Dual Constrained Single Machine Sequencing to Minimize Total Weighted Completion Time

Yunpeng Pan and Leyuan Shi, *Member, IEEE*

***Abstract*—We study a single-machine sequencing problem with both release dates and deadlines to minimize the total weighted completion time. We propose a branch-and-bound algorithm for this problem. The algorithm exploits an effective lower bound and a dynamic programming dominance technique. As a byproduct of the lower bound, we have developed a new algorithm for the generalized isotonic regression problem; the algorithm can also be used as an $O(n \log n)$-time timetabling routine in earliness–tardiness scheduling. Extensive computational experiments indicate that the proposed branch-and-bound algorithm competes favorably with a dynamic programming procedure.**

***Note to Practitioners*—Real-life production systems usually involve multiple machines and resources. The configurations of such systems may be complex and subject to change over time. Therefore, model-based solution approaches, which aim to solve scheduling problems for specific configurations, will inevitably run into difficulties. By contrast, decomposition methods are much more expressive and extensible. The single-machine problem and its solution procedure studied in this paper will prove useful to a decomposition method that decomposes multiple-machine, multiple-resource scheduling problems into a number of single-machine problems. The total weighted completion time objective is relevant to production environments where inventory levels and manufacturing cycle times are key concerns. Future research can be pursued along two directions. First, it seems to be necessary to further generalize the problem to consider also negative job weights. Second, the solution procedure developed here is ready to be incorporated into a machine-oriented decomposition method such as the shifting bottleneck procedure.**

***Index Terms*—Branch and bound, isotonic regression, scheduling, timetabling, weighted completion time (WCT).**

I. Introduction

SCHEDULING problems with the total weighted time completion objective first caught our attention during an industrial project to develop algorithms for scheduling job shop operations. In the job shop environment that we modeled, the central managerial goal is to sustain production with the minimum level of work-in-process and finished goods inventories while delivering jobs on time. Ideally, earliness–tardiness objectives would be desirable for this kind of application, but no practical solution procedure was foreseeable. Therefore, we introduced a more computationally tractable alternative called the *job shop total inventory minimization problem* (JSTIMP) [32].

As a critical step in developing a solution procedure for JSTIMP, we needed to solve a single-machine sequencing problem that minimizes the total weighted time completion objective, subject to dual constraints, i.e., both release dates and deadlines. This problem also serves as a relaxation for other job shop problems with flow time or completion time objectives. We formalize the problem next.

An instance of our sequencing problem comprises n 4-tuples: $\langle r_j, p_j, \bar{d}_j, w_j \rangle$, $j \in \mathcal{J}$, where $\mathcal{J} = \{1, \ldots, n\}$ designates jobs to be scheduled on a machine, and $r_j(\geq 0)$, $p_j(> 0)$, $\bar{d}_j(\geq r_j + p_j)$, and $w_j(\geq 0)$ are the release date, processing time, deadline, and associated weight, respectively. Job $j \in \mathcal{J}$ is ready to start at time r_j; once started, the job will occupy the machine exclusively for the duration of p_j without interruption. The completion time of job j is denoted by C_j, and the collection, $\{C_j | j \in \mathcal{J}\}$, is called a *schedule*. A schedule that satisfies all the release dates and deadlines is said to be *feasible*. The objective is to find a feasible schedule that minimizes the objective function $\sum_{j=1}^{n} w_j C_j$. This problem, denoted by $1|r_j, \bar{d}_j| \sum w_j C_j$ with the three-field notation [19], can be stated mathematically as follows:

$$\min \sum_{j=1}^{n} w_j C_j$$

$$(\mathbf{WCT})\ \text{s.t.}\ r_j + p_j \leq C_j \qquad \forall j \tag{1}$$

$$C_j \leq \bar{d}_j \qquad \forall j \tag{2}$$

$$C_i \leq C_j - p_j \text{ or } C_j \leq C_i - p_i \qquad \forall i, j, i \neq j. \tag{3}$$

Even the problem of satisfying constraints (1)–(3) is NP-complete [16], but it can be efficiently dealt with in practice [8] and [27]. In the literature, only some special cases of the weighted completion time (WCT) have been considered thus far. $1|\bar{d}_j| \sum w_j C_j$ has been studied by numerous authors, including [14], [29], [30], [33]. Ahmadi and Bagchi [1] and Chand and Schneeberger [10] investigate enumerative procedures for $1|\bar{d}_j, nowait| \sum w_j E_j$, where $E_j = \bar{d}_j - C_j$ is the earliness. Bianco and Ricciardelli [7], Hariri and Potts [20], and Belouadah *et al.* [6] examine $1|r_j| \sum w_j C_j$. Dell'Amico *et al.* [13] extend the job splitting idea of Posner [29] and apply it to the static problem with general weights, while the same problem was also studied by Bard *et al.* [5]. None of the above authors consider both release dates and deadlines.

Manuscript received August 31, 2004; revised November 26, 2004 and January 30, 2005. This paper was recommended for publication by Associate Editor C. Teo and Editor N. Viswanadham upon evaluation of the reviewers' comments.This work was supported in part by the National Science Foundation under Grant DMI-0100220, Grant DMI-0217924, and Grant DMI-0431227, in part by the Air Force Office of Scientific Research under Grant FA9550-04-1-0179, and in part by John Deere Horicon Works.

The authors are with the Department of Industrial & Systems Engineering, University of Wisconsin-Madison, Madison, WI 53706 USA (e-mail: pany@cae.wisc.edu; leyuan@engr.wisc.edu).

Digital Object Identifier 10.1109/TASE.2005.853474

While they model the dual-constrained situation through a flow shop problem, Ahmadi and Bagchi [2] nevertheless assume nonconstraining release dates in their solution approach. Only recently has a solution procedure for the general problem, i.e., WCT, been proposed by Gélinas and Soumis [18] who use dynamic programming and consider general weights. Apart from [18], there are few relevant results for WCT. Neither lower bounds nor branch-and-bound algorithms have been proposed for WCT in the scheduling literature.

In this paper, we propose a branch-and-bound algorithm for solving this problem. In Section II, we derive a new lower bound and then reduce the task of computing the lower bound to one of solving a linear programming problem that has special structure. The problem is shown to be an isotone optimization problem. In Section III, we discuss the preliminaries of isotone optimization problems and propose a new algorithm for the generalized isotonic regression problem with immediate application to our lower bound calculation. Section IV discusses dominance conditions and a new dynamic programming dominance technique. The branch-and-bound algorithm is presented in Section V and then compared with a dynamic programming algorithm through extensive computational experiments in Section VI. Some concluding remarks are given in Section VII. We begin by deriving a lower bound.

II. Lower Bound

Our lower bounding technique utilizes Lagrangian relaxation and the multiplier adjustment method first introduced by van Wassenhove [39]. This lower bound can be viewed as a synthesis and extension of those of Hariri and Potts [20] and Potts and van Wassenhove [30].

A. Derivation

Two notions are necessary for our subsequent discussion. First, a job sequence σ is called a *nondelay sequence* if it implies a *nondelay schedule*, where the machine is never kept idle while some job is waiting to be processed [4]. It should be noted that σ is generated using only the release dates and processing times, and therefore, may violate some of the deadlines. This does not cause any problems since σ merely serves as a stepping stone toward computing the lower bound. Renumber the jobs such that $\sigma = (1, \ldots, n)$. Second, we adopt the notion of *block* [23]. Job v is called the *last job of a block* if $C_v \leq r_i$ for $i = v+1, \ldots, n$. Let $B = \{u, \ldots, v\}$ be a set of jobs in adjacent positions of σ. B is called a *block* if (*i*) job $u-1$ (provided that $u > 1$) and job v are the last jobs of two consecutive blocks, and (*ii*) job i is not the last job of a block for $i = u, \ldots, v-1$. Thus, the nondelay sequence σ can be partitioned into a number of blocks.

Omitting the deadlines in WCT yields $1|r_j|\sum w_jC_j$. The nondelay sequence σ is optimal for $1|r_j|\sum w_jC_j$ if the following condition by Hariri and Potts [20] is satisfied.

Proposition 1: The nondelay sequence σ is optimal for $1|r_j|\sum w_jC_j$ if the jobs within each block $B_k (k = 1, \ldots, K)$ are sequenced in nonincreasing order of w_j/p_j.

To derive a lower bound for our problem, we perform a Lagrangian relaxation of each release date constraint (1) and each deadline constraint (2). The resulting Lagrangian problem is denoted by $\mathbf{LR}$, and the value of the Lagrangian is

$$\begin{aligned} L(\mathbf{a},\mathbf{b}) &= \mathbf{min} \sum_{j=1}^{n} w_jC_j + a_j(r_j + p_j - C_j) + b_j(C_j - \bar{d}_j) \\ &= \sum_{j=1}^{n}(w_j - a_j + b_j)C_j + \sum_{j=1}^{n} a_j(r_j + p_j) - b_j\bar{d}_j \\ &\quad \mathbf{s.t.}(1), (3) \end{aligned}$$

where $\mathbf{a} = (a_1, \ldots, a_n)$ and $\mathbf{b} = (b_1, \ldots, b_n)$ are vectors of nonnegative multipliers. Note that we have purposefully retained the release date constraints (1) in $\mathbf{LR}$.

It is easy to see that with $\mathbf{a}$ and $\mathbf{b}$ fixed, $\mathbf{LR}$ (after ignoring the constant terms in the objective function) is of the same form as $1|r_j|\sum w_jC_j$, but each job j has a new weight

$$w'_j = w_j - a_j + b_j. \tag{4}$$

For any choice of multipliers satisfying $\mathbf{a} \geq \mathbf{0}$ and $\mathbf{b} \geq \mathbf{0}$, $L(\mathbf{a},\mathbf{b})$ is a lower bound on the optimal value of our problem WCT.

To find proper values for our multipliers, we proceed as follows. First, we can immediately rule out those multiplier values that cause any w'_j to be negative, as it is shown next that their $L(\mathbf{a},\mathbf{b})$ values are always dominated. Consider a particular choice of multipliers denoted by $(\mathbf{a},\mathbf{b})$ such that $w'_{j_0} = w_{j_0} - a_{j_0} + b_{j_0} < 0$ for some j_0. Now define new multipliers $(\hat{\mathbf{a}},\hat{\mathbf{b}})$ that are identical to $(\mathbf{a},\mathbf{b})$ except that $\hat{a}_{j_0} = a_{j_0} + w'_{j_0}$. By construction, $\hat{w}'_{j_0} = w_{j_0} - \hat{a}_{j_0} + \hat{b}_{j_0} = 0$, and the new multipliers are valid since $\hat{a}_{j_0} = a_{j_0} + w'_{j_0} = w_{j_0} + b_{j_0} \geq 0$. Also, for any fixed schedule $\{C_j | j \in \mathcal{J}\}$ that satisfies (1) and (3), redefining a_{j_0} results in an increase of $w'_{j_0}(r_{j_0} + p_{j_0} - C_{j_0})$ (note that $w'_{j_0} < 0$, $r_{j_0} + p_{j_0} - C_{j_0} \leq 0$). Minimize over all such schedules and we have $L(\mathbf{a},\mathbf{b}) \leq L(\hat{\mathbf{a}},\hat{\mathbf{b}})$. This is repeated until all $w'_j \geq 0$.

Second, we impose an additional restriction on the multipliers so that the Lagrangian problem can be solved easily by applying Proposition 1. Assume that σ is a given nondelay sequence with K blocks, $B_1, \ldots, B_K$, and that the jobs are renumbered such that $\sigma = (1, \ldots, n)$. Based on the above two considerations, we restrict our choice of multipliers to those that satisfy

$$w'_j \geq 0, \qquad j = u_k, \ldots, v_k \tag{5}$$

$$\frac{w'_j}{p_j} \geq \frac{w'_{j+1}}{p_{j+1}}, \qquad j = u_k, \ldots, v_k - 1 \tag{6}$$

for each block $B_k = \{u_k, \ldots, v_k\}$. If the multipliers $\mathbf{a}$ and $\mathbf{b}$ are chosen as such, then by Proposition 1, the nondelay sequence σ is optimal for the Lagrangian problem, with $\{C^\sigma_j | j \in \mathcal{J}\}$ being an optimal schedule. Hence, we have

$$L(\mathbf{a},\mathbf{b}) = \sum_{j=1}^{n}(w_j - a_j + b_j)C^\sigma_j + \sum_{j=1}^{n} a_j(r_j + p_j) - b_j\bar{d}_j.$$

To get a lower bound as tight as possible, we further require that $L(\mathbf{a},\mathbf{b})$ be maximized while $\mathbf{a}$ and $\mathbf{b}$ satisfy (5) and (6). Let y_k be the contribution of the jobs in block B_k to $L(\mathbf{a},\mathbf{b})$. Maximizing $L(\mathbf{a},\mathbf{b})$ is then equivalent to maximizing each y_k

independently, which can be accomplished by solving the following maximization problem $\mathbf{LP}_k$:

$$\bar{y}_k = \max y_k = \sum_{j=u_k}^{v_k} a_j \left(r_j + p_j - C_j^\sigma\right) + b_j \left(C_j^\sigma - \bar{d}_j\right) + w_j C_j^\sigma$$
$$\text{s.t.} \qquad (4),(5),(6), \qquad a_j \geq 0, b_j \geq 0,\ j = u_k, \ldots, v_k.$$

It is interesting to note that $\mathbf{LP}_k$ is a linear program (LP) in which the decision variables are the multipliers a_j, b_j, $j = u_k, \ldots, v_k$ (ignore the constant terms in y_k). The solution of $\mathbf{LP}_k$ degenerates to triviality for the special cases where only release dates or only deadlines are considered, as in [30] and [20].

Let $\bar{a}_j, \bar{b}_j, j = u_k, \ldots, v_k$ denote an optimal solution to $\mathbf{LP}_k$. We define the following lower bound for WCT:

$$\text{LBPS}(\sigma) = \sum_{k=1}^{K} \bar{y}_k = L(\bar{\mathbf{a}}, \bar{\mathbf{b}})$$

where the dependency of LBPS on σ is emphasized. If $\mathbf{LP}_k$ is unbounded for some k, then the bound $\text{LBPS} = \infty$, indicating that WCT is infeasible. Otherwise, LBPS is a valid lower bound, since $L(\bar{\mathbf{a}}, \bar{\mathbf{b}})$ is a valid lower bound.

Because all the blocks receive the same treatment, we hereafter assume, without loss of generality, that σ comprises only one block B_1 with $K = 1$, $u_1 = 1$, $v_1 = n$. Accordingly, we will drop all subscripts k for indexing blocks; e.g., $\mathbf{LP}_k$ and $\bar{y}_k$ become $\mathbf{LP}$ and $\bar{y}$, respectively.

LBPS can be improved using a tightening technique suggested in [20]. Supposed that $\bar{a}_j$, $\bar{b}_j$, $j = 1, \ldots, n$ is an optimal solution to $\mathbf{LP}$. We sort the multipliers $\bar{a}_j$, $j = 1, \ldots, n$ in ascending order to get $\bar{a}_{[i]}$, $i = 1, \ldots, n$. Define $\delta_1 = \bar{a}_{[1]}$, $\delta_i = \bar{a}_{[i]} - \bar{a}_{[i-1]}$, $i = 2, \ldots, n$. Then, $\delta_i \geq 0$ for all i, and $\bar{y}_1$ can be rewritten as

$$\bar{y} = \sum_{i=1}^{n} \delta_i \left[\left(\sum_{h=i}^{n} r_{[h]} + p_{[h]}\right) - \sum_{h=i}^{n} C_{[h]}^\sigma\right] + \sum_{j=1}^{n} w_j C_j^\sigma + \bar{b}_j \left(C_j^\sigma - d_j\right).$$

Note that the term $\left(\sum_{h=i}^{n} r_{[h]} + p_{[h]}\right)$ in the above equation is a lower bound on $\sum_{h=i}^{n} C_{[h]}^\sigma$, because the completion times C_j^σ always satisfy the release date constraints. To obtain a tighter lower bound, we solve an auxiliary problem of the form $1|r_j|\sum C_j$, where there are $(n - i + 1)$ jobs with unit weights. This problem is also known to be NP-hard, but its preemptive version is solved by the shortest-remaining-processing-time rule [4], which permits job preemption. Suppose that the minimum objective value of the preemptive problem is $\sum_{h=i}^{n} C'_{[h]}$, then

$$\sum_{h=i}^{n} r_{[h]} + p_{[h]} \leq \sum_{h=i}^{n} C'_{[h]} \leq \sum_{h=i}^{n} C_{[h]}^\sigma.$$

Therefore, $\sum_{h=i}^{n} C'_{[h]}$ is a better lower bound on $\sum_{h=i}^{n} C_{[h]}^\sigma$ than $\sum_{h=i}^{n} r_{[h]} + p_{[h]}$. Define

$$\bar{y}' = \sum_{i=1}^{n} \delta_i \left[\left(\sum_{h=i}^{n} C'_{[h]}\right) - \sum_{h=i}^{n} C_{[h]}^\sigma\right] + \sum_{j=1}^{n} w_j C_j^\sigma + \bar{b}_j \left(C_j^\sigma - d_j\right).$$

Then, $\bar{y}' \geq \bar{y}$. As a result, we have a strengthened lower bound, which we refer to as LBPS$'$.

B. Three Nondelay Sequences

We propose to consider three $O(n \log n)$ nonpreemptive heuristics, all of which can potentially be employed to generate nondelay sequences for lower bounding purposes. The first two are generic ones taken from the literature; they are the earliest-due-date (EDD) rule and the weighted-shortest-processing-time (WSPT) rule. The third is a new heuristic that we developed specifically for WCT. It is outlined next.

We refer to this heuristic as COMP (for "composite"). Each job $j \in \mathcal{J}$ is associated with a number $t_j = \max\{r_j, \bar{d}_j - p_j\}$. If a job arrives at a time when the machine is busy, the job becomes a waiting job. Suppose that a job needs to be selected for processing at time t when the machine is free. If all jobs have already arrived (i.e., $r_j \leq t$ for all j), we sequence those jobs that have not been processed using Smith's backward scheduling rule [33]. There is a chance that at this time t, some of the waiting jobs are bound to be overdue no matter how they are sequenced. To account for this situation, we modify Smith's rule by pretending that deadlines can be extended whenever necessary during the backward scheduling process. On the other hand, if at time t not all jobs have arrived, we select a job j_0 for processing according to the following rule. Let E be the set of all the jobs currently waiting and let $E_1 = \{j \in E | t_j \leq t\}$. If $E_1 \neq \emptyset$, then set j_0 such that $w_{j_0}/p_{j_0} = \max_{j \in E_1}\{w_j/p_j\}$; otherwise, set j_0 such that $w_{j_0}/p_{j_0} = \max_{j \in E}\{w_j/p_j\}$. This procedure can be implemented to run in $O(n \log n)$ time if E and E_1 are implemented using two heaps (or priority queues) and a bitmap of n elements is used to indicate whether a job has already been processed at time t. Like EDD and WSPT, COMP generates a nondelay sequence σ that respects the release dates, but may violate some deadlines. As stated before, this does not pose any problems since the sole purpose of σ is for computing the lower bound LBPS.

The heuristic COMP is designed to strike a good balance between minimizing the objective function and reducing deadline violation. Its superior performance is later confirmed by experimentation (Section VI-B), in relation to the performance of EDD and WSPT.

C. Transformation of the LP

The remaining issue concerning the lower bound calculation is how to solve the LP efficiently. Before developing a solution procedure, we apply a change of variable based on the following property ofthe LP.

Theorem 2: If the LP is bounded, then there exists an optimal solution that satisfies

$$a_j \cdot b_j = 0, \qquad j = 1, \ldots, n. \tag{7}$$

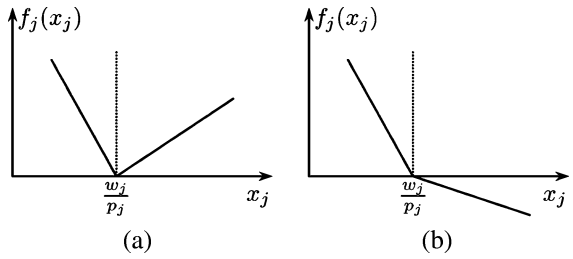


Fig. 1. Convexity of $f_j(x_j)$. (a) $\bar{d}_j - C_j^\sigma \geq 0$. (b) $\bar{d}_j - C_j^\sigma < 0$.

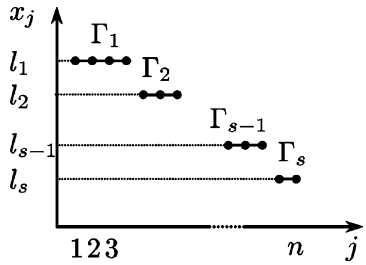


Fig. 2. Segments of a solution to the isotone optimization problems.

Otherwise, for any given Δ, there exists a solution that satisfies (7) and yields an objective value greater than Δ.

Proof: It suffices to show that from any given feasible solution $a_j = \tilde{a}_j, b_j = \tilde{b}_j, j = 1, \ldots, n$ with $\tilde{a}_{j_0} > 0, \tilde{b}_{j_0} > 0$ for some j_0, we can construct a new solution of the required form without decreasing the objective value. Because $r_j + p_j \leq \bar{d}_j$ for all j, it is easily verified that the construction below is valid

$$a_j = \tilde{a}_j - \min\{\tilde{a}_j, \tilde{b}_j\}, \quad b_j = \tilde{b}_j - \min\{\tilde{a}_j, \tilde{b}_j\}$$

for $j = 1, \ldots, n$. ∎

Theorem 2 enables us to perform a change of variable as follows: For each j, define a new variable $x_j = w'_j/p_j$ (recall that $w'_j = w_j - a_j + b_j$). The pair of variables a_j and b_j is then replaced by the single variable x_j using $a_j = (w_j - p_j x_j)^+$, $b_j = (p_j x_j - w_j)^+$, where $x^+ = x$ if $x \geq 0$ and $x^+ = 0$ otherwise. As a result, we get

$$\bar{y} = -\bar{z} + \sum_{j=1}^{n} w_j C_j^\sigma$$

where

$$\bar{z} = \min z = \sum_{j=1}^{n} f_j(x_j) \quad \text{s.t. } x_1 \geq \ldots \geq x_n \tag{8}$$

and

$$f_j(x_j) = p_j \left[\left(C_j^\sigma - r_j - p_j\right) \left(\frac{w_j}{p_j} - x_j\right)^+ + \left(\bar{d}_j - C_j^\sigma\right) \left(x_j - \frac{w_j}{p_j}\right)^+ \right] \tag{9}$$

is a continuous, piecewise-linear function of x_j with a single kink point at w_j/p_j. By definition, $C_j^\sigma - r_j - p_j$ is always nonnegative, whereas $\bar{d}_j - C_j^\sigma$ can be negative since the given σ may violate the deadline $\bar{d}_j$. Moreover, $f_j(x_j)$ is convex since $\bar{d}_j - C_j^\sigma \geq r_j + p_j - C_j^\sigma$ (see Fig. 1). It should be noted that Problem (8) had an additional constraint $x_n \geq 0$; we dropped this constraint because doing so does not change the optimal value $\bar{z}$ (this can be easily shown using the fact that $w_j/p_j > 0$ for all j). To simplify the notation in (9), we define $\alpha_j = p_j(C_j^\sigma - r_j - p_j)$, $\beta_j = p_j(\bar{d}_j - C_j^\sigma)$, and $c_j = w_j/p_j$. Then, (9) becomes

$$f_j(x_j) = \alpha_j (c_j - x_j)^+ + \beta_j (x_j - c_j)^+ \tag{10}$$

with $\beta_j \geq -\alpha_j$. We state without proof the following obvious property of Problem (8) with $f_j(x_j)$ defined by (10).

Proposition 3: Problem (8) is bounded, i.e., $\bar{z} > -\infty$, if and only if $\sum_{j=1}^{i} \beta_j \geq 0$ and $\sum_{j=i}^{n} \alpha_j \geq 0$ for all $i = 1, \ldots, n$.

In fact, the additivity of z, convexity of $f_j(x_j)$, and chain constraints $x_1 \geq \ldots \geq x_n$ in Problem (8) together characterize an extensively researched class of problems called the *isotone optimization problems*, which find applications in operations research, statistics, and image processing. In particular, if the functions $f_j(x_j), j = 1, \ldots, n$ are assumed to be general convex functions, Problem (8) is called the *generalized isotonic regression problem* [3].

III. Solving Isotone Optimization Problems

A. Preliminaries

The isotone optimization problems have been studied by numerous authors (see [3] for a comprehensive list of references). Thus far, the best and most general result is due to Ahuja and Orlin [3], who study the generalized isotonic regression problem, where each $f_j(x_j)$ is an arbitrary convex function. It is also assumed that (*i*) each $f_j(x_j)$ can be evaluated in $O(1)$ time for a given x_j, and that (*ii*) the optimal values of all x_j lie on the interval $[x_{lb}, x_{ub}]$ (this implies that the minimization problem has to be bounded). Let $U = x_{ub} - x_{lb}$ and let ϵ be a tolerance. These authors improve the $O(n^2 \log(U/\epsilon))$ running time of the pool adjacent violators (PAV) algorithm [31] to $O(n \log(U/\epsilon))$ using a scaling technique.

The scaling PAV algorithm [3] can be adapted to solve Problem (8), where $f_j(x_j)$ is specified by (10) with $\beta_j \geq -\alpha_j$. We first determine whether Problem (8) is bounded by verifying the conditions of Proposition 3 in $O(n)$ time. A mapping is then used to modify the problem so that all c_j are mapped onto the integers $1, \ldots, n$. For the modified problem, we can set $U = n$ and fix ϵ to any value less than 1; thus, the algorithm runs in $O(n \log n)$ time. (This mapping is suggested in [3] for the special case when $\alpha_j = \beta_j$, but remains valid for our more general problem.) A serious handicap of this approach, however, is that it requires sorting, which enlarges the hidden constant in the big-O time bound.

To facilitate our subsequent discussion, we introduce the notion of a segment. For any vector $(x_1, \ldots, x_n)$, a *segment* is defined as a maximal subset of consecutive indexes $\{p, p+1, \ldots, q\}$ such that $x_p = x_{p+1} = \ldots = x_q = l$, where l is called the *level*. For example, Fig. 2 shows a feasible solution $(x_1, \ldots, x_n)$ that comprises s segments, $\Gamma_1, \ldots, \Gamma_s$, with the corresponding levels being $l_1, \ldots, l_s$.

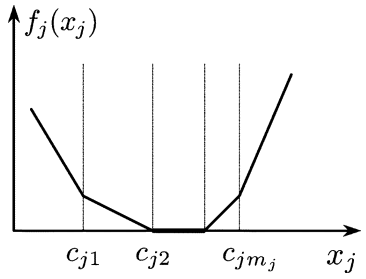


Fig. 3. Continuous, piecewise-linear, convex function with more than one kink point.

B. New Algorithm

In order to solve Problem (8) more efficiently, we propose a new algorithm that is different than the PAV algorithm. Unlike the latter, which is a dual method from a linear programming perspective and achieves primal feasibility only upon termination [9], the proposed algorithm is a primal method where the iterates always maintain primal feasibility (i.e., the chain constraints are satisfied during every iteration). The proposed algorithm uses fairly simple data structure and can be coded with very little overhead (hence, the hidden constant in the big-O notation is very small). Another significant advantage of the proposed algorithm is that it easily generalizes to cases where there are any number of kink points for $f_j(x_j)$, as depicted in Fig. 3. By contrast, the time bound of Ahuja and Orlin's algorithm deteriorates under such circumstances, because the cost of evaluating $f_j(x_j)$ at an x_j is no longer $O(1)$ (i.e., assumption (*i*) of Ahuja and Orlin is violated), but generally depends on the number of kink points.

Assume that each $f_j(x_j)$ is a general convex function and that x_{lb} is a lower bound on the optimal value of x_n (by default, $x_{lb} = -\infty$). For any pair of u and v with $u \leq v$, define a subproblem $\mathcal{P}_u^v$ of Problem (8) as

$$\bar{z}_u^v = \mathbf{min}\ z_u^v = \sum_{j=u}^{v} f_j(x_j)$$
$$\mathbf{s.t.}\ x_u \geq \ldots \geq x_v.$$

Hence, $\bar{z}_1^n = \bar{z}$. The proposed algorithm sequentially solves a series of n subproblems with the ith one being $\mathcal{P}_1^i$. Initially, we set $x_1 = \ldots = x_n = x_{lb}$. Now, assume that the $(i-1)$th subproblem is already solved with $x_1, \ldots, x_{i-1}$ set to appropriate values, and $x_i = \ldots = x_n = x_{lb}$; further, suppose that $x_1, \ldots, x_{i-1}$ form s segments denoted by $\Gamma_1, \ldots, \Gamma_s$ with levels $l_1 > \ldots > l_s$. For convenience, we initially also set $\Gamma_0 = \emptyset$, $l_0 = \infty$. To solve the ith subproblem, we first create a new segment using i; hence, set $s := s+1$, $\Gamma_s := \{i\}$, and $l_s = x_{lb}$. Clearly, the feasibility is satisfied. We then attempt to merge the last two segments, i.e., Γ_s and Γ_{s-1}. To this end, a method of choice is applied to find an x' that minimizes the single-variable convex function $\sum_{j\in\Gamma_s} f_j(x)$ over the interval $[l_s, l_{s-1}]$; we update the current solution by setting $x_j := x'$ for all $j \in \Gamma_s$. Two cases arise: Either it holds that $x' < l_{s-1}$, in which case the updated solution optimally solves the ith subproblem; or it holds that $x' = l_{s-1}$, in which case segments Γ_{s-1} and Γ_s merge into one. In the latter case, we set $\Gamma_{s-1} := \Gamma_{s-1} \cup \Gamma_s$, $s := s-1$. At this point, if $s = 0$, then the ith subproblem is solved ($\bar{z}$ may be unbounded); otherwise, we once again attempt to merge Γ_s and Γ_{s-1}. The ith subproblem will be solved after a finite number of mergers since every time two segments merge, s decreases by one. Because the addition of one more variable x_i may trigger a cascade of mergers and the objective value z associated with an iterate $(x_1, \ldots, x_n)$ is nonincreasing, we call the proposed algorithm the *Cascading Descent* algorithm. An algorithmic description of this algorithm is given below.

Algorithm. Cascading Descent

Step 0) $i = 0$, $s = 0$, $\Gamma_0 = \emptyset$, $l_0 = \infty$.
Step 1) Set $i := i+1$. If $i > n$, then go to 5.
Step 2) *Create a new segment*: Set $s := s+1$, $l_s := x_{lb}$, $\Gamma_s := \{i\}$.
Step 3) Apply any method of choice to find an x' that minimizes $F(x) = \sum_{j\in\Gamma_s} f_j(x)$ over $[l_s, l_{s-1}]$. If $x' < l_{s-1}$, set $l_s := x'$ and go to 1.
Step 4) *Merge two segments*: Set $\Gamma_{s-1} := \Gamma_{s-1} \cup \Gamma_s$, $s := s-1$. If $s = 0$, go to 1; otherwise, go to 3.
Step 5) *Postprocessing*: For each segment $h = 0, \ldots, s$ and for each $j \in \Gamma_s$, set $x_j := l_h$. Compute $\bar{z} = \sum_{j=1}^{n} f_j(x_j)$. **STOP**.

Lemma 4: Let $(x_u, \ldots, x_v)$ and $(x_{v+1}, \ldots, x_w)$ be optimal solutions to subproblems $\mathcal{P}_u^v$ and $\mathcal{P}_{v+1}^w$, respectively. If $x_v \geq x_{v+1}$, then the concatenated solution $(x_u, \ldots, x_w)$ is optimal to subproblem $\mathcal{P}_u^w$.

Proof: By dropping the constraint $x_v \geq x_{v+1}$ in subproblem $\mathcal{P}_u^w$, we obtain a relaxed problem, which then separates into $\mathcal{P}_u^v$ and $\mathcal{P}_{v+1}^w$. Hence, $\bar{z}_u^v + \bar{z}_{v+1}^w$ is a valid lower bound for $\mathcal{P}_u^w$. Since $(x_u, \ldots, x_w)$ attains this lower bound and is feasible, it must be optimal. ■

Lemma 5: Suppose that $(x_u, \ldots, x_v) = (a, \ldots, a)$ and $(x_{v+1}, \ldots, x_w) = (b, \ldots, b)$ are optimal solutions to subproblems $\mathcal{P}_u^v$ and $\mathcal{P}_{v+1}^w$, respectively. If $a \leq b$, then subproblem $\mathcal{P}_u^w$ has an optimal solution of the form $(x_u, \ldots, x_w) = (c, \ldots, c)$.

Proof: Refer to Lemma 1 in [3]. ■

Theorem 6: The proposed Cascading Descent algorithm is correct.

Proof: We show through mathematical induction that (I) after the ith iteration $(i = 1, \ldots, n)$, the algorithm correctly solves subproblem $\mathcal{P}_1^i$. Furthermore, we show that (II) for $h = 1, \ldots, s$, the solution, $x_j = l_h$, $j \in \Gamma_h$, optimally solves subproblem $\mathcal{P}_{\min\{\Gamma_h\}}^{\max\{\Gamma_h\}}$. The base case when $i = 1$ is easily verified.

Now, assume that the above Part I and Part II hold after the $(i-1)$th iteration. Consider the first time when Step 3) is executed for this particular i value (i.e., when $\Gamma_s = \{i\}$). By definition, x' minimizes $F(x) = f_i(x)$ over $[x_{lb}, l_{s-1}]$. In the first case, we have $x' < l_{s-1}$. We claim that x' also minimizes $F(x)$ over $[x_{lb}, \infty)$. For the sake of drawing contradiction, assume that there exists $x'' > l_{s-1}$ such that $F(x'') < F(x')$. By convexity, we have $F(l_{s-1}) \leq [F(x')(l_{s-1} - x') + F(x'')(x'' - l_{s-1})]/(x'' - x')$. Replacing $F(x'')$ with $F(x')$ in the right-hand side yields $F(l_{s-1}) < F(x')$, which is a contradiction to the

fact that x' minimizes $F(x)$ over $[x_{lb}, l_{s-1}]$. Hence, x' optimally solves subproblem $\mathcal{P}^{\max\{\Gamma_s\}}_{\min\{\Gamma_s\}}$. Noting also the induction assumption, it follows from Lemma 4 that subproblem $\mathcal{P}^i_1$ is optimally solved. Part II evidently follows.

In the second case, we have $x' \geq l_{s-1}$ and Step 4) is therefore executed to merge Γ_{s-1} and Γ_s. The induction assumption states that before merger, the solution, $x_j = l_{s-1}, j \in \Gamma_{s-1}$, optimally solves $\mathcal{P}^{\max\{\Gamma_{s-1}\}}_{\min\{\Gamma_{s-1}\}}$. From Lemma 5, it can be seen that subproblem $\mathcal{P}^{\max\{\Gamma_{s-1}\}}_{\min\{\Gamma_{s-1}\}}$ still has single-valued optimal solution even after merger. This ensures that the same argument can be used to analyze subsequent executions of Steps 3) and 4). Since there can be only a finite number of mergers, the ith iteration eventually will terminate. This completes the induction step of the proof. ■

A small technicality in the algorithm is the use of the auxiliary segment Γ_0 with level $l_0 = \infty$. If $s = 1$ and $x' = l_0$ in Step 3) of the algorithm, a merger of segments Γ_0 and Γ_1 will occur. This means that all x_i, $i \in \Gamma_1$ should take on the value of ∞. However, this does not necessarily imply that $\bar{z}^i_1 = -\infty$ since the single-variable function $\sum_{j=1}^{i} f_j(x)$ could be asymptotically a horizontal line.

We now turn to the focus of this subsection, which is to specialize the above algorithm for situations where $f_j(x_j)$ is not only convex, but also continuous, piecewise linear. Clearly, the single-variable function $F(x) = \sum_{j\in\Gamma_s} f_j(x)$ in Step 3), which is the summation of a finite number of such functions, is also a continuous, piecewise-linear convex function. For this type of function, it is easy to verify the following fact.

Lemma 7: Let $F(x)$ be a continuous, piecewise-linear function on interval $[l_s, l_{s-1}]$ (could be $(-\infty, \infty)$). Let $F'_-(x)$ and $F'_+(x)$ denote the left-hand and right-hand derivatives, respectively. Then, the minimum of $F(x)$ occurs in three different scenarios: 1) $x \in (l_s, l_{s-1})$ and $F'_-(x) \leq 0$ and $F'_+(x) \geq 0$; 2) $F'_+(l_s) \geq 0$; 3) $F'_-(l_{s-1}) \leq 0$.

Because the values of $F'_-(x)$ and $F'_+(x)$ only change at kink points, it suffices to restrict our attention to these points. Here, it is assumed that each $f_j(x_j)$ has m_j kink points denoted by $c_{j1}, \ldots, c_{jm_j}$; let $m = \sum_{j=1}^{n} m_j$. Note that for kink point c_{jk}, the left-hand derivative $f'_{j-}(c_{jk})$ is the slope of the line segment to the left of this point, and the right-hand derivative $f'_{j+}(c_{jk})$ is the slope of the line segment to the right of this point. Hence, all the left-hand and right-hand derivatives are known beforehand.

The following *specialized cascading descent* algorithm keeps track of kink points using a *heap* (also called a *priority queue*) data structure. Let the heap be denoted by H. An element in H is a kink point c_{jk} ($j = 1, \ldots, n$; $k = 1, \ldots, m_j$). Our algorithm performs three types of operations on H, i.e., INSERT (insert an element into H), MIN (return the element with the smallest key value), and DELETE-MIN (remove from H the element with the smallest key value). Because there is no need to merge two heaps into a new heap, such a heap data structure can be implemented efficiently as a complete binary tree stored in an array object (see [12] for more details). By contrast, heaps that support merge operations are called *mergeable heaps*, whose implementations are much more tedious and inefficient (for examples of mergeable heap implementations, see *binomial heap* and *Fibonacci heap* in [12] and *leftist heap* in [35]).

In the following algorithm, RHD_s represents the right-hand derivative of the function $\sum_{j\in\Gamma_s} f_j(x)$ at $x = l_s$.

Algorithm. Specialized Cascading Descent

Step 0) *Initialization*: $i = 0$, $s = 0$, $\Gamma_0 = \emptyset$, $l_0 = \infty$, $\mathrm{RHD}_0 = \infty$, $H = \emptyset$.

Step 1) Set $i := i + 1$. If $i > n$, then go to 5.

Step 2) *Create a new segment*: Set $s := s + 1$, $l_s := -\infty$, $\Gamma_s := \{i\}$, $\mathrm{RHD}_s := f'_{i+}(-\infty)$. INSERT each kink point $c_{ik} (k = 1, \ldots, m_j)$ into the heap H.

Step 3) Set $x' := l_s$.

While $\mathrm{RHD}_s < 0$ do

- If $H = \emptyset$, set $x' := l_{s-1}$ and terminate the while-do loop; otherwise, let c_{jk} be the kink point that corresponds to $\mathrm{MIN}(H)$.
- If $c_{jk} \leq l_{s-1}$, set $x' := c_{jk}$, $\mathrm{RHD}_s := \mathrm{RHD}_s - f'_{i-}(c_{jk}) + f'_{i+}(c_{jk})$; otherwise, set $x' = l_{s-1}$ and terminate the while-do loop.
- DELETE-MIN (H).

If $x' < l_{s-1}$, set $l_s := x'$ and go to 1.

Step 4) *Merge two segments*: Set $\Gamma_{s-1} := \Gamma_{s-1} \cup \Gamma_s$, $\mathrm{RHD}_{s-1} := \mathrm{RHD}_{s-1} + \mathrm{RHD}_s$, $s := s - 1$. If $s = 0$, go to 1; otherwise, go to 3.

Step 5) *Postprocessing*: For each segment $h = 0, \ldots, s$ and for each $j \in \Gamma_s$, set $x_j := l_h$. Compute $\bar{z} = \sum_{j=1}^{n} f_j(x_j)$. *STOP*.

To show that this specialized algorithm is correct, we only need to argue that Step 3) correctly solves the problem of minimizing $\sum_{j\in\Gamma_s} f_j(x)$ over interval $[l_s, l_{s-1}]$. When a new segment $\Gamma_s = \{i\}$ is created in Step 2), the level l_s is set to $-\infty$. In the subsequent execution of Step 3), we start with $x' = l_s = -\infty$, $\mathrm{RHD}_s = f'_{i+}(-\infty) = \lim_{x\to-\infty} f'_{i+}(x)$ and gradually increase x' ("hopping" on kink points) until one of the following two scenarios occurs: 1) $f'_{i+}(x') \geq 0$ for the first time or 2) $x' = l_{s-1}$ is reached. If scenario 1) occurs, then $f'_{i+}(x') \geq 0$ holds either at $x' = l_s$ (i.e., at the left boundary), or at some kink point $x' > l_s$ (hence, $f'_{i-}(x') = f'_{i+}(x' - \epsilon) < 0$ for some sufficiently small $\epsilon > 0$); by Lemma 7, x' minimizes $f'_{i+}(x)$ over interval $[l_s, l_{s-1}]$. Note that the proof so far covers the tie situation when 1) and 2) hold simultaneously. Now, if scenario 2) occurs and $f'_{i+}(l_s) < 0$, then $f'_{i-}(l_s) < 0$, following from the fact $f'_{i-}(l_s) \leq f'_{i+}(l_s)$; hence, Lemma 7 can be invoked once again. Using the same argument, we can also show that Step 3) correctly solves $\sum_{j\in\Gamma_s} f_j(x)$ over interval $[l_s, l_{s-1}]$ for a segment Γ_s resulting from a merger.

In Step 3), as x' passes through kink points, these points are removed from H and will never be used again; this simplifies the problem for later iterations. It is also worth pointing out that the kink points that belong to functions $f_j(x_j), j \in \Gamma_0 \cup \ldots \cup \Gamma_{s-1}$ do not interfere with those of segment Γ_s, even though all kink points are stored in the same heap. This is because all the kink points associated with a segment are always above the current level of the segment; i.e., during the execution of Step 3), we have $c_{jk} > l_{s-1}$ for any c_{jk} in H with $j \notin \Gamma_s$.

Theorem 8: The specialized Cascading Descent algorithm runs in $O(m \log m)$ time.

Proof: Note that there are m INSERTs and consequently, no more than m DELETE-MINs. Since both types of operations take $O(\log m)$ time on a heap with at most m elements [12], the total cost of these operations is $O(m \log m)$ time. Also, there are $O(m)$ MIN operations, each taking $O(1)$ time. The remaining algorithmic steps can be completed in $O(m)$ time. ■

As an immediate application of the specialized algorithm, we can use it to solve, in $O(m \log m)$ time, the isotonic median regression problem, where $f_j(x_j) = \sum_{i=1}^{m_j} |x_j - c_{jk}|$ for all j. This time bound is an improvement over that of Pardalos *et al.* [28], which is $O(m \log^2 m)$. Also, it is a significant advantage from a practical standpoint that our algorithm does not require merging two heaps (or balanced binary search trees in [28]). Furthermore, it is difficult to adapt Ahuja and Orlin's scaling algorithm [3] to this problem and achieve a competitive time bound. The difficulty stems from the fact $f_j(x_j)$ cannot be evaluated in $O(1)$ time, with the exception when $m_j = 1$ for all j (Ahuja and Orlin indeed propose an adaptation for this special case with an $O(n \log n)$ time bound).

Now, with the specialized algorithm in hand, we can compute the lower bound LBPS in $O(n \log n)$ time. Note that in this particular application, the functions $f_j(x_j)$, $j = 1, \ldots, n$ are defined by (10). Therefore, $m_j = 1$ for all j and $m = n$. The total cost of computing LBPS consists of two parts. One part is for generating the nondelay sequence σ, which takes $O(n \log n)$ time. The other part is the cost of executing the specialized algorithm for the blocks $B_1, \ldots, B_K$. The suggested time bound follows from the fact that

$$\sum_{k=1}^{K} |B_k| \cdot \log |B_k| \leq \log n \sum_{k=1}^{K} |B_k| = n \log n.$$

IV. Dominance Conditions and Elimination Technique

We made some straightforward extensions of those in [6] and [20], which deal with $1|r_j| \sum w_j C_j$. It was also noted that there is no need to consider a node that represents an infeasible instance.

A. Dominance Conditions

In our dominance theorems, we implicitly assume that the problem is feasible. However, these theorems can be applied even if feasibility is undetermined, since nothing is lost if the problem turns out to be infeasible. For a sequence π, the notation π_k denotes the job in the kth position of the sequence. The first two of the following theorems extend the results of Bianco and Ricciardelli [7] originally proposed for $1|r_j| \sum w_j C_j$, so as to take into account the deadlines $\bar{d}_j$. The proofs of these theorems are straightforward and can be found in Pan [26].

Theorem 9: If $w_t/p_t = \max_{i=1}^{n}\{w_i/p_i\}$ and $\bar{d}_t = \min_{i=1}^{n}\{\bar{d}_i\}$, then there exists an optimal solution with job t preceding job i for $i = 1, \ldots, n$, $i \neq t$ and $r_i \geq r_t$.

Theorem 10: If $r_t + p_t = \min_{i=1}^{n}\{r_i + p_i\}$, then for some optimal sequence π^*, $\pi_1^* \neq i$ for $i = 1, \ldots, n$ and $r_i \geq r_t + p_t$.

Theorem 11: Let $\pi = ji\pi^1$ be any feasible sequence with job j and job i in the initial two positions and let $\pi' = ij\pi^1$ be the result of interchanging job j and job i in the sequence π. If π' is feasible and $C_j^{\pi'} \leq C_i^{\pi}$ and $w_i C_i^{\pi'} + w_j C_j^{\pi'} \leq w_j C_j^{\pi} + w_i C_i^{\pi}$, then π is dominated.

Theorem 11 follows from the principle of optimality. The idea is effectively used in [20] and [30] for their respective problems. The proof is straightforward and requires no comment.

Unlike the dominance conditions presented thus far, the next one does not depend on the objective function. Rather, it is simply a feasibility check.

Theorem 12: Job i can be assigned to the first position only if there exists a feasible sequence π with $\pi_1 = i$.

The feasibility of the subproblem with job i fixed in the first position can be determined by solving a related problem of minimizing the maximum lateness. The following corollary is not as strong as Theorem 12 but can be verified quickly.

Corollary 13: Consider the subproblem in which job i is fixed in the first position and the objective is to minimize the maximum lateness. If an optimal preemptive solution yields a positive objective value, then job i cannot be in the first position.

B. Elimination by Recursion

The above dominance conditions are not very strong after so much adaptation to our dual-constrained problem. As a remedy, we developed an elimination technique. It was first tried on the static problem, since at the time, we had already implemented Posner's [29] branch-and-bound algorithm for $1|\bar{d}_j| \sum w_j C_j$ to be used as a module in our own algorithm for WCT, and incorporating this new feature took little effort. To our surprise, the improved algorithm doubled the size of problems that can be solved; problems with up to 120 jobs can now be solved efficiently [25]. In the following, we discuss in detail how to apply this technique to WCT. We hope to advocate the use of this technique through this example so that it will eventually become as a standardized element as dominance conditions to special-purpose branch-and-bound algorithms.

A search tree node corresponds to a feasible partial sequence π in which jobs in the first l positions are fixed. For all $i \leq l$, the completion time C_{π_i} of job π_i is also computed. We define $S = \{\pi_1, \ldots, \pi_l\}$ as the set of scheduled jobs and $\bar{S}$ as the set of unscheduled jobs. Under the partial sequence π, no jobs in $\bar{S}$ can start before time $T = \max\{C_{\pi_l}, \min_{j \in \bar{S}} r_j\}$. If jobs in S can be resequenced such that they finish by time T, observe their respective release dates and deadlines, and make a total contribution to the objective function smaller than $\sum_{i=1}^{l} w_{\pi_i} C_{\pi_i}$, then the node can be eliminated from further consideration. This is a particular application of the well-known principle of optimality (see [15] for a general discussion of this principle applied to branch-and-bound algorithms). Our innovation here, however, is to provide a method to exploit this powerful property to a much greater extent than what was achieved in the past.

We first recognize that the embedded problem of resequencing jobs in S is, in fact, an instance of WCT, the exact problem that we set out to solve. More precisely, the instance consists of l jobs π_i, $i = 1, \ldots, l$, each of which has a release date r_{π_i}, a deadline $\min\{\bar{d}_{\pi_i}, T\}$, and a weight w_{π_i}. We then proceed to solve this embedded problem using the same algorithm developed for WCT. The node is eliminated if we can disprove the optimality of the partial sequence π with respect to the embedded problem.

What we described above is essentially a recursive process. Several measures are taken so as to prevent excessive stalling inside a recursion. First, the recursion depth is limited to one; i.e., the embedded problem is solved by a stripped-down version of the branch-and-bound algorithm that does not carry out any further recursion. Second, the recursion returns once an objective value less than $\sum_{i=1}^{l} w_{\pi_i} C_{\pi_i}$ is attained for the embedded problem. Third, the recursion returns once the execution inside the recursion exceeds a preset CPU time limit. Finally, we define a parameter η called the *retrospect depth*. If $l \leq \eta$, then the embedded problem involves all those l jobs; otherwise, only the most recently fixed η jobs $\pi_{l-\eta+1}, \ldots, \pi_l$ are involved, and the release date of job $\pi_i (i = l - \eta + 1, \ldots, l)$ is set to $\max\{C_{\pi_{l-\eta}}, r_{\pi_i}\}$. Clearly, larger η gives us a better chance of eliminating a node, but it also means longer computation time required for solving the subproblem.

V. Branch-and-Bound Algorithm

In this section, we present a complete branch-and-bound algorithm for solving WCT. The lower bounds developed in Section II are employed in this algorithm.

Algorithm. PS

Step 0) Determine the feasibility of WCT by solving the associated $1|r_j|L_{\max}$ problem. If the optimal $L_{\max}$ value is positive (meaning that WCT is infeasible), then **STOP**. Otherwise, derive precedence relations between jobs, tighten the release dates r_j and the deadlines $\bar{d}_j$, and initialize the root node.

Step 1) Find the node with the smallest lower bound. If the lower bound meets the upper bound or the CPU time or memory usage reaches a preset limit, then **STOP**. Otherwise, find a feasible sequence in this node. If no feasible sequence exists, discard this node and repeat Step 1); otherwise, improve the feasible sequence found using an iterative pairwise interchange heuristic.

Step 2) Find the set of unscheduled jobs H that can be put in position $l + 1$. Create a new node for each job in H.

Step 3) Initialize each of the new nodes and, if not fathomed, add it to the search tree. Go to 1.

We discuss the algorithmic details step by step. Preprocessing is carried out in Step 0). First, to determine the feasibility of WCT, we solve the $1|r_j|L_{\max}$ problem (where the due dates $d_j = \bar{d}_j$ and $L_{\max} = \max_j\{C_j - d_j\}$) using a procedure suggested in [27], which improves upon Carlier's algorithm [8]. This procedure is adapted such that it exits as soon as it finds a sequence with $L_{\max} \leq 0$ (in this case, WCT is feasible). Next, if the algorithm does not terminate due to problem infeasibility, precedence relations between jobs are derived using the simple fact that if $r_j + p_j + p_i > \bar{d}_i$ then job i must precede job j. Meanwhile, additional precedence relations are deduced from known ones through an updating method described in [30]. Also, we use a method in [24] to tighten the constraints: To increase the release date r_j of job j, we impose the requirement that job j must start exactly at time r_j, and if this results in infeasibility, we set the new release date to $r_j + 1$. Similarly, we may be able to decrease the deadline $\bar{d}_j$. This process is repeated until dates cannot be tightened any further.

In Step 1), we seek a feasible sequence (again using the procedure in [27]). If no feasible sequence can be found by the procedure, the subproblem represented by this node is infeasible and the node is therefore fathomed. If a feasible sequence is indeed found, we run a standard iterative pairwise interchange heuristic to improve the sequence by way of swapping job pairs. This helps attain a tight upper bound quickly, thereby limiting the size of the search tree.

During the branching in Step 2), H is the set of jobs that can be put in position $(l + 1)$, i.e., the first free position. To keep the cardinality of H small, we first use the fact that only active schedules need to be considered (Theorem 10). Moreover, an unscheduled job i does not belong to H if putting job i in position $(l + 1)$ would violate the precedence relations derived in Step 0), or if putting job i and another unscheduled job j in positions $(l+1)$ and $(l+2)$ in that order would cause job j to violate its deadline (Theorem 12). The set H is further reduced using Theorems 9 and 11 and Corollary 13. During the verification of the dominance condition given by Corollary 13, the preemptive EDD rule is applied to the 10 most imminent jobs (we arrived at this number through experimentation). In addition, as the completion times of these jobs are determined by the preemptive EDD rule, they are checked against the deadlines, and there is no need to continue as soon as any deadline is violated.

Node initialization is carried out in Steps 0) and 3). Before the initialization begins, it is assumed that jobs in the first l positions are fixed (e.g., $l = 0$ in Step 0) and that these l jobs are specified by a partial sequence π. Let $\bar{S}$ be the set of unscheduled jobs. For each job $j \in \bar{S}$, its *effective release date*—the earliest time when the job can start under the partial sequence π—is $r'_j = \max\{C_{\pi_l}, r_j\}$. If all r'_j, $j \in \bar{S}$ are equal, then the node represents an instance of the static problem $1|\bar{d}_j|\sum w_j C_j$ and is subsequently solved using the branch-and-bound algorithm of Posner [29]; the node is then fathomed. Otherwise, we generate a nondelay sequence σ using the dispatching heuristic of our choosing (to be discussed in Section VI-B) and in turn calculate the lower bounds LBPS and LBPS'. The node is fathomed if either of the lower bounds is greater than or equal to the upper bound. If the sequence σ is feasible and leads to a better upper bound, then the same iterative pairwise interchange heuristic as mentioned before is applied to σ, in hope of finding additional improvement. Finally, the node is examined using the node elimination technique of Section IV-B, and it is added to the search tree afterwards, provided that it is not fathomed. It should be noted that during the elimination test, the embedded problem as defined in Section IV-B is solved as a recursion using a stripped-down version of algorithm PS.

In the stripped-down version of algorithm PS, no further recursion is invoked and no attempt is made to derive any precedence relations or to tighten release dates or deadlines. Moreover, in Step 1), we skip the feasibility check and the iterative

pairwise interchange heuristic. To improve the detection of infeasibility, Corollary 13 is applied with regard to all the unscheduled jobs at a node. This also ensures that in the branching step, any job in the set H will satisfy its deadline, provided that it is scheduled immediately in position $(l+1)$.

The value of the parameter η (see Section IV-B) is found by experimentation. Preliminary experiments indicate that the computational results are quite insensitive to the choice of the η value as long as η is between 8 and 15. The best performance is achieved when η is set to 10, which is the parameter value assumed in the subsequent numerical experiments (Section VI).

VI. Computational Results

We coded both our branch-and-bound algorithm (PS) and the dynamic programming method (GS) [18] in Visual C++ and ran them on a Pentium III 733 personal computer. The version of GS that we implemented exploits the fact that all job weights w_j are nonnegative, and it uses a large hash table consisting of 500 000 entries to minimize the possibility of a collision. For both procedures, we set the maximum CPU time allowed on each test problem to 120 s. With regard to the storage limit, PS abandons a particular test problem if there are more than 100 000 unexplored nodes, and GS terminates if more than 500 000 labels are required or the number of states exceeds 200 000 for states of any given cardinality. Next, we explain the test problems used in our experiments.

A. Test Problems

Three sets of problems were created. Infeasible ones were ignored because they pose little challenge. Let $U[a,b]$ be an integer uniform distribution on interval $[a,b]$. For problem set (I), we took $p_j \in U[1,100]$ and $w_j \in U[1,10]$ (see [20], [30]). The release dates (r_j) were generated from $U[0, \alpha\sum_{j=1}^{n} p_j]$, where the parameter $\alpha \in \{0.5, 1\}$. To generate the deadlines, we followed two steps. First, we computed the earliest job completion times (denoted by $\tilde{C}_j$) under the first-in–first-out (FIFO) rule, which only takes into account the release dates and processing times. Then, we set the deadlines: For all j, set $\bar{d}_j := \tilde{C}_j + V_j$, where $V_j \in U[0, \beta(\tilde{C}_j - r_j)]$ with the parameter $\beta \in \{1,2,4,8,16\}$. The deadlines, together with the release dates, processing times, and weights, defined a feasible problem. Ten problems were created for each combination of the problem size $(n \in \{20,30,40,50\})$, α, and β. This problem set was specifically designed to test the algorithms' responses to different input data characteristics. In addition to varying problem size, the distribution of the release dates was controled by the parameter α. Imagine that there are two systems that handle comparable workloads, but one with $\alpha = 0.5$ and the other with $\alpha = 1$. On average, all jobs will have arrived by the time when half of them have been finished in the first system. By contrast, the workload is distributed more evenly over time in the second system; jobs arrive within a time interval that is twice as large as that of the first system. Clearly, jobs in the first system are bound to experience longer waiting time, and their time windows therefore ought to be larger in order to have a feasible schedule. Finally, it should be pointed out that we have chosen to only report on the two α values $(\alpha \in \{0.5, 1\})$ to avoid an excess of computational results—especially those on easy instances, where deadlines are either very constraining or not constraining at all. The two chosen values are most representative of the characteristics of nontrivial instances.

Problem set (II) was created in a similar fashion as in [18]. We only describe the parameter settings below and refer the reader to [18] for more details. The processing times and weights were generated from $U[1,50]$ and $U[1,10]$, respectively. Let W be a parameter that controls the average of time window widths $\bar{d}_j - r_j$. W took on values in {150, 200, 250, 300, 400, 800}, covering a greater range of time window widths than in [18]. Ten problems were generated for each combination of the problem size $(n \in \{20,30,40,50\})$ and W. We used this set of problems to study the sensitivity of the solution procedures to increase in the time window width.

Problem set (III) consists of problems in which release dates and deadlines are uncorrelated. To generate the release dates, processing times, and weights, we followed the same scheme as in problem set (I). The deadlines $(\bar{d}_j)$ were initially set to random samples of $U[0, \gamma\sum_{j=1}^{n} p_j]$, where $\gamma \in \{1,2,4,8,16\}$ is the slackness parameter. Then, we solved the associated $1|r_j|L_{\max}$ problem with respect to the obtained r_j, p_j, and $d_j = \bar{d}_j$ using the procedure in [27]. Let δ denote the minimum value. If the problem instance was infeasible (i.e., $\delta > 0$), we extended all the deadlines of WCT by setting $\bar{d}_j := \bar{d}_j + \delta$ for all j (thus, the modified instance became feasible). Clearly, the release dates and deadlines remain uncorrelated. For each combination of the problem size $(n \in \{20,30,40,50\})$, α, and γ, ten problem instances were created.

B. Choice of Nondelay Sequence σ

Our first experiment compares the performance of EDD, WSPT, and COMP, discussed in Section II-B for choosing the nondelay sequence σ. Combining each of the heuristics with the PS algorithm results in three variants: PS-EDD, PS-WSPT, and PS-COMP. Problem set (I) is used in this experiment. For each combination of the values of n, α, and β, Table I lists the number of unsolved problems, the mean and maximum solution times in seconds, and the mean and maximum numbers of nodes. Because the calculation of means and maximums takes into account both solved and unsolved problems, their values should be interpreted as lower bounds on the true values if there are unsolved problems. When an algorithm attempts to solve a difficult problem, it may halt prematurely due to storage overflow, which quite often occurs long before the CPU time limit is reached. This can make the mean values and the maximum values look better, which is misleading. For this reason, means and maximums are reported only when over half of the test problems for any given parameter setting are solved; otherwise, they are replaced by "—" (this rule will be followed in our subsequent experiments as well). The performance of PS-EDD turns out to be rather poor; the computational results for this algorithm are therefore omitted. The results in Table I indicate the overall dominance of PS-COMP over PS-WSPT. Therefore, PS-COMP merits further investigation, and the PS algorithm involved in our other experiments should be taken as this particular variant.

TABLE I
COMPARISON OF PS-WSPT VERSUS PS-COMP USING PROBLEM SET (I)

n	α	β	No. Unsolved		CPU Seconds				No. of Nodes			
			PS-WSPT	PS-COMP	PS-WSPT		PS-COMP		PS-WSPT		PS-COMP	
					Mean	(Max)	Mean	(Max)	Mean	(Max)	Mean	(Max)
20	0.5	1	0	0	0.02	(0.03)	0.02	(0.03)	119	(337)	84	(333)
		2	0	0	0.03	(0.08)	0.02	(0.07)	201	(558)	113	(405)
		4	0	0	0.03	(0.18)	0.02	(0.09)	146	(1008)	69	(415)
		8	0	0	0.02	(0.06)	0.01	(0.02)	113	(333)	44	(98)
		16	0	0	0.01	(0.05)	0.01	(0.02)	58	(296)	31	(51)
	1	1	0	0	0.01	(0.01)	0.01	(0.01)	36	(104)	33	(100)
		2	0	0	0.00	(0.01)	0.00	(0.01)	39	(116)	38	(113)
		4	0	0	0.01	(0.01)	0.01	(0.01)	49	(97)	47	(97)
		8	0	0	0.01	(0.01)	0.01	(0.01)	55	(124)	54	(124)
		16	0	0	0.01	(0.01)	0.01	(0.01)	41	(59)	41	(57)
30	0.5	1	0	0	0.26	(0.71)	0.10	(0.26)	734	(1747)	240	(731)
		2	0	0	0.77	(1.50)	0.16	(0.27)	1992	(4260)	384	(645)
		4	0	0	2.75	(13.16)	0.10	(0.19)	7402	(31734)	257	(486)
		8	0	0	0.73	(4.58)	0.07	(0.13)	2025	(10527)	188	(341)
		16	0	0	2.11	(18.51)	0.08	(0.18)	6300	(54500)	197	(419)
	1	1	0	0	0.04	(0.11)	0.04	(0.12)	179	(502)	132	(342)
		2	0	0	0.11	(0.72)	0.07	(0.26)	412	(2003)	266	(832)
		4	0	0	0.18	(1.35)	0.05	(0.13)	671	(4984)	190	(488)
		8	0	0	0.06	(0.14)	0.05	(0.14)	240	(510)	192	(408)
		16	0	0	0.03	(0.05)	0.04	(0.06)	142	(218)	135	(209)
40	0.5	1	0	0	5.30	(19.08)	0.57	(1.80)	10206	(41959)	1072	(2900)
		2	0	0	13.51	(32.12)	1.02	(4.05)	24967	(59500)	1553	(5499)
		4	0	0	12.52	(74.71)	0.36	(1.26)	23806	(146577)	642	(2249)
		8	1	0	11.14	(48.00)	0.59	(3.92)	29533	(117399)	908	(5825)
		16	0	0	6.54	(53.27)	0.32	(0.79)	17148	(145573)	605	(2176)
	1	1	0	0	0.10	(0.47)	0.09	(0.38)	377	(2109)	279	(1395)
		2	0	0	0.18	(0.57)	0.18	(0.56)	578	(1860)	497	(1565)
		4	0	0	0.29	(0.82)	0.27	(0.68)	923	(2239)	769	(2064)
		8	0	0	0.20	(0.49)	0.19	(0.55)	670	(1945)	563	(1938)
		16	0	0	0.28	(0.82)	0.31	(0.89)	784	(1642)	774	(1640)
50	0.5	1	4	0	48.83	(120.00)	14.33	(75.56)	98824	(162634)	21882	(139057)
		2	3	0	77.04	(120.00)	2.35	(7.26)	120419	(229314)	2789	(9245)
		4	7	0	—		4.04	(15.14)	—		5314	(17678)
		8	4	0	34.07	(120.00)	4.03	(28.58)	66947	(176218)	3990	(26063)
		16	2	0	13.93	(38.83)	1.05	(2.03)	35150	(117276)	1361	(2567)
	1	1	0	0	0.37	(0.96)	0.27	(0.60)	970	(2360)	609	(1390)
		2	0	0	4.14	(35.01)	0.63	(2.01)	7488	(60635)	1226	(3892)
		4	0	0	12.84	(99.93)	2.44	(9.10)	20621	(156388)	3739	(13628)
		8	0	0	3.80	(24.91)	2.26	(12.63)	6613	(41385)	3751	(19131)
		16	0	0	0.95	(3.11)	1.01	(3.09)	1979	(5158)	1885	(5054)

TABLE II
RESULTS OF GS VERSUS PS ON PROBLEM SET (I)

n	α	β	$\frac{\hat{E}(\bar{d}_j - r_j)}{\hat{E}(p_j)}$	No. Unsolved		CPU Seconds		n	α	β	$\frac{\hat{E}(\bar{d}_j - r_j)}{\hat{E}(p_j)}$	No. Unsolved		CPU Seconds	
				GS	PS	GS	PS					GS	PS	GS	PS
						Mean (Max)	Mean (Max)							Mean (Max)	Mean (Max)
20	0.5	1	9.42	0	0	0.01 (0.05)	0.02 (0.03)	40	0.5	1	18.06	3	0	9.85 (24.95)	0.57 (1.80)
		2	11.44	0	0	0.03 (0.14)	0.02 (0.07)			2	22.13	8	0	—	1.02 (4.05)
		4	16.00	0	0	0.29 (0.99)	0.02 (0.09)			4	33.70	10	0	—	0.36 (1.26)
		8	31.04	0	0	1.99 (7.01)	0.01 (0.02)			8	61.08	10	0	—	0.59 (3.92)
		16	57.48	0	0	5.58 (12.61)	0.01 (0.02)			16	96.89	10	0	—	0.32 (0.79)
	1	1	4.65	0	0	0.00 (0.01)	0.01 (0.01)		1	1	5.34	0	0	0.07 (0.68)	0.09 (0.38)
		2	5.39	0	0	0.00 (0.00)	0.00 (0.01)			2	8.04	0	0	0.09 (0.80)	0.18 (0.56)
		4	7.96	0	0	0.01 (0.05)	0.01 (0.01)			4	12.02	1	0	1.96 (8.56)	0.27 (0.68)
		8	12.80	0	0	0.02 (0.07)	0.01 (0.01)			8	21.69	5	0	—	0.19 (0.55)
		16	20.82	0	0	0.23 (0.80)	0.01 (0.01)			16	32.25	8	0	—	0.31 (0.89)
30	0.5	1	12.19	0	0	0.17 (1.37)	0.10 (0.26)	50	0.5	1	20.43	7	0	—	14.33 (75.56)
		2	18.30	0	0	6.01 (13.89)	0.16 (0.27)			2	27.29	10	0	—	2.35 (7.26)
		4	27.64	7	0	—	0.10 (0.19)			4	42.78	10	0	—	4.04 (15.14)
		8	43.64	8	0	—	0.07 (0.13)			8	66.47	10	0	—	4.03 (28.58)
		16	80.73	10	0	—	0.08 (0.18)			16	123.11	10	0	—	1.05 (2.03)
	1	1	6.27	0	0	0.01 (0.03)	0.04 (0.12)		1	1	7.13	0	0	0.04 (0.25)	0.27 (0.60)
		2	7.41	0	0	0.09 (0.71)	0.07 (0.26)			2	8.57	0	0	0.72 (4.70)	0.63 (2.01)
		4	11.07	0	0	0.35 (2.89)	0.05 (0.13)			4	17.30	5	0	—	2.44 (9.10)
		8	15.49	0	0	1.50 (9.63)	0.05 (0.14)			8	24.11	5	0	—	2.26 (12.63)
		16	31.08	5	0	—	0.04 (0.06)			16	44.65	10	0	—	1.01 (3.09)

C. Comparative Study of GS Versus PS

In our second experiment, we compare the performance of GS versus our proposed algorithm PS using problem set (I). The average time window width is measured by $\hat{E}(\bar{d}_j - r_j)/\hat{E}(p_j)$, where $\hat{E}(X)$ stands for the sample mean of a random variable X. This ratio is independent of the time unit in which the input data is given.

The results in Table II indicate that PS performs consistently well over the entire set of 400 problems. PS is able to solve all the test problems to optimality, including all the 50-job prob-

TABLE III
RESULTS ON PROBLEM SET (II): ALGORITHMS' SENSITIVITY TO TIME WINDOW WIDTH

n	W	$\frac{\dot{E}(d_j-r_j)}{\dot{E}(p_j)}$	No. Unsolved GS	No. Unsolved PS	CPU Seconds GS Mean (Max)	CPU Seconds PS Mean (Max)	n	W	$\frac{\dot{E}(d_j-r_j)}{\dot{E}(p_j)}$	No. Unsolved GS	No. Unsolved PS	CPU Seconds GS Mean (Max)	CPU Seconds PS Mean (Max)
20	150	5.96	0	0	0.00 (0.01)	0.01 (0.01)	40	150	6.08	0	0	0.00 (0.01)	0.06 (0.11)
	200	7.87	0	0	0.01 (0.02)	0.00 (0.01)		200	7.87	0	0	0.01 (0.02)	0.12 (0.29)
	250	10.65	0	0	0.02 (0.07)	0.00 (0.01)		250	9.53	0	0	0.06 (0.14)	0.21 (0.84)
	300	11.64	0	0	0.25 (2.11)	0.01 (0.01)		300	11.21	0	0	0.26 (0.70)	0.19 (0.42)
	400	15.86	0	0	0.66 (1.68)	0.01 (0.01)		400	16.66	4	0	7.29 (16.86)	0.15 (0.34)
	600	21.49	0	0	2.28 (10.85)	0.00 (0.01)		600	23.20	9	0	—	0.14 (0.39)
	800	30.06	0	0	4.83 (15.33)	0.00 (0.01)		800	32.38	10	0	—	0.06 (0.17)
30	150	5.59	0	0	0.00 (0.01)	0.02 (0.04)	50	150	5.93	0	0	0.01 (0.01)	0.13 (0.18)
	200	7.76	0	0	0.00 (0.01)	0.03 (0.07)		200	7.87	0	0	0.02 (0.04)	0.23 (0.58)
	250	10.05	0	0	0.05 (0.29)	0.03 (0.06)		250	9.41	0	0	0.08 (0.16)	0.40 (0.87)
	300	11.56	0	0	0.34 (1.62)	0.05 (0.11)		300	11.93	0	0	1.31 (6.76)	1.39 (5.22)
	400	16.33	0	0	1.98 (6.22)	0.06 (0.20)		400	16.84	4	0	11.65 (25.81)	0.79 (2.96)
	600	24.95	10	0	—	0.02 (0.07)		600	23.84	10	0	—	1.01 (6.14)
	800	30.68	10	0	—	0.02 (0.05)		800	31.17	10	0	—	0.22 (0.42)

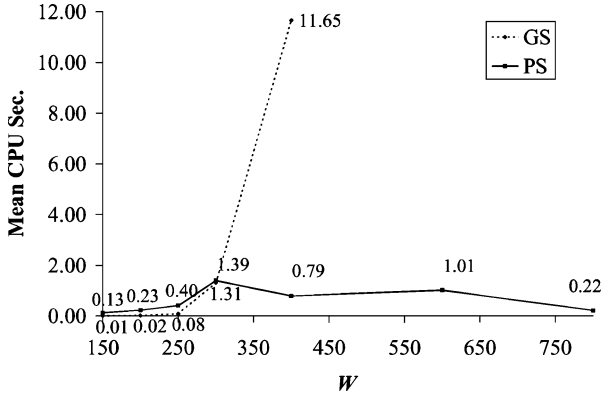


Fig. 4. Algorithms' sensitivity to average time window width ($n = 50$).

lems. By contrast, GS is unable to solve 152 problems; these unsolved problems occur with both $\alpha = 0.5$ and $\alpha = 1$, and some of them are as small as having 30 jobs. The computation time required by PS is small and does not have as much variability as that required by GS. However, GS has some advantage on problems with small time windows. It is almost always the case that GS either solves a problem in few seconds or has to abandon it due to excessive memory requirement.

The above experiment is repeated using problem set (II). Individual problems in this second problem set are similar to those problems with $\alpha = 1$ in problem set (I) in terms of the average window width and the distribution of release dates, but are generally less difficult. Because GS is able to solve a greater portion of the problems (213 out of 280), more insights can be drawn from the results regarding computation time. The results in Table III indicate that the average time window width has a much smaller impact on the computation time of PS than on the computation time of GS. For example, with n fixed at 50, the algorithms' sensitivity to the average time window width is depicted by Fig. 4. For $W \leq 300$, both algorithms perform quite well, but GS is even faster than PS. However, this occasional speed advantage of GS over PS occurs only when the computation time is fairly small (<1–2 s) for both algorithms [this observation is also consistent with the results on problem set (I)].

We also tested the algorithms using problem set (III), where the release dates are deadlines are not correlated. As indicated by the results in Table IV, this set of problems are much more challenging for both algorithms. However, the proposed PS algorithm fares considerably better than GS, since the latter experiences difficulty even for 30-job problems.

Overall, it is perhaps fair to say that our algorithm (PS) performs consistently well across different types of instances, and more importantly, it is more robust when faced with instances that take the dynamic programming algorithm (GS) a long time to solve.

VII. DISCUSSION AND FUTURE RESEARCH

In this paper, we introduced a lower bound for WCT and developed a fast algorithm to compute it in $O(n \log n)$ time. We also proposed several dominance conditions and used an effective node elimination technique, to curtail the size of the branch-and-bound search tree. Using these ingredients, we constructed a branch-and-bound solution procedure. The procedure was able to effectively solve test problems of up to 50 jobs within the time limit. Our method proves to be quite robust over a wide range of input data characteristics, compared to a dynamic programming method in the literature. Clearly, the techniques demonstrated here can be used to achieve superior results in solving $1|r_j|\sum w_j C_j$—a special case of WCT.

Admittedly, the proposed branch-and-bound algorithm follows the standard track of combinatorial branch-and-bound algorithms (as opposed to **LP**-based ones). However, our successful solution of WCT brings this paradigm into uncharted territories. Combinatorial branch-and-bound previously have not been applied to sequencing problems with nontrivial feasibility issues. Usually, all sequences are feasible, and in situations where there is infeasibility, all feasible sequences can be enumerated without ever running into infeasible ones (e.g., $1|\bar{d}_j|\sum w_j C_j$). The implication of infeasibility on lower bounds and dominance conditions has never been studied. Meanwhile, there are indications that researchers have at least pondered upon this due to its practical relevance (e.g., [2]).

In future research, we would incorporate the lower bound and solution method developed for WCT into solution procedures

TABLE IV
RESULTS ON PROBLEM SET (III): UNCORRELATED RELEASE DATES AND DEADLINES

n	α	γ	$\frac{\hat{E}(d_j-r_j)}{\hat{E}(p_j)}$	No. Unsolved		CPU Seconds		n	α	γ	$\frac{\hat{E}(d_j-r_j)}{\hat{E}(p_j)}$	No. Unsolved		CPU Seconds	
				GS	PS	GS Mean (Max)	PS Mean (Max)					GS	PS	GS Mean (Max)	PS Mean (Max)
20	0.5	1	12.51	0	0	0.24 (0.68)	0.03 (0.10)	40	0.5	1	25.76	9	0	—	6.05 (45.76)
		2	18.63	0	0	1.16 (5.87)	0.02 (0.04)			2	42.45	10	0	—	1.35 (3.07)
		4	36.13	0	0	3.93 (10.82)	0.01 (0.02)			4	79.18	10	0	—	0.77 (2.39)
		8	75.17	0	0	8.27 (15.78)	0.01 (0.02)			8	163.04	10	0	—	0.36 (1.46)
		16	167.08	1	0	26.07 (38.02)	0.01 (0.02)			16	317.80	10	0	—	0.37 (1.19)
	1	1	15.81	0	0	1.18 (4.20)	0.02 (0.09)		1	1	34.15	10	0	—	0.49 (1.48)
		2	24.03	0	0	1.65 (6.48)	0.01 (0.02)			2	52.13	10	0	—	1.06 (7.13)
		4	40.30	0	0	5.12 (11.75)	0.01 (0.02)			4	80.41	10	0	—	0.39 (2.21)
		8	77.68	0	0	9.66 (29.36)	0.01 (0.01)			8	153.58	10	0	—	0.36 (1.21)
		16	162.15	0	0	12.38 (30.81)	0.01 (0.02)			16	310.17	10	0	—	0.27 (0.74)
30	0.5	1	20.36	8	0	—	0.99 (6.37)	50	0.5	1	31.45	10	3	—	46.39 (120.01)
		2	30.81	10	0	—	0.18 (0.57)			2	56.08	10	1	—	16.05 (120.00)
		4	59.01	10	0	—	0.15 (0.57)			4	100.57	10	0	—	9.10 (51.28)
		8	116.51	10	0	—	0.09 (0.20)			8	197.80	10	0	—	0.86 (1.59)
		16	249.62	10	0	—	0.09 (0.26)			16	393.73	10	0	—	1.89 (5.66)
	1	1	24.45	9	0	—	0.16 (0.69)		1	1	41.90	10	0	—	1.05 (2.24)
		2	34.56	10	0	—	0.09 (0.18)			2	59.47	10	0	—	2.49 (16.60)
		4	58.38	10	0	—	0.11 (0.41)			4	109.72	10	0	—	3.61 (11.10)
		8	120.28	10	0	—	0.07 (0.20)			8	196.81	10	0	—	1.35 (5.52)
		16	246.35	10	0	—	0.06 (0.16)			16	387.60	10	0	—	1.27 (5.05)

for JSTIMP [32] and other job shop problems with inventory and cycle time-related objectives. Another promising direction is to utilize our tailored algorithm within a mathematical programming framework based on the Dantzig-Wolfe decomposition and column generation (e.g., [11], [36]–[38]).

Additionally, the isotone optimization problem defined in (8) can be extended by adding the following bound constraints on the variables:

$$x_j^0 \le x_j \le x_j^1 \qquad \forall j \tag{11}$$

where x_j^0 and x_j^1 are given lower and upper bounds on the variable x_j. It turns out that these bound constraints do not complicate the problem and can be handled easily. Specifically, we first solve the problem without taking into account the bounds; let $x_1, \ldots, x_n$ be the optimal solution found. The solution to the problem with bounds can be obtained in two passes.

Algorithm. Two-Pass

Step 0) Let $x_1, \ldots, x_n$ be an optimal solution to the problem without bound constraints.
Backward Pass:
Step 1) Set $x_n := \max\{x_n, x_n^0\}$.
Step 2) For $j = n-1, \ldots, 1$, set $x_j := \max\{x_j, x_{j+1}, x_j^0\}$.
Forward Pass:
Step 3) Set $x_1 := \min\{x_1, x_1^1\}$.
Step 4) For $j = 2, \ldots, n$, set $x_j := \min\{x_j, x_{j-1}, x_j^1\}$. If $x_j < x_j^0$ for any j, the chain constraints and bound constraints as a whole are not consistent, i.e., the problem is infeasible.

This procedure clearly runs in $O(n)$ time, and its correctness can be shown by a fairly elementary argument.

Finally, we would like to shed some light on the relationship between the isotone optimization problems and the timetabling algorithms in scheduling theory. The timetabling algorithms are also extensively researched and play an important role in scheduling with *nonregular* objectives. Take the earliness–tardiness problem, $1||\sum \mu_j(d_j - C_j)^+ + \nu_j(C_j - d_j)^+$, as an example, where for each job j, $\mu_j \ge 0$, $\nu_j \ge 0$, and d_j denotes the *due date* (which can be violated at a penalty). Suppose without loss of generality that we are given a sequence $(1, \ldots, n)$. The task of timetabling is to compute optimal completion times, C_j, $j = 1, \ldots, n$, that solve the following problem (C_j's are variables here):

$$\begin{aligned} \textbf{min} & \sum_{j=1}^{n} \mu_j(d_j - C_j)^+ + \nu_j(C_j - d_j)^+ \\ \textbf{s.t.}\ & 0 \le C_1 - p_1, \\ & C_{j-1} \le C_j - p_j,\ j = 2, \ldots, n. \end{aligned} \tag{12}$$

Quite a number of authors have studied variations of the timetabling problem (see [21] for an extensive list of references). $O(n^2)$-time timetabling algorithms are straightforward (e.g., the algorithm of Szwarc and Mukhopadhyay [34]). $O(n \log n)$-time implementations are often adapted from that of Garey *et al.* [17], who study the unit-weight case where $\mu_j = \nu_j = 1$ for all j.

In fact, the timetabling problem defined above can be converted into one just like (8) by the following linear transformation

$$x_j = C_{n+1-j} - \sum_{i=1}^{n+1-j} p_i \qquad \forall j.$$

This is a remarkable coincidence, considering the distinct origins of the isotone optimization problems and timetabling problem. It should be noted that the $f_j(x_j)$ function associated with the timetabling problem corresponds to the special case depicted in Fig. 1(a), because $\mu_j \ge 0$ and $\nu_j \ge 0$ in the objective function. Consequently, the optimal objective value of the timetabling problem is bounded, whereas the isotone

problem originated from our lower bound calculation can be unbounded. After the conversion, constraint (12) results in a lower bound constraint on x_n, which can be handled using the backward pass discussed above. As a result, our Specialized Cascading Descent algorithm, together with the backward pass, offers a new $O(n \log n)$ procedure for timetabling.

Although our algorithm is not the first $O(n \log n)$ procedure for timetabling, it does offer a more practical alternative to the algorithm of Garey *et al.* and its variations. This is because Garey *et al.*'s algorithm (the $O(n \log n)$-time version) requires the so-called "meld" operation—i.e., the merger of two heaps to form a new heap. Because there can be as many as $O(n)$ melds, each meld has to be done in $O(\log n)$ in order to achieve the $O(n \log n)$-time bound. Unfortunately, this means that the aforementioned mergeable heaps must be used. Due to the difficulty and overhead in the implementation of mergeable heaps, less efficient $O(n^2)$ procedures are often used instead (see, e.g., [22]). Our proposed procedure, on the other hand, does not require meld operations, and therefore avoids this difficulty.

ACKNOWLEDGMENT

The authors are grateful to the Associate Editor and two anonymous referees, whose critiques and suggestions have greatly influenced the content and presentation of this paper, in particular, to the Associate Editor who brought isotone optimization problems to their attention, and to one referee who led them to the study of problem set (III).

REFERENCES

[1] R. H. Ahmadi and U. Bagchi, Just-in-time scheduling in single machine systems, Dep. of Manag., Univ. of Texas, Austin, 1986.

[2] ——, "Minimizing job idleness in deadline constrained environments," *Oper. Res.*, vol. 40, pp. 972–985, 1992.

[3] R. K. Ahuja and J. B. Orlin, "A fast scaling algorithm for minimizing separable convex functions subject to chain constraints," *Oper. Res.*, vol. 49, no. 5, pp. 784–789, 2001.

[4] K. R. Baker, *Introduction to Sequencing and Scheduling*. New York: Wiley, 1974.

[5] J. F. Bard, K. Venkatraman, and T. A. Feo, "Single machine scheduling with flow time and earliness penalties," *J. Global Optim.*, vol. 3, no. 3, pp. 289–309, 1993.

[6] H. Belouadah, M. E. Posner, and C. N. Potts, "Scheduling with release dates on a single machine to minimize total weighted completion time," *Discr. Appl. Math.*, vol. 36, no. 3, pp. 213–231, 1992.

[7] L. Bianco and S. Ricciardelli, "Scheduling of a single machine to minimize total weighted completion time subject to release times," *Nav. Res. Logist. Q.*, vol. 29, pp. 151–167, 1982.

[8] J. Carlier, "The one-machine sequencing problem," *Eur. J. Oper. Res.*, vol. 11, pp. 42–47, 1982.

[9] N. Chakravarti, "Isotonic median regression: a linear programming approach," *Math. Oper. Res.*, vol. 14, no. 2, pp. 303–308, 1989.

[10] S. Chand and H. Schneeberger, "Single machine scheduling to minimize weighted earliness subject to no tardy jobs," *Eur. J. Oper. Res.*, vol. 34, pp. 221–230, 1988.

[11] Z.-L. Chen and W. B. Powell, "Solving parallel machine scheduling problems by column generation," *INFORMS Comp.*, vol. 11, pp. 78–94, 1999.

[12] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1997.

[13] M. Dell'Amico, S. Martello, and D. Vigo, "Minimizing the sum of weighted completion times with unrestricted weights," *Discr. Appl. Math.*, vol. 63, no. 1, pp. 25–41, 1995.

[14] H. Emmons, "A note on a scheduling problem with dual criteria," *Nav. Res. Logist. Q.*, vol. 22, pp. 615–616, 1975.

[15] S. French, *Sequencing and Scheduling: An Introduction to the Mathematics of the Job Shop*. Chichester, U.K.: Horwood, 1982.

[16] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1979.

[17] M. R. Garey, R. Tarjan, and G. Wilfong, "One-processor scheduling with symmetric earliness and tardiness penalties," *Math. Oper. Res.*, vol. 13, pp. 330–348, 1988.

[18] S. Gélinas and F. Soumis, "A dynamic programming algorithm for single machine scheduling with ready times," *Ann.Oper. Res.*, vol. 69, pp. 135–156, 1997.

[19] R. Graham, E. Lawler, J. Lenstra, and A. H. G. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: A survey," *Ann. Discr. Math.*, vol. 5, pp. 287–326, 1979.

[20] A. M. A. Hariri and C. N. Potts, "Algorithm for single machine sequencing with release dates to minimize total weighted completion time," *Discr. Appl. Math.*, vol. 5, pp. 99–109, 1983.

[21] J. J. Kanet and V. Sridharan, "Scheduling with inserted idle time: Problem taxonomy and literature review," *Oper. Res.*, vol. 48, no. 1, pp. 99–110, 2000.

[22] C. Koulamas, "Single-machine scheduling with time windows and earliness/tardiness penalties," *Eur. J. Oper. Res.*, vol. 91, no. 1, pp. 190–202, 1996.

[23] B. J. Lageweg, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Minimizing maximum lateness on one machine: computational experience and some applications," *Stat. Neerland.*, vol. 30, pp. 25–41, 1976.

[24] P. D. Martin and D. B. Shmoys, "A new approach to computing optimal schedules for the job-shop scheduling problem," *Lect. Notes Comp. Sci.*, vol. 1084, pp. 389–403, 1996.

[25] Y. Pan, "An improved branch and bound algorithm for single machine scheduling with deadlines to minimize total weighted completion time," *Oper. Res. Lett.*, vol. 31, no. 6, pp. 492–496, 2003.

[26] ——, "Production Scheduling for Suppliers in the Extended Enterprise," Ph.D. dissertation, Dep. of Ind. Eng., Univ. of Wisconsin-Madison, Madison, WI, 2003.

[27] Y. Pan and L. Shi, "Branch-and-bound algorithms for solving hard instances of the one-machine sequencing problem," *Eur. J. Oper. Res.*, 2004.

[28] P. M. Pardalos, G. Xue, and L. Yong, "Efficient computation of an isotonic median regression," *Appl. Math. Lett.*, vol. 8, no. 2, 1995.

[29] M. Posner, "Minimizing weighted completion times with deadlines," *Oper. Res.*, vol. 33, no. 3, pp. 562–574, 1985.

[30] C. N. Potts and L. N. van Wassenhove, "Algorithm for single machine sequencing with deadlines to minimize total weighted completion time," *Eur. J. Oper. Res.*, vol. 12, pp. 379–387, 1983.

[31] T. Robertson and P. Waltman, "On estimating monotone parameters," *Ann. Math. Stat.*, vol. 39, pp. 1030–1039, 1968.

[32] L. Shi and Y. Pan, "Minimizing job shop inventory with on-time delivery guarantees," *J. Syst. Sci. Syst. Eng.*, vol. 12, no. 4, pp. 449–469, 2003.

[33] W. E. Smith, "Various optimizers for single-stage production," *Nav. Res. Logist. Q.*, vol. 3, pp. 59–66, 1956.

[34] W. Szwarc and S. K. Mukhopadhyay, "Optimal timing schedules in earliness–tardiness single machine sequencing," *Nav. Res. Logist. Q.*, vol. 42, pp. 1109–1114, 1995.

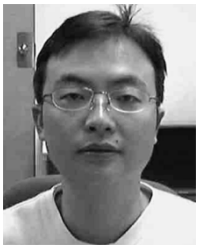
[35] R. Tarjan, *Data Structures and Network Algorithms*. Philadelphia, PA: SIAM, 1983, vol. 44.

[36] J. M. van den Akker, J. A. Hoogeveen, and S. L. van de Velde, "Parallel machine scheduling by column generation," *Oper. Res.*, vol. 47, no. 6, pp. 862–872, 1999.

[37] J. M. van den Akker, C. A. J. Hurkens, and M. W. P. Savelsbergh, "Time-indexed formulations for machine scheduling problems: column generation," *INFORMS Comp.*, vol. 12, no. 2, pp. 111–124, 2000.

[38] M. van den Akker, H. Hoogeveen, and S. van de Velde, "Combining column generation and lagrangean relaxation to solve a single-machine common due date problem," *INFORMS Comp.*, vol. 14, no. 1, pp. 37–51, 2002.

[39] L. N. van Wassenhove, "Special-Purpose algorithms for one-machine sequencing problems with single and composite objectives," Ph.D. dissertation, Katholieke Univ., Leuven, Belgium, 1979.



Yunpeng Pan received the B.S. degree in computational mathematics from Nanjing University, Nanjing, China, in 1995, the M.S. degree in operations research from the University of Delaware, Newark, in 1998, and the M.S. degree in computer sciences, and the Ph.D. degree in industrial engineering from the University of Wisconsin-Madison, in 2001 and 2003, respectively.

He is a Research Associate in the Department of Industrial and Systems Engineering, University of Wisconsin-Madison. His current research interest is hybrid combinatorial and mathematical programming-based approaches to practical shop scheduling problems that arise from extended enterprise supply chain networks. His work appears in *Operational Research Letters*, IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING, the *European Journal of Operational Research*, and the *Journal of Systems Science and Systems Engineering*.

Dr. Pan is a member of INFORMS.

Leyuan Shi (M'92) received the B.S. degree in mathematics from Nanjing Normal University, Nanjing, China, in 1982, the M.S. degree in applied mathematics from Tsinghua University, Beijing, China, in 1985, and the M.S. degree in engineering, and Ph.D. degree in applied mathematics from Harvard University, Cambridge, MA, in 1990, and 1992, respectively.

She is a Professor in the Department of Industrial and Systems Engineering, University of Wisconsin-Madison. She has been involved in undergraduate and graduate teaching, as well as research and professional service. Her research is devoted to the theory and applications of large-scale optimization algorithms, discrete event simulation and modeling and analysis of discrete dynamic systems. She has published many papers in these areas. Her work has appeared in *Discrete Event Dynamic Systems*, *Operational Research*, *Management Science* and in IEEE and IIE Transactions.

Dr. Shi is currently a member of the editorial board for *Journal of Manufacturing and Service Operations Management*, and is an Associate Editor of *Journal of Discrete Event Dynamic Systems*. She is a member of INFORMS.