POLITECNICO DI TORINO
Repository ISTITUZIONALE

Solution of a bilevel optimistic scheduling problem on parallel machines

Original
Solution of a bilevel optimistic scheduling problem on parallel machines / Schau, Q., Ploton, O., T'Kindt, V., Hoogeveen, H., Croce, F.D., Hoogeveen, J.. - In: ANNALS OF OPERATIONS RESEARCH. - ISSN 0254-5330. - (2026). [10.1007/s10479-026-07281-z]

Availability:
This version is available at: 11583/3011798 since: 2026-06-08T22:28:39Z

Publisher:
Springer

Published
DOI:10.1007/s10479-026-07281-z

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

18 November 2026

Annals of Operations Research
https://doi.org/10.1007/s10479-026-07281-z

ORIGINAL RESEARCH

Solution of a bilevel optimistic scheduling problem on parallel machines

Quentin Schau[1,2] · Olivier Ploton[1] · Vincent T'kindt[1] · Han Hoogeveen[4] · Federico Della Croce[2,3] · Jippe Hoogeveen[4]

Received: 9 October 2025 / Accepted: 11 May 2026
© The Author(s) 2026

Abstract
We consider the uniform parallel machines scheduling problem in the context of optimistic bilevel optimization, where two speed options are considered. In this scenario, the leader selects and assigns a subset of jobs to the follower to minimize the weighted number of tardy jobs, while the follower schedules these jobs on a set of uniform machines to minimize the total completion time. This problem has practical applications in Industry 4.0. We show that this problem is $\mathcal{NP}$-hard in the strong sense by providing a reduction from the Numerical 3-Dimensional Matching problem and we provide a moderately exponential-time dynamic programming algorithm. The problem is solved by means of a concise MIP formulation and a branch-and-bound algorithm that embeds a column generation approach for the lower bound computation. Computational experiments are presented for instances with up to 100 jobs and 4 machines while larger problems are out of reach for the proposed approaches.

Keywords Bilevel scheduling · Exact algorithms · Complexity results

1 Introduction

Bilevel scheduling problems model the situation in which two agents interact to schedule a set of jobs, but under a specific hierarchical setting. The first agent, called the leader, takes part of the scheduling decisions to minimize its objective function f^L. Next, the second agent, called the follower, takes the remaining decisions to minimize its objective function f^F. The particularity of such problems is that these two objective functions can only be evaluated when all scheduling decisions have been taken. The literature on bilevel optimization is rich and has focused on both continuous linear and non-linear problems (Bard, 1998; Dempe, 2002, 2020). It appears that, when the leader has taken his decisions, the follower's problem may have multiple optimal solutions: this leads to different classes of bilevel problems. In optimistic (OPT) bilevel problems, the follower chooses an optimal solution for its problem

✉ Quentin Schau
quentin.schau@univ-tours.fr; quentin.schau@polito.it

1 University of Tours, LIFAT, Tours, France

2 Politecnico di Torino, DIGEP, Torino, Italy

3 CNR, IEIIT, Torino, Italy

4 Utrecht University, Department of Information and Computing Sciences, Utrecht, The Netherlands

that is the best for the leader's problem. In pessimistic (PES) bilevel problems, the follower chooses an optimal solution for its problem that is the worst for the leader's problem. Finally, in adversarial (ADV) bilevel problems, the leader takes decisions to ensure that the optimal solution of the follower's problem is the worst possible. For these problems there is a one-to-one mapping between the optimal solution for the follower and the leader, leading to the classical min − max problems.

In contrast, bilevel scheduling is a challenging and emerging research topic that relates to bilevel optimization. To the best of our knowledge, the literature on bilevel scheduling is relatively limited. To solve a bilevel flow shop problem, Karlof and Wang (1996) proposed an exact two level branch-and-bound algorithm, where the leader assigns the operators to the machines in order to minimize the total flowtime, while the follower decides on the jobs' schedule in order to minimize the makespan. The considered approach was limited to instances with up to 10 jobs. Abass (2005) studied an extension of this problem by considering fuzzy processing times. Kasemset and Kachitvichyanukul (2010) proposed a bilevel scheduling multi-objective job shop problem that operates under the Theory of Constraints (TOC) policy. It was extended in Kasemset and Kachitvichyanukul (2012) who proposed a heuristic approach based on Particle Swarm Optimization. In this problem, the leader aims to minimize idle time on bottleneck machines before the follower, that schedules jobs in order to minimize the weighted aggregating function, i.e., the makespan, the maximum tardiness and the maximum earliness. Brown et al. (2005) considered a bilevel project scheduling problem where the leader takes decisions in order to maximize the completion time of the follower. We can see this problem as an adversarial setting. They investigated the computational complexity of variants of the problem. Lukač et al. (2008) proposed a heuristic based on Tabu search to solve a bilevel scheduling problem, where the objective of the leader is to assign the products to the machines to minimize the total sequence dependent setup time, while the objective of the follower is to minimize the production, storage and setup costs of the machine. Kovács and Kis (2011) used an exact constraint programming approach to model and solve discrete bilevel problems, and they applied it on optimistic bilevel single machine problems in which the leader selects the set of jobs the follower next schedules. Their approach failed finding the optimal solution already for several instances with 25 jobs. Following the three-field classification scheme (Graham et al. 1979) extended by T'kindt, et al. (2024), this problem is denoted by $1|OPT-n, r_j, \tilde{d_j}|\sum_j w_j^L R_j, \sum_j w_j^F C_j^F$, with $OPT-n$ meaning that the leader selects n jobs in an optimistic setting. Moreover, the Boolean variable $R_j = 1$ means that job J_j is rejected by the leader, who minimizes the total cost of the rejected jobs $\sum_j w_j^L R_j$, after which the follower schedules the jobs in order to minimize the weighted sum of completion times. This problem was shown weakly $\mathcal{NP}$-hard (Kis and Kovács, 2012). Furthermore, Kis and Kovács (2012) studied the two versions *optimistic* and *pessimistic* of the bilevel scheduling problem: 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, where A_k is a set of jobs assigned to any machine k by the leader, while the follower sequences jobs on each machine. These problems are shown to be strongly $\mathcal{NP}$-hard. Konur and Golias (2013) studied scheduling of inbound trucks at the inbound doors of a cross-dock facility under truck arrival time uncertainty. A Genetic Algorithm was proposed to solve both *optimistic* and *pessimistic* bilevel problems. The leader's problem is the minimization of the expected total service time of the cross-dock operator while the follower's problem is the minimization or maximization, regarding the setting, of the expected total waiting time by determining the expected arrival times. Bianco et al. (2015) proposed a bilevel scheduling problem occurring in grid computing. The leader assigns jobs to the follower aiming to minimize the total cost for rejecting

or delaying tasks while the follower wants to maximize the computational resource usage efficiency. They also proposed an exact algorithm based on a bilevel programming formulation, as well as, a reformulation as a single-level Mixed Integer Programming (MIP) by using Karush-Kuhn-Tucker (KKT) optimality conditions. This MIP is often referred to as a Mathematical Program with Equilibrium Constraints (MPEC) (Dempe, 2003). Finally, the paper proposes a heuristic algorithm to solve large size instances. T'kindt, et al. (2024) studied several bilevel scheduling problems, with one machine and the adversarial setting. The paper considers job selection and data modification for the leader's decision and several usual objective functions from the scheduling literature. They provided some complexity proofs as well as exact polynomial-time algorithms. At the current state of the art, these problems appear very challenging and exact solution approaches can just handle instances very small in size.

This study focuses on a scenario with one leader and one follower composed of uniform parallel machines operating at two speeds. We consider a periodic decision-making framework with a time horizon T to account for dynamic effects. The leader selects jobs and minimizes the weighted number of tardy jobs, which directly reflects satisfying on-time delivery for customers, with delivery priority reflected by the associated weights. The follower, in turn, schedules the selected jobs and minimizes total completion times, thereby reducing the average time jobs spend on the shop floor and improving overall throughput. These objective functions are both common performance metrics that are non-decreasing in the job completion times. Furthermore, they are largely uncorrelated: minimizing total completion time does not necessarily minimize the weighted number of tardy jobs. Thus, the leader's optimal strategy is influenced by the follower's actions; a leader who neglects the follower's schedule may find that their efforts to minimize tardy jobs are undermined. A key point to get is that both objective functions cannot be evaluated before all decisions are taken, otherwise the problem boils down to a single-level problem. The bilevel formulation allows us to model a hierarchical decision-making process, where the follower operates independently, responding to the leader's actions. A single-level formulation such as a bicriteria optimization problem, in contrast, cannot capture this hierarchical structure; it only provides a Pareto-optimal solution set, requiring the decision-maker to choose a specific outcome based on the inherent trade-offs between the objectives. From a bilevel perspective, we consider an optimistic case where the follower returns a schedule that achieves the minimum number of weighted tardy jobs among all optimal schedules for the total completion time.

Such a problem may occur in the context of the Industry of the Future in which the manufacturing system is composed of autonomous cyber-physical systems (CPS) interacting with each other in a hierarchical way. Thus, the problem is to dispatch production orders to these CPS. In such an organization, the leader is the system and the followers are the CPS. The integration of maintenance in the scheduling process may lead to setup the speed to potentially defective machines to its lowest value while healthy machines work at their highest speed.

Our contributions focus on two main aspects. First, we present several complexity results and, in particular, we establish that the considered bilevel scheduling problem is strongly $\mathcal{NP}$-hard. To this extent, we introduce a MIP formulation with a polynomial number of variables and constraints that rules out the possibility that the bilevel problem belongs to the second level of the polynomial hierarchy (Woeginger, 2021) and provide a polynomial-time reduction from the Numerical 3-Dimensional Matching problem which is known to be $\mathcal{NP}$-complete in the strong sense. Next, we present a moderately exponential-time exact dynamic programming procedure and a dedicated branch-and-bound algorithm embedding a column generation scheme for the computation of the lower bound. Experimental results reveal that

the bilevel problem is extremely challenging to solve and only instances with up to 100 jobs and 4 machines can be attacked.

The remainder of this paper is structured as follows. Sect. 2 presents structural properties and a MIP formulation as well as complexity results and a dynamic programming formulation. Sect. 3 focuses on the branch-and-bound algorithm. Sect. 4 reports on computational experiments conducted using randomly generated instances. Finally, Sect. 5 draws conclusions and outlines potential directions for future research.

2 Properties, modeling and complexity issues

We assume that the leader has taken his maintenance decisions, i.e., changed the floor shop configuration: we have m_1 machines with high speed denoted by set $\mathcal{V}_1$, and m_0 machines with low speed denoted by set $\mathcal{V}_0$. Correspondingly, we have $m = m_1 + m_0$ machines. A set $\mathcal{J}$ of N jobs is given among which $n \leqslant N$ have to be scheduled, with n a given value. Each job J_j is characterized by its processing time p_j, weight w_j, and due date d_j. Thus, the leader selects a subset $\mathcal{I} \subset \mathcal{J}$ of n jobs and assigns them to the follower who, next, schedules them on the machines. The leader aims to minimize the number of weighted tardy jobs $\sum_j w_j U_j^L$ whereas the follower aims to minimize the sum of completion times $\sum_j C_j^F$. Let $\mathfrak{S}_\mathcal{I}$ be the set of machine schedules that contain n jobs from $\mathcal{I}$, and let $C_j^F(s)$ be the completion time of job J_j in machine schedule s.

Then, we define $U_j^L(s) = \begin{cases} 1 \text{ if } C_j^F(s) > d_j \\ 0 \text{ otherwise} \end{cases}$. In the follower's problem, the optimistic scenario implies that there is a lexicographical objective function, since we want the schedule that minimizes the leader's objective function among all optimal schedules for the follower's objective function. Consequently, the follower minimizes $Lex\left(\sum_j C_j^F, \sum_j w_j U_j^L\right)$. The bilevel problem can be formulated as:

$$\begin{aligned} &\min_{\substack{\mathcal{I}\subset\mathcal{J} \\ |\mathcal{I}|=n}} \sum_{j\in\mathcal{I}} w_j U_j^L(s^*) \\ &\text{s.t.} \\ &\quad s^* \in \operatorname*{argmin}_{s\in\mathfrak{S}_\mathcal{I}} \left(Lex\left(\sum_{j\in s} C_j^F(s), \sum_{j\in s} w_j U_j^L(s) \right) \right) \end{aligned} \tag{1}$$

Correspondingly, we denote this bilevel problem by $Q|V_i \in \{V_0, V_1\}, OPT - n|\sum_j C_j^F, \sum_j w_j U_j^L$. The related follower's problem is denoted by $Q|V_i \in \{V_0, V_1\}|Lex\left(\sum_j C_j, \sum_j w_j U_j\right)$. For the rest of the paper, we assume that all jobs are sorted by Shortest Processing Time (SPT) rule, i.e., $p_1 \leqslant p_2 \leqslant \ldots \leqslant p_N$. In case of equal-size jobs, i.e., jobs with same processing times, we sort them according to the Earliest Due Date (EDD) rule.

2.1 Block structure

In this section, we consider structural properties for the $Q||\sum_j C_j$ problem. We know from Conway et al. (1967) that the problem can be solved in $\mathcal{O}(n \log n)$ time using the Shortest

Processing Time job on the First Available Machine (STP-FAM) rule. The follower's problem is a special case where there are only two different speeds. This implies a special property for optimal schedules that is obtained by adapting the proof of Conway et al. (1967).

Given n jobs and m machines M_i with a speed V_i, the processing time of a job J_j on the machine M_i is p_j/V_i. Our objective is to minimize $\sum_j C_j$. We represent a schedule by a non-rectangular and two-dimensional list that is a function of the machine number i and of the chronological order k. We denote by $J_{[i,-\ell]}$ the ℓ^{th} job on the machine M_i from the end, with $p_{[i,-\ell]}$ its processing time. We also denote by $J_{[i,k]}$ the k^{th} job on the machine M_i from the beginning, with $p_{[i,k]}$ its processing time.

A schedule σ is given by : $\sigma = \begin{pmatrix} J_{[1,1]} & \dots & J_{[1,\eta_1]} \\ & \dots & \\ J_{[m,1]} & \dots & J_{[m,\eta_m]} \end{pmatrix}$ where η_i denotes the number of jobs assigned on machine M_i, with $\sum_{i=1}^{m} \eta_i = n$.

Since all η_i depend on the schedule σ, we can have multiple values of η_i. To simplify the notation, let us introduce some dummy jobs with a processing time and a weight equal to zero. We will ensure that each machine is assigned exactly n jobs, including the dummy ones. It is clear that in an optimal schedule, these dummy jobs are scheduled on the left and can be removed without modifying the objective function nor the optimality of the schedule. To facilitate notation, we will denote all dummy jobs with the index 0, thus, $p_0 = 0$. Thus we add $(mn - n)$ dummy jobs. This results in the modified schedule σ being defined as :

$$\sigma = \begin{pmatrix} 0 & \overbrace{\dots}^{n-\eta_1 \ times} & 0 & J_{[1,n-\eta_1+1]} & \dots & J_{[1,n]} \\ & & \dots & & & \\ 0 & \overbrace{\dots}^{n-\eta_m \ times} & 0 & J_{[m,n-\eta_m+1]} & \dots & J_{[m,n]} \end{pmatrix} = \begin{pmatrix} J_{[1,1]} & \dots & J_{[1,n]} \\ & \dots & \\ J_{[m,1]} & \dots & J_{[m,n]} \end{pmatrix}$$

We focus on a ranking from right to left, i.e., in anti-chronological order $\ell = n + 1 - k$ (with $\ell = 1$ for the last, 2 for the second-to-last job, and so forth). Since the completion time of the job on machine M_i at position k can be expressed as $C_{[i,k]} = \sum_{h=1}^{k} p_{[i,h]}/V_i$, we see that $\sum_{k=1}^{n} C_{[i,k]} = \sum_{k=1}^{n} \sum_{h=1}^{k} p_{[i,h]}/V_i = \sum_{h=1}^{n} (n + 1 - h) p_{[i,h]}/V_i$. Hence, we find that

$$\sum_{j=1}^{n} C_j = \sum_{\substack{1 \leqslant i \leqslant m \\ 1 \leqslant \ell \leqslant n}} \ell/V_i \times p_{[i,-\ell]}$$

Correspondingly, we must minimize the weighted sum of processing times $p_{[i,-\ell]}$ with positional weights $\omega_{[i,\ell]} = \ell/V_i$, for position ℓ ($\ell = 1, \dots, n$) from the end on machine M_i. The $\omega_{[i,\ell]}$'s do not depend on the jobs or on the schedule. It is well known that such a sum of pairwise products of two sequences of reals is achieved if one sequence is arranged in increasing order and the other in decreasing order (Hardy et al., 1934). To get the optimal solution, we have to sort the $\omega_{[i,\ell]}$'s in non-decreasing order and the jobs in the reverse order, i.e. Longest Processing Time (LPT) rule. Specifically, the k^{th} smallest coefficient $\omega_{[k]}$ is associated with the k^{th} largest processing time $p_{[k]}$. Consequently, we have the following theorem.

Theorem 2.1 *The $Q|V_i \in \{V_0, V_1\}| \sum_j C_j$ problem is solvable in $\mathcal{O}(n \log n)$ time.*

□

Remark Firstly, the proof of the general case, i.e., $Q|| \sum_j C_j$, provides the algorithm by constructing the schedule from the right to the left, i.e., from the end of the machine schedule to

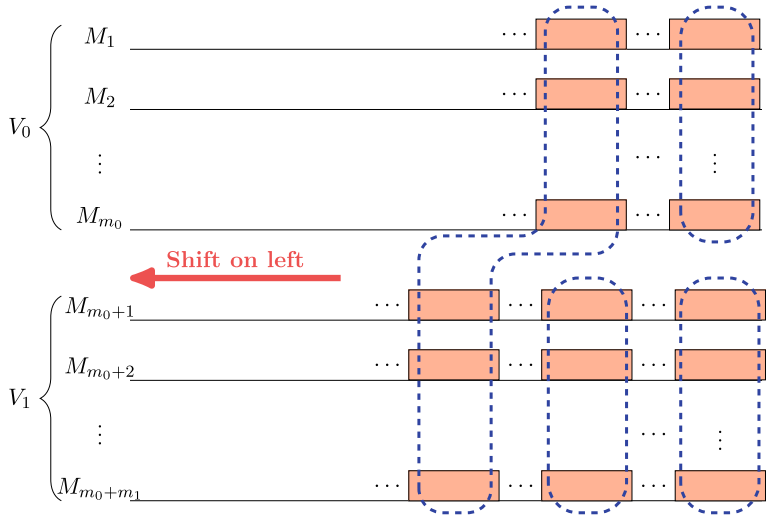


Fig. 1 Scheduling pattern with block structure for the solution of $Q|V_i \in \{V_1, V_0\}|\sum_j C_j$

the beginning. Alternatively, we can equivalently consider an SPT-FAM rule, thus scheduling jobs from the left to the right.

Secondly, the Theorem 2.1 establishes that sets with identical positional weights correspond to batches of jobs. Furthermore, when there are only two machine speeds, i.e. V_0, V_1 involved, an optimal schedule is characterized by a repetition of blocks. These blocks are formed by identical positional weights satisfying $\ell/V_0 = k/V_1$, where ℓ denotes a position on a low-speed machine and k denotes a position on a high-speed machine. If no such k or ℓ satisfies this equality, then the block will consist exclusively of positions on either high-speed or low-speed machines. For instance, if $V_0 = 1$ and $V_1 = 2$, then every odd position on high-speed machines forms a block exclusively on those machines, whereas every even position $2k$ forms a block spanning both machines, together with position k on low-speed machines, since $2k/V_1 = 2k/2V_0 = k/V_0$. Figure 1 illustrates the block structure, where jobs within the dashed group signify their membership to a block and can subsequently undergo permutation without changing the value of $\sum_j C_j$. We denote such a block as $\mathcal{B}_b$, which is a set composed of pairs (i, k) where i is the machine index and k is the position on the machine schedule.

This is an important result because it enables us to characterize the set of optimal schedules for the first criterion ($\sum_j C_j$). Thus, the problem is to determine which permutation of jobs within a block should be applied to obtain an optimal schedule for the second criterion ($\sum_j w_j U_j$).

2.2 Mixed integer programming

In this subsection we show that the bilevel problem can be modeled by means of Mixed Integer Programming. Before presenting the model, we introduce a few notations.

We have $b_{\max}$ blocks $\mathcal{B}_b$, $1 \leqslant b \leqslant b_{\max}$, given by the block structure described in Section 2.1. Let $E = \{(i, k)\}$ be a set of locations (just like $\mathcal{B}_b$) where i is the machine index and k the position on the machine schedule in chronological order; we have $E =$

$\mathcal{B}_1 \cup \ldots \cup \mathcal{B}_{b_{\max}}$. We need to assign a job to each position in the blocks $\mathcal{B}_2, \ldots, \mathcal{B}_{b_{\max}}$, but the first block $\mathcal{B}_1$ may not be completely filled. However, we can compute the number of jobs Q that should go into $\mathcal{B}_1$ as $Q = n - \sum_{b=2}^{b_{\max}} |\mathcal{B}_b|$.

We use binary variables $x_{i,j,k}$, such that $x_{i,j,k} = 1$ if job J_j is scheduled at location (i,k); and zero otherwise. We also use binary variables $U_{i,j,k}$, with $U_{i,j,k} = 1$ if job J_j is scheduled at location (i,k) and is penalized. The equations only state that tardiness implies penalty. So, a job J_j may be a priori unduly penalized, but this situation is, at the end, avoided by the minimization of penalties. So, we have $U_j \leqslant \sum_{(i,k)\in E} U_{i,j,k}$, and the minimization of $\sum_j \sum_{(i,k)\in E} w_j\, U_{i,j,k}$ leads to minimizing $\sum_j w_j U_j$. We use variables $C_{i,k}$ to express the completion time of the job in position k on machine M_i.

The following MIP model holds:

$$\min \sum_{j=1}^{N} \sum_{(i,k)\in E} w_j U_{i,j,k} \tag{2}$$

s.t.

$$\sum_{(i,k)\in E} x_{i,j,k} \leqslant 1 \qquad \forall j \in \{1,\ldots,N\} \tag{3}$$

$$\sum_{j=1}^{N} x_{i,j,k} \leqslant 1 \qquad \forall (i,k) \in E \tag{4}$$

$$\sum_{j=1}^{N} \sum_{(i,k)\in E\setminus\mathcal{B}_1} x_{i,j,k} = n - Q \tag{5}$$

$$\sum_{j=1}^{N} \sum_{(i,k)\in\mathcal{B}_1} x_{i,j,k} = Q \tag{6}$$

$$\sum_{j=1}^{N} x_{i,j,k}\, p_j \leqslant \sum_{j=1}^{N} x_{i',j,k'}\, p_j$$
$$\forall 1 \leqslant b \leqslant b_{\max} - 1\,, \forall (i,k) \in \mathcal{B}_b\,, \forall (i',k') \in \mathcal{B}_{b+1} \tag{7}$$

$$C_{i,k} = C_{i,k-1} + \sum_{j=1}^{N} x_{i,j,k}\, p_j / V_i \qquad \forall (i,k) \in E \tag{8}$$

$$C_{i,0} = 0 \qquad \forall i \in \{1,\ldots,m\} \tag{9}$$

$$U_{i,j,k} \leqslant x_{i,j,k} \qquad \forall j \in \{1,\ldots,N\}\,, \forall (i,k) \in E \tag{10}$$

$$C_{i,k} \leqslant \sum_{j=1}^{N} x_{i,j,k}\, d_j + \left(\sum_{j=1}^{N} U_{i,j,k}\right) H(i,k) \qquad \forall (i,k) \in E \tag{11}$$

$$U_{i,j,k} \in \{0,1\}\,,\ x_{i,j,k} \in \{0,1\} \qquad \forall j\,,\ \forall (i,k) \in E$$

Here $H(i,k) = k/V_i \times \max_j p_j$, which is an upper bound on the completion time of the kth job on machine M_i.

The objective function (2) minimizes the weighted number of tardy jobs. Constraints (3) ensure that no jobs are duplicated. The capacity constraints (4) ensure that each location has

at most one job. Constraints (5) and (6) guarantee that all blocks, except maybe the first one, must be completely filled: these constraints imply that we select exactly n jobs. Constraints (7) enforce that the processing times of assigned jobs between consecutive blocks do not decrease. Constraints (8) allow to compute the completion times. Constraints (10) ensure that only a scheduled job can be penalized. Finally, constraints (11) guarantee that a job that is not penalized is necessarily early, and so, we need to check that its completion time is smaller than its due date.

This mathematical formulation has $\mathcal{O}(N(n+m))$ variables and $\mathcal{O}(N^2+Nm^2)$ constraints.

2.3 Complexity results

2.3.1 The follower's problem

Firstly, we demonstrate that in the case of equal-size jobs, the follower's problem, with an arbitrary number of machine speeds can be solved in polynomial time.

Theorem 2.2 *The* $Q|p_j = p|Lex\left(\sum_j C_j, \sum_j w_j U_j\right)$ *problem is polynomially solvable.*

Proof We know that all optimal schedules for the $Q||\sum_j C_j$ problem are characterized by the block structure, and the SPT-FAM rule gives an optimal solution (Conway et al., 1967) in $\mathcal{O}(n \log n)$ time. Thus, we can compute all completion times, as jobs' processing times are equal. Moreover, by extending the algorithm mentioned by Dessouky et al. (1990), we can solve this problem. The idea is to determine which job is on time if assigned to a completion time, and then choose among them the one with the maximum weight. To do this, the jobs are initially sorted according to the reverse EDD order, in $\mathcal{O}(n \log n)$ time. Then, we start with the largest completion time and assign the early job with the largest weight. If no early job exists then this completion time is temporarily left unassigned, and we move to the second largest completion time, repeating the above operation. At the end of this first pass, when all completion times have been considered, then the remaining jobs can be randomly assigned to the remaining unassigned completion times: they are all tardy and with minimum weight. This can be implemented to run in $\mathcal{O}(n \log n)$ time. □

Now, we focus on the general case with arbitrary processing times. In the case of 2 identical machines, we show that the follower's problem is $\mathcal{NP}$-hard in the weak sense.

Theorem 2.3 *The* $P2||Lex\left(\sum_j C_j, \sum_j U_j\right)$ *problem is* $\mathcal{NP}$*-hard in the weak sense.*

Proof Let us consider the following problem which is known to be $\mathcal{NP}$-complete in the weak sense (Garey et al., 1988):

`Even-Odd Partition (EOD)`

Data: $2n$ positive integers $a_1 < a_2 < \ldots < a_{2n}$ where $A = 1/2 \times \sum_{j=1}^{2n} a_j$.

Question: Does there exist a partition into two sets A_1 and A_2, each containing exactly one element from the pair $\{a_{2j-1}, a_{2j}\}$ for $j = 1, \ldots, n$, such that $\sum_{j \in A_1} a_j = \sum_{j \in A_2} a_j = A$?

We show that this problem reduces to the decision version of problem $P2||Lex\left(\sum_j C_j, \sum_j U_j\right)$:

`DEC`

Data: A set of jobs J_j with a processing time p_j, a weight w_j and a due date d_j, to be processed on 2 identical machines. Let $C^\star$ be the value of an optimal solution of $P2||\sum_j C_j$, and U a given value.

Question: Does there exist a schedule σ such that $\sum_j C_j(\sigma) = C^\star$ and $\sum_j U_j(\sigma) \leq U$?

We show that there exists a reduction $\propto$ from EOD to DEC. Let I be a given instance of EOD and $\propto(I)$ is the corresponding instance of DEC. This instance $\propto(I)$ contains $2n$ jobs that must be scheduled on 2 identical machines. Job J_j corresponds to a_j. The processing times are defined as follows: $\forall k = 0, \ldots, n-1$ we put $p_{2k+1} = 2kA + a_{2k+1}$, $p_{2k+2} = 2kA + a_{2k+2}$. Furthermore, each job J_j $(j = 1, \ldots, 2n)$ has due date $d_j = D = 1/2 \times \sum_{i=1}^{2n} p_i = \sum_{k=0}^{n-1} (p_{2k+1}/2 + p_{2k+2}/2) = \sum_{k=0}^{n-1} 2A + A = n(n-1)A + A$. Finally, we put $U = 0$.

We demonstrate that the EOD problem's answer is true if and only if there exists a schedule σ that has $\sum_j C_j(\sigma) = C^\star$ and in which all jobs are on time, i.e., $\sum_j U_j(\sigma) = 0$.

We show the forward implication ($\Rightarrow$). Let us consider the scenario where the answer to the EOD problem is true. This implies that there exists a partition in two sets, A_1 and A_2, each containing exactly one element from the pair $\{a_{2j-1}, a_{2j}\}$ for $j = 1, \ldots, n$, such that $\sum_{j \in A_1} a_j = \sum_{j \in A_2} a_j = A$. We create our schedule by putting J_j on M_1 if $j \in A_1$, and on M_2, otherwise. Consequently, the completion times of the last job on each machine are $C_{M_1} = \sum_{j \in A_1} p_j = \sum_{j \in A_1} 2\lfloor (j-1)/2 \rfloor A + a_j = 2A \sum_{k=0}^{n-1} k + A = n(n-1)A + A$ and similarly $C_{M_2} = \sum_{j \in A_2} p_j = n(n-1)A + A$. Since the common due date equals $n(n-1)A + A$, this indicates that all jobs can, indeed, be scheduled on time and $\sum_j C_j$ is minimum as this schedule follows the SPT-FAM rule (see Figure 2).

Now suppose that there exists a schedule σ with $\sum_j C_j(\sigma) = C^\star$ and $\sum U_j(\sigma) \leq U = 0$. Because of the choice of the processing times, we know that the kth position (counted from the start) on machines M_1 and M_2 is occupied by the jobs J_{2k-1} and J_{2k}. Moreover, since the total processing time of all jobs is equal to $2D$ and each job is on time, both machines must finish exactly at time D. It is easily verified that the desired sets A_1 and A_2 can be obtained as the sets containing the indices of the jobs executed by machines M_1 and M_2 in σ, respectively. □

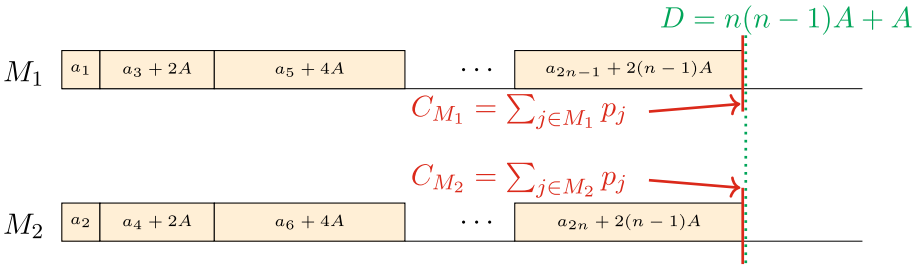


Fig. 2 Machine schedules for the problem $P2||Lex\left(\sum_j C_j, \sum_j U_j\right)$

We also show that when the number of machines is a part of the instance, then the follower's problem is $\mathcal{NP}$-hard in the strong sense.

Theorem 2.4 *The* $P||Lex\left(\sum_j C_j, \sum_j U_j\right)$ *problem is* $\mathcal{NP}$*-hard in the strong sense.*

Proof Let us consider the following problem which is known to be $\mathcal{NP}$-complete in the strong sense (Garey and Johnson, 1979):

`Numerical 3-Dimensional Matching (NUM-3DM)`

Data: Three disjoint sets of integers $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$, and $Z = \{z_1, \ldots, z_n\}$ along with a bound b.

Question: Can $X \cup Y \cup Z$ be partitioned into n disjoint sets $A_1, A_2, \ldots, A_n$ such that each A_h $(h = 1, \ldots, n)$ contains exactly one element from each of X, Y, and Z, i.e., $A_h = \{x_{[h]}, y_{[h]}, z_{[h]}\}$ and for $1 \leqslant h \leqslant n$, $x_{[h]} + y_{[h]} + z_{[h]} = b$?

We show that this problem reduces to the decision version of problem $P||Lex\left(\sum_j C_j, \sum_j U_j\right)$:

`DEC`

Data: A set of jobs J_j with a processing time p_j and a due date d_j, to be processed on m identical machines. Let $C^\star$ be the value of an optimal solution of $P||\sum_j C_j$ and U a given value.

Question: Does there exist a schedule σ such that $\sum_j C_j(\sigma) = C^\star$ and $\sum_j U_j(\sigma) \leq U$?

We show that there exists a reduction $\propto$ from `NUM-3DM` to `DEC`. Let I be a given instance of `NUM-3DM` and let $\propto (I)$ be the corresponding instance of `DEC`. We have: $U = 0$ and given three multisets of integers X, Y, and Z, each containing n elements, we create $3n$ jobs with the following processing times $\forall i = 1, \ldots, n$:

$p_i = x_i$, $p_{n+i} = y_i + \sum_{x \in X} x$, $p_{2n+i} = z_i + \sum_{y \in Y} y + \sum_{x \in X} x$ and a common due date

$D = b + \sum_{y \in Y} y + 2\sum_{x \in X} x$. Furthermore, we introduce $m = n$ machines.

Since we have $\forall i, j, k = 1, \ldots, n$, $p_i < p_{n+j} < p_{2n+k}$, $\sum_j C_j$ is minimized by scheduling exactly one triplet $(x_{[h]}, y_{[h]}, z_{[h]}) = (x_i, y_j, z_k)$, with $i, j, k \in \{1, \ldots, n\}$, on each machine M_h $(h = 1, \ldots, n)$ (see Figure 3).

We demonstrate that the NUM-3DM problem's answer is true if and only if all jobs can be scheduled in time.

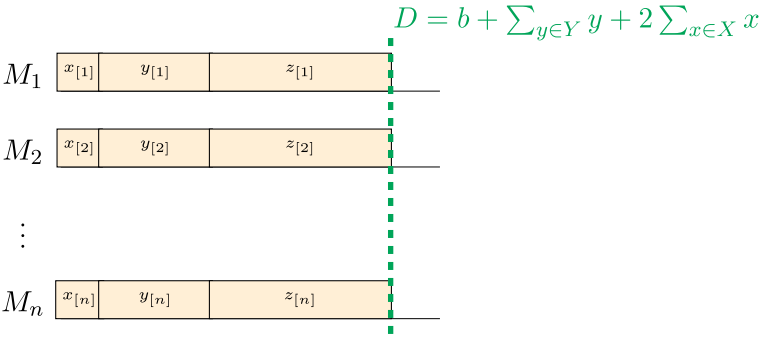


Fig. 3 Machine schedules for the $P||Lex\left(\sum_j C_j, \sum_j U_j\right)$

Let us demonstrate the forward implication ($\Rightarrow$). Assuming the solution to the NUM-3DM problem is true implies the existence of n disjoint triples $A_h = \{x_i, y_j, z_k\}$ such that $x_i + y_j + z_k = b$. Consequently, we schedule the jobs $\{p_i, p_{n+j}, p_{2n+k}\}$ on the machine M_h. The completion time of the last job on machine M_h is:

$C_{M_h} = x_i + y_j + z_k + \sum_{y \in Y} y + 2\sum_{x \in X} x = b + \sum_{y \in Y} y + 2\sum_{x \in X} x = D$ ensuring all jobs are scheduled in time. Now, we establish the reverse implication ($\Leftarrow$). Suppose that there exists a schedule σ such that $\sum_j C_j(\sigma) = C^\star$ and $\sum_j U_j(\sigma) \leq U = 0$. Because of the choice of the processing times, we see that each machine M_h ($h = 1, \ldots, n$) executes exactly one job from the sets $\{J_1, \ldots, J_n\}$, $\{J_{n+1}, \ldots, J_{2n}\}$, and $\{J_{2n+1}, \ldots, J_{3n}\}$. Furthermore, since the total processing time of all jobs is equal to nD and all jobs are on time, all machine must finish exactly at time D. A feasible solution to the NUM-3DM problem is immediately obtained by constructing triplet h ($h = 1, \ldots, n$) according to the indices of the jobs executed by machine M_h. □

We can also derive the following result.

Corollary 2.4.1 *The* $P||Lex\left(\sum_j C_j, \sum_j U_j\right)$ *problem is not approximable in polynomial or pseudo-polynomial time unless* $\mathcal{P} = \mathcal{NP}$.

Proof This result directly follows from the proof of Theorem 2.4 where it is shown that the particular problem when $\sum_j U_j^\star = 0$ is strongly $\mathcal{NP}$-hard. This rules out the existence of an approximation algorithm for ratio $\rho = \frac{\sum_j U_j}{\sum_j U_j^\star}$ □

From Theorem 2.4 and Corollary 2.4.1, the following result immediately holds (as the considered problem is a generalization of the $P||Lex\left(\sum_j C_j, \sum_j U_j\right)$ problem).

Corollary 2.4.2 *The* $Q|V_i \in \{V_0, V_1\}|Lex\left(\sum_j C_j, \sum_j w_j U_j\right)$ *problem is* $\mathcal{NP}$*-hard in the strong sense and is not approximable in polynomial or pseudo-polynomial time unless* $\mathcal{P} = \mathcal{NP}$.

2.3.2 The bilevel problem

From the reductions of the previous section, we can derive that the bilevel problem is at least $\mathcal{NP}$-hard in the strong sense. With respect to the polynomial hierarchy in complexity theory, the bilevel problem cannot be complete with respect to class Σ_2^p of level 3 due to the existence of the polynomial-size MIP formulation in Section 2.2 (Woeginger, 2021): consequently, the bilevel problem is $\mathcal{NP}$-hard in the strong sense.

Here, we show that when m is fixed, the problem is weakly $\mathcal{NP}$-hard by providing a moderately exponential-time dynamic programming algorithm based on the block structure of an optimal schedule as described in Section 2.1. The main idea of the dynamic programming approach is to fill the blocks one by one with the correct set of jobs. We use state variables that keep track of each possible set of completion times, the set of machines that are available, the current block, and the currently considered job. Thus, we try to assign the job to each available machine in the current block; once it is assigned, we can update the corresponding machine's completion time and proceed to the next job. When the block is full, we update the state variables for the next block and continue until all blocks are fulfilled.

It is worth noting that bilevel formulations are generally $\mathcal{NP}$-hard to solve. Specifically, the continuous linear bilevel problem is $\mathcal{NP}$-hard, but it is not Σ_2^p complete (Buchheim,

2023; Jeroslow, 1985; Lodi et al., 2014). Furthermore, the general mixed-integer bilevel problem is Σ_2^p-complete (Jeroslow, 1985).

We need to select and schedule n jobs from a set $\mathcal{J}$ containing N jobs. Initially, we consider the case where there are no equal-size jobs, i.e., $p_j \neq p_\ell$ for all $j, \ell = 1, \ldots, N$, $j \neq \ell$. We index blocks $\mathcal{B}_b$, which represent sets of pairs (machine index, position on machine schedule), in chronological order with $1 \leqslant b \leqslant b_{\max}$, where block $B_{b_{\max}}$ is the final block in the schedule, which contains the final jobs on the machines with speed V_0. Additionally, let $\vec{C} = \left(C_{M_i}\right)_{1 \leqslant i \leqslant m}$ denote the vector of machine completion times. To facilitate reading, we define

$$\omega(\vec{C}, i, j) = \begin{cases} w_j \text{ if } C_{M_i} + p_j/V_i > d_j \\ 0 \text{ otherwise} \end{cases},$$

and $M(\mathcal{B}_b) = \{i / \exists (i,k) \in \mathcal{B}_b\}$ as the set of machine indices in block $\mathcal{B}_b$. We also define $\vec{C}' = \vec{C} \oplus (i, p_j)$ as ${C'}_{M_i} = C_{M_i} + p_j/V_i$ and ${C'}_{M_\ell} = C_{M_\ell}$ for $\ell \neq i$.
In our dynamic programming algorithm we work from behind. We assume that we have a schedule that follows the block structure in which each machine M_i $(i = 1, \ldots, m)$ has a given completion time C_{M_i}; the number of such possible completion times amounts to $\mathcal{O}(\sum p_j)$ per machine. In each step we select one job to fill a position on one of the machines in the current block (starting with block $b_{\max}$), until this block is full, after which we continue with the next block. To facilitate notation, we denote the set of *available* machines in the current step of the dynamic program by $\mathcal{A} \subset \mathcal{V}_0 \cup \mathcal{V}_1$. Here a machine is available, if it has a position in the currently considered block $\mathcal{B}_b$ to which we have not yet assigned a selected job.

We define $OPT[\vec{C}, b, \mathcal{A}, g]$ as the optimal value of the total weighted number of tardy jobs in the bilevel problem, given that all machine completion times are equal to $\vec{C}$, the next job will be scheduled on the current block $\mathcal{B}_b$, with the available machines $\mathcal{A}$ and where J_g is the smallest indexed job that the follower can schedule. Remark here that $OPT[\vec{C}, b, \mathcal{A}, g]$ only accounts for the cost of the jobs that have been selected. Since, all blocks, except maybe the first one, must be completely filled, we have that the block $\mathcal{B}_b$ with the set of available machines $\mathcal{A}$ is completely filled when full$(b, \mathcal{A})$ is true with

$$\text{full}\,(b, \mathcal{A}) \iff \begin{cases} |\mathcal{A}| = n - \sum_{2 \leqslant b' \leqslant b_{\max}} |B_{b'}| & \text{if } b = 1 \\ |\mathcal{A}| = 0 \ (i.e.\ \mathcal{A} = \emptyset) & \text{if } b > 1 \end{cases}.$$

We initialize the recursion with $OPT[\vec{C}, b, \mathcal{A}, g] = +\infty$ whenever $g > N$, and $OPT[\vec{C}, b, \mathcal{A}, g] = 0$ if $b = b_{\max}$ and full$(b, \mathcal{A})$ is true; this enforces that we have a schedule containing n (yet unselected) jobs. Our recurrence relation is then

$$OPT[\vec{C}, b, \mathcal{A}, g] = \begin{cases} \min\limits_{\substack{i \in \mathcal{A} \\ g \leqslant j \leqslant N}} \left(w(\vec{C}, i, j) + OPT[\vec{C} \oplus (i, p_j), b, \mathcal{A} \setminus \{i\}, j+1] \right) & \text{if full}(b, \mathcal{A}) \text{ is false} \\ OPT[\vec{C}, b+1, M(\mathcal{B}_{b+1}), g] & \text{otherwise} \end{cases} \tag{12}$$

The first line corresponds to the case that the current block is not completely full. Hence, we select the best job J_j with $g \leqslant j \leqslant N$ to be scheduled on the best available machine $M_i \in \mathcal{A}$. In the second line, the current block is completely filled, and we need to pass to the next block.

The value of the optimal solution is given by $OPT[\vec{0}, 1, M(\mathcal{B}_1), 1]$.

Next, we focus on the general case with arbitrary processing times. We apply the formula described above and when we face k equal-size jobs, we apply an algorithm that selects ℓ jobs, with $\ell = 0, \ldots, k$. The main idea of the algorithm is that these ℓ jobs can completely

fill the current block $\mathcal{B}_b$ and some more, and partially fill a last block. Consequently, we need to choose which machines on the last block will be selected; we enumerate all these possibilities. For each choice, we can determine the machine completion times since we have $\vec{C}$ and ℓ equal-size jobs with a processing time p_j. Then, we can find the best set ℓ out of these k equal-size jobs by applying the algorithm that we used to prove Theorem 2.2 in Section 2.3.1. Note that we can update the dynamic programming state when we use this algorithm for any value of ℓ and any choice of selected machines on the last block. This general case multiplies the time complexity by a factor of 2^m since we have to make $\mathcal{O}(2^m)$ choices and by a factor of $n \log n$ for the time complexity of the jobs' assignment.

Therefore, this dynamic programming algorithm requires $\mathcal{O}\left(2^m \left(\sum_{j=1}^{N} p_j\right)^m N n^2 \log n\right)$ time and space. Correspondingly, the following theorem holds.

Theorem 2.5 *The $Q|V_i \in \{V_0, V_1\}, OPT - n|Lex\left(\sum_j C_j^F, \sum_j w_j U_j^L\right)$ problem is $\mathcal{NP}$-hard in the strong sense and when the number of machines m is fixed it turns to be $\mathcal{NP}$-hard in the ordinary sense.*

Proof We have shown these results for the follower's problem, which is a special case of the bilevel problem, where the leader selects $n = N$ jobs. In the scenario where the number of machines is fixed, the moderate-exponential worst-case time complexity dynamic programming runs in pseudo-polynomial time. □

2.3.3 A related polynomially solvable bilevel scheduling problem

As a side result, we consider the adversarial bilevel optimization problem, where the follower still minimizes $\sum_j C_j^F$. In this scenario, the leader takes decisions so that the optimal solution of the follower's problem is the worst possible. For the rest of the section, we assume that the jobs are sorted according the LPT rule.

Theorem 2.6 *The $Q|ADV - n|\sum_j C_j^F$ problem can be solved in $\mathcal{O}(N \log(N))$ time as follows:*

(i) The leader selects the $n \leqslant N$ jobs with the largest processing times,
(ii) The follower applies the SPT-FAM rule, on these n jobs.

Proof The objective value of any schedule σ_s with n jobs determined by the follower can be rewritten as $\sum_{k=1}^{n} \omega_{[k]} p_{[k]}$, where $\omega_{[k]}$ and $p_{[k]}$ denote the kth smallest positional weight and the kth largest processing time of the selected jobs, for $k = 1, \ldots, n$.

Let us assume that the leader selects an optimal set $\mathcal{I}_1$ of n jobs such that they are not the ones with the largest processing times. Suppose that the leader selects job J_u instead of job J_v, whereas $p_u < p_v$. Let $\mathcal{I}_2$ denote the set of n jobs that we obtain by replacing job J_u with J_v in $\mathcal{I}_1$. Let $p_{[k]}^1$ and $p_{[k]}^2$ denote the kth largest processing time of the jobs in the set $\mathcal{I}_1$ and $\mathcal{I}_2$, respectively. Then we see that $p_{[k]}^1 \leq p_{[k]}^2$ for all $k = 1, \ldots, n$, where at least one of the inequalities is strict. Since the positional weights $w_{[k]}$ are positive and independent of the selected jobs, we see for the resulting schedules σ_s^1 and σ_s^2 created by the follower that

$$\sum_{j=1}^{n} C_j(\sigma_s^1) - \sum_{j=1}^{n} C_j(\sigma_s^2) = \sum_{k=1}^{n} w_{[k]}(p_{[k]}^1 - p_{[k]}^2) < 0,$$

which contradicts the optimality of the set $\mathcal{I}_1$. □

2.4 Properties of the solutions

In this section, we describe some dominance conditions between two partial solutions. Let s and s' be two partial solutions. We have s, that is composed of a partial schedule σ_s, a set of remaining jobs Ω_s, and a number of jobs k_s^1 (respectively k_s^0) that must be scheduled on high-speed (respectively low-speed) machines to complete the solution, i.e. $|\sigma_s| + k_s^1 + k_s^0 = n$. Similarly, we have s', that is composed of a partial schedule $\sigma_{s'}$, a set of remaining jobs $\Omega_{s'}$, and a number of jobs $k_{s'}^1$ (respectively $k_{s'}^0$) that must be scheduled on high-speed (respectively low-speed) machines to complete the solution, i.e. $|\sigma_{s'}| + k_{s'}^1 + k_{s'}^0 = n$. Schedules σ_s and $\sigma_{s'}$ start at time zero without any unnecessary idle time. We denote by $C_{M_i}(\sigma_s)$ the completion time of machine M_i in any partial solution s.

Theorem 2.7 *If the following conditions hold:*

(i) $C_{M_i}(\sigma_s) \leqslant C_{M_i}(\sigma_{s'}) \quad \forall i \in \{1, \ldots, m\}$,
(ii) $\sum_{j \in \sigma_s} w_j U_j(\sigma_s) \leqslant \sum_{j \in \sigma_{s'}} w_j U_j(\sigma_{s'})$,
(iii) $k_s^1 \leqslant k_{s'}^1$,
(iv) $k_s^0 \leqslant k_{s'}^0$,
(v) $\Omega_{s'} \subseteq \Omega_s$,

then the best solution s^* reachable from the partial solution s is better than the best solution s'^* reachable from the partial solution s'. We say that s dominates s'.

Proof Let $(\sigma_s||\alpha_s^1 \cup \alpha_s^0)$ be the best schedule that begins with σ_s where α_s^1 is a partial schedule on high-speed machines and α_s^0 is a partial schedule on low-speed machines. Both are composed only of jobs coming from Ω_s and $|\alpha_s^1| = k_s^1$ and $|\alpha_s^0| = k_s^0$. We say that $\alpha_s^1 \cup \alpha_s^0$ is the best complementary schedule from Ω_s containing $k_s^1 + k_s^0$ jobs.

We denote by $\tilde{\alpha}_s^1 \cup \tilde{\alpha}_s^0$ the best complementary schedule to σ_s with $k_s^1 + k_s^0$ jobs from $\Omega_{s'}$. As we have $\Omega_{s'} \subseteq \Omega_s$, we obtain:

$$\sum_{j \in (\sigma_s||\alpha_s^1 \cup \alpha_s^0)} w_j U_j(\sigma_s||\alpha_s^1 \cup \alpha_s^0) \leqslant \sum_{j \in (\sigma_s||\tilde{\alpha}_s^1 \cup \tilde{\alpha}_s^0)} w_j U_j(\sigma_s||\tilde{\alpha}_s^1 \cup \tilde{\alpha}_s^0)$$

Let $\alpha_{s'}^1 \cup \alpha_{s'}^0$ be the best complementary schedule to $\sigma_{s'}$ with $k_{s'}^1 + k_{s'}^0$ jobs from $\Omega_{s'}$, and as we have $k_s^1 \leqslant k_{s'}^1$ and $k_s^0 \leqslant k_{s'}^0$, scheduling $\alpha_{s'}^1 \cup \alpha_{s'}^0$ after σ_s guarantees that jobs in $\alpha_{s'}^1$ are executed on high-speed machines, while those in $\alpha_{s'}^0$ are executed on low-speed machines. We have:

$$\sum_{j \in \left(\sigma_s||\alpha_{s'}^1 \cup \alpha_{s'}^0\right)} w_j U_j(\sigma_s||\alpha_{s'}^1 \cup \alpha_{s'}^0) \geqslant \sum_{j \in (\sigma_s||\tilde{\alpha}_s^1 \cup \tilde{\alpha}_s^0)} w_j U_j(\sigma_s||\tilde{\alpha}_s^1 \cup \tilde{\alpha}_s^0) \geqslant \sum_{j \in (\sigma_s||\alpha_s^1 \cup \alpha_s^0)} w_j U_j(\sigma_s||\alpha_s^1 \cup \alpha_s^0)$$

Since the completion times of the machines in s are smaller than those in s', i.e. $\forall i \in \{1, \ldots, m\}$, $C_i(s) \leqslant C_i(s')$, we have:

$$\sum_{j \in \left(\sigma_{s'}||\alpha_{s'}^1 \cup \alpha_{s'}^0\right)} w_j U_j(\sigma_{s'}||\alpha_{s'}^1 \cup \alpha_{s'}^0) \geqslant \sum_{j \in \left(\sigma_s||\alpha_{s'}^1 \cup \alpha_{s'}^0\right)} w_j U_j(\sigma_s||\alpha_{s'}^1 \cup \alpha_{s'}^0) \geqslant \sum_{j \in (\sigma_s||\alpha_s^1 \cup \alpha_s^0)} w_j U_j(\sigma_s||\alpha_s^1 \cup \alpha_s^0)$$

So, s dominates s'. □

Conditions (iii) and (iv) state that more jobs have to be added to s' than to s; whereas condition (v) states that there are more remaining jobs for s than for s'. Given that the total number of jobs for the complete schedule is fixed, this does not imply that all these conditions are satisfied with an equality relation. We can construct an example where there are strict inequalities.

Remark For the leader's problem which is a selection problem, we could think about Knapsack-like dominance conditions, e.g. given two jobs J_j and J_ℓ, J_j dominates J_ℓ if and only if $p_j \leqslant p_\ell$ and $d_j \geqslant d_\ell$ and $w_j \leqslant w_\ell$. Such a condition would possibly hold in a single level optimization problem. However, in our bilevel setting, the following counterexample holds, where the leader has to select two jobs, with one machine of speed $V = 1$ and three jobs defined as follow:

Job	p_j	d_j	w_j
1	1	7	4
2	2	2	3
3	3	5	5

Here, the Knapsack-like dominance condition would indicate that J_1 dominates J_3 and, correspondingly, the schedule $(J_2\ J_3)$ should be worse than $(J_1\ J_2)$. However, $s = (J_2\ J_3)$ is better than $s = (J_1\ J_2)$ with a weighted number of tardy jobs equal to 0 against 3. Indeed, the follower's block structure imposes a specific assignment for the jobs that may invalidate the above dominance condition.

3 A branch-and-bound algorithm

3.1 Branching scheme

We propose a branch-and-bound algorithm that relies on a lower bound based on column generation. We use a branching strategy based on the block structure property of the follower's problem, i.e. we branch on the blocks in the order $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_{b_{\max}}$. At a given node, we have taken decisions on jobs $J_1, \ldots, J_{j-1}$, so we need to take a decision for job J_j. We have the following decisions:

- We select job J_j. Then we iterate over all available locations in the first available block from the parent node: let us denote by (i, k) such location. Then, we assign job J_j on the machine M_i at the position k. Thus, we create as many children nodes as available locations in this block.
- We do not select job J_j, so we remove it in the corresponding child node.

Consequently, if N_a is the number of available locations in the first available block, we create exactly $(N_a + 1)$ children nodes.

The nice aspect of this branching scheme is that we know the completion time of each machine. If we have k equal-size jobs of which we want to select and schedule ℓ ($\ell = 1, \ldots, k$) jobs, then we can use the algorithm that we used to prove Theorem 2.2 to find these ℓ jobs. We create a node for each ℓ value and for each machine's selection on the last block, resulting in $\mathcal{O}(2^m n)$ nodes.

3.2 Memorization

Shang et al. (2021) introduce the Branch-and-memorize paradigm, describing how to apply memorization within search tree algorithms for sequencing problems. The authors present several memorization schemes that can be used to prune the search tree and avoid solving identical or dominated subproblems.

The first scheme is solution memorization, which involves storing in memory the best solution reachable from each node. This approach can be used when the problem is decomposable, meaning that an optimal solution of a subproblem does not depend on the decisions taken to generate the subproblem. Before branching on a node, we check if there exists an optimal solution of the subproblem that has already been explored in memory. If so, we do not need to branch on this node.

The second scheme is called passive node memorization and involves storing information about the explored nodes in memory. This scheme requires a function $check(s, s')$ that returns *true* if partial solution s dominates partial solution s'. The idea is to check before branching on a node whether there exists a node in memory that dominates it.

The third and final scheme is predictive node memorization. Like passive node memorization, it shares the same idea: storing information about explored nodes in memory. Before branching on a node, we check whether there exists a node in memory that dominates it. If no such node exists, we try to create a partial solution that has not been explored in the tree and dominates the current branching node. If such a better partial solution is found, we store it in memory, and we do not branch on the current node since it is dominated.

Our problem is non-decomposable because the decision made during the partial schedule has a significant impact on the optimal solution for the remaining jobs. Moreover, each node $\mathcal{N}$ of the search tree represents a partial solution s, that is composed of a partial schedule σ_s, a set of remaining jobs Ω_s, and a number of jobs k_s^1 (respectively k_s^0) that must be scheduled on high-speed (respectively low-speed) machines to complete the solution.

In this context, we employ passive node memorization based on the function $check(s, s')$. To that end, we associate a data structure with each node. This structure consists of: a bit set $\{0, 1\}^N$ encoding the set of remaining jobs; an array with two parts representing the total completion time of high-speed (respectively low-speed) machines, both sorted in non-increasing order; the partial solution value $\sum_{j \in \sigma_s} w_j U_j(\sigma_s)$; and integers for k_s^1 and k_s^0. Thus, given two nodes $\mathcal{N}_1$ and $\mathcal{N}_2$, their data structures enable implementation of the dominance condition from Theorem 2.7 as a function $check(s_1, s_2)$.

We store the data structure of each explored node in a database Db. To accelerate the search for a dominating node, we group data structures in the database by the size $|\Omega_s|$. Thus, $Db[k]$ is the set of data structures associated with explored nodes for which $|\Omega_s| = k$. Consequently, given a node $\mathcal{N}_1$, we search for a dominating node $\mathcal{N}_2$ in $Db[k]$ (for $k = 1, \ldots, |\Omega_{s_1}|$).

However, the database can become full due to the exponential number of explored nodes. Therefore, we need to define a policy to clear it: at any time the database is full, we remove all nodes that have never dominated any other node.

As authored by Shang et al. (2021), the memorization mechanism has an impact on the choice of the most efficient search strategy. Experiments on that matter are reported in Sect. 4.

3.3 Initial upper bound computation

To compute a good upper bound at the root node of the search tree, we first compute a solution with the MIP by setting a time limit to 20 seconds (experimental results showed that 20 seconds is enough to often get near-optimal solutions).

We also use a heuristic that improves this solution. Let $s = (\sigma_s, \Omega_s)$ be a solution, with σ_s the schedule, $\mathcal{I}_s$ the set of jobs that have been scheduled, and Ω_s the set of jobs that have not been selected. The main idea, is to loop over each block $\mathcal{B}_b$, for $b = 1, \ldots, b_{\max}$. For a given block, we make all jobs in it available. Afterwards, we solve an assignment problem with all the jobs that can be scheduled in $\mathcal{B}_b$, i.e., jobs $J_j \in \mathcal{I}_s \cup \Omega_s$ with the processing time p_j such as $\max_{(i,k)\in\mathcal{B}_{b-1}} (p_{[i,k]}) \leqslant p_j \leqslant \min_{(i,k)\in\mathcal{B}_{b+1}} (p_{[i,k]})$ where $p_{[i,k]}$ is the processing time of the job in position k on machine M_i. Finally, we optimize all equal-size jobs using the same algorithm as described in the proof of Theorem 2.2 in Section 2.3.1.

3.4 Lower bound computation by Column Generation

The first lower bound we can easily obtain is based on the relaxed MIP. Decisions within the branch-and-bound algorithm can be reflected in the MIP formulation by fixing the corresponding variables to values of either 0.0 or 1.0. Furthermore, if the relaxation yields an integer solution, the node can be pruned immediately, since the optimal reachable solution for that branch has been identified. However, more efficient lower bounds for the leader's objective function exist in the literature.

We propose an extension of the column generation approach presented in Van den Akker et al. (1999) in order to compute a lower bound at each node of the search tree.

The master problem is formulated as a set-covering problem with an exponential number of binary variables and a linear number of constraints. We use *machine schedules*, which are defined as a sequence of jobs that can be assigned to any machine. In the master problem, we cannot enforce the block structure, because we do not know the selected jobs. We do know, however, that the machine schedules must be part of a potential schedule constructed by the follower that follows the block structure. Hence, we can impose several constraints on the machine schedules that we consider. First of all, we know that the jobs must be in SPT order. Second, since the block structure is independent of the selection of the jobs, we know that we must fill the blocks $\mathcal{B}_b$, for $b = 1, \ldots, b_{\max}$, which provides information on the number of jobs per machine. Since the blocks $\mathcal{B}_b$ are full for $b = 2, \ldots, b_{\max}$, we know the resulting number of jobs for each fast and slow machine. The only freedom with respect to the number of jobs per machine schedule that we have concerns the Q jobs in the first block, $\mathcal{B}_1$. If this block is not filled completely, then we must distribute these Q jobs over the relevant machines. This leads to machine schedules with a minimum or maximum number of jobs, which corresponds to a difference of one job. Finally, we can limit the maximum number of jobs with equal processing time that can occur in a machine schedule. We will work this out later.

Each machine schedule s is characterized by the following set of parameters:

$$a_{j,s} = \begin{cases} 1 \text{ if the job } J_j \text{ is in the schedule } s \\ 0 \text{ otherwise} \end{cases}$$

$$b_{s\gamma+} = \begin{cases} 1 \text{ if the machine schedule } s \text{ is to be scheduled on a machine with speed } V_\gamma \\ \quad \text{and contains the maximum number of jobs} \\ 0 \text{ otherwise} \end{cases}$$

$$b_{s\gamma-} = \begin{cases} 1 \text{ if the machine schedule } s \text{ is to be scheduled on a machine with speed } V_\gamma \\ \quad \text{and contains the minimum number of jobs} \\ 0 \text{ otherwise} \end{cases}$$

$$t_{j,s} = \begin{cases} 1 \text{ if the job } J_j \text{ is late in the schedule } s \\ 0 \text{ otherwise} \end{cases}$$

These constants can be computed for any given schedule s.

In the column generation approach, we aim to compute a lower bound for $\sum_j w_j U_j$. Notice that we cannot compute it as in the bilevel scenario, because we cannot impose a constraint that ensures that, when n jobs are selected, they must be scheduled to minimize $Lex(\sum_j C_j, \sum_j w_j U_j)$: this is a consequence of the block structure that cannot be enforced before we know the n selected jobs. As a result, we focus on minimizing the single-level $\sum_j w_j U_j$ problem in a scenario where we select n jobs. So, the optimal solution of our restricted master problem is a lower bound to the optimal solution of the master problem that is itself a lower bound to the optimal solution of the bilevel problem. For a schedule s, we associate the cost c_s defined as follows: $c_s = \sum_{j=1}^{N} w_j t_{j,s}$.

Let S be the set containing all feasible machine schedules. We introduce variables $x_s \in \{0, 1\}$, with $s \in S$, and $x_s = \begin{cases} 1 \text{ if the machine schedule } s \text{ is selected} \\ 0 \text{ otherwise} \end{cases}$

Now, our master problem consists of selecting m machine schedules that contain exactly n distinct jobs corresponding to the leader's decision, and must minimize the total cost. These m machine schedules are partitioned in m_0 and m_1 machine schedules for the machines with speed V_0 and V_1, respectively. Furthermore, these m_0 machine schedules are partitioned into m_{0+} and m_{0-} machine schedules with a maximum and minimum number of jobs; a similar partition holds for the m_1 machine schedules. If $\mathcal{B}_1$ corresponds to positions on machines with speed V_0 (or V_1) only, then we can compute the corresponding values of $m_{0+}, m_{0-}, m_{1+}, m_{1-}$, and we find the following master problem formulation:

$$\min \sum_{s \in S} c_s x_s \tag{13}$$

s.t.

$$\sum_{s \in S} a_{j,s} x_s \leqslant 1 \qquad \forall j = 1, \ldots, N \tag{14}$$

$$\sum_{s \in S} b_{s0+} x_s = m_{0+} \tag{15}$$

$$\sum_{s \in S} b_{s0-} x_s = m_{0-} \tag{16}$$

$$\sum_{s \in S} b_{s1+} x_s = m_{1+} \tag{17}$$

$$\sum_{s \in S} b_{s1-} x_s = m_{1-} \tag{18}$$

$$x_s \in \{0, 1\} \qquad \forall s \in S$$

In this formulation Constraints (14) ensure that each job is processed at most once. Constraints (15) and (16) ensure that we select the correct number of machine schedules for fast machines with the maximum and minimum number of scheduled jobs. Constraints (17) and (18) ensure the same for the slow machines. Constraints (15),(16),(17) and (18) ensure that we select exactly n jobs, reflecting the leader's decisions.

If $\mathcal{B}_1$ is processed on both high and low speed machines, then we don't know the exact values $m_{0+}, m_{0-}, m_{1+}, m_{1-}$. Nevertheless, we know that there are Q jobs to be scheduled in $\mathcal{B}_1$. Since there are m_0 fast machines, we know that we need a total of m_0 machine schedules for fast machines. Similarly, we need m_1 machine schedules for slow machines. Furthermore, we need a total of exactly Q machines schedules with a maximum number of jobs.

Given these considerations, we can rewrite the last four constraints as follows:

- Constraints (15) and (16) become : $\sum_{s\in S}(b_{s0+}+b_{s0-})x_s = m_0$. This constraint ensures that the total number of machine schedules for fast machines with minimal and maximal number of scheduled jobs selected is equal to the number of high speed machines.
- Constraints (17) and (18) become : $\sum_{s\in S}(b_{s1+}+b_{s1-})x_s = m_1$. This constraint ensures that the total number of machine schedules for slow machines with minimal and maximal number of scheduled jobs selected is equal to the number of low speed machines.
- We add a constraint $\sum_{s\in S}(b_{s0+}+b_{s1+})x_s = Q$ that ensures that we select exactly the number of machines with the maximum number of scheduled jobs to be equal to the number of jobs in the first block.

We obtain the linear programming relaxation by replacing the definition set of variables $x_s \in \{0, 1\}$ by $x_s \in [0, 1]$.

Generating Columns In the analysis below we refer to the model with constraints (15), (16), (17) and (18). The analysis can easily be adjusted to deal with the second model. We compute the reduced cost r_s of any machine schedule s as follows:

$$r_s = c_{s^\gamma} - \sum_{j=1}^{N} \lambda_j \, a_{j,s^\gamma} - \tilde{\lambda}_\gamma \tag{19}$$

$$\text{where } \tilde{\lambda}_\gamma = \begin{cases} \lambda_{0+} \text{ if } s \text{ is a fast machine with the maximum number of jobs} \\ \lambda_{0-} \text{ if } s \text{ is a fast machine with the minimum number of jobs} \\ \lambda_{1+} \text{ if } s \text{ is a slow machine with the maximum number of jobs} \\ \lambda_{1-} \text{ if } s \text{ is a slow machine with the minimum number of jobs} \end{cases}$$

and respectively λ_{0+},λ_{0-},λ_{1+} and λ_{1-} represent the given values of the dual variables corresponding to Constraints (15),(16), (17) and (18). Notice that $\lambda_1, \ldots, \lambda_N$ constitute the given values of the dual variables corresponding to constraints (14).

It is well-known from the linear programming theory that a solution to a minimization problem is optimal if the reduced cost of each non basic variable is nonnegative. Therefore, we need to determine if there exists a machine schedule s with such a negative cost. We solve the *pricing problem*, that involves finding a machine schedule in S with the minimum reduced cost. Removing the constant, we need to minimize:

$$r_s = c_s - \sum_{j=1}^{N} \lambda_j \, a_{j,s} = \sum_{j=1}^{N} \left(w_j t_{j,s} - \lambda_j \, a_{j,s} \right)$$

We achieve the calculation of the schedule with a minimal reduced cost by dynamic programming. It is applied for each speed V_0 and V_1 and for the maximum and minimum number of scheduled jobs.

Define the job groups $G_1, \ldots, G_c$ such that jobs within each group have equal processing time. We assume that $p(G_1) < p(G_2) < \cdots < p(G_c)$, where $p(G_g)$ denotes the processing time of any job in G_g. Thus, the set of jobs $\mathcal{J}$ is partitioned as $\mathcal{J} = G_1 \cup \cdots \cup G_c$. Assume that we solve the dynamic programming for speed V_γ. Define $F_k^\gamma(t, g)$ as the minimum reduced cost of a machine schedule with speed V_γ such that the first job from sets $G_g, \ldots, G_c$ starts at time t and where exactly k jobs have been selected from the job sets $G_g, \ldots, G_c$. The initialization is $F_k^\gamma(t, g) = 0$ for $k = 0$, and is $+\infty$ otherwise; this implies that we must start with an empty schedule. We find the best machine schedule for a minimum and maximum number of scheduled jobs in one run of the dynamic programming, adding the multiplier $\tilde{\lambda}_\gamma$ for the corresponding constraint at the end. Given the relevant values of $F_k^\gamma(t, g)$, we compute the values of $F_k^\gamma(t, g-1)$ by taking the jobs from G_{g-1} into account. If we select ℓ jobs from set G_{g-1}, then the first of these jobs will start at time t and the ℓ-th job will end at time $t + \ell p/V_\gamma$, where p is the processing time of all jobs in set G_{g-1}. We can use a subroutine to compute the cost of completing these ℓ jobs, which is a polynomial-time algorithm and is described at the end of the section. We denote $S(\ell, g-1, t)$ as the cost for completing ℓ jobs from group G_{g-1} that start at time t.

The advantage of recording the number of jobs already scheduled is two-fold. First, we only select machine schedules with the right number of jobs. Secondly, we can reduce the number of jobs that we select from a group G_g by setting $\ell_{\max}$ as the maximum number of jobs allowed from that group. Suppose that we want to determine $F_k^\gamma(t, g)$, including the jobs from G_g; we need to schedule k jobs on the machine with speed V_γ. Thus, using the block structure, we can identify which block $\mathcal{B}_h$ contains the k^{th} job from the end on the machine. If the leader selects all jobs from G_g with at least one job in block $\mathcal{B}_h$, then this will fill subsequent blocks, and we denote the last filled block as $\mathcal{B}_u$. Thus, $\ell_{\max}$ is the maximum number of jobs that can be assigned to one machine with speed V_γ from the relevant blocks $\mathcal{B}_h$ up to and including $\mathcal{B}_u$. These computations are performed as preprocessing – prior to solving any pricing problem – since the block structure remains unchanged throughout the column generation process.

The recurrence relation then becomes equal to:

$$F_k^\gamma(t, g-1) = \min_{0 \leqslant \ell \leqslant \ell_{\max}} \left\{ F_{k-\ell}^\gamma(t + \ell p/V_\gamma, g) + S(\ell, g-1, t) \right\}$$

In this formula ℓ corresponds to the number of jobs from G_{g-1} that we add; here $\ell = 0$ implies that no jobs from G_{g-1} are added.

The optimal value $\overset{\star}{F}{}^\gamma$ for the machine with the speed V_γ is obtained by computing $F_{\eta_{\gamma+}}^\gamma(0, 1)$, where $\eta_{\gamma+}$ is the maximum number of jobs that can be scheduled on machine with speed V_γ.

Now, let us return to the subroutine $S(\ell, g, t)$. First, if G_g contains only one job denoted J_g, then the cost of completing this single job is given by $S(\ell, g, t) = \begin{cases} -\lambda_g & \text{if } t + p_g/V_\gamma \leqslant d_g \\ w_g - \lambda_g & \text{if } t + p_g/V_\gamma > d_g \end{cases}$

Secondly, we consider a set of equal-size jobs that start at time t with n_g jobs $J_1, \ldots, J_{n_g}$ in G_g, where $p_1 = p_2 = \ldots = p_{n_g} = p$. We aim to minimize the reduced cost by selecting and scheduling some of these jobs. We consider all the possible scenarios where 1 job is selected, 2 jobs are selected, …, $\ell_{\max}$ jobs are selected. We solve the equivalent maximization reduced

cost problem, where each coefficient is multiplied by -1. Thus, each job has a contribution defined by $(\lambda_j - w_j U_j)$, with U_j equals 1 if the job is late 0 otherwise.

Let $v_j = \lambda_j$ be the contribution of job J_j when it is executed on time, and let $q_j = \lambda_j - w_j$ be its contribution of J_j when it executed tardy. There are three possibilities for job J_j: either it is selected as an on-time job, or as a tardy job, or not at all. A naive approach to solving this problem involves solving $\ell_{\max}$ assignment problems, one for each scenario where we select $\ell = 1, \ldots, \ell_{\max}$ jobs. The total time complexity of this naive algorithm that computes all possible schedules of negative reduced cost is $\mathcal{O}({n_g}^4)$. We propose below a dedicated algorithm with a total time complexity of $\mathcal{O}({n_g}^3)$.

Since we do not know the number of jobs that need to be selected beforehand, this may force us to select a job even if its contribution is negative when tardy. To do so, we subtract some value M from each job's contribution. This step aims to reduce to two the number of choices for each job. If the job is on-time or tardy but its contribution is positive, i.e. $v_j - M \geqslant 0$ or $q_j - M \geqslant 0$, then we select it. Otherwise, the job's contribution is negative, and we do not select it at all. We can adjust the number of jobs that are selected by modifying the value of M.

Assume that M is given, and define, for each job J_j, the value w'_j as the difference between being on time with adjusted value $(v_j - M)$ and the second option where either the job is executed tardy with the adjusted value $(q_j - M)$ or it is not selected. Thus, the value w'_j becomes $w'_j = (v_j - M) - \max(0, q_j - M) = \min(v_j - M, v_j - q_j)$. The problem now essentially becomes a $1||\sum w'_j U_j$ problem, which can be solved using Dynamic Programming which assumes that the jobs are in EDD order. Let $f_j(k)$ be the total cost of the on-time jobs from $\{J_1, \ldots, J_j\}$, where we have exactly k jobs on time. The recurrence relation is

$$f_{j+1}(k) = \max\{f_j(k),\, f_j(k-1) + w'_{j+1}\} \qquad \text{if } d_{j+1} \geq t + kp,$$

and $f_j(k)$, otherwise. The running time of this algorithm is $\mathcal{O}({n_g}^2)$, since k and j run from 0 to n_g. We can reduce the running time of the above dynamic programming algorithm to $\mathcal{O}(n_g)$ by renumbering the jobs in order of non-increasing w'_j value and using a clever data structure; we describe this algorithm in Appendix A.

We focus on choosing the value of M. First, we select a small perturbation $\epsilon > 0$; since scheduling or removing a job is allowed by the differences $v_j - M$ and $q_j - M$, we have at least four possible values of M: $M = v_j - \epsilon$, $M = v_j + \epsilon$, $M = q_j - \epsilon$, and $M = q_j + \epsilon$. Furthermore, the value of M impacts $w'_j = \min(v_j - M, v_j - q_j)$; for all $M < q_j$, w'_j is constant and equal to $v_j - q_j$, while it becomes linear for $M \geqslant q_j$. Since w'_j determines the order in which jobs are carried out, we need to add values of M where the linear part of w'_j intersects the constant part of w'_k for all distinct jobs $j, k = 1, \ldots, n_g$, specifically at $M = v_j - v_k + q_k + \epsilon$ and $M = v_j - v_k + q_k - \epsilon$. If we must select all jobs, the value of M is set to $\min\limits_{1 \leqslant j \leqslant n_g} \{q_j\} - \epsilon$.

If there are equal contributions for on-time jobs, we sort them in non-increasing order of weight w_j, since we prefer to select the job with the greatest weight first. Conversely, if there are equal contributions for tardy jobs, we sort them in non-decreasing order of weight w_j, since we prefer to select the job with the smallest weight first. After sorting, we apply a slight perturbation to the contributions to ensure that all M values become distinct, leading to the computation of $\mathcal{O}(n_g^2)$ distinct values of M.

Finally, as the value of M decreases, the number of selected jobs increases. Therefore, we sort the M values in non-increasing order. If, for a given value of M, the number of selected jobs exceeds $\ell_{\max}$, then we can terminate the subroutine.

So, the total complexity of the algorithm that computes all possible schedules of negative reduced cost is $\mathcal{O}({n_g}^3)$ time.

Notably, in many cases, $\ell_{\max}$ is small, typically less than or equal to three. In these situations, we employ a brute-force approach that exhaustively tests all possible combinations of job selection. For the cases where $\ell_{\max} \geqslant 4$ jobs, we use the algorithm described above. For implementation details on how to optimize the column generation approach we refer to B. Also, we attempted to use heuristics to accelerate the generation of improved columns; unfortunately, experimental results indicate that this approach does not enhance the method. Conversely, generating multiple columns during the backtracking phase in dynamic programming has shown to improve the method, as evidenced by our experiments where generating three columns during this phase yields best performances.

To resume, the pricing problem is solved by a dynamic programming that requires $\mathcal{O}\left(\sum_{j=1}^{N} p_j \times N \times n^3\right)$ time and space in the worst case.

4 Experimental results

In this section we computationally compared CPLEX solver version 22.1.1. applied to the MIP formulation (denoted as MIP) and the branch-and-bound algorithm (denoted as BaB). All algorithms were coded in langague C++, and the tests were run on a PC 2GHz AMD Epyc 7702. The source code can be found on the GitHub repository at (Schau, 2025). We randomly generated instances where we have m_1 high-speed machines with their speed $V_1 = 2$ and m_0 low-speed machines with their speed $V_0 = 1$. The processing times were drawn at random from the uniform distribution $U[1, 100]$ and weights from the uniform distribution $U[1, 10]$. Due dates were generated using the uniform distribution $U[P \times (1 - tf - rdd/2), P \times (1 - tf + rdd/2)]$ with $tf, rdd \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$ and $P = \frac{\sum_{j=1}^{N} p_j}{m_1 \times V_1 + m_0 \times V_0}$. Here, tf and rdd define a class of instances and we have 25 classes of instances. For each class, we randomly generated 10 instances of size $N = 40, 50, 60, 70, 80, 90, 100$. For each value of N we set $n = N/4, N/2, 3N/4$, where values were rounded down in case they were fractional. Moreover, we set the total number of machines to $m = 2, 4$, where we have the same number of high-speed and low-speed machines.

We divide these instances into easy instances for which, all exact methods solve them to optimality in at most 300 seconds and hard instances for which, at least one method does not solve them to optimality within this time budget.

Primary experimental results indicated that using a depth-first search strategy yields the best results for the BaB algorithm. In the remainder, we consider this version of BaB.

Table 1 presents a comparison between the impact of memorization and lower bounds derived from column generation and the MIP relaxation. In this table, we report the number of instances solved to optimality (#OPT) out of 250 by the BaB, where the lower bound used is either column generation or MIP relaxation. For each combination, the impact of memorization was measured. The average computation time and the average number of nodes explored are also presented.

Table 1 shows that memorization has an impact on the average number of nodes explored, especially for the column generation lower bound, where it can be reduced by a factor of 2. Furthermore, the number of instances solved optimally improves significantly with memorization, and this improvement tends to decrease as the values of N and n increase (on 2 machines, there were 246 optimally solved instances for $N = 50$, $n = 3N/4$, with memorization, versus 133 without; while 135 optimally solved instances were achieved for

Table 1 Contribution of the lower bounds and the memorization in the branch and bound

			Column Generation						MIP Relaxation					
			With Memorization			Without Memorization			With Memorization			Without Memorization		
m	N	n	#OPT	T_{avg}	$\mathcal{N}_{avg}$	#OPT	T_{avg}	$\mathcal{N}_{avg}$	#OPT	T_{avg}	$\mathcal{N}_{avg}$	#OPT	T_{avg}	$\mathcal{N}_{avg}$
2	40	10	250/250	0.10	0	250/250	0.09	0	250/250	0.10	0	250/250	0.09	0
		20	250/250	1.69	16.12	250/250	1.72	39.34	250/250	2.71	626.18	240/250	12.82	5782.82
		30	**250/250**	12.21	491.15	244/250	34.50	6028.60	237/250	50.91	14607.12	180/250	86.91	30469.61
	50	12	250/250	0.16	0	250/250	0.16	0	250/250	0.16	0	250/250	0.16	0
		25	250/250	4.97	80.70	250/250	6.13	456.10	242/250	24.22	7189.92	216/250	42.40	13521.18
		37	**246/250**	57.20	3029.46	133/250	168.66	23076.10	135/250	168.74	32689.37	85/250	199.32	48292.93
	60	15	250/250	0.33	0	250/250	0.32	0	250/250	0.36	0	250/250	0.32	0
		30	**250/250**	13.27	489.26	239/250	39.55	6769.91	204/250	71.98	15547.62	177/250	89.32	20396.14
		45	**196/250**	164.37	5697.46	68/250	226.14	25165.11	74/250	229.49	30087.56	53/250	237.33	38622.33
	70	17	250/250	0.67	0.09	250/250	0.66	0.09	250/250	1.75	375.49	249/250	1.78	339.27
		35	**245/250**	51.73	2490.22	165/250	124.96	13576.08	145/250	142.90	21987.64	121/250	155.69	26548.24
		52	**77/250**	234.45	5103.07	48/250	244.50	19766.14	53/250	243.91	23408.31	46/250	245.56	29230.37
	80	20	250/250	1.98	11.35	250/250	2.08	30.11	247/250	8.18	1971.52	239/250	14.16	3603.16
		40	**218/250**	91.11	3423.01	142/250	146.04	16070.72	127/250	156.38	19083.84	117/250	161.66	21377.36
		60	**42/250**	252.81	5373.04	39/250	253.91	22316.36	39/250	253.94	17270.48	40/250	252.80	20944.55
	90	22	250/250	2.55	11.09	250/250	2.67	19.95	246/250	11.10	2277.82	236/250	17.99	3773.05
		45	**171/250**	138.07	4219.24	123/250	161.00	16664.28	116/250	166.33	15553.80	111/250	168.08	17249.09
		67	**39/250**	254.37	3511.56	38/250	255.57	10029.45	38/250	255.34	13565.66	37/250	256.45	16578.62
	100	25	250/250	5.03	44.84	250/250	6.25	126.71	233/250	29.90	4946.79	220/250	37.27	6319.21
		50	**135/250**	154.50	4157.50	115/250	166.83	16531.24	113/250	169.47	12707.86	109/250	171.91	14281.18
		75	36/250	257.93	1945.08	36/250	257.91	7521.99	37/250	256.80	10568.54	36/250	257.91	12612.74

Table 1 continued

			Column Generation						MIP Relaxation					
			With Memorization			Without Memorization			With Memorization			Without Memorization		
m	N	n	#OPT	T_{avg}	$\mathcal{N}_{avg}$	#OPT	T_{avg}	$\mathcal{N}_{avg}$	#OPT	T_{avg}	$\mathcal{N}_{avg}$	#OPT	T_{avg}	$\mathcal{N}_{avg}$
4	40	10	250/250	0.11	0	250/250	0.11	0	250/250	0.11	0	250/250	0.11	0
		20	250/250	2.50	229.96	250/250	3.51	2384.26	245/250	8.66	4221.73	244/250	9.05	4231.72
		30	**178/250**	126.06	72809.14	124/250	169.23	203121.43	84/250	200.26	86176.71	84/250	200.25	89429.28
	50	12	250/250	0.15	0	250/250	0.15	0	250/250	0.16	0	250/250	0.15	0
		25	**248/250**	20.21	10369.75	230/250	47.34	46246.86	161/250	108.03	38936.98	161/250	108.03	36031.48
		37	**97/250**	201.22	86874.20	77/250	214.33	178115.77	62/250	226.12	60112.54	63/250	224.99	66197.84
	60	15	250/250	0.43	0	250/250	0.42	0	250/250	0.42	0	250/250	0.42	0
		30	**230/250**	59.73	30900.20	178/250	99.98	92046.02	142/250	130.55	34588.36	142/250	130.55	31595.90
		45	**71/250**	219.24	93502.40	70/250	219.49	182885.89	59/250	229.90	45350.09	60/250	228.77	54392.71
	70	17	250/250	0.91	0	250/250	0.90	0	250/250	0.91	0	250/250	0.89	0
		35	**180/250**	106.57	46348.34	151/250	125.57	86967.41	134/250	140.28	26114.50	134/250	140.27	24994.90
		52	**59/250**	234.09	88192.28	55/250	236.85	163612.06	52/250	238.32	34086.12	52/250	238.31	40623.52
	80	20	250/250	1.60	28.73	250/250	1.75	174.03	246/250	6.04	1630.64	246/250	6.04	1478.28
		40	**146/250**	135.89	51624.48	132/250	146.24	90280.77	125/250	151.29	20910.31	126/250	150.16	20484.02
		60	51/250	239.83	85068.54	51/250	239.82	154665.02	51/250	239.84	24196.32	51/250	239.82	28044.93
	90	22	250/250	3.60	406.64	250/250	5.66	2149.84	228/250	27.60	6421.28	228/250	27.58	5733.84
		45	**134/250**	148.26	53423.87	124/250	154.84	91833.30	120/250	157.56	16315.30	120/250	157.54	16987.01
		67	**44/250**	249.39	74288.59	43/250	249.46	137636.36	43/250	249.49	18098.15	43/250	249.46	22541.43
	100	25	250/250	5.85	927.34	244/250	14.18	7379.41	217/250	40.80	7737.79	217/250	40.79	7155.38
		50	**131/250**	148.22	53438.56	128/250	150.64	89807.43	124/250	153.12	13138.64	124/250	153.10	13513.20
		75	44/250	248.84	63985.30	45/250	247.69	110859.90	44/250	248.86	14377.85	44/250	248.81	18612.44

Table 2 Computation times on easy instances

				MIP			*BaB*		
m	N	n	#OPT	T_{min}	T_{avg}	T_{max}	T_{min}	T_{avg}	T_{max}
2	40	10	250/250	0.05	0.08	0.33	0.06	0.10	0.34
		20	250/250	0.16	4.41	238.83	0.17	**1.69**	**23.49**
		30	222/250	0.38	17.08	281.96	0.41	**9.65**	**57.58**
	50	12	250/250	0.08	0.14	3.18	0.08	0.16	3.42
		25	240/250	0.30	10.02	289.95	0.32	**4.15**	**29.08**
		37	153/250	0.76	46.59	271.72	0.79	**36.94**	**257.11**
	60	15	250/250	0.15	0.30	3.23	0.16	0.33	3.57
		30	219/250	0.49	15.57	208.50	0.52	**9.33**	**78.00**
		45	68/250	1.26	**25.57**	**214.27**	1.35	33.67	280.27
	70	17	250/250	0.20	0.72	43.22	0.22	**0.67**	**20.62**
		35	163/250	0.76	25.52	284.26	0.84	**23.29**	**250.97**
		52	48/250	1.82	12.93	225.60	1.97	**11.08**	**213.92**
	80	20	249/250	0.25	4.05	280.34	0.25	**1.90**	**24.71**
		40	130/250	1.05	**10.17**	235.28	1.13	11.08	**232.34**
		60	40/250	2.46	10.56	256.70	2.69	**7.25**	**109.49**
	90	22	248/250	0.41	4.94	123.05	0.38	**2.38**	**28.40**
		45	120/250	1.62	15.53	295.09	1.42	**14.48**	**244.23**
		67	39/250	4.12	13.25	235.88	3.99	**7.36**	**58.42**
	100	25	241/250	0.39	8.38	223.46	0.42	**4.02**	**57.06**
		50	115/250	2.16	14.36	**252.40**	2.10	**10.65**	270.16
		75	35/250	5.88	8.10	12.06	4.86	7.33	10.58

Table 2 continued

				MIP			*BaB*		
m	N	n	#OPT	T_{min}	T_{avg}	T_{max}	T_{min}	T_{avg}	T_{max}
4	40	10	250/250	0.06	0.10	0.20	0.07	0.11	0.27
		20	250/250	0.17	2.63	69.37	0.19	**2.50**	**48.74**
		30	130/250	0.41	38.95	292.22	0.43	**32.01**	**273.55**
	50	12	250/250	0.08	0.14	0.44	0.09	0.15	0.48
		25	230/250	0.35	29.90	294.48	0.36	**15.29**	**290.51**
		37	75/250	0.74	25.00	286.44	0.76	**13.31**	**163.28**
	60	15	250/250	0.19	0.39	4.18	0.21	0.43	4.16
		30	169/250	0.46	20.27	**264.98**	0.50	**12.59**	277.34
		45	62/250	1.22	7.46	155.13	1.31	**4.18**	**29.91**
	70	17	250/250	0.15	0.84	17.11	0.16	0.91	18.75
		35	141/250	0.67	8.75	292.39	0.71	**4.43**	**165.29**
		52	52/250	1.99	3.18	17.51	2.14	3.45	19.11
	80	20	250/250	0.24	1.70	45.36	0.26	**1.60**	**23.39**
		40	129/250	1.01	**4.19**	**104.82**	1.12	4.86	161.82
		60	51/250	2.71	4.65	14.40	2.82	5.03	15.91
	90	22	247/250	0.38	7.15	172.96	0.37	**3.38**	**54.42**
		45	121/250	1.40	**4.16**	**75.68**	1.26	4.58	163.72
		67	43/250	4.62	6.89	11.51	3.87	6.23	10.21
	100	25	242/250	0.45	14.09	277.84	0.49	**4.79**	**109.76**
		50	125/250	2.03	7.03	199.18	1.87	**4.13**	**37.91**
		75	44/250	5.55	10.25	19.48	5.00	9.29	17.21

Table 3 Computation times on hard instances

				MIP				*BaB*		
m	N	n	#OPT	T_{min}	T_{avg}	T_{max}	#OPT	T_{min}	T_{avg}	T_{max}
2	40	30	0/28	300.00	300.00	300.01	**28/28**	21.74	32.53	47.07
	50	25	0/10	300.00	300.00	300.01	**10/10**	21.42	24.60	29.38
		37	4/97	37.38	293.18	300.01	**93/97**	32.10	89.17	300.01
	60	30	0/31	300.00	300.01	300.01	**31/31**	23.86	41.09	96.98
		45	8/182	38.16	293.03	300.01	**128/182**	32.74	213.20	300.02
	70	35	3/87	48.70	292.15	300.01	**82/87**	36.66	105.01	300.03
		52	18/202	55.45	284.28	300.02	**29/202**	90.13	287.53	300.08
	80	20	0/1	300.01	300.01	300.01	**1/1**	21.52	21.52	21.52
		40	7/120	39.73	290.75	300.02	**88/120**	41.38	177.81	300.08
		60	**13/210**	18.50	289.43	300.02	2/210	254.36	299.58	300.31
	90	22	0/2	300.01	300.01	300.01	**2/2**	22.77	24.48	26.19
		45	10/130	33.40	286.69	300.02	**51/130**	27.35	252.15	300.10
		67	**10/211**	23.97	292.05	300.04	0/211	300.00	300.02	300.31
	100	25	0/9	300.01	300.01	300.01	**9/9**	22.86	32.05	46.26
		50	9/135	55.04	289.72	300.08	**20/135**	30.04	277.04	300.09
		75	**8/206**	31.68	294.01	300.05	0/206	300.00	300.01	300.05
4	40	30	21/120	31.58	272.54	300.01	**48/120**	22.66	227.94	300.01
	50	25	2/20	26.30	273.00	300.01	**18/20**	21.23	76.85	300.01
		37	11/175	19.01	288.75	300.01	**22/175**	29.77	281.76	300.01
	60	30	2/81	122.81	296.64	300.01	**61/81**	21.34	158.07	300.01
		45	**10/188**	18.75	288.37	300.02	9/188	30.36	290.16	300.01
	70	35	6/109	32.79	287.05	300.02	**39/109**	22.74	238.69	300.01
		52	7/198	22.14	293.49	300.02	7/198	88.86	294.66	300.05
	80	40	9/121	21.02	283.28	300.02	**17/121**	34.14	275.59	300.02
		60	**13/199**	30.08	288.91	300.02	0/199	300.00	300.00	300.06
	90	22	0/3	300.01	300.01	300.01	**3/3**	21.40	22.03	22.98
		45	**15/129**	26.69	278.60	300.21	13/129	52.13	283.04	300.02
		67	**14/207**	33.33	287.40	300.06	1/207	278.99	299.90	300.05
	100	25	0/8	300.01	300.01	300.01	**8/8**	22.02	37.87	65.46
		50	**18/125**	27.24	277.83	300.03	6/125	49.57	292.31	300.10
		75	**8/206**	31.68	294.01	300.05	0/206	300.00	300.01	300.05

$N = 100, n = N/2$, with memorization, versus 115 without). The average computation time also decrease when memorization is used with column generation.

However, memorization has less impact when the MIP relaxation is used as the lower bound. Indeed, the reduction in the average number of nodes is less significant. Moreover, with memorization, the number of instances solved optimally does not increase and the computation time does not decrease on 4 machines. On 2 machines, memorization provides a slight improvement in both the average number of optimally solved instances and the computational times.

Lastly, column generation with memorization solves all instances optimally over 250 up to $N = 40$ when $n = 3N/4$ for 2 machines, and up to $N = 60$ for 2 machines and $N = 40$ for 4 machines when $n = N/2$. For $n = N/4$, all instances over 250 were solved optimally. In contrast, the BaB with the MIP relaxation and memorization can solve instances optimally for $n = N/4$ until $N = 70$ for 2 and 4 machines, and only for $N = 40$, $n = N/2$, for 2 machines.

Table 2 presents the results on easy instances. In this table, we report the number of instances solved to optimality (#OPT) out of the 250 per class by both algorithms. For each algorithm, we provide the minimum, average, and maximum computation times, in seconds, for each value of N and n.

Table 3 presents the results for hard instances. There, #OPT shows the number of instances solved to optimality by each method, as well as the minimum, average, and maximum computation times.

Tables 4 and 5 present the minimum, average, and maximum number of nodes explored by both methods for easy and hard instances, respectively.

Tables 2 and 3 demonstrate that BaB enables solving more instances to optimality with a significant improvement in computation time compared to MIP. MIP can solve to optimality all instances with $n = N/2$ and $N = 40$ on two and four machines, whereas BaB can solve all instances for $n = N/2$ until $N = 60$ for two machines and until $N = 40$ for four machines. Additionally, for $n = 3N/4$ and for two machines, MIP does not solve to optimality all instances for $N = 40$, whereas BaB can solve these instances. Furthermore, although MIP can solve more instances for $n = 3N/4$, $N \geqslant 80$ on two machines, for $N = 60, 80, 90, 100$ on four machines, and for $n = N/2$, $N \geqslant 90$ on four machines on hard instances, on average on easy instances, BaB is faster. Nevertheless, in terms of computation time, BaB is generally faster than MIP.

Moreover, Tables 4 and 5 demonstrate that for instances solved optimally with $n = N/4$ and $N = 40, 50, 60$ on two machines and for $N = 40, 50, 60, 70$ on four machines, the number of nodes explored by BaB is 0. Since we run MIP for several seconds to compute an upper bound, it follows that MIP has identified and validated the optimal solution. Moreover, with $n = 3N/4$ and $N = 100$ on two machines and $N = 90, 100$ on four machines both MIP and BaB have explored 0 nodes meaning that the MIP found the optimal solution at the root node. For hard instances, BaB explores significantly fewer nodes (at least a factor of 2 reduction on average until $N = 90$ and $n = N/2$) compared to MIP for two machines. However, for instances with four machines, MIP appears to explore fewer nodes. This disparity can be attributed to the branching schemes used by each algorithm, specifically $\mathcal{O}((m+1)^N)$ for BaB. Although this scheme remains competitive with MIP for two machines, its exponential growth with respect to the number of machines has a negative impact on BaB's performance due to the increased number of lower bound computations, which are relatively time-consuming.

Since the leader must select n out of N jobs, leading to $\binom{N}{n}$ possibilities, the combinatorics of selection is greatest when $n = N/2$. Furthermore, we observe that this value does not necessarily lead to the most challenging scenario for the bilevel problem, whereas it does for the leader's problem. This difference can be attributed to the block structure of the follower's problem, as having more jobs allows for more permutations within this block compared to the selection options of the leader.

Table 4 Number of explored nodes on easy instances

			MIP			*BaB*		
m	N	n	$\mathcal{N}_{min}$	$\mathcal{N}_{avg}$	$\mathcal{N}_{max}$	$\mathcal{N}_{min}$	$\mathcal{N}_{avg}$	$\mathcal{N}_{max}$
2	40	10	0	11.12	1223	0	**0**	**0**
		20	0	1750.36	97543	0	**16.12**	**1296**
		30	0	4743.21	125273	0	**351.29**	**6982**
	50	12	0	2.19	158	0	**0**	**0**
		25	0	2815.82	59516	0	**58.85**	**3028**
		37	0	8805.35	83637	0	**1883.87**	**35868**
	60	15	0	73.30	2637	0	**0**	**0**
		30	0	3955.66	62184	0	**322.83**	**10733**
		45	0	2400.21	**20040**	0	**983.43**	25055
	70	17	0	255.98	23073	0	**0.09**	**22**
		35	0	3760.82	47013	0	**1788.77**	**45878**
		52	0	666.58	15805	0	**330.10**	**11252**
	80	20	0	1257.62	87631	0	**11.22**	**451**
		40	0	1099.03	16593	0	**477.63**	**14868**
		60	0	200.78	6455	0	**147.43**	**5897**
	90	22	0	1185.97	27810	0	**10.14**	**721**
		45	0	1343.53	31843	0	**950.58**	**31798**
		67	0	338.74	11758	0	**46.77**	**1824**
	100	25	0	1593.78	43199	0	**27.44**	**1028**
		50	0	782.28	**12303**	0	**478.90**	14185
		75	0	0	0	0	0	0
4	40	10	0	11.53	651	0	**0**	**0**
		20	0	1678.14	**29660**	0	**229.96**	47091
		30	0	**11227.68**	142538	0	13356.42	**123613**
	50	12	0	25.62	1939	0	**0**	**0**
		25	0	10473.10	**115001**	0	**7011.52**	185475
		37	0	3184.11	**45725**	0	**1862.84**	57337
	60	15	0	120.92	3581	0	**0**	**0**
		30	0	**4412.47**	**59521**	0	4906.64	165832
		45	0	489.77	11873	0	**29.31**	**781**
	70	17	0	486.33	15517	0	**0**	**0**
		35	0	1080.67	53012	0	**305.06**	**33456**
		52	0	44.92	1125	0	**0**	**0**
	80	20	0	765.20	29961	0	**28.73**	**2848**
		40	0	**425.82**	**12639**	0	908.82	67507
		60	0	29.67	1000	0	**0**	**0**
	90	22	0	1952.83	53102	0	**402.90**	**23698**
		45	0	**104.36**	**5175**	0	1450	175450
		67	0	0	0	0	0	0

Table 4 continued

			MIP			*BaB*		
m	N	n	$\mathcal{N}_{min}$	$\mathcal{N}_{avg}$	$\mathcal{N}_{max}$	$\mathcal{N}_{min}$	$\mathcal{N}_{avg}$	$\mathcal{N}_{max}$
	100	25	0	3079.48	73059	0	**640.06**	**50943**
		50	0	184.66	**11039**	0	**170.06**	20998
		75	0	0	0	0	0	0

Table 5 Number of explored nodes on hard instances

			MIP			*BaB*		
m	N	N	$\mathcal{N}_{min}$	$\mathcal{N}_{avg}$	$\mathcal{N}_{max}$	$\mathcal{N}_{min}$	$\mathcal{N}_{avg}$	$\mathcal{N}_{max}$
2	40	30	13370	32334.43	107382	88	**1600.04**	**4529**
	50	25	31952	63335.10	120191	93	**605**	**1598**
		37	3510	28988.97	94478	192	**4836.41**	**30263**
	60	30	14447	39547.10	151066	126	**1665**	**7376**
		45	1062	17375.68	65564	150	**7458.74**	**53426**
	70	35	5318	31274.66	115348	252	**3804.43**	**30983**
		52	905	11083.59	**31534**	47	**6237.24**	33393
	80	20	87389	87389	87389	44	**44**	**44**
		40	4078	21749.87	46693	199	**6613.83**	**33435**
		60	651	8811.28	**23618**	7	**6368.40**	33021
	90	22	30112	35857.50	41603	68	**129**	**190**
		45	1907	13106.02	**31343**	218	**7236.45**	43447
		67	453	5651.15	**15086**	5	**4151.98**	48365
	100	25	24904	38067.78	50699	56	**511**	**1169**
		50	1132	10312.82	**24680**	31	**7291.13**	77726
		75	120	4521.07	**15387**	0	**2261.73**	23174
4	40	30	7650	**51431.79**	**119710**	2031	137216.27	280826
	50	25	13338	56591.95	**104747**	358	**48989.40**	272139
		37	5024	**26850.71**	**56562**	1674	123307.63	279241
	60	30	20591	**46902.17**	**108168**	265	85133.67	294619
		45	1144	**11866.77**	**34792**	851	124328.63	286789
	70	35	4252	**25217.14**	**62082**	276	105908.92	382526
		52	668	**8579.62**	**27620**	1670	111353.89	300045
	80	40	1498	**18249.76**	**64581**	1459	105693.24	307970
		60	532	**6532.13**	**21029**	6779	106870.03	283827
	90	22	29022	47880	73080	304	**714.33**	**1198**
		45	559	**11355**	**39587**	1988	102174.56	283924
		67	0	**4608.57**	**14373**	1898	89720.52	282359

Table 5 continued

			MIP			*BaB*		
m	N	N	$\mathcal{N}_{min}$	$\mathcal{N}_{avg}$	$\mathcal{N}_{max}$	$\mathcal{N}_{min}$	$\mathcal{N}_{avg}$	$\mathcal{N}_{max}$
	100	25	25421	49045.75	81798	811	**9617.50**	**30812**
		50	899	**8238.67**	**21670**	1527	106707.06	310613
		75	83	**3053.86**	**10523**	3100	77652.06	279453

5 Conclusion

In this paper, we tackled a challenging bilevel parallel machine scheduling problem in an optimistic setting, i.e. when the follower returns among all its optimal solutions, the one that leads to the leader's optimal value. We have shown the $\mathcal{NP}$-hardness in a strong sense of the bilevel problem. We also developed several exact approaches, namely, a moderately exponential-time dynamic programming algorithm, a MIP formulation and a branch-and-bound algorithm using memorization mechanism in which we use column generation to compute a lower bound. The algorithms can be adapted to the pessimistic setting by considering the lexicographic objective $Lex\left(\sum_j C_j^F, -\sum_j w_j U_j^L\right)$, where the follower selects, among its optimal schedules, the one that minimizes $-\sum_j w_j U_j^L$.

While the moderately exponential-time dynamic programming algorithm is only relevant for theoretical purposes, our experimental results demonstrate that the branch-and-bound algorithm outperforms the MIP except for large instances where its performance deteriorates especially when dealing with more machines. Both approaches can efficiently solve all instances with up to 40 jobs and 2 or 4 machines.

These results show that the considered bilevel scheduling problem is very challenging and remains difficult to solve in practice, even for a dedicated exact algorithm. At first glance, this seems surprising as optimizing only the follower's objective function can be done in polynomial time. Moreover, the hardest scenario for the leader's problem is not the hardest for the bilevel problem. As a future research direction, we intend to develop heuristic approaches to tackle large instances. These heuristics could exploit the BaB algorithm as a beam-search heuristic or leverage machine learning techniques to approximate the follower's decisions

Acknowledgements The authors benefitted from the use of the cluster at the Centre de Calcul Scientifique en région Centre-Val de Loire. The authors would like to thank the two anonymous referees for their helpful comments and suggestions on the earlier version of this article.

Funding Open access funding provided by Politecnico di Torino within the CRUI-CARE Agreement.

Declarations

Conflicts of Interest The authors declare that they have no conflict of interest.

Ethical Approval This article does not contain any studies with human participants or animals performed by any of the authors.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give

appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

A Optimization of the pricing algorithm

In this section, we describe an algorithm with improved computational complexity compared to the dynamic programming approach for computing all possible schedules with negative reduced cost in the case of n_g equal-sized jobs. Recall that this problem boils down to $1|p_j = p|\sum w'_j U_j$, where $w'_j = \min\{v_j - M, q_j - M\}$. To ease notation, we assume without loss of generality that $p = 1$ and that we start at time 0 on the machine instead of starting at time t. To compensate for this, we put the due dates equal to

$$d_j \leftarrow \lfloor \frac{d_j - t}{p} \rfloor;$$

rounding down the due dates does not change the problem, since the jobs must finish at fixed times t, which are limited to values within the set $\{1, 2, \ldots, n_g\}$. For the remainder of this section, whenever we refer to t, it will represent any possible value from this predetermined set. We renumber the jobs in order of non-increasing w'_j value; initially the set of on-time jobs is empty. We add the jobs one by one in order of index to the current set of on time jobs. If the job can be on time together with the already selected on-time jobs, then we put it on time. Otherwise, we mark the job as tardy and execute it at the end of the schedule if $q_j - M > 0$, and discard it if $q_j \leq M$. To test whether the next job J_j can be set as on time, we need an appropriate data structure. Any job with a due date smaller than or equal to zero will be marked as tardy immediately and not be considered as a candidate for the on time set.

We first focus on an efficient way to decide whether or not a job J_j can be scheduled on time together with the already selected on-time jobs. For a given set of on-time jobs, we keep track of the number of jobs n_t in the on time set that have a due date that is smaller than or equal to t, for all $t = 1, \ldots, n_g$. It is easily verified that the jobs in the on time set can only be scheduled on time together if $n_t \leq t$, for all t. If we add job J_j to the on-time set, then this implies that n_t will increase by 1 for all t such as $t \geq d_j$ and remain equal for all t with $t < d_j$. If all new n_t values remain smaller than or equal to t, then job J_j is added to the on-time set. Updating this in a straightforward manner takes $O(n_g)$ time, and leads to an $O({n_g}^2)$ algorithm. To speed up the process of updating the n_t values, we compute the values $k_t = t - n_t$. The feasibility of the set of on-time jobs is guaranteed as long as $k_t \geqslant 0, \forall t$.

Determine for each $t = 1, 2, \ldots, n_g$ the value m_t as the minimum $k_{t^\star}$ value for all $t^\star \geq t$, and $t^\star \in \{1, 2, \ldots, n_g\}$ that is, $m_t = \min_{t^\star \geq t} k_{t^\star}$. If we plot the m_t values as a function of t, then we would see a number of horizontal line segments that move up. We are only interested in the jump points (m_t, t). Therefore, we store the t values corresponding to a jump point and add a pointer to the predecessor and successor t value for each jump point.

When we consider a new job J_j with due date d_j, we first determine the largest value t_1 such that $t_1 \leqslant d_j < t_2$, where t_1 and t_2 correspond to jump points. Let t_0 be the value of the jump point that precedes t_1. Adding job J_j to the set of on time jobs implies that m_t is decreased by 1, $\forall t \geqslant t_1$, which is feasible if they all remain nonnegative. To avoid useless updates, we rather store the difference between two consecutive jump points, i.e. $(m_{t_{k+1}} - m_{t_k})$.

Consequently, adding job J_j only requires to decrease by 1 the value $(m_{t_1} - m_{t_0})$. If this difference becomes 0, then this means that t_1 does not correspond to a jump point anymore. This is easily taken care of by adjusting the pointers: the successor of t_0 becomes t_2 and the predecessor of t_2 becomes t_0. We further mark job J_j as on-time and add it to the on time set. If t_1 is the origin point, i.e., t_0 does not exist then it means that the job cannot be on time because $m_{t_1} = 0$. At the end of the algorithm, we then process the jobs in the on-time set by increasing order of their due date.

The remaining point is how to efficiently compute the value t_1 for each job J_j. This can be done using a disjoint set union find. This data structure supports taking the union of two segments and finding the segment to which a certain value belongs. Its running time is $\mathcal{O}(a(n))$, where $a(n)$ is the inverse of the Ackermann function (Tarjan and Van Leeuwen, 1984). This is less than 5 for reasonable n values.

We now focus on how to efficiently update the w'_j when the value M changes. We initially create two lists of the jobset: one has the jobs sorted by non-increasing order of the v_j's while, in the other, the jobs are sorted by non-increasing order of the $(v_j - q_j)$'s. As $w'_j = \min(v_j - M, v_j - q_j)$ we compute them by a simple pass through the two lists. Notice that, due to the sorting of the two lists, the w'_j's are computed in non-increasing order of their computed value. Consequently, the running time of computing and sorting the w'_j decreases from $\mathcal{O}(n_g \log(n_g))$ to $\mathcal{O}(n_g)$.

So, the time complexity of this algorithm to compute a machine schedule, for a given value M, is $\mathcal{O}(n_g)$.

B Optimization of the Column Generation within the Branch and bound

Since Column Generation is used to provide a lower bound, and solving the master problem does not necessarily lead to obtaining a feasible solution of the bilevel problem, we can stop the column generation process earlier. Thereto, we compute at each iteration k of the column generation a lower bound $LB(k)$ on the optimal value z^*_{MP} of the master problem. Let $r_{\gamma+}$ and $r_{\gamma-}$ denote the outcome values of the pricing problem for a machine with speed V_γ that contains a maximum or minimum number of jobs. Then it is well known that $LB(k) = z_{RMP}(k) + m_{0+}r_{0+} + m_{0-}r_{0-} + m_{1+}r_{1+} + m_{1-}r_{1-} \leq z^*_{MP}$, where $z_{RMP}(k)$ is the optimal value of the restricted master problem (Desrosiers et al., 2024). Thus, we have a global lower bound $LB = \max_k LB(k)$. We stop the column generation at iteration k once the condition $z_{RMP}(k) - LB < 1$ is satisfied. Given that the objective function— the weighted number of tardy jobs— is integer-valued, the valid lower bound for the node is then defined as the ceiling of the global lower bound, $\lceil LB \rceil$.

Then, when exploring a node in the branch-and-bound algorithm, the scheduling decisions for the jobs have already been made. Consequently, at a given node $\mathcal{N}_{child}$, the dynamic programming procedure begins with the last undecided set of jobs, $G_{g'}$, and the current total completion time, t', for machine M_i, reflecting the previously made scheduling decisions. We then use these parameters, (t', g'), to compute the minimal reduced cost for machine M_i at node $\mathcal{N}_{child}$. Furthermore, if $\mathcal{B}_1$ is processed on both high and low speed machines, then for the given node N_{child}, we might have already scheduled jobs on $\mathcal{B}_1$. Let $\overline{m}_{0+}$ denote the number of high speed machines with a maximum of jobs in the node N_{child}. Thus we add the constraint $\sum_{s \in S} b_{s0+} x_s \geqslant \overline{m}_{0+}$, which helps the master problem and gives better experimental results in practice. Remark, at the root node, if the number of jobs to schedule

in the first block is greater than the number of low-speed machines, then we can have at least $\max(Q - m_1, 0) = \overline{m}_{0+}$ high-speed machines with a maximum number of jobs.

Moreover, we reuse columns from parent nodes; however, as we explore the search tree using branch-and-bound, a subset of these columns becomes infeasible. Rather than removing each infeasible column immediately, we count the number of times this occurs and define a threshold ϵ. If the proportion of all infeasible columns exceeds ϵ, then we remove them.

Finally, in a search tree, when facing the subproblem of filling the last block, instead of exploring the tree further, we solve it as an assignment problem. Indeed, by the time we need to fill the last block, we already know the completion times of the machines and the jobs that can be scheduled in this last block. Therefore, we can formulate and solve the assignment problem, where our goal is to assign the remaining jobs to the last block in order to minimize the weighted number of tardy jobs.

References

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

Bard, J.F. (1998). Practical Bilevel Optimization. volume 30 of *Nonconvex Optimization and Its Applications*. Springer US. https://doi.org/10.1007/978-1-4757-2836-1.

Bianco, L., Caramia, M., Giordani, S., & Mari, R. (2015). Grid scheduling by bilevel programming: A heuristic approach. *European Journal of Industrial Engineering, 9*, 101–125. https://doi.org/10.1504/EJIE.2015.067450

Brown, G.G., Carlyle, W.M., Royset, J.O., & Wood, R.K. (2005). On the Complexity of Delaying an Adversary's Project, in: The Next Wave in Computing, Optimization, and Decision Technologies, Springer US. pp. 3–17. https://doi.org/10.1007/0-387-23529-9_1.

Buchheim, C. (2023). Bilevel linear optimization belongs to NP and admits polynomial-size KKT-based reformulations. *Operations Research Letters, 51*, 618–622. https://doi.org/10.1016/j.orl.2023.10.006

Conway, R. W., Maxwell, W. L., & Miller, L. W. (1967). Theory of Scheduling. Addison-Wesley Publishing Company.

Dempe, S. (2002). Foundations of Bilevel Programming. volume 61 of *Nonconvex Optimization and Its Applications*. Kluwer Academic Publishers. https://doi.org/10.1007/b101970.

Dempe, S. (2003). Annotated Bibliography on Bilevel Programming and Mathematical Programs with Equilibrium Constraints. *Optimization, 52*, 333–359. https://doi.org/10.1080/0233193031000149894

Dempe, S. (2020). Bilevel Optimization: Theory, Algorithms, Applications and a Bibliography, in: Bilevel Optimization: Advances and Next Challenges. Springer International Publishing, pp. 581–672. https://doi.org/10.1007/978-3-030-52119-6_20.

Desrosiers, J., Lübbecke, M., Desaulniers, G., & Gauthier, J. B. (2024). Branch-and-Price. *Springer*. https://doi.org/10.1007/978-3-031-96917-1

Dessouky, M. I., Lageweg, B. J., Lenstra, J. K., & Van de Velde, S. L. (1990). Scheduling identical jobs on uniform parallel machines. *Statistica Neerlandica, 44*, 115–123. https://doi.org/10.1111/j.1467-9574.1990.tb01276.x

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A guide to the theory of NP-Completeness*. Freeman and Co: W.H.

Garey, M. R., Tarjan, R. E., & Wilfgong, G. T. (1988). One-processor scheduling with symmetric earliness and tardiness penalties. *Mathematics of Operations Research, 13*, 330–348.

Graham, R.L., Lawler, E.L., Lenstra, J.K., & Kan, A.H.G.R. (1979). Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey, in: Annals of Discrete Mathematics. Elsevier. volume 5 of *Discrete Optimization II*, pp. 287–326. https://doi.org/10.1016/S0167-5060(08)70356-X.

Hardy, G. H., Littlewood, J. E., & Pólya, G. (1934). *Inequalities (Cambridge Mathematical Library)*. Cambridge University Press.

Jeroslow, R. G. (1985). The polynomial hierarchy and a simple model for competitive analysis. *Mathematical Programming, 32*, 146–164. https://doi.org/10.1007/BF01586088

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

Kasemset, C., & Kachitvichyanukul, V. (2010). Bi-level multi-objective mathematical model for job-shop scheduling: The application of Theory of Constraints. *International Journal of Production Research, 48*, 6137–6154. https://doi.org/10.1080/00207540903176705

Kasemset, C., & Kachitvichyanukul, V. (2012). A PSO-based procedure for a bi-level multi-objective TOC-based job-shop scheduling problem. *International Journal of Operational Research, 14*, 50. https://doi.org/10.1504/IJOR.2012.046343

Kis, T., & Kovács, A. (2012). On bilevel machine scheduling problems. *OR Spectrum, 34*, 43–68. https://doi.org/10.1007/s00291-010-0219-y

Konur, D., & Golias, M. M. (2013). Analysis of different approaches to cross-dock truck scheduling with truck arrival time uncertainty. *Computers & Industrial Engineering, 65*, 663–672. https://doi.org/10.1016/j.cie.2013.05.009

Kovács, A., & Kis, T. (2011). Constraint programming approach to a bilevel scheduling problem. *Constraints, 16*, 317–340. https://doi.org/10.1007/s10601-010-9102-3

Lodi, A., Ralphs, T. K., & Woeginger, G. J. (2014). Bilevel programming and the separation problem. *Mathematical Programming, 146*, 437–458. https://doi.org/10.1007/s10107-013-0700-x

Lukač, Z., Šorić, K., & Rosenzweig, V. V. (2008). Production planning problem with sequence dependent setups as a bilevel programming problem. *European Journal of Operational Research, 187*, 1504–1512. https://doi.org/10.1016/j.ejor.2006.09.029

Schau, Q. (2025). Code of the exact algorithms for the bilevel scheduling problem. https://archive.softwareheritage.org/swh:1:dir:0568db95ce103ecafca6a2d970a4ef0e42b608cb;origin=https://github.com/QuentinSchau/Exact_Algorithms_Bilevel_Scheduling_Industry_Futur;visit=swh:1:snp:bb6e651f8e73095820aef9588a9466a3aba36d74;anchor=swh:1:rev:49eea1d7848ab81ea21b85a4f9b4a776c35dab2d. software Heritage identifier.

Shang, L., T'Kindt, V., & Della Croce, F. (2021). Branch & Memorize exact algorithms for sequencing problems: Efficient embedding of memorization into search trees. *Computers & Operations Research,128*, Article 105171. https://doi.org/10.1016/j.cor.2020.105171

Tarjan, R. E., & van Leeuwen, J. (1984). Worst-case Analysis of Set Union Algorithms. *J. ACM,31*, 245–281. https://doi.org/10.1145/62.2160

T'kindt, V., Della Croce, F., Agnetis, A.,. (2024). Single machine adversarial bilevel scheduling problems. *European Journal of Operational Research,315*, 63–72. https://doi.org/10.1016/j.ejor.2023.11.018

Van den Akker, J. M., Hoogeveen, J. A., & Van de Velde, S. L. (1999). Parallel Machine Scheduling by Column Generation. *Operations Research, 47*, 862–872. https://doi.org/10.1287/opre.47.6.862

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

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.