Computing the worst-case due dates violations with budget uncertainty

Igor Malheiros, Artur Pessoa, Michael Poss, Anand Subramanian

Computing the worst-case due dates violations with budget uncertainty

Igor Malheiros[a,b], Artur Pessoa[c], Michaël Poss[a], Anand Subramanian[d]

[a]*LIRMM, Université de Montpellier, CNRS, Rue Ada 156, 34095, Montpellier, France*
[b]*Atoptima, Rue Marc Sangnier 2, Bègles, 33130, France*
[c]*TEP, Universidade Federal Fluminense, Rua Passo da Pátria, 156, 24210-240, Niterói, Brazil*
[d]*DSC, Universidade Federal da Paraíba, Rua dos Escoteiros s/n, 58055-000, João Pessoa, Brazil*

Abstract

We study the problem of maximizing the violation of due dates when considering either the total violation, or the number of jobs that are tardy. We consider either classical completion times or those involving time warp. The four problems are motivated by the solution of scheduling problems with release and due dates/deadlines and processing time uncertainty, and also routing problems with (soft) time windows and travel time uncertainty. We provide polynomial dynamic programming algorithms for the four problems.

Keywords: robust optimization, release date, due date, deadlines, budget uncertainty

1. Introduction

We consider a set of n jobs scheduled in the order $[n] = \{1, 2, \ldots, n\}$. For each job, we are given a release date r_i, a due date d_i, a nominal processing time $\bar{p}_i$ and a deviation $\hat{p}_i$. The actual processing time of job i is given by $\bar{p}_i + \delta_i \hat{p}_i$, where $\delta_i \in \{0, 1\}$ is a binary optimization variable that satisfies the constraint

$$\sum_{i \in [n]} \delta_i \leq \Gamma, \tag{1}$$

for a given positive integer Γ. The goal of the optimization problems studied in this paper is to find a binary δ satisfying (1) that maximizes the violation of due dates. In what follows, we use the notation $\Delta_\Gamma = \left\{\delta \in \{0,1\}^n \;\middle|\; \sum_{i \in [n]} \delta_i \leq \Gamma\right\}$ to represent the feasibility set of δ, which is widely used in the robust routing and scheduling literature, e.g. [1, 2, 3, 4, 5].

We consider more specifically two models for computing the completion times of each job. In the *basic* model, the completion time of job i, denoted $t_i^B(\delta)$, is given by $t_1^B(\delta) = r_1 + \bar{p}_1 + \delta_1 \hat{p}_1$ and

$$t_i^B(\delta) = \max(t_{i-1}^B(\delta), r_i) + \bar{p}_i + \delta_i \hat{p}_i \tag{2}$$

for each $i \in \{2, \ldots, n\}$. Observe that the maximum in (2) models the wait until r_i in case job $i-1$ is completed earlier than r_i. In the *time-warp* model, the completion time is given by $t_1^{TW}(\delta) = r_1 + \bar{p}_1 + \delta_1 \hat{p}_1$ and

$$t_i^{TW}(\delta) = \max(\min(t_{i-1}^{TW}(\delta), d_{i-1}), r_i) + \bar{p}_i + \delta_i \hat{p}_i \tag{3}$$

for each $i \in \{2, \ldots, n\}$. The innermost minimum sets the completion time back to d_{i-1} in case violation occurs, the so-called time-warp, while the inner maximum models the waiting time as in the basic model.

For each of these two models $M \in \{B, TW\}$, we seek to maximize the total violation

$$\sum_{i \in [n]} f(t_i^M(\delta) - d_i)$$

for two choices of functions f:

- $f_0(x) = 0$ for $x \leq 0$ and $f_0(x) = 1$ for $x > 0$;
- $f_1(x) = \max(0, x)$, also denoted by $(x)^+$;

where f_0 indicates whether a job is tardy (or fails) while f_1 considers the tardiness of the job. We denote the resulting functions of δ as $\phi_i^M(\delta) = f_0(t_i^M(\delta) - d_i)$ and $\tau_i^M(\delta) = f_1(t_i^M(\delta) - d_i)$, and $\phi^M(\delta) = \sum_{i \in [n]} \phi_i^M(\delta)$ and $\tau^M(\delta) = \sum_{i \in [n]} \tau_i^M(\delta)$, respectively. Therefore, this paper studies the optimization problems defined on set Δ_Γ, for the objective functions τ^B, τ^{TW}, ϕ^B, and ϕ^{TW}.

1.1. Motivation

Basically, the four optimization problems studied in this paper are motivated by the exact and heuristic solution of robust scheduling problems with release and due dates/deadlines and processing time uncertainty, and also robust routing problems with (soft) time windows and travel time uncertainty.

Let us start with model B and consider a robust scheduling problem seeking to minimize the tardiness. One exact approach to such problems is based on a row-and-column generation algorithm [6], the separation problem of which corresponds exactly to maximizing τ^B, see [4]. Similar models arise in the vehicle routing literature with release and due dates, where one wishes to minimize the total tardiness [7, 8]. Hence, the separation problems studied herein could be leveraged to develop algorithms also for the robust counterparts of the above routing problems. The above row-and-column generation algorithm could be extended to the problem minimizing the number of tardy jobs, proved $\mathcal{NP}$-hard in [9], which leads to maximizing ϕ^B.

Second, these scheduling problems, as well as their routing counterparts, are also often addressed heuristically through local-search algorithms [10]. In this case, one needs to compute the objective of the schedules and routes explored by the algorithm, which amounts again to maximizing τ^B and ϕ^B.

Model TW is useful in local search algorithms applied to scheduling and routing problems with release dates and deadlines, also called time windows in the routing literature. Because the search algorithm frequently modifies subpaths of the current solution, it is crucial to design efficient methods to assess violation levels. A popular approach is to evaluate this violation without accumulating the delays along the entire path, known as *time-warp relaxation*, see [11, 12]. Assessing the violation of the current solution without accumulation leads to the maximization of functions τ^{TW} and ϕ^{TW}.

1.2. Contributions

Checking whether the optimal solution cost of any of the four problems is equal to 0 can be answered in polynomial time by using the dynamic programming algorithm proposed in [1], and subsequently leveraged to provide compact formulations for routing and scheduling problems with deadlines [2, 3]. However, prior to this paper, it was unknown whether optimizing functions $\tau^B, \tau^{TW}, \phi^B$, and ϕ^{TW} could be done in polynomial time, and these were typically solved by using dynamic programming algorithms running in pseudo-polynomial time or Mixed-Integer Linear Programming formulations, e.g. [13, 14, 4].

The contribution of this work is thus to prove that these four problems are polynomially solvable, proposing three different dynamic programming algorithms detailed in the following three sections. While the algorithms for τ^B and τ^{TW} rely on different ideas, presented in Sections 3 and 4, respectively, those for ϕ^B and ϕ^{TW} are nearly identical for the two models, and presented in the following section.

2. Number of failures

The total number of failures among the n jobs is bounded by n. This allows one to design a dynamic programming algorithm to compute $V(i, \gamma, \alpha)$, the maximum completion time of job i given that $\alpha \in [i]$ failures have occurred along the jobs in $[i]$ and using $\gamma \in [\Gamma] \cup \{0\}$ deviations so far. We consider next the basic case ϕ^B and propose a recursion for computing $V(i, \gamma, \alpha)$.

The recursion returning the value of $V(i, \gamma, \alpha)$ is split into different cases, depending on the value of i, γ, α. Notice the presence of modulus for some of the values $|V(i, \gamma, \alpha)|$ when subject to an upper bound. This implies that if $V(i, \gamma, \alpha) = -\infty$, meaning that the state has not been reached, the condition is always violated. In particular, these absolute values are not needed when the condition requires to be greater than some value since in that case $V(i, \gamma, \alpha) = -\infty$ immediately invalidates the condition. In addition to the following cases, we also have the initial condition that $V(i, \gamma, \alpha) = -\infty$ if $\alpha > i$ as well as $V(i, \gamma, \alpha) = -\infty$ if $\gamma > i$.

$i = 1$ **and** $\gamma = 0$ **and** $\alpha = 0$**:**

$$\begin{cases} \textbf{if } r_1 + \bar{p}_1 \leq d_1: & r_1 + \bar{p}_1 \\ \textbf{else}: & -\infty \end{cases}$$

$i = 1$ **and** $\gamma = 0$ **and** $\alpha = 1$**:**

$$\begin{cases} \textbf{if } r_1 + \bar{p}_1 \leq d_1: & -\infty \\ \textbf{else}: & r_1 + \bar{p}_1 \end{cases}$$

$i = 1$ **and** $\gamma = 1$ **and** $\alpha = 0$**:**

$$\begin{cases} \textbf{if } r_1 + \bar{p}_1 + \hat{p}_1 \leq d_1: & r_1 + \bar{p}_1 + \hat{p}_1 \\ \textbf{else}: & -\infty \end{cases}$$

$i = 1$ **and** $\gamma = 1$ **and** $\alpha = 1$**:**

$$\begin{cases} \textbf{if } r_1 + \bar{p}_1 + \hat{p}_1 \leq d_1: & -\infty \\ \textbf{else}: & r_1 + \bar{p}_1 + \hat{p}_1 \end{cases}$$

$i \geq 2$ **and** $\gamma = 0$ **and** $\alpha = 0$**:**

$$\begin{cases} \textbf{if } |V(i-1,0,0)| + \bar{p}_i \leq d_i: & \max\{r_i, V(i-1,0,0) + \bar{p}_i\} \\ \textbf{else}: & -\infty \end{cases}$$

$i \geq 2$ **and** $\gamma = 0$ **and** $1 \leq \alpha \leq i$**:**

$$\begin{cases} \textbf{if } V(i-1,0,\alpha-1) + \bar{p}_i > d_i: & V(i-1,0,\alpha-1) + \bar{p}_i \\ \textbf{elsif } |V(i-1,0,\alpha)| + \bar{p}_i \leq d_i: & \max\{r_i, V(i-1,0,\alpha) + \bar{p}_i\} \\ \textbf{else}: & -\infty \end{cases}$$

$i \geq 2$ **and** $1 \leq \gamma \leq i$ **and** $\alpha = 0$**:**

$$
\begin{cases}
\textbf{if } |V(i-1,\gamma,0)| + \bar{p}_i \le d_i \textbf{ and} \\
\quad |V(i-1,\gamma-1,0)| + \bar{p}_i + \hat{p}_i \le d_i : \\
\quad \max\{r_i, V(i-1,\gamma,0) + \bar{p}_i, V(i-1,\gamma-1,0) + \bar{p}_i + \hat{p}_i\} \\
\textbf{elsif } |V(i-1,\gamma-1,0)| + \bar{p}_i + \hat{p}_i \le d_i : \\
\quad \max\{r_i, V(i-1,\gamma-1,0) + \bar{p}_i + \hat{p}_i\} \\
\textbf{elsif } |V(i-1,\gamma,0)| + \bar{p}_i \le d_i : \\
\quad \max\{r_i, V(i-1,\gamma,0) + \bar{p}_i\} \\
\textbf{else}: \ -\infty
\end{cases}
$$

$i \ge 2$ **and** $1 \le \gamma \le i$ **and** $1 \le \alpha \le i$**:**

$$
\begin{cases}
\textbf{if } V(i-1,\gamma-1,\alpha-1) + \bar{p}_i + \hat{p}_i > d_i \textbf{ or} \\
\quad V(i-1,\gamma,\alpha-1) + \bar{p}_i > d_i : \\
\quad\quad \max\{V(i-1,\gamma,\alpha-1) + \bar{p}_i, \\
\quad\quad\quad V(i-1,\gamma-1,\alpha-1) + \bar{p}_i + \hat{p}_i\} \\
\textbf{elsif } |V(i-1,\gamma-1,\alpha)| + \bar{p}_i + \hat{p}_i \le d_i \textbf{ and} \\
\quad\quad |V(i-1,\gamma,\alpha)| + \bar{p}_i \le d_i : \\
\quad\quad \max\{r_i, V(i-1,\gamma,\alpha) + \bar{p}_i, \\
\quad\quad\quad V(i-1,\gamma-1,\alpha) + \bar{p}_i + \hat{p}_i\} \\
\textbf{elsif } |V(i-1,\gamma-1,\alpha)| + \bar{p}_i + \hat{p}_i \le d_i \\
\quad\quad \max\{r_i, V(i-1,\gamma-1,\alpha) + \bar{p}_i + \hat{p}_i\} \\
\textbf{elsif } |V(i-1,\gamma,\alpha)| + \bar{p}_i \le d_i : \\
\quad\quad \max\{r_i, V(i-1,\gamma,\alpha) + \bar{p}_i\} \\
\textbf{else}: \ -\infty
\end{cases}
$$

Observe that in the last brace, it may happen for example that $V(i-1,\gamma-1,\alpha-1) + \bar{p}_i + \hat{p}_i > d_i$ and $|V(i-1,\gamma-1,\alpha)| + \bar{p}_i + \hat{p}_i \le d_i$ simultaneously, in which case the maximum completion time is given by $V(i-1,\gamma-1,\alpha-1) + \bar{p}_i + \hat{p}_i$ and we fall into the first case. Similarly, if $V(i-1,\gamma,\alpha-1) + \bar{p}_i > d_i$ and $|V(i-1,\gamma,\alpha)| + \bar{p}_i \le d_i$ simultaneously, we fall again in the first case as the max is given by $V(i-1,\gamma,\alpha-1) + \bar{p}_i$.

Finally, we obtain the maximum total numbers of failures α as follows:

$$
\max\{\alpha \mid V(n,\gamma,\alpha) \neq -\infty \text{ for some } \gamma \in \{0,\dots,\Gamma\}\}.
$$

Therefore, it is possible to implement a dynamic programming algorithm to maximize the total number of failures with the time complexity of $O(n^2\Gamma)$.

We conclude this section by mentioning how to extend the above dynamic programming algorithm to the case with time-warp, ϕ^{TW}. In this case, it is enough to replace the value of $V(i,\gamma,\alpha)$ by d_i when a failure occurs at a given job.

3. Tardiness with the basic model

We omit the superscript B of τ^B, τ_i^B, and t^B throughout the section, and compute τ using a dynamic programming algorithm described next. For each $i \in [n]$ and $\gamma \in [\Gamma] \cup \{0\}$, we denote by Δ_γ^i the projection of Δ_γ on the components $[i]$, and for each $i \in [n]$, we denote by $\tau^i(\delta) = \sum_{j\in[i]} \tau_j(\delta)$ the sum over the tardiness for the first i jobs only. Observe that $\tau^i(\delta)$ depends only on the first i components of δ, so we assume next that the argument of τ^i belongs to Δ_γ^i for some γ.

Our dynamic programming algorithm is based on the value-function

$$
F(i,\gamma,\beta) = \max_{\delta\in\Delta_\gamma^i} \tau^i(\delta) + \beta \cdot t_i(\delta),
$$

where $\beta \in [n] \cup \{0\}$. Note that the optimal solution to the problem minimizing τ over Δ_Γ is given by $F(n,\Gamma,0)$. Furthermore, for $i = 1$, the value-function can be computed as $F(1,\gamma,\beta) =$

$$
\begin{cases}
(r_1 + p_1(\gamma) - d_1)^+ + \beta \cdot (r_1 + p_1(\gamma)), & \text{if } \gamma \in \{0,1\} \\
-\infty & \text{otherwise}
\end{cases},
$$

where we use the notation $p_i(\delta) = \bar{p}_i + \delta\hat{p}_i$. We obtain a general recurrence for the remaining cases by relying on the recursive definition (2) and the definition $\tau_i(\delta) = (t_i(\delta) - d_i)^+$. Denoting the concatenation of δ and δ' as $\delta\|\delta'$, we see that $F(i,\gamma,\beta) =$

$$
\begin{aligned}
&\max_{\delta'\in\{0,1\}} \max_{\delta\in\Delta_{\gamma-\delta'}^{i-1}} \tau^{i-1}(\delta) + (t_i(\delta\|\delta') - d_i)^+ + \beta \cdot t_i(\delta\|\delta') \\
&= \max_{\delta'\in\{0,1\}} \max_{\delta\in\Delta_{\gamma-\delta'}^{i-1}} \\
&\qquad \max\left(\tau^{i-1}(\delta) + (\beta+1)t_i(\delta\|\delta') - d_i, \tau^{i-1}(\delta) + \beta \cdot t_i(\delta\|\delta')\right) \\
&= \max_{\delta'\in\{0,1\}} \max_{\delta\in\Delta_{\gamma-\delta'}^{i-1}} \max\Big(\tau^{i-1}(\delta) + (\beta+1)t_{i-1}(\delta) + (\beta+1)p_i(\delta') - d_i, \\
&\qquad \tau^{i-1}(\delta) + (\beta+1)r_i + (\beta+1)p_i(\delta') - d_i, \\
&\qquad \tau^{i-1}(\delta) + \beta \cdot t_{i-1}(\delta) + \beta \cdot p_i(\delta'), \\
&\qquad \tau^{i-1}(\delta) + \beta \cdot r_i + \beta \cdot p_i(\delta')\Big) \\
&= \max_{\delta'\in\{0,1\}} \max\Big(\max_{\delta\in\Delta_{\gamma-\delta'}^{i-1}} \{\tau^{i-1}(\delta) + (\beta+1)t_{i-1}(\delta)\} + (\beta+1)p_i(\delta') - d_i, \\
&\qquad \max_{\delta\in\Delta_{\gamma-\delta'}^{i-1}} \{\tau^{i-1}(\delta)\} + (\beta+1)r_i + (\beta+1)p_i(\delta') - d_i, \\
&\qquad \max_{\delta\in\Delta_{\gamma-\delta'}^{i-1}} \{\tau^{i-1}(\delta) + \beta \cdot t_{i-1}(\delta)\} + \beta \cdot p_i(\delta'), \\
&\qquad \max_{\delta\in\Delta_{\gamma-\delta'}^{i-1}} \{\tau^{i-1}(\delta)\} + \beta \cdot r_i + \beta \cdot p_i(\delta')\Big) \\
&= \max_{\delta'\in\{0,1\}} \max\Big(F(i-1,\gamma-\delta',\beta+1) + (\beta+1)p_i(\delta') - d_i, \\
&\qquad F(i-1,\gamma-\delta',0) + (\beta+1)r_i + (\beta+1)p_i(\delta') - d_i, \\
&\qquad F(i-1,\gamma-\delta',\beta) + \beta \cdot p_i(\delta'), \\
&\qquad F(i-1,\gamma-\delta',0) + \beta \cdot r_i + \beta \cdot p_i(\delta')\Big)
\end{aligned}
$$

Overall, all values of $F(i,\gamma,\beta)$ can be computed in $O(n^2\Gamma)$.

Remark 1. *The above ideas can be extended to provide polynomial-time algorithms for robust lot-sizing problems with budget uncertainty when considering scenario generation algorithms [13, 14]. Stated shortly, the separation problem of this case involves two main differences with the problem considered so far. First, the completion time of job i is replaced by the total demand at the end of period i, so t_i for $i \in \{2, \dots, n\}$ satisfies the recurrence $t_i(\delta) = t_{i-1}(\delta) + \bar{p}_i + \delta_i \hat{p}_i - x_i$, where x_i is the production during period i and $t_1(\delta) = \bar{p}_1 + \delta_1 \hat{p}_1 - x_1$. Notice the productions are assumed constant for the separation problem as they are variables of the corresponding master problem. Second, the cost at every period is given by the formula* $\max(b_i t_i(\delta), -s_i t_i(\delta))$*, where s_i and b_i are given non-negative parameters. Adapting the above ideas to this case leads to considering negative and positive values for β, yielding a pseudo-polynomial time algorithm for the problem, where the pseudo-polynomial factors involve s_i and b_i. In contrast, the pseudo-polymonial time algorithm proposed by [13] depends on $\hat{p}_i$.*

4. Tardiness with time-warp

Again we omit the superscript TW of τ^{TW}, τ_i^{TW}, and t^{TW} throughout the section, and study the optimization problem

$$\max_{\delta \in \Delta_\Gamma} \tau(\delta). \tag{4}$$

We first handle the case where intermediate release dates are not restrictive, formally

$$r_i \leq r_1 = r^*, \quad \forall i = 2, \dots, n. \tag{5}$$

We have the following proposition for this case.

Proposition 1. *Assume (5) holds, let $\delta \in \Delta_\Gamma$ and i^* be the largest i such that $\tau_i(\delta) > 0$. Then, $\tau(\delta) = r^* - d_{i^*} + \sum_{i \in [i^*]} (\bar{p}_i + \delta_i \hat{p}_i)$.*

Proof. Let $I = \{i \in [n] \mid \tau_i(\delta) > 0\}$. Note that, for all $i \in I \setminus \{i^*\}$, we have

$$t_i(\delta) = t_{i-1}(\delta) + \bar{p}_i + \delta_i \hat{p}_i - \tau_i(\delta) \tag{6}$$

Moreover, for all $i \in [i^*] \setminus I$, we have

$$t_i(\delta) = t_{i-1}(\delta) + \bar{p}_i + \delta_i \hat{p}_i \tag{7}$$

Combining (6) for all $i \in I$ and (7) for all $i \in [i^*] \setminus I$, we obtain that

$$t_{i^*}(\delta) = r^* + \sum_{i \in [i^*-1]} (\bar{p}_i + \delta_i \hat{p}_i) - \sum_{i \in I \setminus \{i^*\}} \tau_i(\delta).$$

Then, $\tau_{i^*}(\delta) = t_{i^*}(\delta) - d_{i^*}$, proving the result. □

Proposition 1 shows that the objective value of any solution only depends on the last release date that generates a time warp. Hence, an effective approach to solve the problem under non-restrictive intermediate release dates is to probe for the last time warp index i^*. From Proposition 1, for each candidate index $i \in [n]$, the best solution for the case where $i^* = i$ is to set $\delta_j = 1$ for the $\min\{i, \Gamma\}$ largest values of $\hat{p}_j$, for $j \in [i]$, and $\delta_j = 0$ for the remaining components. This leads to the following proposition:

Proposition 2. *Problem (4) under assumption (5) can be solved in $O(n \log \Gamma)$.*

Proof. The algorithm performs one pass for $i \in [n]$, keeping track of $\bar{P} = \sum_{j \in [i]} \bar{p}_j$ and $\hat{P}$ is equal to the sum of the $\min\{i, \Gamma\}$ largest values of $\hat{p}_j$. Updating $\bar{P}$ takes $O(1)$ time but updating $\hat{P}$ requires that the terms of the sum be also maintained, which takes $O(\log \Gamma)$. The objective value in each case is computed in $O(1)$ time as $r^* + \bar{P} + \hat{P} - d_i$ following Proposition 1. □

In order to solve the complete problem, we use a reformulation more suited to the above results. Let σ_i be an additional binary decision variable for $i \in [n]$ that models whether or not we apply the release date for job i. We adapt definition (3) and define $\tilde{t}_1(\delta, \sigma) = r_1 + \bar{p}_1 + \delta_1 \hat{p}_1$ (observe $\tilde{t}_1(\delta, \sigma)$ does not depend on σ), and for each $i \in \{2, \dots, n\}$

$$\tilde{t}_i(\delta, \sigma) = \begin{cases} \min(d_{i-1}, \tilde{t}_{i-1}(\delta, \sigma)) + \bar{p}_i + \delta_i \hat{p}_i & \text{if } \sigma_i = 0 \\ r_i + \bar{p}_i + \delta_i \hat{p}_i & \text{if } \sigma_i = 1 \end{cases}$$

We adapt τ_i similarly and define

$$\tilde{\tau}_i(\delta, \sigma) = (\tilde{t}_i(\delta, \sigma) - d_i)^+$$

for each $i \in [n]$, as well as $\tilde{\tau}(\delta, \sigma) = \sum_{i \in [n]} \tilde{\tau}_i(\delta, \sigma)$. Consider consequently the following optimization problem

$$\max_{\delta \in \Delta_\Gamma, \sigma \in \{0,1\}^n} \tilde{\tau}(\delta, \sigma). \tag{8}$$

Proposition 3. *Let (δ^*, σ^*) be an optimal solution of (8). Then, δ^* is an optimal solution to (4) and $\tau(\delta^*) = \tilde{\tau}(\delta^*, \sigma^*)$.*

Proof. Let (δ^*, σ^*) be an optimal solution of (8). We define $\bar{\sigma} \in \{0, 1\}^n$ by fixing $\bar{\sigma}_i = 1$ if $t_i(\delta^*) \leq r_i$ and 0 otherwise. Note that $(\delta^*, \bar{\sigma})$ is feasible for (8) and $\tilde{\tau}(\delta^*, \bar{\sigma}) = \tau(\delta^*)$. Hence,

$$\tilde{\tau}(\delta^*, \sigma^*) \geq \tilde{\tau}(\delta^*, \bar{\sigma}) = \tau(\delta^*). \tag{9}$$

Moreover, since $\bar{\sigma}_i = 1$ if and only if $\tilde{t}_i(\delta^*, \bar{\sigma}) = \tilde{t}_i(\delta^*) \leq r_i$, $\tilde{t}_i(\delta^*, \bar{\sigma}) \geq \tilde{t}_i(\delta^*, \sigma)$ for any $\sigma \in \{0,1\}^n$. As a result, $\tilde{\tau}(\delta^*, \bar{\sigma}) \geq \tilde{\tau}(\delta^*, \sigma^*)$, and therefore, $\tau(\delta^*) \geq \tilde{\tau}(\delta^*, \sigma^*)$. Together with (9), we obtain that $\tau(\delta^*) = \tilde{\tau}(\delta^*, \sigma^*)$.

Finally, assume δ^* is not optimal for (4). Then, there exists $\bar{\delta}$ such that $\tau(\bar{\delta}) > \tau(\delta^*)$, and repeating the above procedure, we construct $\bar{\sigma}$ such that $\tilde{\tau}(\bar{\delta}, \bar{\sigma}) = \tau(\bar{\delta}) > \tau(\delta^*) = \tilde{\tau}(\delta^*, \sigma^*)$. □

The algorithm we propose next for (8) relies on the following observation. On the one hand, setting σ_i to 0 relaxes the release date of job i, in line with (5). On the other hand, setting σ_i to 1 splits the problem into the one involving jobs in $[i-1]$, and the subsequent one with jobs $[n] \setminus [i-1]$. For the sake of deriving a dynamic programming algorithm, let us denote by $T(i, \gamma)$ the maximum total tardiness obtained for jobs in $[i-1]$ setting at most γ values of δ to 1 among the indices of $[i]$; in particular, $T(n+1, \Gamma)$ is equal to the optimal solution cost of problem (8). Let also $T^u(i, j, \gamma)$ be the maximum total tardiness obtained for the jobs in $\{i, \ldots, j-1\}$, using r_{i-1} for the release date of job $i-1$, ignoring release dates for jobs in $\{i+1, \ldots, j-1\}$, and setting at most γ values of δ to 1 among the indices of $\{i, \ldots, j-1\}$. Following the above discussion on the value of σ_i, $T(i, \gamma)$ satisfies the following recursion:

$$T(i,\gamma) = \begin{cases} 0, & \text{if } i = 1 \\ \max\limits_{j \in [i], \theta \in [\gamma] \cup \{0\}} T(j,\theta) + T^u(j, i, \gamma - \theta) & \text{if } i > 0 \end{cases},$$

where we assume that $T^u(i, i, \gamma) = 0$ for $i \in [n]$, and, $\gamma = 0, \ldots, \Gamma$. The maximization over j in the recursion seeks the optimal index $j \in [i]$ such that $\sigma_j = 1$ and $\sigma_k = 0$ for $k \in \{j+1, \ldots, i\}$.

Proposition 4. *$T(n+1, \Gamma)$ can be computed in $O(n^2\Gamma^2)$.*

Proof. From the procedure proposed in the proof of Proposition 2, $T^u(i, j, \gamma)$ can be computed in $O(n \log \Gamma)$ for a fixed i, and γ, and $j = i, \ldots, n$. Thus, computing the whole table of T^u takes $O(n^2 \Gamma \log \Gamma)$. Then, it remains to fill the $O(n\Gamma)$ values of T, each one taking $O(n\Gamma)$ time. This gives the stated complexity. □

References

[1] A. Agra, M. Christiansen, R. M. V. Figueiredo, L. M. Hvattum, M. Poss, C. Requejo, The robust vehicle routing problem with time windows, Comput. Oper. Res. 40 (2013) 856–866. URL: https://doi.org/10.1016/j.cor.2012.10.002. doi:10.1016/J.COR.2012.10.002.

[2] P. A. Munari, A. Moreno, J. D. L. Vega, D. J. Alem, J. Gondzio, R. Morabito, The robust vehicle routing problem with time windows: Compact formulation and branch-price-and-cut method, Transp. Sci. 53 (2019) 1043–1066. URL: https://doi.org/10.1287/trsc.2018.0886. doi:10.1287/TRSC.2018.0886.

[3] M. Bold, M. Goerigk, A compact reformulation of the two-stage robust resource-constrained project scheduling problem, Comput. Oper. Res. 130 (2021) 105232. URL: https://doi.org/10.1016/j.cor.2021.105232. doi:10.1016/J.COR.2021.105232.

[4] M. Silva, M. Poss, N. Maculan, Solution algorithms for minimizing the total tardiness with budgeted processing time uncertainty, Eur. J. Oper. Res. 283 (2020) 70–82. URL: https://doi.org/10.1016/j.ejor.2019.10.037. doi:10.1016/J.EJOR.2019.10.037.

[5] B. Tadayon, J. C. Smith, Algorithms and complexity analysis for robust single-machine scheduling problems, J. Sched. 18 (2015) 575–592. URL: https://doi.org/10.1007/s10951-015-0418-0. doi:10.1007/S10951-015-0418-0.

[6] B. Zeng, L. Zhao, Solving two-stage robust optimization problems using a column-and-constraint generation method, Oper. Res. Lett. 41 (2013) 457–461. URL: https://doi.org/10.1016/j.orl.2013.05.003. doi:10.1016/J.ORL.2013.05.003.

[7] B. C. Shelbourne, M. Battarra, C. N. Potts, The vehicle routing problem with release and due dates, INFORMS J. Comput. 29 (2017) 705–723. URL: https://doi.org/10.1287/ijoc.2017.0756. doi:10.1287/IJOC.2017.0756.

[8] W. Yang, L. Ke, D. Z. Wang, J. S. L. Lam, A branch-price-and-cut algorithm for the vehicle routing problem with release and due dates, Transportation Research Part E: Logistics and Transportation Review 145 (2021) 102167. URL: http://dx.doi.org/10.1016/j.tre.2020.102167. doi:10.1016/j.tre.2020.102167.

[9] M. Bougeret, A. A. Pessoa, M. Poss, Single machine robust scheduling with budgeted uncertainty, Oper. Res. Lett. 51 (2023) 137–141. URL: https://doi.org/10.1016/j.orl.2023.01.007. doi:10.1016/J.ORL.2023.01.007.

[10] T. Vidal, T. G. Crainic, M. Gendreau, C. Prins, Time-window relaxations in vehicle routing heuristics, J. Heuristics 21 (2015) 329–358. URL: https://doi.org/10.1007/s10732-014-9273-y. doi:10.1007/S10732-014-9273-Y.

[11] Y. Nagata, O. Bräysy, W. Dullaert, A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows, Comput. Oper. Res. 37 (2010) 724–737. URL: https://doi.org/10.1016/j.cor.2009.06.022. doi:10.1016/J.COR.2009.06.022.

[12] T. Vidal, T. G. Crainic, M. Gendreau, C. Prins, A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows, Comput. Oper. Res. 40 (2013) 475–489. URL: https://doi.org/10.1016/j.cor.2012.07.018. doi:10.1016/J.COR.2012.07.018.

[13] A. Agra, M. C. Santos, D. Nace, M. Poss, A dynamic programming approach for a class of robust optimization problems, SIAM J. Optim. 26 (2016) 1799–1823. URL: https://doi.org/10.1137/15M1007070. doi:10.1137/15M1007070.

[14] D. Bienstock, N. Özbay, Computing robust basestock levels, Discret. Optim. 5 (2008) 389–414. URL: https://doi.org/10.1016/j.disopt.2006.12.002. doi:10.1016/J.DISOPT.2006.12.002.