\{v_{it}\,|\,v_{it}\in\mathscr{P}\}\right| \\ &= \sum_{\substack{v_{it}\in\mathscr{P}\\ i\in\mathscr{N}}} (f_i(t)-\mu_i) + \sum_{i\in\mathscr{N}} \mu_i \\ &= L_{\mathrm{R}}(\mathscr{P};\,\mu) + \sum_{i\in\mathscr{N}} \mu_i. \end{aligned} \tag{17}$$

It implies that (LR_1) for a fixed set of multipliers is equivalent to find the unconstrained shortest path from v_{00} to $v_{n+1,T_{\max}}$ on $G' = (V, A, W')$ where W' is

$$W'(e) = \begin{cases} f_i(t) - \mu_i, & \text{if } e = (v_{j,t-s_{ji}-p_i}, v_{it}) \in A_{\mathrm{A}} \cup A_{\mathrm{B}}, \\ 0, & \text{otherwise.} \end{cases} \tag{18}$$

(LR_1) can be solved by dynamic programming. In the forward dynamic programming, the length of a path from v_{00} to every other node $v_{it} \in V \setminus \{v_{00}\}$ is computed recursively by increasing t. This computation can be done in $O(n)$ time for one node because the number of incoming arcs to a node (except $v_{n+1,T_{\max}}$) is at most $n-1$. Since there are at most $O(nT_{\max})$ nodes, the time complexity of the dynamic programming is given by $O(n^2 T_{\max})$. If we note that each arc is checked only once, the time complexity is given simply by $O(|A|)$ (clearly, the number of arcs is $O(n^2 T_{\max})$). As a matter of fact, the backward dynamic programming can be performed with the same time complexity.

To improve the lower bound obtained by this relaxation, the following two types of constraints are imposed on it. The first are on successively visited nodes on a path, which are described by

> For any $i \in \mathscr{N}$, nodes corresponding to job i, i.e. v_{it}, should not be visited more than once in any $k+1$ ($k \geq 2$) successive nodes in a path.

These correspond to the k-cycle elimination constraints that often appear in routing problems (e.g. [41]). It should also be noted that this type of constraint is also utilized in the exact algorithm for $1|s_{ij}|\sum T_i$ by Bigras et al. [31]. The set of $\mathscr{P}$ satisfying the constraints is denoted by $\mathscr{Q}^k$ and the problem to find the shortest path in $\mathscr{Q}^k$ on G is denoted by (LR_k). This problem can be solved by dynamic programming in $O(n^k T_{\max})$ time [35]. Unlike the relaxations of the problems without sequence-dependent setup times for which the time complexities of (LR_1) and (LR_2) differ, they are the same due to the existence of sequence-dependent setup

times. Therefore, only (LR_2), whose time complexity is $O(n^2T_{\max})$ or $O(|A|)$, is employed in this paper.

The second are the relaxed constraints (15) and some of them are recovered to (LR_2). Let $\mathscr{M} = \{\phi_1, \phi_2, \ldots, \phi_m\} \subseteq \mathscr{N}$ and the constraints (15) for ϕ_i $(1 \leq i \leq m)$ are imposed on (LR_2). The problem (LR_2) with these constraints is denoted by (LR_2^m). Clearly, (LR_2^m) is equivalent to the original problem (P) when $\mathscr{M} = \mathscr{N}$.

To solve (LR_2^m), we introduce a new acyclic directed graph $G^m = (V^m, A^m, W')$. First, let us define an m-dimensional vector q_i^m by $q_i^m = (q_{i1}, \ldots, q_{im})$, where

$$q_{ij} = \begin{cases} 1, & \text{if } i = \phi_j, \\ 0, & \text{otherwise.} \end{cases} \tag{19}$$

Next, the node set V^m is defined by

$$V^m = \{v_{00}^{0_m}\} \cup V_{\mathrm{O}}^m \cup \{v_{n+1,T_{\max}}^{1_m}\}, \tag{20}$$

$$V_{\mathrm{O}}^m = \{v_{it}^b \,|\, v_{it} \in V_{\mathrm{O}},\ q_i^m \leq b \leq 1_m\}, \tag{21}$$

where 0_m and 1_m denote m-dimensional vectors whose elements are all zero and all one, respectively. The arc set A^m is defined by

$$A^m = A_{\mathrm{A}}^m \cup A_{\mathrm{B}}^m \cup A_{\mathrm{C}}^m, \tag{22}$$

$$A_{\mathrm{A}}^m = \{(v_{00}^{0_m}, v_{i,s_{0i}+p_i}^{q_i^m}) \,|\, (v_{00}, v_{i,s_{0i}+p_i}) \in A_{\mathrm{A}}\}, \tag{23}$$

$$A_{\mathrm{B}}^m = \{(v_{j,t-s_{ji}-p_i}^{b-q_i^m}, v_{it}^b) \,|\, (v_{j,t-s_{ji}-p_i}, v_{it}) \in A_{\mathrm{B}},\ q_i^m + q_j^m \leq b \leq 1_m\}, \tag{24}$$

$$A_{\mathrm{C}}^m = \{(v_{jt}^{1_m}, v_{n+1,T_{\max}}^{1_m}) \,|\, (v_{jt}, v_{n+1,T_{\max}}) \in A_{\mathrm{C}}\}. \tag{25}$$

Then, (LR_2^m) is equivalent to the shortest path problem from $v_{00}^{0_m}$ to $v_{n+1,T_{\max}}^{1_m}$ on G^m under the 2-cycle elimination constraints. The time complexity of this problem is $O(n^2 2^m T_{\max})$ [35]. It can also be given by $O(|A^m|)$.

3.2. Network reduction

The primary bottleneck of the algorithm is heavy memory usage for storing the network structure (dynamic programming states). To reduce it, unnecessary nodes and arcs are removed from the network. Since the time complexities of (LR_2) and (LR_2^m) are given by $O(|A|)$ and $O(|A^m|)$, respectively, this network reduction also enables us to reduce the computation time. All the techniques in [36] are exploited except the one based on the dominance check that is inapplicable to our problem. It is because this reduction also requires that interchanging jobs does not affect the other jobs as the constraints on adjacent pairs of jobs described in the top of this section.

3.2.1. Network reduction by an upper bound

In this network reduction [38], a lower bound of the length of paths that pass through a node (an arc) is computed and the node (resp. the arc) is removed from the network if it is greater than or equal to an upper bound of the path length. This lower bound can be computed by applying dynamic programming in both forward and backward manners when the shortest path problem on the network is solved.

3.2.2. Network reduction by constraint propagation

The constraint propagation technique is applicable to reduce the size of the network. Several consistency tests have been proposed so far [42] and their extensions to the problem with setup times were also proposed [43, 44]. However, we simply apply the same consistency tests exploited in [36] by assuming the processing time of job i to be $p'_i = p_i + \min_{0 \leq j \leq n} s_{ji}$.

3.2.3. Network reduction by node compression

When the network structure is stored, successive nodes are compressed into one super-node. This contributes to improving a lower bound as well as reducing memory usage [36]. In [36], a maximum of four successive nodes are compressed into one super-node, while a maximum of six are in the proposed algorithm.

3.3. Upper bound computation

It is crucial to obtain a good upper bound for the algorithm because the efficiency of the network reduction in 3.2.1 highly depends on the tightness of an upper bound. To compute such an upper bound, a solution of a Lagrangian relaxation is converted to a feasible one by first removing duplicated jobs and next adding the unprocessed jobs greedily as in [36]. Then, it is improved by a local search. As this local search, the enhanced dynasearch [45, 46] and its extension to the problem with idle times [47] were employed in [35] and [36], respectively. Following these, we adopt the (extended) dynasearch in the proposed algorithm.

The dynasearch is a local search algorithm that adopts the dynasearch swap neighborhood, This neighborhood is defined by a set of solutions generated by applying pairwise interchanges (PIs) to the original solution. The number of these PIs is not restricted, but they should not intersect with each other. The primary advantage of this neighborhood is that the best solution in the neighborhood can be obtained in polynomial time for the problem without setup times nor idle time, although the neighborhood is composed of an exponential number of solutions. An extension of the dynasearch to the problem with sequence-dependent setup times is already proposed in [47]. Let us denote the current solution (job sequence) by

$\sigma(1)$, $\sigma(2)$, …, $\sigma(n)$ and let $\sigma(0) = 0$. Then, the recurrence equations of the dynamic programming to obtain the best solution in the dynasearch swap neighborhood are given as follows.

$$F(k,0,t) = \begin{cases} 0 & k = 0,\ t = 0, \\ +\infty & \text{otherwise}, \end{cases} \tag{26}$$

$$F(k,i,t) = \begin{cases} \min\limits_{0 \le j \le i-1} F(i-1, j, t - s_{\sigma(j)\sigma(k)} - P^{\mathrm{PI}}(k,i)) + F^{\mathrm{PI}}(k,i,t), \\ \qquad\qquad 1 \le i \le k \le n,\ 1 \le t \le T_{\max}, \\ +\infty, \qquad\qquad \text{otherwise}, \end{cases} \tag{27}$$

where the best objective value is $\min_{T_{\min} \le t \le T_{\max}} \min_{1 \le i \le n} F(n,i,t)$. Here, $s_{ii} = 0$,

$$P^{\mathrm{PI}}(k,i) = \sum_{j=i}^{k-1} s_{\sigma^{ki}(j)\sigma^{ki}(j+1)} + \sum_{j=i}^{k} p_{\sigma^{ki}(j)}, \tag{28}$$

$$F^{\mathrm{PI}}(k,i,t) = \sum_{j=i}^{k} f_{\sigma^{ki}(j)} \left(t - \sum_{l=j}^{k-1} s_{\sigma^{ki}(l)\sigma^{ki}(l+1)} - \sum_{l=j+1}^{k} p_{\sigma^{ki}(l)} \right), \tag{29}$$

and $\sigma^{ki}(j)$ is defined by

$$\sigma^{ki}(j) = \begin{cases} \sigma(k), & j = i, \\ \sigma(i), & j = k, \\ \sigma(j), & \text{otherwise}. \end{cases} \tag{30}$$

Since this recursion takes $O(n^4 T_{\max})$ time ($O(n^3 T_{\max})$ time even if the method in [48] is applied), it is a little too slow. Therefore, we propose a simpler extension where the *modified* dynasearch swap neighborhood is employed. On this neighborhood, an additional constraint is imposed that each PI should be separated by at least one job. For example, 4,2,3,1,7,6,5,8 belongs to the dynasearch swap neighborhood of 1,2,3,4,5,6,7,8, but does not to the modified dynasearch swap neighborhood. On the other hand, 4,2,3,1,5,8,7,6 belongs to both the neighborhoods because the PIs ($1 \leftrightarrow 4$ and $6 \leftrightarrow 8$) are separated by job 5. Let us assume that the $(n+1)$-th job in the current solution is a dummy job $n+1$ ($\sigma(n+1) = n+1$) such that $p_{n+1} = 0$, $f_{n+1}(t) \equiv 0$, and $s_{i,n+1} = 0$ for any i. Then, the best solution in the modified dynasearch swap neighborhood is given by $\min_{T_{\min} \le t \le T_{\max}} F'(n+1,t)$,

$$1, \underline{2,} 3, 4, 5, 6, \underline{7,} 8 \;\Rightarrow\; 1, \underline{7,} 3, 4, 5, 6, \underline{2,} 8$$

(a) PI (Pairwise Interchange)

$$1, \underline{2,} 3, 4, 5, 6, \underline{7,} 8 \;\Rightarrow\; 1, \underline{7, 2, 3, 4, 5, 6,} 8$$

(b) EBSR (Extraction and Backward Shifted Reinsertion)

$$1, \underline{2,} 3, 4, 5, 6, \underline{7,} 8 \;\Rightarrow\; 1, \underline{3, 4, 5, 6, 7, 2,} 8$$

(c) EFSR (Extraction and Forward Shifted Reinsertion)

$$1, \underline{2,} 3, 4, 5, 6, \underline{7,} 8 \;\Rightarrow\; 1, \underline{7, 6, 5, 4, 3, 2,} 8$$

(d) twist

$$1, \underline{2,} 3, 4, 5, 6, \underline{7,} 8 \;\Rightarrow\; 1, \underline{6, 7,} 4, 5, \underline{2, 3,} 8$$

(e) PI^2

$$1, \underline{2,} 3, 4, 5, 6, \underline{7,} 8 \;\Rightarrow\; 1, \underline{6, 7, 2, 3, 4, 5,} 8$$

(f) EBSR^2

$$1, \underline{2,} 3, 4, 5, 6, \underline{7,} 8 \;\Rightarrow\; 1, \underline{4, 5, 6, 7, 2, 3,} 8$$

(g) EFSR^2

$$1, \underline{2,} 3, 4, 5, 6, \underline{7,} 8 \;\Rightarrow\; 1, \underline{5, 6, 7, 2, 3, 4,} 8$$

(e) BI (Block Interchange)

Figure 1: Operators in the modified dynasearch neighborhood

where

$$F'(0,t) = \begin{cases} 0 & t = 0, \\ +\infty & \text{otherwise}, \end{cases} \tag{31}$$

$$F'(k,t) = \begin{cases} \min \Bigg[F'(k-1, t - s_{\sigma(k-1)\sigma(k)} - p_{\sigma(k)}) + f_{\sigma(k)}(t), \\ \min\limits_{0 \le i \le k-1} \left\{ F'(i-1, t - s_{\sigma(i-1)\sigma(k-1)} - P^{\mathrm{MPI}}(k,i)) + F^{\mathrm{MPI}}(k,i,t) \right\} \Bigg], \\ \qquad\qquad 1 \le k \le n+1,\ 1 \le t \le T_{\max}, \\ +\infty, \qquad\qquad \text{otherwise}, \end{cases} \tag{32}$$

and

$$P^{\mathrm{MPI}}(k,i) = \sum_{j=i}^{k-1} s_{\sigma^{k-1,i}(j)\sigma^{k-1,i}(j+1)} + \sum_{j=i}^{k} p_{\sigma^{k-1,i}(j)}, \tag{33}$$

$$F^{\mathrm{MPI}}(k,i,t) = \sum_{j=i}^{k} f_{\sigma^{k-1,i}(j)} \left(t - \sum_{l=j}^{k-1} s_{\sigma^{k-1,i}(l)\sigma^{k-1,i}(l+1)} - \sum_{l=j+1}^{k} p_{\sigma^{k-1,i}(l)} \right). \tag{34}$$

It follows that the time complexity reduces to $O(n^3 T_{\max})$ although the neighborhood becomes smaller. To compensate the smaller neighborhood size, we introduce several new operators as well as EBSR (Extraction and Backward Shifted Reinsertion) and EFSR (Extraction and Forward Shifted Reinsertion) [49] in the enhanced dynasearch [46], and twist in [48]. These new operators, PI^2, EBSR^2, EFSR^2 and BI (Block Interchange), are defined as in Figure 1. Preliminary experiments showed that this extension can obtain better solutions in shorter time than the direct extension in [47].

3.4. Outline of the Algorithm

Now, our proposed algorithm is summarized. The algorithm is composed of two stages. In the first stage, Lagrangian multipliers μ_i ($1 \in \mathscr{N}$) are adjusted for (LR_2). Next, in the second stage, (LR_2^m) is solved by increasing m.

3.4.1. Stage 1

An initial upper bound UB is computed by applying the dynasearch in 3.3 to a greedily obtained job sequence. The conjugate subgradient algorithm [50, 51] is applied to adjust Lagrangian multipliers of (LR_2). More specifically, the Lagrangian multipliers at the $(k+1)$th iteration, $\boldsymbol{\mu}^{(k+1)}$, are calculated by

$$\boldsymbol{\mu}^{(k+1)} = \boldsymbol{\mu}^{(k)} + \gamma^{(k)} \frac{\mathrm{UB} - \mathrm{LB}^{(k)}}{\|\boldsymbol{d}^{(k)}\|^2} \boldsymbol{d}^{(k)}, \tag{35}$$

where

$$\boldsymbol{d}^{(k)} = \boldsymbol{g}^{(k)} + \frac{\|\boldsymbol{g}^{(k)}\|}{\|\boldsymbol{d}^{(k-1)}\|} \boldsymbol{d}^{(k-1)}, \tag{36}$$

$$g_i^{(k)} = 1 - \mathscr{V}_i(\mathscr{P}^{(k)}), \tag{37}$$

$$\mathscr{P}^{(k)} = \arg\min_{\mathscr{P}} L_{\mathrm{R}}(\mathscr{P};\, \boldsymbol{\mu}^{(k)}), \tag{38}$$

$$\mathrm{LB}^{(k)} = L_{\mathrm{R}}(\mathscr{P}^{(k)};\, \boldsymbol{\mu}^{(k)}) + \sum_{i \in \mathscr{N}} \mu_i^{(k)}, \tag{39}$$

and $\gamma^{(k)}$ is the step size parameter, which is controlled as in [36]. In the course of the algorithm, upper bounds are computed by the method in 3.3 and the best upper bound UB is updated if necessary. The network reduction in 3.2.1 is performed every time when the best lower bound or UB is updated. If $\mathrm{UB} - \mathrm{LB}^{(\mathrm{k})} < 1$, the algorithm is terminated without entering Stage 2. The obtained multipliers and best lower bound in this stage are denoted by μ^{stage1} and $\mathrm{LB}^{\mathrm{stage1}}$, respectively.

3.4.2. Stage 2

We repeat Subprocedure($\mathrm{UB}^{\mathrm{tent}}$) with $\mathrm{UB}^{\mathrm{tent}}$ increased by Δ from $\mathrm{LB}^{\mathrm{stage1}}$ until $\mathrm{UB}^{\mathrm{tent}} = \mathrm{UB}$ ($\mathrm{UB}^{\mathrm{tent}}$ is rounded off to the nearest integer). Here, Δ is a parameter to determine the increment size of the tentative upper bound $\mathrm{UB}^{\mathrm{tent}}$.
Subprocedure($\mathrm{UB}^{\mathrm{tent}}$)**:**

(0) Let $\mathscr{M} := \emptyset$ and $\mathrm{LB} := \mathrm{LB}^{\mathrm{stage1}}$.

(1) Add a maximum of three jobs to $\mathscr{M}$ from $\mathscr{N} \setminus \mathscr{M}$, and (LR^m) for μ^{stage1} ($m = |\mathscr{M}|$) is solved by forward or backward dynamic programming in turns to compute the lower bound LB. All the network reduction techniques in 3.2 are performed, where $\mathrm{UB}^{\mathrm{tent}}$ is used instead of UB for the network reduction in 3.2.1.

(2) A new upper bound is searched for by the method in 3.3 if the best lower bound is updated. Update UB and $\mathrm{UB}^{\mathrm{tent}}$ if they are dominated by it.

(3) Stop if $\mathrm{UB}^{\mathrm{tent}} - \mathrm{LB} < 1$. Otherwise, go to (1).

4. Numerical Experiments I

In this section, the proposed algorithm is applied to the benchmark instances of $1|s_{ij}|\sum w_i T_i$ by Cicirello [8, 16][1]. In these instances with 60 jobs, processing times p_i are generated from the integer uniform distribution between 50 and 150, and the weights w_i from the integer uniform distribution between 0 and 10. The duedates d_i and the setup times s_{ij} are generated from integer uniform distributions by changing three parameters. The first is on the tightness of the duedates and there are three settings: “loose”, “moderate” and “tight”. The second is on the duedate range and has two settings: “loose” and “tight”. The last is on the setup

[1] Available from http://loki.stockton.edu/~cicirelv/benchmarks.html.

times and has two settings: "mild" and "severe." For every 12 combinations of the settings, 10 instances were generated. Therefore, the set consists of 120 instances.

The algorithm is coded in C (gcc) and we run it on a desktop computer with an Intel Core i7 980X Extreme Edition CPU (3.33GHz) and 24GB RAM. The maximum memory size for storing the network structure is restricted to 512MB. The parameter Δ in Stage 2 of the algorithm is set to 0.01UB by some preliminary experiments.

Before applying our proposed algorithm, the problem was formulated by a mixed-integer programming problem as in Appendix A and a general-purpose MIP solver, IBM ILOG CPLEX 12.1, was applied to 24 instances (Nos. 1, 6, ..., 116) out of the 120 instances. However, no instance could be solved within one day. This fact validates the necessity of constructing an exact algorithm for this problem.

Tables 1 and 2 summarize the results of our algorithm for the instances Nos. 41–120 ("moderate duedates" and "tight duedates"). In these tables, the optimal objective values and CPU times are given together with the best known objective values summarized in [28]. From these tables, we can see that optimal solutions are obtained for all the 80 instances in at most 320 seconds. It is also revealed that the current best objective values are already optimal for 62 out of 80 instances, but those for the two instances Nos. 69 and 116, which were given in [17], turned out to be incorrect. We verified that the solutions sent to us from the corresponding author of [17] do not yield the objective values in [17]. It seems that corrupted instance data was used in [17].

For the instances Nos. 1–40 ("loose duedates"), the conjugate subgradient algorithm in Stage 1 did not work well for adjusting Lagrangian multipliers and hence failed in obtaining good lower bounds. Therefore, the algorithm was terminated due to shortage of memory without finding optimal solutions except for the 17 instances. For these instances (Nos. 12, 21, 22, 23, 25, 26, 28, 29, 31, 32, 33, 34, 35, 36, 38, 39, 40) zero (and hence obviously optimal) solutions are obtained as initial upper bounds.

5. Improvement of the Algorithm

As described in the preceding section, 23 instances with "loose duedates" could not be solved by the proposed method due to shortage of memory. To solve them, the algorithm is improved so that a good lower bound is obtained in Stage 1 and that memory usage is reduced as much as possible.

Table 1: Computational Results for Cicirello's Benchmark Instances Nos. 41–80 (Moderate Due-dates)

No.	Best	Optimal	Time (s)	No.	Best	Optimal	Time (s)
41	**69102**	**69102**	86.52	61	**75916**	**75916**	97.78
42	**57487**	**57487**	100.11	62	**44769**	**44769**	82.03
43	**145310**	**145310**	155.39	63	**75317**	**75317**	75.17
44	35289	**35166**	126.43	64	**92572**	**92572**	93.77
45	**58935**	**58935**	155.92	65	**126696**	**126696**	97.09
46	**34764**	**34764**	144.62	66	**59685**	**59685**	40.76
47	**72853**	**72853**	149.97	67	**29390**	**29390**	58.17
48	**64612**	**64612**	205.29	68	**22120**	**22120**	56.96
49	77641	**77449**	134.73	69	64632*	**71118**	113.73
50	31292	**31092**	123.48	70	**75102**	**75102**	83.80
51	49761	**49208**	233.89	71	**145007**	**145007**	235.71
52	93106	**93045**	223.24	72	**43286**	**43286**	157.42
53	**84841**	**84841**	265.00	73	**28785**	**28785**	200.35
54	**118809**	**118809**	240.62	74	30136	**29777**	164.01
55	65400	**64315**	247.24	75	**21602**	**21602**	180.27
56	74940	**74889**	291.87	76	**53555**	**53555**	161.94
57	64552	**63514**	271.78	77	**31817**	**31817**	195.42
58	**45322**	**45322**	283.05	78	**19462**	**19462**	177.77
59	51649	**50999**	204.01	79	**114999**	**114999**	155.78
60	**60765**	**60765**	301.68	80	**18157**	**18157**	180.31

Bold: optimal solution, *: incorrect solution.

5.1. Removal of Zero Cost Jobs

In the Cicirello's benchmark instances, some jobs has the zero tardiness weight ($w_i = 0$). These jobs are removed as far as the problem structure is kept unchanged.

If job i with $w_i = 0$ satisfies

$$s_{ji} + p_i + s_{ik} \geq s_{jk} \quad j \in \{0\} \cup \mathscr{N},\ k \in \mathscr{N},\ j \neq i,\ j \neq k,\ k \neq i, \tag{40}$$

it can be moved to the last position in any solution without increasing the objective value. Hence we can ignore such a job without loss of optimality and the proposed algorithm is applied to the remaining jobs. This makes the the problem smaller and thus easier to solve.

Please note that (40) should always be satisfied from a practical point of view. Indeed, it is quite often the case for the problem with sequence-dependent setup

Table 2: Computational Results for Cicirello's Benchmark Instances Nos. 81–120 (Tight Due-dates)

No.	Best	Optimal	Time (s)	No.	Best	Optimal	Time (s)
81	**383485**	**383485**	71.02	101	**352990**	**352990**	133.18
82	**409479**	**409479**	149.37	102	**492572**	**492572**	130.15
83	**458752**	**458752**	97.84	103	**378602**	**378602**	102.38
84	**329670**	**329670**	108.49	104	**357963**	**357963**	134.51
85	**554766**	**554766**	179.19	105	**450806**	**450806**	92.60
86	**361417**	**361417**	173.87	106	**454379**	**454379**	138.62
87	**398551**	**398551**	109.23	107	**352766**	**352766**	90.57
88	**433186**	**433186**	155.25	108	**460793**	**460793**	87.82
89	**410092**	**410092**	109.08	109	**413004**	**413004**	131.40
90	**401653**	**401653**	141.64	110	**418769**	**418769**	142.07
91	**339933**	**339933**	152.97	111	**342752**	**342752**	232.68
92	**361152**	**361152**	233.03	112	**367110**	**367110**	268.30
93	404548	**403423**	289.54	113	**259649**	**259649**	240.20
94	332949	**332941**	220.14	114	**463474**	**463474**	238.46
95	517011	**516926**	236.01	115	457189	**456890**	311.72
96	457631	**455448**	176.50	116	527459*	**530601**	242.62
97	**407590**	**407590**	207.59	117	**502840**	**502840**	271.33
98	**520582**	**520582**	199.14	118	**349749**	**349749**	134.14
99	363977	**363518**	248.33	119	**573046**	**573046**	278.86
100	**431736**	**431736**	181.39	120	**396183**	**396183**	192.17

Bold: optimal solution, *: incorrect solution.

times to assume that the triangle inequality

$$s_{ji} + s_{ik} \geq s_{jk} \quad j \in \{0\} \cup \mathcal{N},\ k \in \mathcal{N},\ j \neq i,\ j \neq k,\ k \neq i \tag{41}$$

holds, which yields (40). However, some jobs in the Cicirello's benchmark instances break (40) and zero weight jobs cannot always be removed.

5.2. Column Generation

To avoid the failure in adjusting Lagrangian multipliers in Stage 1, the column generation algorithm is applied instead of the conjugate subgradient algorithm.

More specifically, multipliers are updated by solving the following problem.

$$(\mu^{(k+1)}, \lambda^{(k+1)}) = \arg\max_{\mu,\lambda} \sum_{i \in \mathcal{N}} \mu_i + \lambda, \tag{42}$$

$$\text{s.t.} \quad L(\mathscr{P}^{(j)}) - \sum_{i \in \mathcal{N}} \mu_i \mathscr{V}_i(\mathscr{P}^{(j)}) \geq \lambda, \quad 0 \leq j \leq k, \tag{43}$$

where $\mathscr{P}^{(0)}$ is the path corresponding to the initial upper bound. It is terminated if $\mathscr{P}^{(k)}$ satisfies

$$L_{\mathrm{R}}(\mathscr{P}^{(k)};\, \mu^{(k)}) = L(\mathscr{P}^{(k)}) - \sum_{i \in \mathcal{N}} \mu_i^{(k)} \mathscr{V}_i(\mathscr{P}^{(k)}) \geq \lambda^{(k)}. \tag{44}$$

Since this solves the Lagrangian dual corresponding to (LR_2) optimally, we can obtain the best Lagrangian multipliers that maximize the lower bound.

The problem (42)–(43) is written in the form of the cutting plane algorithm, but its dual is actually solved in the proposed algorithm. Therefore, we refer to it as column generation here. In this column generation algorithm, the dual stabilization technique in [52] is employed to speed up the convergence.

5.3. Iterated Dynasearch

To obtain a better upper bound, the modified dynasearch in 3.3 is employed in the framework of the iterated local search as in [45]. In this search, a locally optimal solution obtained by the dynasearch is perturbed by applying random pairwise interchanges α times (this is called "kick") and then the dynasearch is applied again. For every β applications of the dynasearch, the current locally optimal solution is replaced by the best solution obtained so far (this is called "backtrack") and it is perturbed by the kick. These iterations are terminated after γ applications of the dynasearch. In our algorithm, α, β and γ are set to 3, 2 and 100, respectively, by some preliminary experiments.

5.4. Interval Branching

It is true that the two improvements in the preceding subsections improve the algorithm to some extent, but the algorithm still terminates in Stage 2 due to shortage of memory. Therefore, we integrate a branching strategy into Stage 2.

If shortage of memory occurs in Subprocedure($\mathrm{UB}^{\mathrm{tent}}$), branching is performed by dividing the time window of a job into two intervals. For example, assume that the completion time of job 1 belongs to the interval $[\underline{a}_1, \overline{d}_1]$. That is,

$$\underline{a}_1 = \min_{v_{1t} \in V_{\mathrm{O}}} t, \quad \overline{d}_1 = \max_{v_{1t} \in V_{\mathrm{O}}} t, \tag{45}$$

where V_{O} is the node set of the network G at the start of Stage 2. Then, we calculate c_1 so that

$$|\{v_{1t} \,|\, v_{1t} \in V_{\mathrm{O}},\, t \leq c_1\}| \simeq |\{v_{1t} \,|\, v_{1t} \in V_{\mathrm{O}},\, t > c_1\}| \tag{46}$$

is satisfied. In other words, the numbers of the occurrences of job 1 in the intervals $[\underline{a}_1, c_1]$ and $[c_1+1, \overline{d}_1]$, respectively, balance with each other. Next, two networks G_{L1} and G_{U1} are generated from G by restricting the occurrence of job 1 in $[\underline{a}_1, c_1]$ and $[c_1+1, \overline{d}_1]$, respectively. Then, Subprocedure($\mathrm{UB}^{\mathrm{tent}}$) is performed for both G_{L1} and G_{U1} separately, instead of Subprocedure($\mathrm{UB}^{\mathrm{tent}}$) for G. Although the former requires a longer computation time, the memory usage reduces.

The job with the largest number of occurrences in G is selected first and two jobs are selected at once. Hence four networks are generated at the first level of the search tree. This branching is performed recursively when Subprocedure($\mathrm{UB}^{\mathrm{tent}}$) is terminated due to shortage of memory. For one value of $\mathrm{UB}^{\mathrm{tent}}$, the maximum depth of the search tree is recorded and it is used for the next value of $\mathrm{UB}^{\mathrm{tent}}$. If, for example, the maximum depth is two for one value of $\mathrm{UB}^{\mathrm{tent}}$, 16 networks (corresponding to four selected jobs) are generated from the start for the next value of $\mathrm{UB}^{\mathrm{tent}}$.

6. Numerical Experiments II

The improved algorithm is applied to the Cicirello's instances Nos. 1–40 ("loose duedates"). To solve the dual of (42)–(43) in the column generation algorithm, lpsolve [53] is used. Since shortage of memory occurs in Stage 2, three settings of the maximum memory size are considered: 512MB, 2GB and 20GB. For the instances with $\mathrm{LB}^{\mathrm{stage1}} = 0$, Δ in Stage 2 is set to 1 to concentrate on finding a solution with the zero objective value.

The results are summarized in Table 3. For instances Nos. 18 and 24, we gave up applying the algorithm with 512MB or 2GB memory size because the search tree became too deep in Stage 2. Although it took too long CPU times for the two (2 weeks and 1 month, respectively, even with 20GB memory size), all the instances were solved optimally. Again, the best objective value for the instance No. 6 given in [17] turned out to be incorrect. It seems that the instances with loose duedates are hard also for the existing metaheuristic approaches because no optimal solutions are obtained by them except for those with the zero optimal objective value. The CPU time can decrease by increasing the memory size for those instances where shortage of memory occurs and hence branching should

be performed in Stage 2 (eg. No. 11). However, it is observed that the CPU time rather increased for some instances (eg. No. 17). This is because the CPU time also depends on when a good upper bound is obtained in the course of the algorithm. For the instances, the algorithm with a smaller memory size happened to find a good upper bound earlier.

Next, the improved algorithm is applied to the instances of $1|s_{ij}|\sum T_i$. Two sets of instances are used here: the set of instances with 15, 25, 35 and 45 jobs in [2][2] and that with 55, 65, 75 and 85 jobs in [7][3]. Hereafter, these are referred to as Rubin's instances and Gagné's instances, respectively.

The results are summarized in Tables 4 and 5. In Table 4, the shorter CPU times of the two branch-and-bound algorithms by Bigras et al. [31] are also presented for comparison. For prob603, the CPU time of a general MILP solver (CPLEX) reported in [31] is given because those of the branch-and-bound algorithms were not reported in [31]. Please note that the computer in this study is 2 or 2.5 times as fast as a 3.4GHz Pentium4 computer in [31]. Nevertheless, from Table 4 we can verify that our algorithm is much faster for the instances with 25 jobs or more, except prob705, even if the difference of the CPU speed is taken into account. Larger instances with $n \geq 55$ can be solved by the proposed algorithm as in Table 5 except prob851 and prob855. The instance prob751 could be solved optimally by the algorithm with 20GB memory size, but the two instances could not be. However, the best objective values of these instances would be almost optimal because the algorithm ensured that optimal objective values of prob851 and prob855 are not better than 357 and 254, respectively.

7. Conclusion

In this study we proposed an exact algorithm for the single-machine total weighted tardiness problem with sequence-dependent setup times based on our previous algorithms for the problem without setup times. Then, the proposed algorithm was applied to well-known benchmark instances and almost all of them were optimally solved. To the best of the authors' knowledge, this is the first attempt to solve the Cicirello's instances and the Gagné's instances optimally. However, it still takes a very long computation time. To reduce it, we should improve the lower bound by, for example, introducing effective cuts such as those proposed in [52]. It is left for future research.

[2]Available from https://www.msu.edu/~rubin/files/research.html.

[3]Available from http://depcom.uqac.ca/~c3gagne/home_fichiers/ProbOrdo.htm.

[1] Allahverdi A, Gupta JND, Aldowaisan T. A review of scheduling research involving setup considerations. OMEGA 1999;27:219–239.

[2] Rubin PA, Ragatz GL. Scheduling in a sequence dependent setup environment with genetic search, Computers & Operations Research 1995;22:85–99.

[3] Lee YH, Bhaskaran K, Pinedo M. A heuristic to minimize the total weighted tardiness with sequence-dependent setups, IIE Transactions 1997;29:45–52.

[4] Tan KC, Narasimhan R. Minimizing tardiness on a single processor with sequence-dependent setup times: a simulated annealing approach, Omega 1997;25:619–634.

[5] Tan KC, Narasimhan R, Rubin PA, Ragatz GL. Comparison of four methods for minimizing total tardiness on a single processor with sequence dependent setup times, Omega 2000;28:313–326.

[6] França PM, Mendes A, Moscato P. A memetic algorithm for the total tardiness single machine scheduling problem. European Journal of Operational Research 2001;132:224–242.

[7] Gagné C, Price WL, Gravel M. Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times. Journal of the Operational Research Society 2002;53:895–906.

[8] Cicirello VA. Weighted tardiness scheduling with sequence-dependent setups: A benchmark library. Technical Report of Intelligent Coordination and Logistics Laboratory, The Robotics Institute, Carnegie Mellon University, USA. 2003.

[9] Cicirello VA, Smith SF. Enhancing stochastic search performance by value-biased randomization of heuristics. Journal of Heuristics 2005;11:5–34.

[10] Gagné C, Gravel M, Price WL. Using metaheuristic compromise programming for the solution of multi-objective scheduling problems, Journal of the Operational Research Society 2005;56:687–698.

[11] Cicirello VA. Non-wrapping order crossover: An order preserving crossover operator that respects absolute position. Proceedings of GECCO'06 2006:1125–1131.

[12] Gupta SR, Smith JS. Algorithms for single machine total tardiness scheduling with sequence dependent setups, European Journal of Operational Research 2006;175:722–739.

[13] Anghinolfi D, Paolucci M. A new ant colony optimization approach for the single machine total weighted tardiness scheduling problem. International Journal of Operations Research 2008;5:44–60.

[14] Anghinolfi D, Paolucci M. A new discrete particle swarm optimization approach for the single-machine total tardiness scheduling problem with sequence-dependent setup times. European Journal of Operational Research 2009;193:73–85.

[15] Bożejko W, Wodecki M. A parallel metaheuristics for the single machine total weighted tardiness problem with sequence-dependent setup times. Proceedings of 3rd Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2007) 2007:96–103.

[16] Cicirello VA. The challenge of sequence-dependent setups: Proposal for a scheduling competition track on one machine sequencing problems. International Workshop on Scheduling a Scheduling Competition 2007.

[17] Liao C-J, Juan H-C. An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups. Computers & Operations Research 2007;34:1899–1909.

[18] Lin S-W, Ying K-C. A hybrid approach for single-machine tardiness problems with sequence-dependent setup times. Journal of the Operational Research Society 2007;58:1–11.

[19] Lin S-W, Ying K-C. Solving single-machine total weighted tardiness problems with sequence-dependent setup times by meta-heuristics. International Journal of Advanced Manufacturing and Technology 2007;34:1183–1190.

[20] Valente JMS, Alves RAFS. Beam search algorithms for the single machine total weighted tardiness scheduling problem with sequence-dependent setups. Computers & Operations Research 2008;35:2388–2405.

[21] Tasgetiren MF, Pan Q-K, Liang Y-C. A discrete differential evolution algorithm for the single machine total weighted tardiness problem with sequence

dependent setup times. Computers & Operations Research 2009;36:1900–1915.

[22] Ying K-C, Lin S-W, Huang C-Y. Sequencing single-machine tardiness problems with sequence dependent setup times using an iterated greedy heuristic. Expert Systems with Applications 2009;36:7087–7092.

[23] Arroyo JEC, Nunes GVP, Kamke EH. Iterative local search heuristic for the single machine scheduling problem with sequence dependent setup times and due dates. Proceedings of 9th International Conference on Hybrid Intelligent Systems (HIS'09) 2009:505–510.

[24] Luo J-X, Liu H-M, Yuan P. A new filter and fan algorithm with kick strategy for single-machine tardiness scheduling with sequence-dependent setups. Proceedings of IEEE International Conference on Intelligent Computing and Intelligent Systems (ICIS 2009) 2009;1:438–442.

[25] Bożejko W. Parallel path relinking method for the single machine total weighted tardiness problem with sequence-dependent setups. Journal of Intelligent Manufacturing 2010;21:777–785.

[26] Akrout H, Jarboui B, Siarry P, Rebaï A. A GRASP based on DE to solve single machine scheduling problem with SDST. Computational Optimization and Applications, available online.

[27] Armadizar F, Hosseini L. A novel ant colony algorithm for the single-machine total weighted tardiness problem with sequence dependent setup times, International Journal of Computational Intelligent Systems 2011;4:456–466.

[28] Kirlik G, Oguz C. A variable neighborhood search for minimizing total weighted tardiness with sequence dependent setup times on a single machine, Computers & Operations Research, available online. DOI:10.1016/j.cor.2011.08.022

[29] Ragatz GL. A branch-and-bound method for minimum tardiness sequencing on a single processor with sequence dependent setup times. Proceedings of the Twenty-Fourth Annual Meeting of the Decision Sciences Institute 1993:1375–1377.

[30] Luo X, Chu F. A branch and bound algorithm of the single machine schedule with sequence dependent setup times for minimizing total tardiness. Applied Mathematics and Computation 2006;183:575–588.

[31] Bigras L-P, Gamache M, Savard G. The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. Discrete Optimization 2008;5:685–699.

[32] Rabadi G, Mollaghasemi M, Anagnostopoulos GC. A branch-and-bound algorithm for the early/tardy machine scheduling problem with a common due-date and sequence-dependent setup time. Computers & Operations Research 2004;31:1727–1751.

[33] Sourd F. Earliness-tardiness scheduling with setup considerations. Computers & Operations Research 2005;32:1849–1865.

[34] Luo X, Chu C. A branch-and-bound algorithm of the single machine schedule with sequence-dependent setup times for minimizing maximum tardiness. European Journal of Operational Research 2007;180:68–81.

[35] Tanaka S, Fujikuma S, Araki M. An exact algorithm for single-machine scheduling without machine idle time, Journal of Scheduling 2009;12:575–593.

[36] 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.

[37] Ibaraki T. Enumerative approaches to combinatorial optimization. Annals of Operations Research 1987;10 and 11.

[38] Ibaraki T, Nakamura Y. A dynamic programming method for single machine scheduling. European Journal of Operational Research 1994;76:72–82.

[39] Kuhn HW. The Hungarian method for the assignment problem. Naval Research Logistics Quarterly 1955;2:83–97.

[40] Sourd F. New exact algorithms for one-machine earliness-tardiness scheduling. INFORMS Journal on Computing 2009;21:167–175.

[41] Christofides N, Mingozzi A, Toth P. Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations, Mathematical Programming 1981;20:255–282.

[42] Baptiste P, Le Pape C, Nuijten W. Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems. Dordrecht, Netherlands: Kluwer Academic Publishers; 2001:21–29.

[43] Brucker P, Thiele O. A branch & bound method for the general-shop problem with sequence dependent setup-times. OR Spektrum 1996;18:145–161.

[44] Artigues C, Feillet D. A branch and bound method for job-shop problem with sequence-dependent setup times. Annals of Operations Research 2008;159:135–159.

[45] Congram RK, Potts CN, van de Velde SL. An iterated dynasearch algorithm for the single machine total weighted tardiness scheduling problem. INFORMS Journal on Computing 2002;14:52–67.

[46] Grosso A, Della Croce F, Tadei R. An enhanced dynasearch neighborhood for the single machine total weighted tardiness scheduling problem. Operations Research Letters 2004;32:68–72.

[47] Sourd F. Dynasearch for the earliness-tardiness scheduling problem with release dates and setup constraints. Operations Research Letters 2006;34:591–598.

[48] Ergun Ö, Orlin JB. Fast neighborhood search for the single machine total weighted tardiness problem, Operations Research Letters 2006;34:41–45.

[49] Della Croce F. Generalized pairwise interchanges and machine scheduling. European Journal of Operational Research 1995;83:310–319.

[50] Sherali HD, Ulular O. A primal-dual conjugate subgradient algorithm for specially structured linear and convex programming problems, Applied Mathematics and Optimization 1989;20:193–221.

[51] Sherali HD, Lim C. Enhancing Lagrangian dual optimization for linear programs by obviating nondifferentiability, INFORMS Journal on Computing 2007;19:3–13.

[52] Pessoa A, Uchoa E, Poggi de Aragăo M, Rodrigues R. Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems, Mathematical Programming Computation 2010;2:259–290.

[53] lpsolve: http://sourceforge.net/projects/lpsolve.

Appendix A. Naive Mixed-Integer Programming Formulation

Our problem is formulated as the following mixed-integer programming problem to apply a general-purpose MIP solver:

$$
\begin{aligned}
(\mathrm{MIP}): \min_{y,C,T} & \sum_{1\le i\le n} w_i T_i, && && \text{(A.1)}\\
\text{s.t.} & \sum_{1\le j\le n} y_{0j} = 1, && && \text{(A.2)}\\
& \sum_{1\le i\le n} y_{i,n+1} = 1, && && \text{(A.3)}\\
& \sum_{\substack{0\le i\le n\\ i\ne j}} y_{ij} = 1, && 1\le j\le n, && \text{(A.4)}\\
& \sum_{\substack{1\le j\le n+1\\ j\ne i}} y_{ij} = 1, && 1\le i\le n, && \text{(A.5)}\\
& C_j \ge (p_j + s_{0j})y_{0j} - M(1-y_{0j}), && 1\le i\le n, && \text{(A.6)}\\
& C_j \ge C_i + (p_j + s_{ij})y_{ij} - M(1-y_{ij}), && 1\le i,j\le n,\ i\ne j, && \text{(A.7)}\\
& T_i \ge C_i - d_i, && 1\le i\le n, && \text{(A.8)}\\
& y_{0j}, y_{i,n+1}, y_{ij} \in \{0,1\},\ C_i \ge 0,\ T_i \ge 0, && && \text{(A.9)}
\end{aligned}
$$

where y_{ij} is a binary decision variable such that $y_{ij} = 1$ if and only if job j is the immediate successor of job i. In addition, M is a sufficiently large integer and is chosen as $M = T_{\max}$.

Table 3: Computational Results for Cicirello's Benchmark Instances Nos. 1–40 (Loose Duedates)

No	Best	Optimal	Time (s)		
			512MB	2GB	20GB
1	471	**453**	221.79	220.63	192.82
2	4878	**4794**	3073.00	1781.17	2895.27
3	1430	**1390**	1812.49	1060.62	1126.00
4	5946	**5866**	467.23	333.16	270.83
5	4084	**4054**	7259.98	4198.60	3292.41
6	5788*	**6592**	293.22	322.82	252.42
7	3330	**3267**	7699.39	4815.34	7600.50
8	108	**100**	220.54	218.62	187.78
9	5751	**5660**	281.37	274.78	255.59
10	1789	**1740**	15454.09	9475.19	9185.96
11	2998	**2785**	123538.53	51443.47	23359.73
12	**0**	**0**	0.93	0.93	0.91
13	4068	**3904**	19599.13	10677.42	8212.44
14	2260	**2075**	15645.10	5049.56	5906.87
15	935	**724**	4404.47	1632.86	841.15
16	3381	**3285**	830.73	585.89	688.30
17	**0**	**0**	181.28	181.12	7268.65
18	845	**767**	—	—	2 weeks
19	**0**	**0**	7.08	7.08	6.41
20	2053	**1757**	1094.24	713.03	477.46
21	**0**	**0**	0.66	0.66	0.63
22	**0**	**0**	0.75	0.75	0.71
23	**0**	**0**	0.58	0.58	0.56
24	920	**761**	—	—	30 days
25	**0**	**0**	0.70	0.70	0.67
26	**0**	**0**	0.70	0.70	0.67
27	**0**	**0**	1.10	1.10	1.05
28	**0**	**0**	1.05	1.05	1.00
29	**0**	**0**	0.70	0.70	0.67
30	**0**	**0**	17.86	17.86	16.30
31	**0**	**0**	0.92	0.92	0.90
32	**0**	**0**	1.02	1.02	0.99
33	**0**	**0**	1.01	1.01	0.97
34	**0**	**0**	0.99	0.99	0.96
35	**0**	**0**	0.94	0.94	0.91
36	**0**	**0**	0.89	0.89	0.87
37	46	**0**	1209.84	1196.33	3707.70
38	**0**	**0**	0.90	0.90	0.88
39	**0**	**0**	0.98	0.98	0.96
40	**0**	**0**	0.96	0.96	0.93

Bold: optimal solution, *: incorrect solution.

Table 4: Computational Results for Rubin's Benchmark Instances

Name	n	Optimal	Time (s)			
			Bigras et al. [31]	512MB	2GB	20GB
prob401	15	**90**	4	0.56	0.56	0.49
prob402	15	**0**	1	0.01	0.01	0.01
prob403	15	**3418**	2	1.00	1.00	0.86
prob404	15	**1067**	1	0.82	0.82	0.74
prob405	15	**0**	1	0.01	0.01	0.01
prob406	15	**0**	1	0.01	0.01	0.01
prob407	15	**1861**	4	0.90	0.90	0.78
prob408	15	**5660**	5	1.55	1.55	1.37
prob501	25	**261**	56	4.29	4.29	3.64
prob502	25	**0**	13	0.03	0.03	0.03
prob503	25	**3497**	55	6.14	6.14	5.24
prob504	25	**0**	12	0.05	0.05	0.05
prob505	25	**0**	7	0.04	0.04	0.04
prob506	25	**0**	9	0.04	0.04	0.04
prob507	25	**7225**	220	8.85	8.85	7.65
prob508	25	**1915**	243	7.69	7.69	7.06
prob601	35	**12**	473	14.53	14.53	13.68
prob602	35	**0**	40	0.09	0.09	0.09
prob603	35	**17587**	(749)	39.95	39.93	34.16
prob604	35	**19092**	7613	53.67	53.68	46.69
prob605	35	**228**	828	30.57	30.57	27.41
prob606	35	**0**	71	0.10	0.10	0.10
prob607	35	**12969**	78768	38.96	38.96	34.31
prob608	35	**4732**	47787	66.89	66.86	61.85
prob701	45	**97**	3400	108.89	135.35	96.51
prob702	45	**0**	374	0.19	0.19	0.18
prob703	45	**26506**	109730	117.27	117.47	100.86
prob704	45	**15206**	18922	152.58	152.45	133.20
prob705	45	**200**	2636	3000.67	1814.05	3536.64
prob706	45	**0**	234	0.22	0.22	0.21
prob707	45	**23789**	630085	118.96	118.91	105.56
prob708	45	**22807**	170879	214.46	214.46	196.18

Bold: optimal solution.

Table 5: Computational Results for Gagné's Benchmark Instances

Name	n	Optimal	Time (s)		
			512MB	2GB	20GB
prob551	55	**183**	354.67	389.65	285.92
prob552	55	**0**	0.35	0.35	0.32
prob553	55	**40498**	255.42	252.59	216.96
prob554	55	**14653**	587.09	586.88	529.32
prob555	55	**0**	0.60	0.60	0.56
prob556	55	**0**	0.37	0.37	0.35
prob557	55	**35813**	310.07	311.56	280.85
prob558	55	**19871**	493.17	492.76	446.43
prob651	65	**247**	12045.00	3441.09	2872.06
prob652	65	**0**	0.50	0.50	0.47
prob653	65	**57500**	542.40	536.89	467.16
prob654	65	**34301**	838.96	838.10	759.70
prob655	65	**0**	679.37	1032.64	2596.73
prob656	65	**0**	0.60	0.60	0.58
prob657	65	**54895**	569.81	570.36	515.91
prob658	65	**27114**	959.32	960.00	880.03
prob751	75	**225**	—	—	34 days
prob752	75	**0**	0.88	0.88	0.84
prob753	75	**77544**	972.78	971.74	883.43
prob754	75	**35200**	1861.19	1861.12	1685.31
prob755	75	**0**	0.89	0.89	0.86
prob756	75	**0**	1.05	1.05	1.01
prob757	75	**59635**	2777.17	2908.40	3800.11
prob758	75	**38339**	2022.47	2022.26	1865.16
prob851	85	*360*	—	—	$>$ 30 days
prob852	85	**0**	1.23	1.23	1.16
prob853	85	**97497**	15833.01	5023.96	1651.31
prob854	85	**79042**	2499.42	2451.33	2198.15
prob855	85	*258*	—	—	$>$ 30 days
prob856	85	**0**	1.30	1.30	1.26
prob857	85	**87011**	6117.44	8156.66	6629.99
prob858	85	**74739**	3983.78	3951.11	3561.79

Bold: optimal solution, *italic*: best solution