Single machine adversarial bilevel scheduling problems

Single machine adversarial bilevel scheduling problems

Vincent t'Kindt, Federico Della Croce, Alessandro Agnetis

Single machine adversarial bilevel scheduling problems

Vincent T'kindt[a,*], Federico Della Croce[b], Alessandro Agnetis[c]

[a] *Université de Tours, Laboratoire d'Informatique Fondamentale et Appliquée (EA 6300), EMR CNRS 7002 ROOT, 64 avenue Jean Portalis, 37200 Tours, France,*
[b] *DIGEP, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy, CNR, IEIIT, Torino, Italy*
[c] *Universita degli Studi di Siena, Dipartimento di Ingegneria dell'Informazion e Scienze Matematiche, via Roma 56, 53100 Siena, Italy.*

Abstract

We consider single machine scheduling problems in the context of adversarial bilevel optimization where two agents, the leader and the follower, take decisions on the same jobset and the leader acts first with the aim of inducing the worst possible solution for the follower. Thus, the follower schedules the jobs in order to optimize a given criterion. The considered criteria are the total completion time, the total weighted completion time, the maximum lateness and the number of tardy jobs. We focus on adversarial bilevel scheduling with job selection and data modification. In the case with job selection, the leader selects a fixed cardinality subset of the jobs that the follower schedules next. In the case with data modification, the leader can modify some of the data (processing times, due dates, weights), given a limited budget Q. Thus, the follower schedules the set of jobs with modified data. For all the considered criteria either we provide polynomial-time algorithms or show that they can be solved in the worst-case in pseudo-polynomial time.

Keywords: Scheduling, Single Machine, Bilevel Optimization, Complexity results

1. Introduction

Scheduling theory deals with the allocation of a given set of jobs to resources over time while minimizing some criteria. Even if it is usually assumed that a single decision maker (called *agent*) takes all scheduling decisions, a significant part of the literature considers that several agents may compete for scheduling their own set of jobs. For example, this is notably the case of multiagent scheduling (*e.g.* Agnetis et al. (2014)) or game theory based scheduling (*e.g.* Pascual et al. (2009)). In this paper we investigate another form of interactions between agents related to bilevel optimization (Dempe et al. (2015)) and leading to bilevel scheduling problems. Bilevel optimization has strong links with other topics like Stackelberg games, set-valued optimization problems (Dempe (2003)) or min-max robust optimization (see Aissi et al. (2009) for a survey of such problems). We first

*Corresponding author
Email addresses: tkindt@univ-tours.fr (Vincent T'kindt), federico.dellacroce@polito.it (Federico Della Croce), agnetis@diism.unisi.it (Alessandro Agnetis)

focus on the main concepts of bilevel optimization before turning to a review of bilevel scheduling literature.

A bilevel discrete optimization problem can be defined as follows:

$$Min_{x \in X^L} \Big(f^L\big(x, y^* = argmin_{y \in X^F} f^F(x,y)\big)\Big), \tag{1}$$

with X^L (resp. X^F) the set of solution of the first (resp. second) level, and f^L (resp. f^F) the objective function to minimize at the first (resp. second) level. Variations with functions to maximize can naturally be introduced. A simple way to present bilevel optimization problems is to consider a first agent, called the *leader* that can take some decisions $x \in X^L$ and wants to minimize its objective function f^L. Next, at the second level the other agent, called the *follower*, makes its own decisions from set $y \in X^F$ taking into account x and minimizing f^F. However, both f^L and f^F depend on the complete set of decisions (x, y). As minimizing the follower's objective function may lead to several optimal solutions, two classes of problems can be introduced (Dempe (2003)): optimistic and pessimistic bilevel problems. In an *optimistic* bilevel problem, it is assumed that the follower returns, among all optimal solutions minimizing f^F, the one that leads to the smallest value of f^L. On the contrary, in a *pessimistic* bilevel problem, it is assumed that the follower returns its optimal solution that is the worst for the leader's objective function. Another class of bilevel optimization problems can be met in the literature, which we call hereafter *adversarial* bilevel problems. In this setting, the leader takes decision so that the optimal solution of the follower's problem is the worst possible. As an example, the bilevel knapsack problem introduced by DeNegre (2011) falls into this category.
From a complexity point of view, bilevel optimization problems can be much harder than traditional single level ones (Woeginger (2021)). This is notably the case of the bilevel adversarial knapsack problem of DeNegre which was shown to be Σ_2^p-hard (Caprara et al. (2014)). And unless $\mathcal{P} = \mathcal{NP}$, we have $\mathcal{P} \subset \mathcal{NP} \subset \Sigma_2^p$ (Stockmeyer (1977)) providing the information that some Σ_2^p-hard problems are harder than any $\mathcal{NP}$-hard problem. It turns out that defining the complexity status of bilevel optimization problem can be really challenging.

To the best of our knowledge, very little results are known in the literature on bilevel scheduling problems. Karlof and Wang (1996) first considered a flowshop scheduling problem with operators where the leader determines the schedule of operators to minimize the sum of job completion times while the follower determines the schedule of jobs to minimize the makespan. An extension of this problem to the case of fuzzy processing times is tackled by Abass (2005). Kovacs and Kis (2011) introduce a general constraint programming formulation for bilevel scheduling problems and applied it to solve an optimistic bilevel single machine problem in which the leader selects the set of jobs the follower next schedules. Following the standard scheduling notation, this problem is denoted by $1|OPT - n, r_j, \tilde{d}_j| \sum_j w_j^L x_j, \sum_j w_j^F C_j^F$, with $OPT - n$ meaning that the optimistic setting is considered and the leader selects the number of jobs n to schedule, minimizing the cost of the selected ones $\sum_j w_j^L x_j$. The follower sequences the jobs so that their weighted sum of completion times is minimum. Kis and Kovacs (2012) consider

both the $P|OPT - A_k|\sum_j w_j^L C_j^L, \sum_j w_j^F C_j^F$ and $P|PES - A_k|\sum_j w_j^L C_j^L, \sum_j w_j^F C_j^F$ problems: they correspond to the optimistic (OPT) and pessimistic (PES) settings of the problem where the leader defines the set of jobs A_k assigned to any machine k while the follower sequence them on each machine. The problem is shown to be strongly $\mathcal{NP}$-hard. Kis and Kovacs (2012) consider again the $1|OPT - n, r_j, \tilde{d}_j|\sum_j w_j^L x_j, \sum_j w_j^F C_j^F$ problem and show that it is weakly $\mathcal{NP}$-hard.

In this paper we focus on a set of basic bilevel adversarial scheduling problems. Let us consider a single machine available for processing a set of n jobs j defined by a processing time p_j and, depending on the problem, by a due date d_j or a weight w_j. For a given schedule s, let $C_j(s)$ be the completion time of job j. It is well-known in the literature that several single machine problems can be solved in polynomial time. But what happens when considering them in a framework of adversarial bilevel optimization? In section 2 we consider scheduling problems for which the leader must select a subset of jobs that the follower schedules. In section 3 we focus on scheduling problems for which the leader has a budget to modify data that the follower next uses to build an optimal schedule to his problem. Section 4 provides conclusions and potential future research directions. The results presented in these sections are summarized in Table 1.

Problem	**Section**	**Note**
Polynomially solvable problems		
$1\|ADV - n\|\sum_j C_j^F$	2.1	$O(N\log(N))$ time.
$1\|ADV - n\|L_{max}^F$	2.3	$O(N\log(N))$ time.
$1\|ADV - n\|\sum_j U_j^F$	2.4	$O(n^2N)$ time.
$1\|ADV - p\|\sum_j C_j^F$	3.1	$O(n\log(n))$ time.
$1\|ADV - p, q_j \in \mathbb{R}\|\sum_j w_j^F C_j^F$ and $1\|ADV - w, q_j \in \mathbb{R}\|\sum_j w_j^F C_j^F$	3.2	$O(n\log(n))$ time.
$1\|ADV - p\|L_{max}^F$	3.3	$O(n)$ time.
$1\|ADV - d, d_j = d\|\sum_j U_j^F$	3.4	$O(n\log(n))$ time.
$1\|ADV - p, d_j = d\|\sum_j U_j^F$	3.4	$O(n\log(n))$ time.
$\mathcal{NP}$-hard problems		
$1\|ADV - n, \mathcal{L}\|-$	2.4	$\mathcal{L}$ is the (known) list of tardy jobs. It is a decision problem.
Open problems		
$1\|ADV - n\|\sum_j w_j C_j^F$	2.2	$O(nN\sum_j p_j)$ time by dynamic programming.
$1\|ADV - p, q_j \in \mathbb{N}\|\sum_j w_j^F C_j^F$ and $1\|ADV - w, q_j \in \mathbb{N}\|\sum_j w_j^F C_j^F$	3.2	An optimal solution does not necessarily preserve the initial WSPT order.
$1\|ADV - d\|L_{max}^F$	3.3	
$1\|ADV - p\|\sum_j U_j^F$	3.4	$O(n^4 p_{max}^3)$ time by dynamic programming.
$1\|ADV - d\|\sum_j U_j^F$	3.4	$O(n^4(p_{max} + d_{max})d_{max}^2)$ time by dynamic programming.

N refers to the number of initial jobs in a selection problem / n refers to the number of jobs to schedule by the follower

Table 1: Complexity status of some bilevel single machine scheduling problems

2. Adversarial bilevel scheduling with job selection

Assume that a set of N jobs are available for the leader who has to select exactly $n < N$ jobs that the follower next schedules minimizing a given criterion f^F. Leader's goal is to select the jobs so that the optimal value of $f^F \in \{\sum_j C_j^F, \sum_j w_j C_j^F, L_{max}^F, \sum_j U_j\}$ is maximum. The following subsections discuss the existence of polynomial-time algorithms for each criterion minimized by the follower.

2.1. Sum of completion times

The problem tackled in this section is referred to as $1|ADV - n|\sum_j C_j^F$. It is well-known that the classic $1||\sum_j C_j$ problem can be solved in $O(n\log(n))$ time by sorting jobs by non-decreasing value of their processing time (SPT rule). We show in the next theorem that this rule can be exploited to solve the bilevel problem still in polynomial time.

Theorem 1. *The $1|ADV - n|\sum_j C_j^F$ problem can be solved in $O(N\log(N))$ time as follows:*

1. *The leader selects the $n \leq N$ jobs with the largest processing times,*
2. *The follower applies the SPT rule on these n jobs.*

Proof. For a given sequence s of n jobs, $\sum_j C_j^F$ can be rewritten as:

$$\sum_j C_j^F(s) = \sum_{j=1}^{n}(n-j+1)p_{s(j)},$$

with $s(j)$ the job in position j is s. Assume now that the leader selects n jobs such that they are not the ones with the largest processing times and let $s = \alpha k \beta \gamma$ be the SPT sequence of these jobs, where α, β and γ are partial sequences and k is a job. Now, let ℓ be a job not previously selected by the leader with $p_\ell > p_k$. We denote by $s' = \alpha\beta\ell\gamma$ the obtained SPT sequence when swapping k and ℓ. Assume that:

$$\sum_j C_j^F(s) > \sum_j C_j^F(s').$$

We have:

$$\begin{cases} \sum_j C_j^F(s) = & \sum_{j=1}^{|\alpha|}(n-j+1)p_{\alpha(j)} + (n-|\alpha|)p_k + \sum_{j=1}^{|\beta|}(n-|\alpha|-j)p_{\beta(j)} \\ & + \sum_{j=1}^{|\gamma|}(n-|\alpha|-|\beta|-j)p_{\gamma(j)} \\ \sum_j C_j^F(s') = & \sum_{j=1}^{|\alpha|}(n-j+1)p_{\alpha(j)} + \sum_{j=1}^{|\beta|}(n-|\alpha|-j+1)p_{\beta(j)} + (n-|\alpha|-|\beta|)p_\ell \\ & + \sum_{j=1}^{|\gamma|}(n-|\alpha|-|\beta|-j)p_{\gamma(j)} \end{cases}$$

$$\sum_j C_j^F(s) > \sum_j C_j^F(s') \quad \Leftrightarrow \quad (n-|\alpha|)p_k > \sum_{j=1}^{|\beta|} p_{\beta(j)} + (n-|\alpha|-|\beta|)p_\ell,$$

as $p_k \leq p_j$, $\forall j \in \beta$:

$$\Leftrightarrow \quad (n-|\alpha|-|\beta|)(p_k - p_\ell) > \sum_{j=1}^{|\beta|} p_{\beta(j)} - |\beta|p_k > 0$$

which is a contradiction with the fact that $p_k < p_\ell$. Consequently, to make the SPT order having the maximum $\sum_j C_j^F$ value, the leader must select the n jobs with the largest processing times.□

2.2. Weighted sum of completion times

Consider the case where each job j has a weight w_j and the follower minimizes the weighted sum of completion times $\sum_j w_j C_j^F$. This problem is referred to as $1|ADV - n|\sum_j w_j C_j^F$. It is well-known that the classic $1||\sum_j w_j C_j$ problem can be solved in $O(n\log(n))$ time by sorting jobs by non-decreasing value of the ratio $\frac{p_j}{w_j}$ (WSPT rule). Thus, whenever the leader selects n jobs, the follower sequences them according to the WSPT rule. We first show that the natural intuition that the leader should select the n jobs with largest ratio, does not lead to an optimal solution of the bilevel problem. Let us consider the following $N = 4$ jobs problem with $p = [10; 1; 3; 1]$ and $w = [1000; 2; 4; 1]$. Suppose $n = 3$ and let $s = (2, 3, 4)$ be the solution obtained if the leader selects the 3 jobs with largest ratio $\frac{p_j}{w_j}$. We have $\sum_j w_j C_j^F(s) = 23$. But if the leader selects jobs $\{1, 2, 3\}$ then WSPT gives $s' = (1, 2, 3)$ and $\sum_j w_j C_j^F(s) = 10078$.

However, We can show the following results.

Lemma 1. *Let be two jobs k and ℓ such that:*

1. $\frac{p_k}{w_k} < \frac{p_\ell}{w_\ell}$, *and*
2. *There are at least n jobs j such that $\frac{p_k}{w_k} < \frac{p_j}{w_j}$, and*
3. *There are strictly less than n jobs j such that $\frac{p_\ell}{w_\ell} < \frac{p_j}{w_j}$.*

Then, the two following conditions hold:

(C1) if $w_\ell \geq w_k$ then there exists an optimal solution to the bilevel problem in which ℓ is selected instead of k.

(C2) if $w_\ell < w_k$ and $p_k \geq p_\ell$ then there exists an optimal solution to the bilevel problem in which k is selected instead of ℓ.

Proof. First, let us provide an analysis common to both conditions. Let be $s = \alpha k \beta \gamma$ and $s' = \alpha \beta \ell \gamma$ two schedules ordered according to WSPT rule, with α, β and γ partial sequences. So, we have:

- $\frac{p_i}{w_i} \leq \frac{p_k}{w_k} \leq \frac{p_j}{w_j} \leq \frac{p_\ell}{w_\ell} \leq \frac{p_u}{w_u}$, $\forall i \in \alpha$, $\forall j \in \beta$ and $\forall u \in \gamma$, (A)
- $\frac{p_k}{w_k} < \frac{p_\ell}{w_\ell}$. (B)

Besides, $\sum_j w_j C_j^F$ on schedules s and s' can be written as follows, where $j \to i$ denotes all the jobs consecutively scheduled from j to i (included):

$$\begin{cases} \sum_j w_j C_j^F(s) = & \sum_{j\in\alpha} p_j\big(w_j + \sum_{j\to i\in\alpha} w_i + w_k + W_\beta + W_\gamma\big) + p_k(w_k + W_\beta + W_\gamma) \\ & + \sum_{j\in\beta} p_j\big(\sum_{j\to i\in\beta} w_i + w_j + W_\gamma\big) + \sum_{j\in\gamma} p_j(\sum_{j\to i\in\gamma} w_i + w_j) \\ \sum_j w_j C_j^F(s') = & \sum_{j\in\alpha} p_j\big(w_j + \sum_{j\to i\in\alpha} w_i + w_\ell + W_\beta + W_\gamma\big) + p_\ell(w_\ell + W_\gamma) \\ & + \sum_{j\in\beta} p_j\big(\sum_{j\to i\in\beta} w_i + w_j + w_\ell + W_\gamma\big) + \sum_{j\in\gamma} p_j(\sum_{j\to i\in\gamma} w_i + w_j) \end{cases}$$

with W_π the sum of the weights of jobs in sequence π. Let us define $\Delta = \sum_j w_j C_j^F(s) - \sum_j w_j C_j^F(s')$. We have:

$$\Delta = (w_k - w_\ell)P_\alpha + W_\gamma(p_k - p_\ell) + p_k W_\beta + p_k w_k - p_\ell w_\ell - w_\ell P_\beta,$$

with P_π the sum of processing times of job in sequence π.
Now, let us focus on condition (C1): we assume that $w_\ell \geq w_k$ and we show that $\Delta > 0$ implies a contradiction.

$$\Delta > 0 \quad \Leftrightarrow \quad (w_k - w_\ell)P_\alpha + W_\gamma(p_k - p_\ell) + p_k W_\beta + p_k w_k - p_\ell w_\ell - w_\ell P_\beta > 0$$

Due to (A) we can write: $W_\beta \leq P_\beta \frac{w_k}{p_k}$,

$$\Rightarrow \quad (w_k - w_\ell)(P_\alpha + P_\beta) + W_\gamma(p_k - p_\ell) > p_\ell w_\ell - p_k w_k$$

but due to (B) and our assumption $w_\ell \geq w_k$, we necessarily have $p_\ell > p_k$ and the left-hand side of the inequality is negative while the right-hand side is positive. This contradicts the fact that $\Delta > 0$ and k cannot be selected instead of ℓ.
Now, let us focus on condition (C2): we assume that $w_\ell < w_k$ and $p_k \geq p_\ell$, and we show that $\Delta < 0$ implies a contradiction.

$$\Delta < 0 \quad \Leftrightarrow \quad (w_k - w_\ell)P_\alpha + W_\gamma(p_k - p_\ell) + p_k W_\beta + p_k w_k - p_\ell w_\ell - w_\ell P_\beta < 0$$

Due to (A) we can write: $P_\beta \leq W_\beta \frac{p_\ell}{w_\ell}$,

$$\Rightarrow \quad (w_k - w_\ell)P_\alpha + (W_\gamma + W_\beta)(p_k - p_\ell) + p_k w_k - p_\ell w_\ell < 0$$

but due to our assumptions $w_\ell < w_k$ and $p_k \geq p_\ell$,the left-hand side is positive. This is a contradiction with $\Delta < 0$ and ℓ cannot be selected instead of k.□

From lemma 1, we can evince that whenever we have $\frac{p_k}{w_k} < \frac{p_\ell}{w_\ell}$, $w_k > w_\ell$ and $p_k < p_\ell$, we cannot decide whether or not one job is preferable to the other.
To solve the $1|ADV - n|\sum_j w_j C_j^F$ problem we can propose a pseudo-polynomial time algorithm as follows. Assume that jobs are indexed such that $\frac{p_1}{w_1} \leq \frac{p_2}{w_2} \leq ... \leq \frac{p_n}{w_n}$ and let $\mathcal{WC}[j,k,\mathcal{C}]$ be the optimal value of the bilevel problem when the leader has selected k jobs among the first j ones and their makespan is equal to $\mathcal{C}$. We have:

$$\mathcal{WC}[j,k,\mathcal{C}] = \max\Big(\underbrace{\mathcal{WC}[j-1,k,\mathcal{C}]}_{j \text{ is not selected}}; \underbrace{\mathcal{WC}[j-1,k-1,\mathcal{C}-p_j] + w_j\mathcal{C}}_{j \text{ is selected}}\Big)$$

with $\mathcal{WC}[j,k,\mathcal{C}] = -\infty$, whenever $k > j$ or $\mathcal{C} < 0$ or ($k = 0$ and $\mathcal{C} > 0$), and $\mathcal{WC}[0,0,0] = 0$.
To solve the $1|ADV - n|\sum_j w_j C_j^F$ problem we need to determine the value $\mathcal{C} \in \{1, ..., \sum_{j=1}^N p_j\}$ such that $\mathcal{WC}[N,n,\mathcal{C}]$ is maximum. This dynamic programming requires $O(Nn\sum_j p_j)$ time and space.

The complexity status of the $1|ADV - n|\sum_j w_j C_j^F$ problem is still open, even if the existence of a pseudo-polynomial time algorithm prevents it from being $\mathcal{NP}$-hard in the strong sense.

2.3. Maximum lateness

In this section, we assume that each job j has a due date d_j and the follower minimizes the maximum lateness $L_{max}^F = \max_{j=1..n}(L_j^F)$ with $L_j^F = C_j^F - d_j$ for a given schedule. This problem is referred to as $1|ADV - n|L_{max}^F$. Without loss of generality, we assume that jobs are indexed following the non-decreasing order of their due date (EDD order), *i.e.* $d_1 \leq ... \leq d_N$. We denote by L_{max}^{EDD} the L_{max} value of schedule $s^{EDD} = (1, ..., N)$:

whenever $n < N$, the optimal solution value of the $1|ADV - n|L^F_{max}$ problem is no more than L^{EDD}_{max}.

First, let us consider a job $j \leq n$. If there exists an optimal solution to the $1|ADV - n|L^F_{max}$ problem in which this job gives the value of L^F_{max}, then, it is necessarily processed in position j and let $s_j = (1, ..., n)$ be the associated schedule. The rationale is that in this position its completion time is maximum.
Next, let us consider a job $j > n$. If there exists an optimal solution in which this job gives the value of L^F_{max}, then, it is necessarily processed in position n. Let P_j be such that $|P_j| = (n-1)$, $P_j = \{k < j\}$ and $\nexists \ell \notin P_j, \ell < j$, such that $\exists k \in P_j$ with $p_\ell > p_k$, i.e., P_j contains the $(n-1)$ longest jobs having index smaller than j. Let the schedule associated to job j be $s_j = s_{P_j}//\{j\}$, i.e. s_{P_j} followed by j, with s_{P_j} the schedule built by sorting jobs in P_j according to their index number (EDD rule).

Theorem 2. *The schedule $s^* = argmax_{1 \leq j \leq N}(L_{max}(s_j))$ is an optimal solution to the $1|ADV - n|L^F_{max}$ problem. It can be computed in $O(N \log(N))$ time.*

Proof. Let s^* be an optimal schedule to the $1|ADV - n|L^F_{max}$ problem and let k be the job giving the value L^F_{max} in s^* (break ties by choosing the one with the smallest position). The jobs in s^* are sorted by increasing value of their index number, and we distinguish between two cases, depending on the position of job k.
First, assume that job $k \leq n$. Then, job k is necessarily in position k otherwise a non selected job $\ell < k$, *i.e.* $\ell \notin s^*$, could be swapped with a job $i > k$, $i \in s^*$, and scheduled before k : this leads to increase L_k and thus contradicts the fact that s^* is optimal. Besides, k is preceded by the $(k-1)$ jobs with smallest due dates, as the follower applies the EDD rule, and followed by any $(n-k)$ jobs with largest due dates.
Second, assume that job $k > n$. Following the same reasoning than in the first case, we can show that job k is necessarily processed in the last position of s^*, i.e. position n. As the follower applies the EDD rule, job k is necessarily preceded by $(n-1)$ jobs with smaller due dates. Besides, to have a maximum L_k value we select, among the $(k-1)$ smallest due date jobs, the ones with the largest processing times, i.e. jobs in P_k.
As we enumerate all schedules s_j with $1 \leq j \leq N$, we necessarily compute s^*.

The complexity result follows from a careful implementation of the algorithm: build all s_j's starting from $j = 1$ and compute P_j from P_{j-1} by a simple update. If $j < n$, $P_j = P_{j-1} \cup \{j-1\}$. Otherwise, remove from P_{j-1} the smallest processing time job p_ℓ and add p_{j-1} if $p_{j-1} > p_\ell$.□

2.4. Number of tardy jobs

In this section, we assume that each job j has a due date d_j and the follower minimizes the number of tardy jobs $\sum_j U^F_j$ with $U^F_j = 1$ if $C^F_j > d_j$ and 0 otherwise. This problem is referred to as $1|ADV - n|\sum_j U^F_j$. Without loss of generality we assume that jobs are indexed following the non-decreasing order of their due date (EDD order), i.e. $d_1 \leq ... \leq d_N$. Whenever the n jobs have been selected by the leader, the follower minimizes the number of tardy jobs by means of Moore's algorithm (Moore (1968)) or, equivalently, by Lin and Wang's algorithm (Lin and Wang (2007)).

Let us denote by s^{Moore} the schedule obtained by applying Moore's algorithm on the N jobs. Roughly speaking, this algorithm works as follows: at iteration k, schedule job k after the partial schedule α which only contains early jobs. If job k is early set $\alpha = \alpha//\{k\}$ and iterate. If job k is late then select job k_{max} from $\alpha//\{k\}$ that has the largest processing time. Set $\alpha = \alpha//\{k\}$, remove job k_{max} from α and iterate. Finally, all the removed jobs are scheduled late at the end of α. We denote by $\sum_j U_j^{Moore}$ the $\sum_j U_j$ value of schedule s^{Moore}. We have the following result.

Lemma 2. *Let k_{last} be the last job detected late in Moore's algorithm. If $k_{last} \leq n$, then the optimal solution of the $1|ADV-n|\sum_j U_j^F$ problem is obtained by selecting the n first jobs in EDD order.*

Proof. Let us denote by $\sum_j U_j^{F*}$ the optimal solution value of problem $1|ADV - n|\sum_j U_j^F$ for a given instance. It is straightforward that $\sum_j U_j^{F*} \leq \sum_j U_j^{Moore}$ as $\sum_j U_j^{F*}$ is computed on a subset of the N jobs. As $k_{max} \leq n$, by selecting jobs $\{1, ..., k_{max}, k_{max}+1, ...n\}$ the follower's optimal solution will have $\sum_j U_j^{F*} = \sum_j U_j^{Moore}$. □

The question whether the $1|ADV - n|\sum_j U_j^F$ problem can or cannot be solved in polynomial time when Lemma 2 does not hold, is answered hereafter. We first consider that the leader has selected n jobs and we want to check if there exists a solution for the follower which has ϵ_U tardy jobs. This problem can be solved by a dynamic programming algorithm which is a simple specialization of the one proposed in Lawler (1990) for the $1|r_j, pmtn|\sum_j w_j U_j$ problem. Let $\mathcal{C}^0(j, \epsilon_U)$ be the value of the smallest makespan when the j first jobs in EDD order have been scheduled and ϵ_U of them are tardy. We have:

$$\mathcal{C}^0(j,\epsilon_U) = \min\Big(\underbrace{\mathcal{C}^0(j-1,\epsilon_U-1)}_{j \text{ is scheduled tardy}}; \underbrace{\mathcal{C}^0(j-1,\epsilon_U)+p_j}_{\text{if } \mathcal{C}^0(j-1,\epsilon_U)+p_j \leq d_j;\ +\infty \text{ otherwise}}\Big)$$

with $\mathcal{C}^0(j, \epsilon_U) = +\infty$ whenever $\epsilon_U < 0$ or $\epsilon_U > j$, and $\mathcal{C}^0(0, 0) = 0$. Notice that this dynamic programming algorithm requires $O(n^2)$ time and space.
Now, for the $1|ADV - n|\sum_j U_j^F$ problem we can generalize the above recursion to take into account the leader's decisions. Let $\mathcal{C}(j, k, \epsilon_U)$ be the value of the smallest makespan when k jobs among the j first ones are selected and ϵ_U of them are tardy. We have:

$$\mathcal{C}(j,k,\epsilon_U) = \max\left(\underbrace{\mathcal{C}(j-1,k,\epsilon_U)}_{j \text{ is not selected}}; \underbrace{\min\big(\mathcal{C}(j-1,k-1,\epsilon_U-1); \mathcal{C}(j-1,k-1,\epsilon_U)+p_j\big)}_{j \text{ is selected}}\right)$$

with $\mathcal{C}(j, k, \epsilon_U) = +\infty$ whenever $\epsilon_U < 0$ or $\epsilon_U > j$, $\mathcal{C}(j, k, \epsilon_U) = -\infty$ whenever $j < k$, and $\mathcal{C}(0, 0, 0) = 0$. Notice that the two terms inside the min() correspond to j being scheduled tardy and, respectively, being scheduled early. Besides, whenever the min() returns $+\infty$, this value must be transformed into $-\infty$ during the recursions.
Solving the bilevel problem requests to determine the greatest value ϵ_U such that $\mathcal{C}(N, n, \epsilon_U) \neq -\infty$. Notice that this dynamic programming algorithm requires $O(Nn^2)$ time and space which implies that the $1|ADV - n|\sum_j U_j^F$ problem can be solved in

polynomial time.

To conclude this section, we prove that a related problem is $\mathcal{NP}$-complete. Assume that the list $\mathcal{L}$ of tardy jobs is imposed, that is the problem turns to a decision problem where the leader has to select $(n - |\mathcal{L}|)$ jobs so that when the follower schedules the n jobs, only those in $\mathcal{L}$ are tardy. This problem is denoted by $1|ADV - n, \mathcal{L}|-$.

Theorem 3. *Let $\mathcal{L}$ be the set of jobs which have to be scheduled tardy by the follower. The $1|ADV - n, \mathcal{L}|-$ problem is $\mathcal{NP}$-complete.*

PROOF. Let us consider the following weakly $\mathcal{NP}$-complete problem:

Equal-size Partition

Data: Let be $A = \{a_1, ..., a_{2n'}\}$ be a finite set of elements and a size $s(a) \in \mathbb{N}, \forall a \in A$. Let be $B = \frac{\sum_{j \in A} s(a)}{2}$

Question: Does there exists $A' \subset A$ such that $\sum_{a \in A'} s(a) = B$ and $|A'| = n'$?

We show that this problem reduces to the special case of agreeable due dates, i.e. with $p_j \leq p_k \Leftrightarrow d_j \leq d_k, \forall j, k = 1..N$. This problem is denoted by $1|ADV - n, agreeable, \mathcal{L}|-$.

Let be given an instance of Equal-size Partition and let us denote by $N' = 2n'$ for simplicity purpose. We build the instance of the bilevel scheduling problem as follows:

- $N = N' + 1$,
- element jobs: $\forall j = 1..N', p_j = a_j$, and $d_j = B$,
- partition job: $p_{N'+1} = 2B$ and $d_{N'+1} = 3B - 1$,
- $\mathcal{L} = \{N' + 1\}$,
- $n = n' + 1$.

To have job $(N' + 1)$ tardy, the total length of the element jobs scheduled before must be at least B. However, if the jobs before $(N' + 1)$ are of total length strictly greater than B, then at least one of these ones is tardy and this violates the fact that only jobs in $\mathcal{L}$ are tardy. Consequently, there exists a feasible solution to the scheduling problem if and only if there exists a feasible solution to Equal-size Partition . □

3. Adversarial bilevel scheduling with data modification

We now turn to problems in which a set of n jobs is available and the leader can modify some of the data (processing times, due dates, weights), given a limited budget Q, before the follower computes a schedule minimizing a given criterion f^F. Leader's goal is to modify data such that the optimal value of $f^F \in \{\sum_j C_j^F, \sum_j w_j^F C_j^F, L_{max}^F, \sum_j U_j^F\}$ is maximum. The following subsections discuss the existence of polynomial-time algorithms for each criterion minimized by the follower.

3.1. Sum of completion times

The problem tackled in this section is referred to as $1|ADV-p|\sum_j C_j^F$. Given a list of n jobs with processing times p_j^F, the follower schedules jobs so that their sum of completion times, denoted by $\sum_j C_j^F$, is minimum. This is doable in polynomial time by applying the SPT rule. Let be the initial processing times p_j so that $p_1 \leq ... \leq p_n$. Thus, the leader has to decide how to fix quantities q_j so that, with $p_j^F = p_j + q_j$, the follower's optimal solution is as bad as possible. In addition, the leader has a budget so that $\sum_j |q_j| \leq Q$, with $Q \in \mathbb{N}$ given. Notice that the leader has no interest in setting a $q_j < 0$ as it would decrease the optimal solution value of the $\sum_j C_j^F$. Thus, without loss of optimality, we consider $q_j \geq 0, \forall j = 1..n$, and $\sum_j q_j \leq Q$. We first show two instrumental results.

Lemma 3. *Let $s = (1, ..., k, k+1, ...n)$ be a SPT schedule with $p_1 = ... = p_k < p_{k+1}$, $q = \frac{Q}{k}$ with $q \in \mathbb{N}$, and $p_k + q \leq p_{k+1}$.*
Let s^1 be the schedule obtained by setting $q_1 = ... = q_k = q$ and $q_j = 0$, $\forall j = k+1, .., n$, and let s^2 be the schedule obtained by setting $q_1 = ... = q_{k'} = q$, $k' < k$ and assigning budget $(Q - (k-k')q)$ to any subset of jobs in $\{(k+1), .., n\}$.
We have $\sum_j C_j^F(s^1) > \sum_j C_j^F(s^2)$.

Proof. First, notice that $s^1 = s$ and that both the first $(k-k')$ jobs in s^2 and the first $(k-k')$ jobs in s have identical processing times. Next, it is well-known that the follower's objective function can be rewritten as:

$$\sum_{j=1}^n C_j^F = \sum_{j=1}^n (n-j+1)p_j^F.$$

Let us define $\Delta(s^\ell, s) = (\sum_j C_j^F(s^\ell) - \sum_j C_j^F(s))$ as the increase in the objective function for a given vector $q^\ell = [q_1^\ell, ..., q_n^\ell]$. We have:

$$\Delta(s^1, s) = qk(n - \tfrac{k-1}{k}).$$

We derive an upper bound on $\Delta(s^2, s)$ by setting $q_{k+1} = (Q-(k-k')q)$ and by remarking that jobs 1 to k' are scheduled from positions $(k-k'+1)$ to k in s^2. Thus, we have:

$$\begin{array}{rcl} \Delta(s^2, s) & \leq & \sum_{j=(k-k')}^{k} q + (n-k)q(k-k') \\ & \leq & q\frac{(k'+1)(n-k+k'+1)}{2} + q(n-k)(k-k') \\ & \leq & \frac{q}{2}\big((n-k)(2k-k'+1) + (k'+1)^2\big) \end{array}$$

Consequently, we have:

$$\begin{array}{rcl} \Delta(s^1, s) - \Delta(s^2, s) & \geq & qk(n - \frac{k-1}{2}) - \frac{q}{2}\big((n-k)(2k-k'+1) + (k'+1)^2\big) \\ & \geq & \frac{q}{2}n(k'+1) + \frac{q}{2}k(k-k'+2) - \frac{q}{2}(k'+1)^2 \\ & \geq & \frac{q}{2}n(k'+1) + \frac{q}{2}(k^2 - kk' + 2k - k'^2 - 2k' - 1) \\ & > & \frac{q}{2}n(k'+1) - \frac{q}{2}(kk'+1) \\ & > & 0 \end{array}$$

It follows that $\sum_j C_j^F(s^1) > \sum_j C_j^F(s^2)$. □

Lemma 4. *Let $s = (1, ..., k, k+1, ...n)$ be a SPT schedule with $p_1 = ... = p_k < p_{k+1}$, $q = \frac{Q}{k}$ with $q \in \mathbb{N}$, and $p_k + q \leq p_{k+1}$.*
Let s^1 be the schedule obtained by setting $q_1 = ... = q_k = q$ and $q_j = 0$, $\forall j = (k+1)..n$, and let s^2 be the schedule obtained by setting $q_1 = ... = q_k = 0$ and assigning budget Q to any subset of jobs in $\{(k+1), .., n\}$.
We have $\sum_j C_j^F(s^1) > \sum_j C_j^F(s^2)$.

PROOF. First, notice that $s^1 = s$ and that the first k jobs in s^2 are identical to those in s^1. Next, it is well-known that the follower's objective function can be rewritten as:

$$\sum_{j=1}^{n} C_j^F = \sum_{j=1}^{n}(n-j+1)p_j^F.$$

Let us define $\Delta(s^\ell, s) = (\sum_j C_j^F(s^\ell) - \sum_j C_j^F(s))$ as the increase in the objective function for a given vector $q^\ell = [q_1^\ell, ..., q_n^\ell]$. We have:

$$\Delta(s^1, s) = Q(n - \tfrac{k-1}{k}), \text{ and}$$
$$\Delta(s^2, s) \leq Q(n-k),$$

as an upper bound to $\Delta(s^2, s)$ is obtained by setting $q_{k+1} = Q$. It follows that $\Delta(s^1, s) > \Delta(s^2, s)$ which proves the result.□

Theorem 4. *The $1|ADV - p|\sum_j C_j^F$ problem can be solved in $O(n\log(n))$ time. The leader sets:*

- $q_j = P - p_j$, $\forall j = 1..\big(k_P - Q - k_P P + \sum_{i=1}^{k_P} p_i\big)$,
- $q_j = P - p_j + 1$, $\forall j = \big(k_P - Q - k_P P + \sum_{i=1}^{k_P} p_i + 1\big)..k_P$,
- $q_j = 0$, $\forall j = k_P + 1..n$,

with $P = argmax_{0\leq t\leq \sum_j p_j}\big((kt - \sum_{j=1}^{k} p_j) \leq Q | p_1 \leq ... \leq p_k \leq t \text{ and } p_{k+1} > t\big)$, and k_P the job such that $p_{k_P} \leq P < p_{k_P+1}$. The follower applies the SPT rule on the $p_j^F = p_j + q_j$'s.

PROOF. The result is obtained by applying repeatedly Lemma 3 and 4, starting from the SPT schedule s^0 with $q^0 = [0, ..., 0]$. Also notice that, from the proof of these two lemma, we can evince that increasing the first k' equal-length jobs j with distinct quantities q_j is sub-optimal with respect to increasing them all by the same quantity q.
Let k^0 be the number of consecutive equal-length jobs in s^0 from position 1 on. First, assume that $Q \geq Q^0 = k^0(p_{k^0+1} - p_{k^0})$. By setting $q = (p_{k^0+1} - p_{k^0})$, $q_j = q$, $\forall j = 1..k^0$, and $q_j = 0, \forall j = (k^0+1)..n$, we obtain a new solution s^1 that provides the largest increase of the $\sum_j C_j^F$ value among all possible assignments of budget Q^0. Notice that the sequence of jobs in s^1 is the same than in s^0, $k^1 > k^0$ and the remaining budget is updated to $Q = (Q - Q^0)$.
The above process is repeated until iteration t when $Q < Q^t = k^t(p_{k^t+1} - (p_{k^t} + q_{k^t})$. In that case we necessarily have $Q < k^t$ and we set $q = 1$ and only increase the q_j's of jobs in positions $(k^t - Q)$ to k^t. This is also the best assignment of quantity Q to increase the optimal value of $\sum_j C_j^F$.

It is possible to directly identify the right values of k and the q_j's as follows. Let $k_P \in \mathbb{N}$, and $P \in \mathbb{N}$, be the largest value such that:

(1) $\sum_{j=1}^{k_P}(P - p_j) \leq Q$, and

(2) $\sum_{j=1}^{k_P+1}(P - p_j) > Q$.

Thus, the k_P smallest jobs will have increased processing times for the follower's problem so that the leader meets its budget constraint.
Notice, that $Q - \sum_{j=1}^{k_P}(P - p_j) < k_P$ necessarily holds, otherwise $(P+1)$ would satisfy (1) and (2), thus contradicting the definition of P. Consequently, the $\left(k_P - Q + k_P P - \sum_{i=1}^{k_P} p_i\right)$ first jobs j will have a processing time $p_j^F = P$, while the $\left(Q - k_P P + \sum_{i=1}^{k_P} p_i\right)$ last jobs j will have a processing time $p_j^F = P + 1$. The remaining $(n - k_P)$ jobs will keep their original processing times values, *i.e.* $p_j^F = p_j$.

Solving the problem requires, in $O(n\log(n))$ time, sorting the jobs according to SPT order. Next, the values of P and k_P can be computed in $O(n)$ time as well as the computation of the q_j's. This yields an overall $O(n\log(n))$ time complexity. □

3.2. Weighted sum of completion times

Consider that jobs are also attached weights w_j^F and the follower is scheduling jobs so that their weighted sum of completion times, denoted by $\sum_j w_j^F C_j^F$, is minimum. Whenever the processing times and weights are fixed, this is doable in polynomial time by applying the WSPT rule. Two adversarial problems are considered hereafter depending on the data the leader can modify. The first problem is denoted by $1|ADV-p|\sum_j w_j^F C_j^F$ and the leader can only modify processing times, while the second problem is denoted by $1|ADV-w|\sum_j w_j^F C_j^F$ and the leader can only modify the weights.

First, we focus on the $1|ADV-p|\sum_j w_j^F C_j^F$ problem. Let p_j be the initial processing times so that $\frac{p_1}{w_1^F} \leq ... \leq \frac{p_n}{w_n^F}$. Again, the leader has to decide how to fix quantities q_j so that with $p_j^F = p_j + q_j$, the follower's optimal solution is as bad as possible. As for the unweighted case, it is of no interest for the leader that some $q_j < 0$. In addition, the leader has a budget so that $\sum_j q_j \leq Q$, with $Q \in \mathbb{N}$ given. We first consider the continuous case, *i.e.* $q_j \in \mathbb{R}$, and show that it can be solved in polynomial time. We start with instrumental results.

Lemma 5. *There exists an optimal solution to the* $1|ADV-p, q_j \in \mathbb{R}|\sum_j w_j^F C_j^F$ *problem in which jobs are sequenced according to the WSPT rule on the* p_j*'s.*

Proof. Suppose by contradiction an optimal solution s^* exists where the original WSPT sequence is not preserved. Thus, in s^* there necessarily exist two adjacent jobs i and j starting at time t_0 whose processing times p_i^F, p_j^F and weights w_i, w_j are such that $\frac{p_i^F}{w_i} > \frac{p_j^F}{w_j}$ (hence job i follows job j in s^*), while in the initial WSPT order we have $\frac{p_i}{w_i} < \frac{p_j}{w_j}$. The contribution to the objective function value of these two jobs in s^* is given by:

$$z_1 = w_j(t_0 + p_j^F) + w_i(t_0 + p_j^F + p_i^F).$$

Let $\epsilon > 0$ be a small enough value such that $\frac{p_i^F - \epsilon}{w_i} > \frac{p_j^F + \epsilon}{w_j}$ and the sequence in s^* remains unchanged. Let s_ϵ be the solution obtained when setting $p_i^F = p_i^F - \epsilon$ and $p_j^F = p_j^F + \epsilon$. Thus, the contribution to the objective function value of these two jobs in s_ϵ is given by:

$$z_2 = w_j(t_0 + p_j^F + \epsilon) + w_i(t_0 + p_j^F + p_i^F) = z_1 + \epsilon w_j > z_1.$$

Consequently, s_ϵ is necessarily better than s^* which contradicts the optimality of s^*. $\square$

Following Lemma 5, we assume in the remainder that an optimal solution to the problem with real valued q_j's preserves the initial WSPT order.

Lemma 6. *Let $s^{WSPT} = (1, .., n)$ be the schedule obtained by the WSPT rule on the p_j's. Let be $k \in \{1..n\}$, $Q = \sum_{j=1}^{k} \frac{p_{k+1}}{w_{k+1}} w_j - p_j$, $q^1 = [\frac{p_{k+1}}{w_{k+1}} w_1 - p_1; ...; \frac{p_{k+1}}{w_{k+1}} w_k - p_k; 0; ...; 0] \in \mathbb{R}^n$ and $q^{1+\epsilon} = q^1 + \epsilon \in \mathbb{R}^n$. Let s^1 (resp. $s^{1+\epsilon}$) be the optimal solution of the follower's problem with q^1 (resp. $q^{1+\epsilon}$).*
We have $\sum_j w_j C_j^F(s^1) \geq \sum_j w_j C_j^F(s^{1+\epsilon})$.

Proof. First, notice that the sequences associated to s^{WSPT}, s^1 and $s^{1+\epsilon}$ are identical. Next, we remind that the follower's objective function can be rewritten as:

$$\sum_{j=1}^{n} w_j^F C_j^F = \sum_{j=1}^{n} p_j^F \sum_{k=j}^{n} w_k^F.$$

Let us define $\Delta(s^\ell, s) = (\sum_j w_j C_j^F(s^\ell) - \sum_j w_j C_j^F(s))$ as the increase in the objective function for a given vector $q^\ell = [q_1^\ell, ..., q_n^\ell]$. We have:

$$\Delta(s^1, s^{WSPT}) = \sum_{j=1}^{k} \left(\frac{p_{k+1}}{w_{k+1}} w_j - p_j\right) \sum_{i=j}^{n} w_i^F, \text{ and}$$
$$\Delta(s^{1+\epsilon}, s^{WSPT}) = \sum_{j=1}^{k} \left(\frac{p_{k+1}}{w_{k+1}} w_j - p_j + \epsilon_j\right) \sum_{i=j}^{n} w_i^F + \sum_{j=k+1}^{n} \epsilon_j \sum_{i=j}^{n} w_i^F.$$

It follows that:

$$\begin{array}{rcl} \Delta(s^1, s^{WSPT}) - \Delta(s^{1+\epsilon}, s^{WSPT}) & = & -\sum_{j=1}^{k} \epsilon_j \sum_{i=j}^{n} w_i^F - \sum_{j=k+1}^{n} \epsilon_j \sum_{i=j}^{n} w_i^F \\ & = & -\sum_{j=1}^{k} \epsilon_j (\sum_{i=j}^{k} w_i^F + \sum_{i=k+1}^{n} w_i^F) \\ & & -\sum_{j=k+1}^{n} \epsilon_j (\sum_{i=k+1}^{n} w_i^F - \sum_{i=k+1}^{j-1} w_i^F) \end{array}$$

As $\sum_{j=1}^{n} \epsilon_j = 0$, $\epsilon_j \geq 0$ $(j = k+1..n)$ and $\epsilon_j \leq \epsilon_{j+1}$ $(j = 1..k)$, we derive that: $\Delta(s^1, s^{WSPT}) - \Delta(s^{1+\epsilon}, s^{WSPT}) = -\sum_{j=1}^{k} \epsilon_j \sum_{i=j}^{k} w_i^F + \sum_{j=k+1}^{n} \epsilon_j \sum_{i=k+1}^{j-1} w_i^F \geq 0$, as both terms are positive. This proves the result. $\square$

Theorem 5. *The $1|ADV - p, q_j \in \mathbb{R}| \sum_j w_j^F C_j^F$ problem can be solved in $O(n \log(n))$ time. The leader sets:*

- $q_j = \frac{(Q + \sum_{\ell=1}^{k_R} p_\ell) w_j^F}{\sum_{\ell=1}^{k_R} w_\ell^F} - p_j$, $\forall j = 1..k_R$
- $q_j = 0$, $\forall j = (k_R + 1)..n$,

with $R = \frac{Q + \sum_{j=1}^{k_R} p_j}{\sum_{j=1}^{k_R} w_j^F}$ and k_R the job such that $\frac{p_{k_R}}{w_{k_R}^F} \leq R < \frac{p_{k_R+1}}{w_{k_R+1}^F}$. The follower applies the WSPT rule on $p_j^F = p_j + q_j$ and $w_j^F = w_j$, $\forall j = 1..n$.

Proof. It is well-known that the follower's objective function can be rewritten as:

$$\sum_{j=1}^{n} w_j^F C_j^F = \sum_{j=1}^{n} p_j^F \sum_{k=j}^{n} w_k^F.$$

Starting with $p_j^F = p_j$, it is always worse for the optimal solution of the follower to increase p_j^F instead of p_k^F, $j < k$, whenever the WSPT order is preserved. To preserve the WSPT order, the leader must ensure that $\frac{p_1^F}{w_1^F} \leq .... \leq \frac{p_n^F}{w_n^F}$. It follows that to worsen

the optimal solution of the follower's problem, the leader increases as much as possible the jobs in the first positions of the WSPT sequence. Assume that a job j has a ratio $\frac{p_j}{w_j^F} < \frac{p_j^F}{w_j^F} = R$. Then, we must set $q_j = w_j^F R - p_j$.
To maintain the WSPT order, and taking into account his budget, the leader must determine the largest ratio $R \in \mathbb{R}$ and the job k_R such that:

$$(P_0) \begin{cases} \frac{p_{k_R}}{w_{k_R}^F} \leq R < \frac{p_{k_R+1}}{w_{k_R+1}^F}, & \text{(I)} \\ \sum_{j=1}^{k_R} q_j = \sum_{j=1}^{k_R} (w_j^F R - p_j) = Q, & \text{(II)} \\ \frac{p_j + q_j}{w_j^F} = R, \forall j = 1..k_R & \text{(III)} \end{cases}$$

Considering (III), for any value of j, and (II) (P_0) implies:

$$(P_1) \begin{cases} \frac{p_{k_R}}{w_{k_R}^F} \leq R < \frac{p_{k_R+1}}{w_{k_R+1}^F}, & \text{(I)} \\ q_j = \frac{(Q + \sum_{\ell=1}^{k_R} p_\ell) w_j^F}{\sum_{\ell=1}^{k_R} w_\ell^F} - p_j, \forall j = 1..k_R & \text{(IV)} \end{cases}$$

It can be easily shown that fixing the q_j's as in (IV) also satisfies (II) and (III), so that (P_0) and (P_1) are equivalent. So, fixing the q_j's as in (IV) solves to optimality the problem.

Solving the problem requires sorting, in $O(n \log(n))$ time, the jobs according to WSPT. The computation of R and k_R can be done in $O(n)$ time, as well as the update of the q_j's. This yields an overall $O(n \log(n))$ time complexity. □

Let us go back to the discrete version of the problem, i.e. when $q_j \in \mathbb{N}, \forall j = 1..n$. It turns out that the WSPT order on the initial p_j's may not be preserved in an optimal solution to the $1|ADV - p, q_j \in \mathbb{N}| \sum_j w_j^F C_j^F$ problem. Let us take the following two jobs example: $p_1 = 999$, $w_1 = 1000$, $p_2 = w_2 = 1$. Consider $Q = 2$. Two solutions $s^1 = (2, 1)$ and $s^2 = (1, 2)$ can be obtained. Solution s^1 can be built from vector $q^1 = [2; 0]$ while solution s^2 can be built from vectors $q^2 = [1; 1]$ and $q^3 = [0; 2]$. It can be easily checked that the optimal solution of the bilevel problem is given by s^1 (by means of vector q^1) which does not follow the initial WSPT order $(1, 2)$. Consequently, it cannot be solved by the same kind of approach as in the continuous case. The complexity status of the $1|ADV - p, q_j \in \mathbb{N}| \sum_j w_j^F C_j^F$ problem remains open.

Next, consider the $1|ADV - w| \sum_j w_j^F C_j^F$ problem and the leader can only modify the weights of the follower. So, for the follower's problem we set $p_j^F = p_j$ and $w_j^F = w_j + q_j$, $\forall j = 1..n$, with $q_j \in \mathbb{N}$. As previously, we first focus on the relaxed version with real valued q_j's, i.e., problem $1|ADV - w, q_j \in \mathbb{R}| \sum_j w_j^F C_j^F$. The proof of Lemma 5 shows that the result also holds in the case the weights can be modified by the leader. So, there exists an optimal solution to the $1|ADV - w, q_j \in \mathbb{R}| \sum_j w_j^F C_j^F$ problem in which jobs are sequenced in the WSPT order on the p_j and w_j.

Theorem 6. *The* $1|ADV-w, q_j \in \mathbb{R}|\sum_j w_j^F C_j^F$ *problem can be solved in* $O(n\log(n))$ *time. The leader sets:*

- $q_j = \frac{(Q+\sum_{\ell=k_R}^{n} w_\ell)p_j^F}{\sum_{\ell=k_R}^{n} p_\ell^F} - w_j$, $\forall j = k_R..n$
- $q_j = 0$, $\forall j = 1..(k_R-1)$,

with $R = \frac{\sum_{j=k_R}^{n} p_j^F}{Q+\sum_{j=k_R}^{n} w_j}$ *and* k_R *the job such that* $\frac{p_{k_R-1}^F}{w_{k_R-1}} < R \leq \frac{p_{k_R}^F}{w_{k_R}}$. *The follower applies the WSPT rule on* $p_j^F = p_j$ *and* $w_j^F = w_j + q_j$, $\forall j = 1.n$.

Proof. The proof follows the same line than that of Theorem 5. As the follower's objective function can be rewritten as:

$$\sum_{j=1}^{n} w_j^F C_j^F = \sum_{j=1}^{n} p_j^F \sum_{k=j}^{n} w_k^F,$$

it follows that it is always worse for the optimal solution of the follower to increase w_j^F instead of w_k^F, $j > k$, whenever the WSPT order is preserved. To preserve the WSPT order, the leader must ensure that $\frac{p_1^F}{w_1^F} \leq .... \leq \frac{p_n^F}{w_n^F}$. It follows that to worsen the optimal solution of the follower's problem, the leader increases as much as possible the jobs in the last positions of the WSPT sequence. Assume that a job j has a ratio $\frac{p_j^F}{w_j} > \frac{p_j^F}{w_j^F} = R$. Then, we must set $q_j = \frac{p_j^F}{R} - w_j$.
To maintain the WSPT order, and taking account of his budget, the leader must determine the smallest ratio $R \in \mathbb{R}$ and the job k_R such that:

$$(P_0)\begin{cases} \frac{p_{k_R-1}}{w_{k_R-1}^F} \leq R < \frac{p_{k_R}}{w_{k_R}^F}, & \text{(I)} \\ \sum_{j=k_R}^{n} q_j = \sum_{j=k_R}^{n} (\frac{p_j^F}{R} - w_j) = Q, & \text{(II)} \\ \frac{p_j^F}{w_j+q_j} = R, \forall j = k_R..n & \text{(III)} \end{cases}$$

Considering (III), for any value of j, and (II) (P_0) implies:

$$(P_1)\begin{cases} \frac{p_{k_R-1}}{w_{k_R-1}^F} \leq R < \frac{p_{k_R}}{w_{k_R}^F}, & \text{(I)} \\ q_j = \frac{(Q+\sum_{\ell=k_R}^{n} w_\ell)p_j^F}{\sum_{\ell=k_R}^{n} p_\ell^F} - w_j, \forall j = k_R..n & \text{(IV)} \end{cases}$$

It can be easily shown that fixing the q_j's as in (IV) also satisfies (II) and (III), so that (P_0) and (P_1) are equivalent. So, fixing the q_j's as in (IV) solves to optimality the problem.
The time complexity is established exactly as in Theorem 5. □

Considering the discrete version of the problem, i.e. when $q_j \in \mathbb{N}, \forall j = 1..n$, it turns out that the WSPT order on the initial w_j's may not be preserved in an optimal solution to the $1|ADV-w, q_j \in \mathbb{N}|\sum_j w_j^F C_j^F$ problem. This can be shown by the same two jobs example than the one used for the $ADV-p$ version of the problem which also implies that the complexity status of the $1|ADV-w, q_j \in \mathbb{N}|\sum_j w_j^F C_j^F$ problem remains open.

3.3. Maximum lateness

Assume that each job j is defined by a processing time p_j and a due date d_j. The aim, for the follower, is to schedule jobs in order to minimize the maximum lateness, defined by $L^F_{max} = \max_{j=1..n}(C^F_j - d^F_j)$. The leader can modify either the processing times or the due dates so that the optimal solution for the follower is as bad as possible. Whenever the p^F_j's and d^F_j's are known, the optimal solution of the follower's problem is built by applying the EDD rule. Without loss of generality, let us assume that $d_1 \leq ... \leq d_n$.

We first focus on the problem where the leader can only modify the processing times, which is referred to as $1|ADV - p|L^F_{max}$. As the due dates remain unchanged, we set $d^F_j = d_j$, $\forall j = 1..n$. Besides, $p^F_j = p_j + q_j$ is the processing time value of the follower's problem. As modifying the processing times has no impact on the EDD order, this bilevel problem can be trivially solved by setting $q_1 = Q$ and $q_j = 0, j = 2..n$.

Now, let us consider the problem in which the leader can only modify the due dates, which is referred to as $1|ADV - d|L^F_{max}$. Thus, we set $p^F_j = p_j$, $d^F_j = d_j - q_j$, $\forall j = 1..n$, and increasing the value of the optimal solution of the follower's problem requires $q_j \geq 0$, with $\sum_{j=1}^{n} q_j \leq Q$. Intuitively, we could think about decreasing the due dates of *critical jobs*, i.e., jobs which give the L_{max} value. However, consider the following 2-job example with $p_1 = 30$, $d_1 = 11$, $p_2 = 3$, $d_2 = 13$ and $Q = 10$. The optimal schedule for these due dates is $s_0 = (1, 2)$, it has $L_{max}(s_0) = 20$ and the critical job is 2. If the leader sets $q_1 = 4$ and $q_2 = 6$ to increase the L_{max} by decreasing d_2, then the optimal solution is $s_1 = (1, 2)$ and has $L_{max}(s_1) = 26$. But if the leader only decreases d_1 by setting $q_1 = 10$ and $q_2 = 0$, then the optimal solution is still $s_2 = (1, 2)$ but with $L_{max}(s_2) = 29 > L_{max}(s_1)$. Consequently, to compute an optimal solution, it is not sufficient to only consider decreasing the due dates of critical jobs.

We provide some properties of an optimal schedule when we are given the critical job j

Theorem 7. *Let j be a job giving the L^F_{max} value of an optimal schedule s^* to the $1|ADV - d|L^F_{max}$ problem. We denote by B_j (resp. A_j) the set of jobs preceding (resp. following) j in s^*. There exists an optimal schedule s^* in which:*

1. *$B_j \subseteq \{1, ..., (j-1)\}$ and jobs in B_j are sequenced by their index order (initial EDD order),*
2. *jobs in A_j are sequenced by their index order (initial EDD order) and $q_k = 0, \forall k \in A_j$,*
3. *$\forall i \in A_j \cap \{1, ..., (j-1)\}, p_i < p_j$.*

Proof. Let $s^{EDD} = (1, ..., n)$ denote the schedule obtained by sequencing jobs according to the EDD rule and let s^* denote an optimal sequence with job j being a job with maximum lateness. In schedule s^* we have $d^F_\ell = d_\ell - q^*_\ell$ and L^*_ℓ refers to the lateness of job ℓ, $\forall \ell = 1...n$. The properties stated in the theorem are shown by proving the five following facts: in s^*, all jobs in B_j are in EDD order (fact 1), all successors of j in s^{EDD} are in A_j (fact 2), all jobs in A_j are in EDD order (fact 3), each job $k \in A_j$ keeps the original due date (fact 4) and for each job $i \in A_j \cap \{1, ..., (j-1)\}$, $p_i < p_j$ must hold

(fact 5). We prove the result by showing that from any presumed optimal schedule, we can always build s^* without increasing the L_{max} value. We distinguish above the five mentioned facts.

1. Suppose there exists another optimal schedule s^1, with values q^1_ℓ, in which j gives the L_{max} and there exists a pair of jobs h, i, preceding j both in s^{EDD} and s^1 such that $d_h < d_i$ but $d^F_i = d_i - q^1_i < d^F_h = d_h - q^1_h$. Hence, job h precedes i in s^{EDD} and h follows i in s^1. In this case, as $L_{\max} = L^1_j$, we have $L^1_i \leq L_{\max}$ and $L^1_h \leq L_{\max}$. By reducing q^1_i so that $d_i - q^1_i = d_h - q^1_h$, we can place job i immediately after job h. Let s'^1 be the corresponding schedule with values $q'^1_\ell = q^1_\ell$, $\forall \ell = 1...n, \ell \neq i$, and $q'^1_i = d_i - d_h + q^1_h > 0$ since $d_h < d_i$. Notice that $\sum_\ell q'^1_\ell < \sum_\ell q^1_\ell \leq Q$. Let us denote by L'^1_h and L'^1_i the associated lateness values. We have $L'^1_h < L'^1_i = L^1_h \leq L_{\max} = L^1_j$, that is $L_{\max}$ does not change. By iteratively applying this argument, we obtain an optimal schedule s^t in which all jobs $i < j$ preceding j also in s^t are in EDD order of the d_ℓ's.
2. Now, we assume that there exists another optimal schedule s^1 with values q^1_ℓ, in which j gives the L_{max} and let job $k > j$ be the largest index job which is scheduled before j in s^1. Notice that by a similar reasoning to that of fact 1, we can assume that job k immediately precedes j. Let us build a schedule s'^1 with values $q'^1_\ell = q^1_\ell$, $\forall \ell = 1...n, \ell \neq k$, and $q'^1_k = d_k - d_j + q^1_j$. Again, note that $q'^1_k \geq 0$ since $d_j \leq d_k$. Thus, $s'^1 = (1, ..., j, k, ..., n)$ with $\sum_\ell q'^1_\ell < \sum_\ell q^1_\ell \leq Q$. We have: $L^1_k < L'^1_j \leq L^*_{max} = L^1_j = L'^1_k$. That is, now k gives the $L_{\max}$ with j preceding k, thus respecting the EDD order. Also for this fact, by iteratively applying this argument, we obtain an optimal schedule s^t where, if job j gives the L_{max} value, then all jobs $k > j$ follow j in s^t.
3. Next, suppose there exists another optimal schedule s^1,with values q^1_ℓ, in which j gives the L_{max} and there exists pairs of jobs h, i following j both in s^{EDD} and s^1 such that $d_h < d_i$ but $d^F_i = d_i - q^1_i < d^F_h = d_h - q^1_h$. Hence, job h precedes i in s^{EDD} and h follows i in s^1. The same analysis considered in fact 1 holds here leading to the conclusion that we can obtain an optimal schedule s^t in which all jobs $i > j$ following j also in s^t are in EDD order of the d_i's.
4. For this fact, suppose there exists another optimal schedule s^1,with values q^1_ℓ, in which j gives the L_{max} and there exists a job k following j both in s^{EDD} and s^1 such that $q^1_k > 0$ and $d^F_k < d_k$. By setting q^1_k to 0 so that $d^F_k = d_k$, job k remains after job j and the value of $L^F_{\max}$ does not change. By iteratively applying this argument, we obtain an optimal schedule s^t in which all successors of j keep their original due date.
5. For the last fact, suppose by contradiction there exists another optimal schedule s^1 with values q^1_ℓ, in which j gives the L_{max} and let i be the smallest index job $i < j$ with $p_i > p_j$ which is scheduled after j in $s^1 = (..., j, ..., i, ...)$. Consider the schedule s'^1 in which j and i are swapped with: $q'^1_i = d_i - d_j + q^1_j$, $q'^1_j = d_j - d_i$ and $q'^1_\ell = q^1_\ell, \forall \ell \neq i, j$. So, we have $d'^F_i = d^F_j$ and $d'^F_j = d^F_i$. Then, $\sum_\ell q'^1_\ell = \sum_\ell q^1_\ell \leq Q$ as $q^1_i + q^1_j = q'^1_i + q'^1_j$, and then s'^1 is feasible as it does not use more budget than s^1. But then, $C'^F_i > C^F_j$ implies $L'^F_i > L^F_j$ which contradicts the optimality of s^1 and i cannot be scheduled after j.□

From Theorem 7 it follows that, if we knew the job j inducing the maximum lateness,

and which jobs in subset B_j precede j in the sequence, then each optimal q_ℓ^* could be immediately determined. Hence, the problem could be trivially solved by testing each job j as the one inducing L_{max}^F and then deciding for each job $i \in \{1, ..., j-1\}$ whether i precedes j or not. However, this algorithm would run with exponential complexity $O(p(n)2^n)$ where we denote by $p(n)$ a polynomial of n.

Even if Theorem 7 provides interesting structural properties of an optimal schedule, the complexity status of the $1|ADV - d|L_{max}^F$ problem remains open.

3.4. Number of tardy jobs

Assume that each job j is defined by a processing time p_j and a due date d_j. The aim, for the follower, is to schedule jobs in order to minimize the number of tardy jobs, defined by $\sum_j U_j^F$, with $U_j^F = 1$ if job j is tardy, i.e. $C_j^F > d_j^F$, and 0 otherwise. Whenever the p_j^F's and d_j^F's are known, the follower minimizes the number of tardy jobs by means of Moore's algorithm (Moore (1968)) or, equivalently, by Lin and Wang's algorithm (Lin and Wang (2007)). Consider the particular case where all jobs share a common due date d: we show that the two possible problems can be solved in polynomial time.

First, consider problem $1|ADV - p, d_j = d|\sum_j U_j$ and notice that, for any given processing times $p_j^F = p_j + q_j$, $\forall j = 1..n$ (with $q_j \geq 0$), an optimal solution for the follower is obtained by sorting the jobs in non-decreasing order of the p_j's. From this sequence, jobs completing after the common due date d are tardy. Assume that the initial processing times p_j are indexed such that $p_1 \leq ... \leq p_n$. Let ℓ be the last early job when jobs are scheduled with the initial p_j's. It is straightforward that tardy jobs j in this initial schedule will have $q_j = 0$. By increasing the processing times of the early jobs, the leader will make jobs ℓ, $(\ell - 1)$, ... tardy. Besides, it is preferable that the budget Q is assigned to the first early jobs and the SPT order is maintained to maximize the number of tardy jobs. Consequently, the procedure described in Theorem 4 can be applied to optimally solve the $1|ADV - p, d_j = d|\sum_j U_j$ problem.
When the leader can only modify the value of the common due date, i.e. problem $1|ADV - d, d_j = d|\sum_j U_j$, it is easy to see that an optimal solution is achieved by setting $d^F = \max(0; d - Q)$. Then, the follower sequences jobs by non-decreasing value of the processing times, inducing a time complexity in $O(n \log(n))$.

We now turn to the general $1|ADV - p|\sum_j U_j$ problem and provide a pseudo-polynomial time algorithm to solve it. We assume hereafter that jobs are indexed such that $d_1 \leq ... \leq d_n$. Let $\mathcal{U}[j, \mathcal{C}, \mathcal{Q}]$ be the maximal minimum number of tardy jobs when the j first jobs are scheduled, the early ones finish at time $\mathcal{C}$ and the leader has a budget $\mathcal{Q}$ to modify their processing time. We have:

$$\mathcal{U}[j, \mathcal{C}, \mathcal{Q}] = \max_{0 \leq q \leq \mathcal{Q}} \Big(\min \big(\underbrace{\mathcal{U}[j-1, \mathcal{C} - p_j - q, \mathcal{Q} - q]}_{\text{if } \mathcal{C} \leq d_j}; \underbrace{\mathcal{U}[j-1, \mathcal{C}, \mathcal{Q} - q] + 1}_{j \text{ is tardy}} \big) \Big)$$

with $\mathcal{U}[j, \mathcal{C}, \mathcal{Q}] = +\infty$, if $j = 0$ and $\mathcal{C} > 0$, and $\mathcal{U}[0, 0, \mathcal{Q}] = 0$. Notice that, by convention, we define $\min(+\infty; +\infty) = -\infty$ and $\max(-\infty; ...; -\infty) = +\infty$. Moreover, the value of q achieving the maximum in the above formula defines the value of q_j. To solve

the $1|ADV-p|\sum_j U_j$ problem, we keep the best solution among all $\mathcal{U}[n,t,Q]$, $\forall t = 0...(\sum_{j=1}^{n} p_j + Q)$. The running time is in $O(n(\sum_{j=1}^{n} p_j + Q)Q^2)$ which can be rewritten as $O(n^4 p_{max}^3)$ since $Q \leq np_{max}$ with $p_{max} = \max_{1\leq j\leq n}(p_j)$.
The $1|ADV-d|\sum_j U_j$ problem can be similarly solved by a slight modification of the above dynamic programming formulation:

$$\mathcal{U}[j,\mathcal{C},\mathcal{Q}] = \max_{0\leq q\leq \mathcal{Q}} \Big(\min \big(\underbrace{\mathcal{U}[j-1,\mathcal{C}-p_j,\mathcal{Q}-q]}_{\text{if } \mathcal{C}\leq (d_j-q_)}; \underbrace{\mathcal{U}[j-1,\mathcal{C},\mathcal{Q}-q]+1}_{j \text{ is tardy}} \big) \Big)$$

with the same initial conditions. The running time is now in $O(n^4(p_{max}+d_{max})d_{max}^2)$ since $Q \leq nd_{max}$ with $d_{max} = \max_{1\leq j\leq n}(d_j)$.

Notice that the complexity status of the two problems with arbitrary dues dates remains open even if the existence of dynamic programming formulations running in pseudo-polynomial time rules out $\mathcal{NP}$-hardness in the strong sense.

4. Conclusions

In this paper we focused on the solution of some bilevel single machine scheduling problems in the adversarial setting, i.e. when the leader modifies the instance to make as bad as possible the optimal solution of the follower's problem. All the considered scheduling problems are polynomially solvable when the leader cannot modify the instance (single level optimization) and some of them remain so in the adversarial bilevel setting. However, surprisingly, others remain open even if pseudo-polynomial time algorithms may have been proposed. It is really intriguing to note that, despite their simple formulation, they seem to be at the frontier between easy and hard problems. These open problems definitely deserve deeper attention.

A natural extension of the problems tackled in this paper could be to consider them in the settings of optimistic or pessimistic bilevel optimization. As the leader minimizes his own criterion, these problems are expected to be much harder than in the adversarial setting. Hopefully, some may be proved to be $\mathcal{NP}$-hard while others may be Σ_2^p-hard, thus requiring challenging complexity proofs.

Finally, we point out that beyond the complexity status of bilevel scheduling problems it is worthwhile to consider the solution of hard ones and establish how effective can be exact algorithms for single machine problems but also for more complex scheduling environment.

Acknowledgements

This work was partially supported by "Ministero dell'Istruzione, dell'Università e della Ricerca" Award "TESUN-83486178370409 finanziamento dipartimenti di eccellenza CAP. 1694 TIT. 232 ART. 6".

References

Abass, S., 2005. Bilevel programming approach applied to the flow shop scheduling problem under fuzsiness. Computational Management Science 2, 279–293. doi:10.1007/s10287-005-0035-z.

Agnetis, A., Billaut, J.C., Gawiejnowicz, S., Pacciarelli, D., Soukhal, A., 2014. Multiagent scheduling. Springer.

Aissi, H., Bazgan, C., Vanderpooten, D., 2009. Min-max and min-max regret versions of combinatorial optimization problems: a survey. European Journal of Operational Research 2, 427–438. doi:10.1016/j.ejor.2008.09.012.

Caprara, A., Carvalho, M., Lodi, A., Woeginger, G., 2014. A complexity and approximability study of the bilevel knapsack problem. SIAM Journal on Optimization 24, 823–838. doi:10.1137/130906593.

Dempe, S., 2003. Annotated bibliography on bilevel programming and mathematical programs with equilibrium constraints. Optimization 52, 333–359. doi:10.1080/0233193031000149894.

Dempe, S., Kalashnikov, V., Pérez-Valdés, G., Kalashnikova, V., 2015. Bilevel Programming Problems: Theory, Algorithms and Applications to Energy Networks. Springer.

DeNegre, S., 2011. Interdiction and discrete bilevel linear programming. Ph.D. thesis. Lehigh University (USA).

Karlof, J., Wang, W., 1996. Bilevel programming applied to the flow shop scheduling problem. Computers & Operations Research 23, 443–451. doi:10.1016/0305-0548(95)00034-8.

Kis, T., Kovacs, A., 2012. On bilevel machine scheduling problems. OR Spectrum 34, 43–68. doi:10.1007/s00291-010-0219-y.

Kovacs, A., Kis, T., 2011. Constraint programming approach to a bilevel scheduling problem. Constraints 16, 317–340. doi:10.1007/s10601-010-9102-3.

Lawler, E., 1990. A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. Annals of Operations Research 26, 125–133. doi:10.1007/BF02248588.

Lin, Y., Wang, X., 2007. Necessary and sufficient conditions of optimality for some classical scheduling problems. European Journal of Operational Research 176, 809–818. doi:10.1016/j.ejor.2005.09.017.

Moore, J., 1968. An n job, one machine sequencing algorithm for minimizing the number of late jobs. Management Science 15, 102–109. doi:10.1007/s10479-018-2852-9.

Pascual, F., Rzadca, K., Trystram, D., 2009. Cooperation in multi-organization scheduling. Concurrency and Computation: Practice and experience 21, 905–921. doi:10.5555/1525995.1525998.

Stockmeyer, L., 1977. The polynomial-time hierarchy. Theoretical Compute Science 3, 1–22. doi:10.1016/0304-3975(76)90061-X.

Woeginger, G., 2021. The trouble with the second quantifier. 4OR 19, 157–181. doi:10.1007/s10288-021-00477-y.