Noname manuscript No.
(will be inserted by the editor)

On bilevel machine scheduling problems

Tamás Kis · András Kovács

Abstract Bilevel scheduling problems constitute a hardly studied area of scheduling theory. In this paper we summarise the basic concepts of bilevel optimisation, and discuss two problem classes for which we establish various complexity and algorithmic results.

The first one is the bilevel total weighted completion time problem in which the leader assigns the jobs to parallel machines and the follower sequences the jobs assigned to each machine. Both the leader and the follower aims to minimise the total weighted completion time objective, but with different job weights. When the leader's weights are arbitrary, the problem is NP-hard. However, when all the jobs are of unit weight for the leader, we provide a heuristic algorithm based on iterative LP-rounding along with computational results, and provide a sufficient condition when the LP-solution is integral. In addition, if the follower weights induce a monotone (increasing or decreasing) processing time order in any optimal solution, the problem becomes polynomially solvable. As a by-product, we characterise a new polynomially solvable special case of the MAX m-CUT problem, and provide a new linear programming formulation for the $P||\sum_j C_j$ problem.

Finally, we present some results on the bilevel order acceptance problem, where the leader decides on the acceptance of orders and the follower sequences the jobs. Each job has a deadline and if a job is accepted, it cannot be late. The leader's objective is to maximise the total weight of accepted jobs, whereas the follower aims at minimising the total weighted job completion times. For this problem we generalise some known single-level machine scheduling algorithms.
Keywords: scheduling, bilevel optimisation, MAX m-CUT, linear programming, approximation algorithms.

Tamás Kis
Computer and Automation Research Institute, Kende str 13-17, H-1111 Budapest, Hungary.
Tel: +36-1-2796156, Fax: +36-1-4667503, E-mail: tamas.kis@sztaki.hu

András Kovács
Computer and Automation Research Institute, Kende str 13-17, H-1111 Budapest, Hungary.

1 Introduction

Scheduling theory is mainly concerned with single-level problems where a decision maker has to compute a schedule of some activities using some resources while minimising an objective function. There are various extensions of this basic decision model involving two or more decision makers whose interactions determine the schedule. One direction is game theory, see e.g., [20], [1], another is integrated planning and scheduling, see e.g., [14], [10].

In this paper we study scheduling problems formulated as bilevel optimisation problems, which has its origins in market economy theory, and in particular in Stackelberg games [7]. Bilevel optimisation is concerned with two-level optimisation problems, in which there is a top level decision maker or *leader*, and one (or more) bottom level decision maker(s) or *follower(s)*. The leader decides first, and fixes those decision variables that are under her control. While making her decisions, she takes into account the possible responses of the follower in order to optimise her own objective function. The leader's decisions affect the constraints and/or the objective function of the follower. The follower decides second, and makes its decisions in view of those of the leader. However, its decisions affect the objective function of the leader or even the feasibility of the leader's solution. The follower also wants to optimise its own objective function. When solving bilevel optimisation problems, we want to support the leader in making optimal decisions. In the *optimistic case*, the leader assumes that the follower chooses an optimal solution which is the most favourable for her, while in the *pessimistic case* the leader assumes that the follower chooses an optimal solution with the worst outcome for her. For overview and references, see [6], [7]. For recent developments of a combinatorial flavor, see e.g., [5], [18], [8] [3].

There are only sporadic results on bilevel machine scheduling problems, see e.g., [12], [17]. As a practical motivation, consider a large organisation with loosely coupled decision makers with conflicting objectives. For instance, planners at a make-to-order company want to satisfy customer orders, while managers at the shop floor wish to minimise production costs, or enforce technological constraints that are neglected by the planners. Another example is construction projects, where subcontractors may not be directly controlled by project planners. A common feature of these examples is that there are independent decision makers, but their decisions are not independent and they have to make decisions sequentially in a pre-defined order. Those who decide first have to take into account the preferences and decision mechanisms of the followers to make optimal decisions. Of course, this is feasible only if the leaders have sufficient knowledge about the followers.

We will frequently use the $\alpha|\beta|\gamma$ notation for scheduling problems, where α is the machining environment, β is the set of restrictions, and γ is the objective function [9]. For instance, $\alpha = 1$ denotes a single machine, while $\alpha = P$ is a parallel machine environment. As for γ, $\sum C_j$ is the total completion time objective. We will also refer to the *weighted shortest processing time order* of the jobs with respect to some job weights w_j (WSPT order for short), in which job j precedes job k, if $w_j/p_j > w_k/p_k$, and there may be additional tie-breaking rules. Finally, we call a schedule *non-delayed*, if no job may be started earlier without violating some of the constraints of the scheduling problem.

1.1 Main results

We will derive various complexity and algorithmic results for two types of bilevel scheduling problems.

The bilevel weighted completion time problem. In this problem, there are n jobs and m identical parallel machines. Each job has a processing time p_j and two non-negative weights, w_j^1 and w_j^2. The leader assigns jobs to machines, and the follower orders the jobs assigned to each machine. Let C_j denote the completion time of job j in a solution. The follower's objective is to minimise $\sum_{i=1}^{m} \sum_{j \in J_i} w_j^2 C_j$, where J_i is the set of those jobs assigned to machine i. In the optimistic case, the leader's objective is to minimise the total completion time $\sum_{j=1}^{n} w_j^1 C_j$, where the minimum is taken over all job assignment $J_1, \ldots, J_m$. In contrast, in the pessimistic case the leader wants to find an assignment of jobs to machines such that the maximum total weighted completion time is minimal by minimising (over all job assignments) $\max \sum_{j=1}^{n} w_j^1 C_j$, where the maximum is taken over all optimal solutions of the follower with respect to a job assignment. The leader aims to find the sets J_i, $i = 1, \ldots, m$, such that her decision is optimal in the optimistic or in the pessimistic sense. Notice that such a solution cannot be modelled by imposing precedence constraints among the jobs, because the assignment of jobs to machines is not known in advance.

For this problem we prove that the decision version is NP-complete in the strong-sense. We will prove that for any instance of the problem, there exists a global ordering of jobs such that in an optimal solution each machine processes the jobs assigned to it in an order compatible with the global job order. On the other hand, we will also prove that the problem is equivalent to finding a MAXIMUM WEIGHT m-CUT in a complete graph (MAX m-CUT) with appropriate edge weights. Assuming the leader has uniform weights, $w^1 \equiv 1$, we will list two polynomially solvable special cases with special follower weights, and provide a heuristic algorithm for general follower weights which performs very well in practice. The heuristic is based on LP rounding techniques, and we will describe two seemingly different linear programs whose solutions will be iteratively rounded. We will show that the two LPs are equivalent, and indeed when the job weights of the follower induce an increasing processing time order, both LPs admit integral optimal solutions. As a by-product, we obtain a new LP based formulation for the $P||\sum_j C_j$ problem. In addition, we characterise a new polynomially solvable special-case of the MAX m-CUT problem (Section 3).

The bilevel order-acceptance problem. There are n jobs and a single machine. Each job has a processing time p_j, a deadline d_j, and two non-negative weights, w_j^1 and w_j^2. The weight w_j^1 is the leader's penalty (loss of profit) of rejecting job j, and w_j^2 is the weight for the follower. The leader has to select a subset of jobs that have to be completed by their respective deadlines. However, the follower aims to minimise the weighted completion time of the accepted jobs, but it is not obliged to take into consideration the deadlines. More formally, the leader's objective is $\min \sum_j w_j^1 R_j$, where $R_j = 1$ if and only if job j is rejected. In turn, the follower's objective is $\min \sum_{j \in J'} w_j^2 C_j$, where $J' = \{j \mid R_j = 0\}$ is the set of accepted jobs, and C_j is the completion time of job j. The leader's objective function is the same in the optimistic and in the pessimistic cases. In the optimistic case, the leader has to accept

a subset of jobs such that there exists *at least one* optimal sequence for the follower which respects all the job-deadlines of the accepted jobs. On the other hand, in the pessimistic case, the leader has to choose a subset of jobs such that *all* the optimal solutions of the follower observes all the deadlines of the accepted jobs.

This is a bilevel optimisation problem, because the leader can only accept or reject the jobs, while the follower determines an optimal sequence without regarding the jobs' deadlines. If the follower's solution violates some of the job-deadlines, then this solution is infeasible.

We prove that this problem is NP-hard in the ordinary sense and modify the method of Lawler and Moore [15], [19] for the (single level) $1||\sum_j w_j U_j$ problem for solving the bilevel order acceptance problem in pseudo-polynomial time. We also provide a polynomial-time algorithm for the special case with $w^1 \equiv 1$ (Section 4).

Relation with multi-criteria optimisation. We emphasise that bilevel optimisation is different from multi-criteria optimisation. This relation is thoroughly studied in the context of bilevel linear programming in [6]. Since we deal with problems of special structure, we will demonstrate that the optimal solutions of the two bilevel scheduling problems are not Pareto optimal (Section 2).

We conclude the paper in Section 5.

2 Bilevel optimisation vs. multi-criteria optimisation

Multi-criteria scheduling problems are thoroughly discussed in the literature, see e.g., the review of Hoogeveen [11]. In those problems, there are two or more criteria to evaluate solutions, and we associate with each solution a vector of objective function values. A central notion of multi-criteria optimisation is that of *Pareto optimality*. Suppose there are k criteria, and S^1, S^2 are two solutions with values $(f_1^1, \ldots, f_k^1)$, and $(f_1^2, \ldots, f_k^2)$, respectively. We say that S^1 *Pareto-dominates* S^2 if $f_i^1 \leq f_i^2$ for each $i = 1, \ldots, k$. The Pareto-dominance is *strict* if $f_i^1 < f_i^2$ for at least one i. Then, a solution is *Pareto optimal* if it is not strictly Pareto-dominated by some other solution.

2.1 Bilevel weighted completion time and Pareto optimality

Now we demonstrate that the optimal solution of the bilevel weighted completion time problem is not Pareto optimal in general. In fact, we describe an instance where every optimal solution of the bilevel problem is strictly dominated by a feasible solution. This shows that the two notions of optimality are different.

Suppose there are 5 jobs, with processing times $p_j = j$ for $j = 1, \ldots, 5$. The weights of the jobs are as follows: $w_j^1 = 1$, and $w_j^2 = p_j(c+p_j)$ for every job j, where c is a constant to be chosen later. Notice that w_j^2 induces a decreasing processing time order, since $w_j^2/p_j \geq w_k^2/p_k$ is equivalent to $c + p_j \geq c + p_k$. Therefore, in any optimal solution of the bilevel scheduling problem, the machines process the assigned jobs in decreasing processing time order, otherwise the solution is not optimal for the follower.

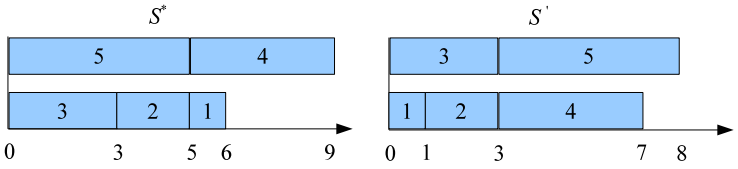

Fig. 1 Two schedules for the bilevel weighted completion time problem.

It can be shown that this problem admits the unique optimal solution S^* (up to permutation of the machines) in which $J_1^* = \{3, 2, 1\}$ and $J_2^* = \{5, 4\}$, and the machines process the assigned jobs in the given order. In this solution the job completion times for $j \in J_1^*$ are $C_3^* = 3$, $C_2^* = 5$, $C_1^* = 6$, and those for $j \in J_2^*$ are $C_5^* = 5$, $C_4^* = 9$, see Figure 1. Therefore, the optimal objective function value of the leader is $\sum_{j=1}^{5} C_j^* = 28$. On the other hand, the followers' objective function value is $\sum_{j=1}^{5} w_j^2 C_j^* = 86c + 322$. Therefore, the vector of objective function values is $(f_1^*, f_2^*) = (28, 86c + 322)$.

We construct another solution S': let $J_1' = \{1, 2, 4\}$ and $J_2' = \{3, 5\}$, and suppose the machines process the assigned jobs in increasing processing time order. Then $C_1' = 1$, $C_2' = 3$, $C_4' = 7$, and $C_3' = 3$, $C_5' = 8$. The leader's objective function value is $\sum_{j=1}^{5} C_j' = 22$. On the other hand, the follower's objective function value on this solution is $\sum_{j=1}^{5} w_j^2 C_j' = 84c + 352$. Therefore, $(f_1', f_2') = (22, 84c + 352)$. Clearly, this solution is not optimal for the follower, since by reversing the processing order on the two machines the objective function would increase.

Now we compare the solutions S^* and S'. S' strictly Pareto-dominates S^*, i.e.,

$$(f_1^*, f_2^*) = (28, 86c + 322) > (22, 84c + 352) = (f_1', f_2')$$

if $c > 200$ (this is not a strict bound). Therefore, S^* is strictly Pareto-dominated.

2.2 Bilevel order acceptance and Pareto optimality

Consider the instance of the bilevel order acceptance problem in Table 1. There are only four candidate job sets that may be selected by the leader: $\{1\}$, $\{2\}$, $\{3\}$, and $\{1, 2\}$, and it is easy to verify that no other job set may be completed on time. The job

Table 1 Problem data for the bilevel order acceptance problem.

Job j	p_j	d_j	w_j^1	w_j^2
1	1	1	2	1
2	1	2	2	3
3	2	2	3	10

set $\{1, 2\}$ admits two sequences, $S^1 = (1, 2)$ and $S^2 = (2, 1)$. The leader's objective

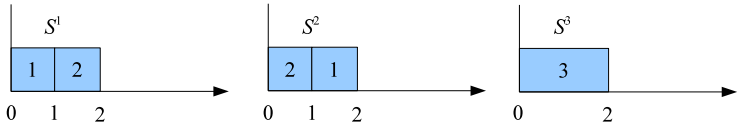

Fig. 2 Three schedules for the bilevel order acceptance problem.

function value is $f_1^1 = f_1^2 = w_3^1 = 3$ for both schedules. However, schedule S^1 is not optimal for the follower, because in the WSPT order job 2 precedes job 1, since $w_2^2/p_2 = 3 > 1 = w_1^2/p_1$. Therefore, S^1 cannot be an optimal solution of the bilevel scheduling problem. On the other hand, in S^2 job 1 completes after its due-date, therefore, it is not feasible for the leader. In S^1 the job completion times are $C_1^1 = 1$ and $C_2^1 = 2$, whence the objective function value of the follower is $\sum_{j\in\{1,2\}} w_j^2 C_j^1 = 7$. The value of S^1 is $(f_1^1, f_2^1) = (3, 7)$, see Fig. 2. Now consider the job set $\{3\}$, the corresponding schedule is $S^3 = (3)$. Clearly, $(f_1^3, f_2^3) = (4, 20)$. It is easy to verify that S^3 is the unique optimal solution of the bilevel scheduling problem.

Comparing the values of S^1 and S^3, we see that S^1 strictly Pareto-dominates S^3.

3 The bilevel weighted completion time problem

3.1 Preliminaries

The bilevel weighted completion time problem is a generalisation of the parallel machine weighted completion time problem, which is as follows. There are m identical parallel machines, and n jobs each with a processing time p_j and weight w_j. Each job has to be assigned to a machine, and the jobs assigned to the same machine have to be sequenced such that the objective function $\sum_j w_j C_j$ is minimised, C_j being the completion time of job j. In the $\alpha|\beta|\gamma$ notation this problem is denoted as $P||\sum_j w_j C_j$. This problem is NP-hard [21], strong NP-hardness is claimed in [16] (cf. Brucker [2]). The special case $P||\sum_j C_j$ is solvable in polynomial time by network flow techniques. An important property of the optimal solutions is that on each machine the jobs are processed in non-decreasing processing time order, see e.g., [2].

3.2 Global ordering of jobs

Given the leader's decision about the job assignments J_i, $i = 1, \ldots, m$, the follower faces m independent $1||\sum_{j\in J_i} w_j^2 C_j$ problems, one for each machine. Hence, the follower is going to minimise the objective function $\sum_{i=1}^m \sum_{j\in J_i} w_j^2 C_j$ by sequencing the jobs according to the WSPT rule using the weights w^2, with ties broken according to w^1 (the direction of tie-braking differs in the optimistic and in the pessimistic case). Since the processing times and weights of the jobs do not depend on

the machine assignment, the sequence on each machine is a sub-sequence of one global partial ordering. The global partial ordering in the optimistic case is:

$$\text{job } j \text{ precedes job } k \text{ if } w_j^2/p_j > w_k^2/p_k \text{ or } (w_j^2/p_j = w_k^2/p_k \text{ and } w_j^1/p_j > w_k^1/p_k) \tag{1}$$

while in the pessimistic case it is:

$$\text{job } j \text{ precedes job } k \text{ if } w_j^2/p_j > w_k^2/p_k \text{ or } (w_j^2/p_j = w_k^2/p_k \text{ and } w_j^1/p_j < w_k^1/p_k) \tag{2}$$

The above ordering is partial because it does not decide which of jobs j and k should be scheduled first if $w_j^2/p_j = w_k^2/p_k$ and $w_j^1/p_j = w_k^1/p_k$.

Proposition 1 *In the optimistic case there always exists an optimal non-delayed schedule for any non-negative job-weights. In the pessimistic case, if there exists a job j with $w_j^2 = 0$ and $w_j^1 > 0$, then the pessimistic bilevel scheduling problem admits no finite optimum. Otherwise, it always has a finite optimum.*

Proof In the optimistic case, the follower always chooses the most favourable schedule for the leader among its optimal solutions with respect to a specific assignment of jobs to machines. Therefore, it always chooses a non-delay schedule, since $w_j^2 \geq 0$ for all jobs, and therefore, there is no gain in delaying a job. Since there exist only a finite number of non-delayed schedules, the problem always admits a finite optimum.

In contrast, in the pessimistic case, the leader assumes that the follower plays against her. Hence, if $w_j^2 = 0$ and $w_j^1 > 0$ for some job j, then the optimum value of the leader is infinite, since the follower has an optimal schedule with an arbitrarily large C_j value for any assignment of jobs to machines. If $w_j^2 = w_j^1 = 0$, then $w_j^1 C_j = 0$, and therefore such jobs can be scheduled arbitrarily by the follower after those jobs with $w_k^2 > 0$ without affecting the objective function of the leader. Finally, if $w_j^2 > 0$ for all the jobs, then any optimal solution of the follower is a non-delayed schedule, and therefore, the problem has a finite optimum. □

The above technical difficulties can be avoided by assuming that the follower always chooses a non-delayed optimal solution, which is quite reasonable in practice.

Lemma 1 *The optimistic weighted completion time problem always admits an optimum solution of finite value such that on each machine the jobs are ordered according to (1).*

Proof The existence of an optimal schedule is guaranteed by Proposition 1. As for the structure of optimal schedules, the key observation is that once the assignment of jobs to machines is fixed, the follower always chooses an optimal ordering of the set of jobs assigned to each machine. However, the single machine problem of machine i with job-set J_i is $1||\sum_{j\in J_i} w_j^2 C_j$, which can be solved by the WSPT rule. If the order of two jobs is arbitrary in an optimal solution, i.e., $w_j^2/p_j = w_k^2/p_k$, then the leader's preference can be taken into account which is expressed in the ordering (1). □

Lemma 2 *If there exists no job j with $w_j^2 = 0$ and $w_j^1 > 0$, or the follower has to choose a non-delayed optimal schedule, then the pessimistic weighted completion time problem admits an optimal solution of finite value such that on each machine the jobs are ordered according to (2).*

Proof Similar to the optimistic case. □

The following lemma shows that in general it is possible to change the followers' weights so that w^2 alone defines one unambiguous complete global ordering of jobs, and the sequences on individual machines will be sub-sequences of that global ordering. This is useful, because it ensures that – whenever this conversion can be performed – our propositions hold both for the optimistic and the pessimistic cases.

Lemma 3 *Any instance Π of the bilevel weighted completion time problem can be converted into an instance $\bar{\Pi}$ such that the followers' WSPT order is unique, and all the optimal solutions of $\bar{\Pi}$ are optimal for Π as well.*

Proof We define new follower's job weights $\bar{w}^2$ as follows: In the optimistic case re-index the jobs (i.e., $j < k$ iff job j precedes k) with respect to partial order (1), break ties arbitrarily, while in the pessimistic case with respect to (2). We define the instance $\bar{\Pi}$ with n jobs having processing times p_j, leader's weights w_j^1 and followers' weights $\bar{w}_j^2 = (n-j+1)p_j$. Since $\bar{w}_j^2/p_j = n-j+1$, it follows that job j precedes job k iff j precedes k in the selected global ordering of jobs. Moreover, the WSPT order with respect to $\bar{w}_j^2$ induces a total order of jobs. □

3.3 Complexity

Below we prove that the decision version of the bilevel weighted completion time problem is NP-complete in the strong sense. The decision version of the bilevel scheduling problem asks whether there is a feasible solution with a leader's objective function value not worse than a given bound K. Notice that such a solution has to be optimal for the follower.

Lemma 4 *The bilevel weighted completion time problem is NP-complete in the strong sense.*

Proof *Membership to NP*: The witness consists of a partitioning $J_1, \ldots, J_m$ of the jobs, and a completion time $\bar{C}_j$ for each job j. One can easily verify whether $\sum_{j=1}^{n} w_j^1 \bar{C}_j \leq K$. Moreover, for each machine i, an instance of the problem $1||\sum_j w_j C_j$ is specified by job-set J_i using weights w_j^2. Each of these problems can be solved in polynomial time by the WSPT rule and let W_i^* denote the optimum value for machine i. The job-completion times $\bar{C}_j$, $j \in J_i$, correspond to an optimal schedule, if the jobs in J_i do not overlap, and $\sum_{j \in J_i} w_j^2 \bar{C}_j = W_i^*$. All these computations can be done in polynomial time in the length of the input and that of the witness given above.

NP-hardness: The bilevel problem contains the strongly NP-hard $P||\sum w_j C_j$ problem, which can be seen by assigning $w_j^1 = w_j^2 := w_j$. Then, any solution is optimal (and feasible) to the bilevel problem if and only if it is an optimal solution to the parallel machine problem as well. □

Problem 1 *An open problem is whether the problem remains NP-hard if* $w_j^1 = 1$ *or* $p_j = p$ *for all jobs* j.

Next we show the connection to the MAX m-CUT problem, which is as follows. Given a complete graph K_n with edge weights $c(i,j)$, determine a partitioning of the nodes into m nonempty subsets such that the total weight of edges connecting the nodes in the different subsets is maximal.

Now, the bilevel scheduling problem is equivalent to a MAX m-CUT problem (unless $m \geq n$, in which case the scheduling problem is trivial). This can be shown by assigning weights to the edges as follows: order the vertices with respect to (1) in the optimistic case, and (2) in the pessimistic case. The weight of edge (j,k) is

$$c(j,k) := p_j w_k^1 \text{ if job } j \text{ precedes job } k \text{ in the ordering.} \tag{3}$$

Lemma 5 *The optimal solution of the MAX* m*-CUT problem with edge weights (3) yields an optimal solution of the bilevel scheduling problem.*

Proof Consider first the optimistic case. By Lemma 1, there is an optimal solution respecting the ordering (1) on each machine. In this ordering, the total weighted completion time on machine i with set of jobs J_i is $\sum_{j\in J_i} w_j^1 C_j = \sum_{j\in J_i} w_j^1 (p_j + \sum_{k\in J_i:k\prec j} p_k)$, where $k \prec j$ iff job k precedes job j. The term $\sum_{j\in J_i} w_j^1 p_j$ being constant, the problem can be reformulated as follows:

$$\min\{\sum_{i=1}^{m} \sum_{j\in J_i} \sum_{k\in J_i:k\prec j} p_k w_j^1 \mid J_1, \ldots J_m \text{ is a partitioning of } J\},$$

where $J = \{1,\ldots,n\}$. Since the total weight of all arcs is $\sum_{j\in J} \sum_{k\prec j} p_k w_j^1$, the total weight of those arcs connecting nodes in different subsets of a partitioning $J_1,\ldots,J_m$ of J is

$$\sum_{j\in J} \sum_{k\prec j} p_k w_j^1 - \sum_{i=1}^{m} \sum_{j\in J_i} \sum_{k\in J_i:k\prec j} p_k w_j^1.$$

Therefore, minimising $\sum_{i=1}^{m} \sum_{j\in J_i} \sum_{k\in J_i:k\prec j} p_k w_j^1$ is equivalent to maximising the total weight of those arcs connecting nodes in different subsets of a partitioning.

The proof of the pessimistic case goes along the same lines using Lemma 2 and ordering (2). □

Consequently, by solving the MAX m-CUT problem in the complete n-graph with appropriate edge-weights, we can solve the bilevel scheduling problem to optimality. Since the MAX m-CUT problem is NP-hard in the strong sense in general even for $m = 2$ [13], we do not have a polynomial time algorithm at hand for solving our bilevel scheduling problem. Next we study some special cases.

3.4 Special cases

In this section we consider two polynomially solvable special cases of the bilevel scheduling problem. We say that weights w^1 and w^2 induce the same ordering of jobs if $w_j^1/p_j \leq w_k^1/p_k$ if and only if $w_j^2/p_j \leq w_k^2/p_k$ for each pair of jobs j and k.

Proposition 2 *Suppose w^1 and w^2 induce the same ordering of jobs. Then an optimal solution of the bilevel scheduling problem is obtained by solving the (single level) problem $P||\sum w_j C_j$ with $w_j = w_j^1$.*

The above parallel machine scheduling problem is NP-hard in general. An important special case is when all the weights are equal to 1.

Further special cases occur when $w^1 \equiv 1$, but w^2 induces an increasing or decreasing processing time order, i.e., $w_j^2/p_j > w_k^2/p_k$ if and only if (1) $p_j < p_k$, or (2) $p_j > p_k$. For instance, the weights $w_j^2 = p_j + 1$ induce an increasing processing time order, whereas the weights $w_j^2 = p_j - 1$ induce a decreasing processing time order.

3.4.1 $w^1 \equiv 1$ and w^2 induces a non-decreasing processing time order

If $w^1 \equiv 1$, then it would be optimal for the leader to process the jobs in non-decreasing processing time order on each machine. Notice that w^2 induces the same order on the machines. Therefore, by Proposition 2 we know that the problem is equivalent to a single level parallel machine scheduling problem. Moreover, since $w^1 \equiv 1$, this single level problem takes the from $P||\sum_j C_j$, which can be solved in polynomial time by linear programming techniques, cf. [2]. Furthermore, we obtain two new linear programming formulations for this problem in Section 3.6. Hence, the bilevel problem with the above job-weights is tractable as well.

3.4.2 $w^1 \equiv 1$, and w^2 induces a non-increasing processing time order

Suppose the jobs are indexed in non-increasing processing time order, i.e., $p_1 \geq p_2 \geq \cdots \geq p_n$. We claim that the bilevel scheduling problem admits an optimal solution such that each of the m machines processes consecutive jobs in the above ordering. Moreover, an optimal solution can be found in polynomial time. We say that job k is a *successor* of job j if k succeeds j in the non-increasing processing time order.

Lemma 6 *If $w^1 \equiv 1$, and w^2 induces a non-increasing processing time order, then the bilevel scheduling problem admits an optimal solution such that the set of jobs J_i assigned to machine i consists of the jobs $\pi_{i-1} + 1, \ldots, \pi_i$, where $\pi_0 = 0$, and $1 \leq \pi_1 \leq \pi_2 \leq \cdots \leq \pi_m = n$.*

Proof We will exploit the equivalence to the MAX m-CUT problem. Since $w_j^1 = 1$ for each job j, and w^2 induces a non-increasing processing time order, we can model the bilevel scheduling problem by a complete graph K_n with nodes identified with the jobs, and for each pair of distinct nodes (j, k), the weight of the edge incident with j and k is $c(j, k) := \max\{p_j, p_k\}$. Clearly, if job k is a successor of job j, then $c(j, k) = p_j$, otherwise $c(j, k) = p_k$.

Any optimal solution of the bilevel scheduling problem can be fully characterised by an assignment of jobs to machines, since the order of jobs assigned to a machine is determined by the decreasing processing time order. In terms of the MAX m-CUT problem, the optimal assignment is equivalent to an m-partition $S_1, \ldots, S_m$ of the nodes of K_n such that the total weight of those edges connecting nodes in distinct subsets in the partitioning is maximal. Let $C = \{(j,k) \mid \exists i \neq \ell$ such that $j \in S_i$ and $k \in S_\ell\}$. W.l.o.g. suppose $1 \in S_1$, and assume that there exist $k_1 \geq 1$ and $k_2 > k_1$ such that $\{1, \ldots, k_1 - 1, k_2\} \subset S_1$, but $k_1 \in S_i$ and $S_i \neq S_1$. We apply the following transformation to $S_1, \ldots, S_m$. Let $S^* = S_1 \cup S_i$, and then let S'_i consist of the last $\max\{|S_1| - k_1 + 1, |S_i|\}$ jobs of S^* in the decreasing processing time order, $S'_1 := S^* \setminus S'_i$, and $S'_\ell := S_\ell$ for $\ell \in \{1, \ldots, m\} \setminus \{1, i\}$. Let C' consist of those edges of K_n connecting nodes in distinct subsets of the partitioning $S'_1, \ldots, S'_m$ of the nodes of K_n. We claim that the total weight of edges in C' is not smaller than that of C. Since $S'_1, \ldots, S'_m$ is equivalent to an assignment of jobs to machines (since the machines are identical), this means that the objective function of the bilevel scheduling problem does not increase with the above transformation. Since this transformation ensures that S_1 contains all the jobs $1, \ldots, k_1$, by repeated application we can make sure that S_1 consists of consecutive jobs. Repeating this argument for each subset of the partitioning, we obtain the desired optimal solution.

To prove our claim, let $C(S_1, S_i) \subseteq C$ denote the set of edges connecting the nodes in S_1 and S_i. Similarly, let $C'(S'_1, S'_i) \subseteq C'$ be the set of edges with endpoints in S'_1 and S'_i. Since C and C' differ only in the set of arcs connecting S_1 and S_i, and S'_1 and S'_i, respectively, we have

$$\sum_{(j,k)\in C} c(j,k) - \sum_{(j,k)\in C(S_1,S_i)} c(j,k) = \sum_{(j,k)\in C'} c(j,k) - \sum_{(j,k)\in C'(S'_1,S'_i)} c(j,k).$$

Therefore, it suffices to show that

$$\sum_{(j,k)\in C(S_1,S_i)} c(j,k) \leq \sum_{(j,k)\in C'(S'_1,S'_i)} c(j,k).$$

For each job $j \in S_1 \cup S_i$, let n_j and n'_j denote the number of arcs (j,k) in $C(S_1, S_i)$ and $C'(S'_1, S'_i)$, respectively, such that k succeeds j. We clearly have

$$\sum_{j\in S_1\cup S_i} n_j \leq |S_1| \cdot |S_i| \leq |S'_1| \cdot |S'_i| = \sum_{j\in S'_1} n'_j.$$

The first inequality trivially holds. The second inequality is ensured by the transformation. The last equality follows from the fact that any job in S'_i succeeds all jobs in S'_1. Since $n'_j \leq |S'_i|$ for every $j \in S'_1$, we also have $n'_j = |S'_i|$. Moreover, for each $j \in S_1$, $n_j \leq |S_i|$, and for each $j \in S_i$, $n_j \leq |S_1| - k_1 + 1$ (since jobs $1, \ldots, k_1 - 1$ all belong to S_1 and all jobs succeed them). Since $|S'_i| = \max\{|S_i|, |S_1| - k_1 + 1\}$, we also have $n_j \leq |S'_i|$. Now, we have

$$\sum_{(j,k)\in C'(S'_1,S'_i)} c(j,k) = \sum_{j\in S'_1} n'_j p_j.$$

On the other hand,

$$\sum_{(j,k)\in C(S_1,S_i)} c(j,k) = \sum_{j\in S_1\cup S_i} n_j p_j \leq \sum_{j\in S'_1} n'_j p_j.$$

Here, the last inequality follows, since $n_j \leq |S'_i|$ for each $j \in S_1 \cup S_i$, $n'_j = |S'_i|$, for each $j \in S'_1$, $\sum_{j\in S_1\cup S_i} n_j \leq \sum_{j\in S'_1} n'_j$, $S_1 \cup S_i = S'_1 \cup S'_i$, and all jobs in S'_i succeed all jobs in S'_1. However, this implies our claim. □

Lemma 7 *The special case with $w^1 \equiv 1$ and w^2 inducing a decreasing processing time order can be solved in polynomial time.*

Proof We define a directed graph. Each node is a tuple $([j_1, j_2], \ell)$, where $[j_1, j_2]$ is an interval of jobs, and ℓ is a depth label with $1 \leq \ell \leq m$. Moreover, there is an initial node $([0, 0], 0)$. We direct an arc from $([j_1, j_2], \ell)$ to $([j_3, j_4], \ell + 1)$ iff $j_3 = j_2 + 1$. In particular, there is an arc from the initial node to each node of the form $([1, j_2], 1)$. The *cost* of any arc directed to node $([j_3, j_4], \ell)$ is the total completion time of the jobs in the interval $[j_3, j_4]$, i.e., the sum $\sum_{j\in[j_3,j_4]}(m_j + 1)p_j$, where m_j is the number of those jobs $k \in [j_3, j_4]$ that succeed j. A shortest path from node $([0, 0], 0)$ to one of the form $([j, n], m)$ gives an optimal solution to the scheduling problem. Namely, such a path has $m+1$ nodes with depth labels 0 through m, the intervals are disjoint and contain all the jobs, and for each $1 \leq \ell \leq m$, the node with depth label ℓ represents the assignment of jobs to the ℓ-th machine. Since this graph has $O(mn^2)$ nodes, and no parallel arcs or loops, the shortest path can be found in polynomial time. □

3.5 The polynomially solvable special case of the MAX m-CUT problem

For the sake of completeness, we reformulate the results of the previous section in terms of the MAX m-CUT problem.

Theorem 1 *Consider the complete graph K_n with a weight p_j associated with each node j. Let $c(j, k) = \max\{p_j, p_k\}$. Then the MAX m-CUT problem for K_n with edge-weights $c(j, k)$ is solvable in polynomial time.*

Proof Order the nodes of K_n in non-increasing p_j order, and apply Lemmas 6 and 7. □

3.6 A heuristic algorithm for $w^1 \equiv 1$, w^2 arbitrary

In this section we provide a heuristic algorithm for the special case with $w^1 \equiv 1$ and w^2 arbitrary, which performs very well in practice. Notice that the complexity status of this problem is open. Firstly, we provide two alternative integer program formulations, and show that their LP relaxations are equivalent, i.e., always produce the same lower bound. Then, we describe a heuristic algorithm based on iterative LP-rounding. We will summarise our computational results showing that the heuristic finds very good solutions, and also describe the structure of those instances on which our algorithm produces the worst results. Notice that even in the worst case the solutions are within 23% from the optimum (verified experimentally).

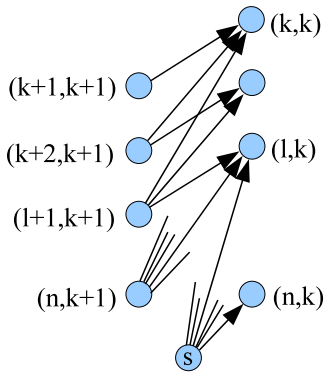

Fig. 3 A fragment of network N.

3.6.1 The two integer program formulations

We exploit that both in the optimistic and in the pessimistic cases there is a global ordering of jobs such that there always exists an optimal solution conforming that order, cf. Section 3.2. In the following, we assume that the jobs are indexed in *reverse* order with respect to (1) or (2), i.e., the last job has index 1, and $j > j'$ iff job j precedes job j' in the global ordering. Likewise, the last job on a machine is in the *first* position, the penultimate job is in the *second* position, etc.

The first integer program is based on the idea of assigning jobs to positions while respecting the global ordering. Notice that in any optimal solution, job j can be assigned to positions 1 through $\min\{j, n-m+1\}$ only. Namely, if j is assigned to a position larger than j, then at least j jobs follow j on the machine (recall that positions are indexed backward), hence, a job with larger index should be scheduled after j (in a smaller position on the machine) which is infeasible. Moreover, if j is assigned to a position larger than $n-m+1$, then there would be a machine without any jobs assigned to it, and the solution would not be optimal.

We define a directed-acyclic network $N = (V, A)$ with a source node s, a sink node t and all the paths connecting the source to the sink. For each job $j \in \{1, \ldots, n\}$ and job position $k \in \{1, \ldots, \min\{j, n-m+1\}\}$ there is a node $(j,k) \in V$. Node $(j', k+1)$ is connected to every node (j,k) with $j < j'$ by a directed arc $((j', k+1), (j,k))$ (recall the jobs are indexed in reverse order). Moreover, source s is connected to every node (j,k) by a directed arc $(s, (j,k))$ and from every node $(j,1)$ there is a directed arc to the sink t. The *length* of every arc entering node (j,k) is kp_j, while all arcs entering the sink t have 0 length. A fragment of network N is shown in Figure 3.

Notice that any directed $s-t$ path P represents an optimal order of jobs on some machine, since the jobs along the path follow the global ordering. The cost of such a path P is

$$c_P = \sum_{(j,k)\in P} kp_j,$$

	pos = 4	3	2	1
M_1			j_4	j_1
M_2		j_5	j_3	j_2

Fig. 4 The positional representation of schedules.

which is the total completion time of the jobs along this path. The integer program requires to find m $s-t$ paths of minimum total weight such that each job occurs on precisely one path. The decision variables are the flow values on the arcs, i.e, for each arc e of the above network, there is a binary variable x_e.

$$\min_x \sum_{j=1}^{n} \sum_{k=1}^{\min\{j,n-m+1\}} \sum_{e\in\delta^{in}(j,k)} kp_j x_e \tag{4}$$

subject to

$$\sum_{e\in\delta^{in}(j,k)} x_e - \sum_{e\in\delta^{out}(j,k)} x_e = 0, \quad \forall (j,k)\in V, \tag{5}$$

$$IP_1: \quad \sum_{k=1}^{\min\{j,n-m+1\}} \sum_{e\in\delta^{in}(j,k)} x_e = 1, \quad \forall j=1,\ldots,n, \tag{6}$$

$$\sum_{e=(s,(j,k))\in A} x_e \le m, \tag{7}$$

$$x_e \in \{0,1\}, \quad \forall e\in A. \tag{8}$$

In this formulation, $\delta^{in}(j,k)$ is the subset of arcs of A entering $(j,k)\in V$. Similarly, $\delta^{out}(j,k)$ is the subset of arcs leaving $(j,k)\in V$. The objective function (4) gives the total weight of selected arcs with edge weights corresponding to the contribution of some arc to the cost of an $s-t$ path. Equations (5) ensure the conservation of flow at each node $(j,k)\in V$. Constraints (6) require that exactly one unit of flow goes through one of the nodes corresponding to each job j. Finally, we require that at most m units of flow leave source s. Let LP_1 denote its linear programming relaxation obtained by replacing (8) with $0\le x_e\le 1$.

The solution of IP_1 is a set of $s-t$ paths, due to (5) and (7). Moreover, by (6) each job is covered exactly once. Notice that IP_1 is feasible by the construction of network N. Therefore, we have:

Proposition 3 *Up to permutation of machines, there is a one-to-one correspondence between the feasible schedules of the bilevel total weighted completion time problem and the feasible solutions of* IP_1.

The second integer program formulation is based on assigning positions to jobs. Clearly, any position can be assigned to at most m jobs, as there are m machines. Moreover, job j can be assigned to positions 1 through $\min\{j, n-m+1\}$ as before. Therefore, we have a decision variable $y_{j,k}$ for each job j and position $k\in$

$\{1, \ldots, \min\{j, n-m+1\}\}$. The second integer program is

$$\min_{y} \sum_{j=1}^{n} \sum_{k=1}^{\min\{j,n-m+1\}} kp_j y_{j,k} \tag{9}$$

subject to

$$\sum_{k=1}^{\min\{j,n-m+1\}} y_{j,k} = 1, \quad \forall j = 1, \ldots, n, \tag{10}$$

$$IP_2: \quad \sum_{j=k}^{n} y_{j,k} \leq m, \quad \forall k = 1, \ldots, n-m+1, \tag{11}$$

$$\sum_{j=k}^{\ell} y_{j,k} \geq \sum_{j=k+1}^{\ell+1} y_{j,k+1}, \quad \forall \ell = 1, \ldots, n-m,\ k = 1, \ldots, \ell, \tag{12}$$

$$y_{j,k} \in \{0,1\}, \quad \forall j = 1, \ldots, n,\ k = 1, \ldots, \min\{j, n-m+1\}. \tag{13}$$

The objective function (9) expresses the contribution of each job to the total completion time of all jobs. Equations (10) prescribe that each job receives exactly one position. Constraints (11) ensure that each position k is assigned to at most m jobs. Finally, constraints (12) make sure that among the last ℓ jobs (indexed by $1, \ldots, \ell$ by our convention) the number of those receiving position k is not less than the number of jobs receiving position $k+1$ among the last $\ell+1$ jobs. To justify this set of constraints, consider any optimal schedule and a fixed job index ℓ and position k. For each machine i, let L_i be the subset of the last $\ell+1$ jobs assigned to machine i. The jobs in L_i are assigned to positions $1, \ldots, |L_i|$. Then clearly, position $k+1$ cannot occur more frequently than position k, because the jobs occupy consecutive positions beginning with the last position (indexed by 1) on each machine. To illustrate this concept, consider Figure 4 with 2 machines and 5 jobs.

Proposition 4 *Any optimal solution of IP_2 is equivalent to an optimal solution of the bilevel total weighted completion time problem under the assumption $w^1 \equiv 1$.*

The correspondence is not one-to-one, since the jobs are assigned to positions only, and the constraints (12) ensure that there is a feasible schedule with the same job positions.

Let LP_2 denote the linear programming relaxation of IP_2 obtained by relaxing (13) to $0 \leq y_{j,k} \leq 1$. Surprisingly, LP_1 and LP_2 are equivalent.

Lemma 8 *The optimum value of LP_1 equals that of LP_2.*

Proof Firstly, we argue that the optimum value of LP_2 is not greater than that of LP_1. To see this, let x^* denote an optimal solution of LP_1 and we define a solution $\bar{y}$ of LP_2 as follows. Let $\bar{y}_{j,k} = \sum_{e \in \delta^{in}(j,k)} x^*_e$, i.e., the total flow entering node (j,k). Clearly, $\bar{y}$ satisfies (10), (11), and $0 \leq \bar{y}_{j,k} \leq 1$. Finally, to justify (12), look at Figure 3. As can be seen, the inflow to the set of nodes $V_{k\ell} = \{(k,k), \ldots, (\ell,k)\}$ comes partly from the nodes $V_{k+1,\ell+1} = \{(k+1,k+1), \ldots, (\ell+1,k+1)\}$, and all the flow through the latter set of nodes is directed toward $V_{k\ell}$. Since x^* satisfies (5),

inequality (12) is satisfied as well. Therefore, $\bar{y}$ is a feasible solution for LP_2 with the same value as x^*. Consequently, the optimum value of LP_2 is at most that of LP_1.

Conversely, we show that if y^* is an optimal solution of LP_2, then LP_1 has a feasible solution of the same value. We construct such a solution using y^* as follows. For any job j and position k, we consider $y^*_{j,k}$ as the total inflow into node (j,k) of network N. We claim that for each k there exists a set of values $\bar{x}_e$, $e \in \delta^{in}(j,k)$ for $j = k, \ldots, n$ such that

$$\sum_{e \in \delta^{in}(j,k)} \bar{x}_e = y^*_{j,k}, \quad j = k, \ldots, n,$$
$$\sum_{e \in \delta^{out}(j,k+1)} \bar{x}_e = y^*_{j,k+1}, \quad j = k+1, \ldots, n.$$

Notice that this system of equations is a transportation problem in a bipartite graph. Recall that $\delta^{in}(j,k)$ contains one arc from each node $(j+1,k+1), \ldots, (n,k+1)$ and an arc from the source s, see Figure 3. By Farkas' Lemma, this system has a non-negative solution iff for all $\alpha_k, \ldots, \alpha_n$, and $\beta_{k+1}, \ldots, \beta_n$,

$$\alpha_j + \beta_{j'} \geq 0, \quad ((j',k+1),(j,k)) \in \delta^{in}(j,k)$$
$$\alpha_j \geq 0, \quad (s,(j,k)) \in \delta^{in}(j,k)$$

imply

$$\sum_{j=k}^{n} \alpha_j y^*_{j,k} + \sum_{j'=k+1}^{n} \beta_{j'} y^*_{j',k} \geq 0.$$

The constraints $\alpha_j \geq 0$ are due to the arcs $(s,(j,k)) \in \delta^{in}(j,k)$ as these arcs do not belong to any of the sets $\delta^{out}(j,k+1)$.

Now we verify that this implication holds if y^* satisfies (12). In fact, for a fixed $\alpha \geq 0$, we can set $\beta_{j'}$ to $-\min\{\alpha_j \mid ((j',k+1),(j,k)) \in \delta^{out}(j',k+1)\}$. Therefore, if $\alpha_j = 0$, then $\beta_{j'} = 0$ for all $j' > j$. Consequently, we may assume that there exists j_1 such that $\alpha_j > 0$, $\beta_{j+1} < 0$ for $k \leq j \leq j_1$ and $\alpha_j = \beta_{j+1} = 0$ for $j_1 < j < n$. Moreover, $\beta_{k+1} \leq \beta_{k+2} \leq \cdots \leq \beta_n \leq 0$, since $\delta^{out}(j,k+1) \subset \delta^{out}(j',k+1)$ for $j < j'$. Hence, we may assume that $\alpha_j = -\beta_{j+1}$ for $j = k, \ldots, n-1$. Notice that this is already quite close to the form of the inequalities (12). Finally, if there are different non-zero α_j values, then letting $\lambda := \min\{\alpha_j \mid \alpha_j > 0\}$, define $\alpha^2_j := \min\{\alpha_j - \lambda, 0\}$, $\beta^2_{j+1} = -\alpha^2_j$ for all $j = k, \ldots, n-1$, and $\alpha^1 = \alpha - \alpha^2$, $\beta^1 := \beta - \beta^2$. Then (α^1, β^1) and (α^2, β^2) meet all of the above properties, and all the non-zeros in α^1, β^1 are λ and $-\lambda$, respectively. Continuing this decomposition with (α^2, β^2) if necessary, we obtain a set of pairs of vectors (α^t, β^t), $t = 1, \ldots, T$, whose sum is (α, β), and each pair (α^t, β^t) satisfies $\lambda_t = \alpha^t_j = -\beta^t_{j+1}$ for $j = k, \ldots, n_t$ for some $\lambda_t > 0$ and $n_t < n$. Hence, the implication holds if y^* satisfies (12), which is true as y^* is an optimal solution of LP_2.

It remains to verify that by choosing any solution for the transportation problem for each k, the total sum of values on the arcs in $\delta^{out}(s)$ is at most m. Notice that

for $1 \leq k < n-m+1$, $\sum_{j=k}^{n} \bar{x}_{(s,(j,k))} = \sum_{j=k}^{n} y^*_{j,k} - \sum_{j=k+1}^{n} y^*_{j,k+1}$, and for $k = n-m+1$, $\sum_{j=n-m+1}^{n} \bar{x}_{(s,(j,n-m+1))} = \sum_{j=n-m+1}^{n} y^*_{j,n-m+1}$. Therefore,

$$\sum_{e \in \delta^{out}(s)} \bar{x}_e = \sum_{k=1}^{n-m} \left(\sum_{j=k}^{n} y^*_{j,k} - \sum_{j=k+1}^{n} y^*_{j,k+1} \right) + \sum_{j=n-m+1}^{n} y^*_{j,n-m+1} = \sum_{j=1}^{n} y^*_{j,1} \leq m$$

as claimed. □

Finally, we show that if w^2 induces an increasing processing time order, i.e., $w^2_j/p_j > w^2_k/p_k$ iff $p_j < p_k$, then LP_2 has an integral optimal solution. Therefore, we have obtained an alternative linear programing formulation for the $P||\sum_j C_j$ problem (cf. Section 3.4.1).

Lemma 9 *If w^2 induces an increasing processing time order of jobs, then LP_2 has an integral optimal solution. Moreover, if all the p_j are different, then all the optimal solutions are integral.*

Proof Since in this section the last job in the global order has index 1, the penultimate job has index 2, etc., the condition of the statement implies that $p_1 \geq p_2 \geq \cdots \geq p_n$. Suppose y^* is an optimal solution with fractional coordinates, and let j be the smallest job index such that $0 < y^*_{j,k} < 1$ for some $k \in \{1, \ldots, \min\{j, n-m+1\}\}$. Let $\bar{k}$ be the smallest position with non-zero $y^*_{j,\bar{k}}$. Since y^* satisfies (10), we have $0 < y^*_{j,\bar{k}} < 1$. We call $(j, \bar{k})$ the *pivot point*.

If $y^*_{j,\bar{k}}$ is the only fractional value among the $y^*_{i,\bar{k}}$ for $j < i \leq n$, then we define a new solution, $\overline{y}$ from y^* as follows. Let $\overline{y}_{j,\bar{k}} = 1$, $\overline{y}_{j,k} = 0$ for all $k \in \{1, \ldots, \min\{j, n-m+1\}\} \setminus \{\bar{k}\}$, and $\overline{y}$ equals y^* on all other coordinates. It is easy to verify that $\overline{y}$ is a feasible solution of LP_2 with a strictly smaller objective function value than y^*, a contradiction.

Now suppose there exists $i > j$ such that $0 < y^*_{i,\bar{k}} < 1$. Then we define a new solution $\overline{y}$ as follows. Let $k' > \bar{k}$ with $y^*_{j,k'} > 0$ (such a position k' necessarily exists), and then we define $\varepsilon := \min\{1 - y^*_{j,\bar{k}}, y^*_{j,k'}, y^*_{i,\bar{k}}, 1 - y^*_{i,k'}\} > 0$. Let $\overline{y}_{j,\bar{k}} = y^*_{j,\bar{k}} + \varepsilon$, $\overline{y}_{j,k'} = y^*_{j,k'} - \varepsilon$, $\overline{y}_{i,\bar{k}} = y^*_{i,\bar{k}} - \varepsilon$, $\overline{y}_{i,k'} = y^*_{i,k'} + \varepsilon$, and $\overline{y}$ and y^* agree on all other coordinates. Again, $\overline{y}$ is a feasible solution of LP_2 and its value is not worse than that of y^*, because the difference of the objective function values is $\varepsilon(\bar{k} - k')p_j - \varepsilon(\bar{k} - k')p_i \leq 0$. Therefore, since y^* is optimal, $\overline{y}$ has to be optimal as well. Moreover, if $p_j > p_i$ then the value of $\overline{y}$ is strictly smaller than that of y^*, a contradiction. Therefore, this transformation is possible only if $p_j = p_i$, which proves the second part of the theorem. If $\overline{y}$ is integral, then we are done. Otherwise, we repeat. To see that this process terminates in a finite number of steps, notice that $\overline{y}_{j,\bar{k}} > y^*_{j,\bar{k}}$, i.e., $\bar{y}$ has increased on the pivot point $(j, \bar{k})$. Moreover, the next pivot point is either $(j, \bar{k})$ (provided $\overline{y}_{j,\bar{k}} < 1$), or involves some job with $i > j$. The former case may occur only finitely many times, hence after a finite number of pivots on $(j, \bar{k})$ a new job $i > j$ is selected. Since there are only n jobs, after a finite number of steps the solution becomes integral. □

We say that LP is a linear programming formulation for a combinatorial problem CP, if LP admits an integral optimal solution which is optimal for CP.

Theorem 2 *Both LP_1 and LP_2 are linear programming formulations for $P||\sum_j C_j$.*

3.6.2 Iterative rounding algorithms

Our algorithms consist of repeatedly solving the LP-relaxation of IP_1 or IP_2, and extracting a sequence of jobs that may be scheduled in that order in each iteration. When using IP_1, the algorithm is the following:

1. Let $LP := LP_1$
2. **for all** ℓ **in** $1, \ldots, m$ **repeat**
   (a) Solve LP and let $\overline{x}$ be an optimal solution.
   (b) Find any $s-t$ path P_ℓ in $N(\overline{x})$. For all arcs $e \in P_\ell$: set the right-hand-side of eq. $(6j)$ to 0 in LP, where $e = ((j', k+1), (j, k))$. Set the supply at the source s to $m - \ell$ in eq. (7).
3. **output** paths $P_1, \ldots, P_m$.

$N(\overline{x})$ is the sub-network of N consisting of all the nodes and those arcs $e \in A$ with $\overline{x}_e > 0$. We call this algorithm Alg_1. Too see that after m iterations all the nodes are covered by a path, we prove the following.

Lemma 10 *In each iteration at least one more job is covered by a path. In the last iteration ($\ell = m$), there is only one $s-t$ path in $N(\overline{x})$ covering all remaining jobs.*

Proof The first part of the lemma follows from the observation that $N(\overline{x})$ contains at least one $s-t$ path. To show the second part, we proceed by induction. First notice that in the last iteration, the supply at node s equals 1 (which is set at the end of iteration $\ell = m-1$). Consider the heads of the arcs leaving node s in $N(\overline{x})$, where $\overline{x}$ is the optimal solution of the LP in the last iteration. Let $F := \{(j_1, k_1), \ldots, (j_r, k_r)\}$ be the set of this nodes. Suppose j_b is the maximum job index over all nodes in F, and there exists $j_a < j_b$ for some $a \neq b$. But then the total inflow into the nodes (j_b, k) over all positions $k = 1, \ldots, \min\{j_b, n-m+1\}$ is

$$\sum_{k=1}^{\min\{j_b, n-m+1\}} \sum_{e \in \delta^{in}(j_b,k)} \overline{x}_e = \sum_{k=1}^{\min\{j_b, n-m+1\}} \overline{x}_{(s,(j_b,k))} < 1,$$

a contradiction. Therefore, $j_1 = \cdots = j_r$. The same argument shows that all successors of the nodes in F must belong to the same job. Repeating this, we obtain that all the $s-t$ paths in $N(\overline{x})$ contain the same jobs, which is possible only if there is one $s-t$ path in $N(\overline{x})$ $\square$

The algorithm has polynomial time complexity : each of the m linear programs can be solved by Tardos' method in strongly polynomial time [22] and each of the m $s-t$ paths in the acyclic digraphs can be found in $O(n^2)$ time.

In another variant of the algorithm, we use LP_2 in place of LP_1. In step (2a) the optimal solution of LP is $\overline{y}$. In step (2b), a sequence of pairs $(j_1, 1), (j_2, 2), \ldots, (j_k, k)$ is sought such that $j_1 < j_2 < \cdots < j_k$, $\overline{y}_{j_i,i} > 0$ for $i = 1, \ldots, k$ and k is maximal, i.e., the sequence cannot be extended. Again, having identified such a sequence of pairs, set the demand to 0 for each job j_i, $i = 1, \ldots, k$ in eq. (10) and replace all the right-hand-sides with $m - \ell$ in (11). We call this algorithm Alg_2. This variant also covers all the jobs in m iterations, which can be seen using the equivalence of LP_1 and LP_2, and Lemma 10.

Table 2 Results of Alg_2 on random instances. Each cell contains the average ratio $(Alg_2 - LP_2)/LP_2$ over 20 instances.

	$m = 5$	$m = 10$
$n = 50$	0.0019	0.00068
$n = 100$	0.0028	0.00204

3.6.3 Computational evaluation

The purpose of computational evaluation has been to (i) evaluate the performance of Alg_2 on randomly generated instances, and (ii) identify the structure of the hardest instances. Below we summarise our experience in both cases.

We implemented both algorithms in the Mosel programming language of FICO Xpress. Preliminary computations had shown that the performance of the two heuristics were the same. In fact, the selection of the $s - t$ path in Alg_1, or the selection of the maximal chain in Alg_2 do not really affect the performance of the methods. On the other hand, Alg_2 is much faster, since the number of decision variables of LP_2 is $O(n^2)$, whereas that of LP_1 is $O(n^3)$. Therefore, all results reported here were obtained by Alg_2. We will slightly abuse notation and denote by Alg_2 and LP_2 the value of the solution produced by Alg_2 and the optimum value of LP_2, respectively, on a problem instance.

The performance measure in all experiments was the ratio $(Alg_2 - LP_2)/LP_2$, i.e., the relative error.

The random instances were generated for different n and m values. For each job, both its processing time p_j and weight w_j^2 were chosen uniformly at random in the interval $[1, 50]$. We generated 20 instances for each combination of $n = 50, 100$ and $m = 5, 10$. Our findings are summarised in Table 2.

- Increasing m while keeping n fixed improves the performance of the method.
- Increasing n while keeping m fixed degrades the performance of the method.
- The relative error was below 1% in all cases.

However, these experiments show the behaviour on random instances only. Therefore, we aimed at identifying hard instances on which the performance of our methods degrade considerably. We have identified the following characteristics of hard instances. Observe that if the global ordering of jobs begins with long jobs, and it continues with short jobs only, then in a relaxed solution fractions of long jobs can be processed at the end of the schedule, thus reducing the costs. On the other hand, in a feasible schedule most of the long jobs are processed before the short jobs provided the number of short jobs is sufficiently large. After some experiments we found that a ratio of $1 : 2$ between the number of long and short jobs yields the worst results. The processing times of the long jobs were random numbers in the interval $[50, 80]$, while those of the short jobs were chosen from the interval $[1, 20]$. To ensure that long jobs precede short jobs in the global order, we let $w_j := (n + 1 - j)p_j$. The number of jobs were $n = 50, 100, 200$, while that of the machines were $m = 3, 5$. For $m = 10$, the relative error diminished. The results are summarised in Table 3. Each cell contains the average relative error as well as the maximum over the 20 randomly

Table 3 Results of Alg_2 on hard random instances. Each cell contains the average as well as the maximum ratio $(Alg_2 - LP_2)/LP_2$ over 20 instances.

	$m = 3$ rel (max)	$m = 5$ rel (max)
$n = 50$	0.186 (0.232)	0.088 (0.105)
$n = 100$	0.196 (0.229)	0.104 (0.115)
$n = 200$	0.203 (0.227)	0.107 (0.113)

generated hard instances. We can draw similar conclusions to those on unstructured random instances, except that the average and the maximum relative error were 20% and 23%, respectively, in the worst case.

3.7 Beyond the $w^1 \equiv 1$ special case

If the job weights of the leader are not uniform, then the bilevel total weighted completion time problem is not easier to approximate than its single level special case. This can be seen by considering the instances with $w^2 \equiv w^1$. Therefore, we cannot hope for stronger approximation results than in the single level case in general. For the problem $P2||\sum_j w_j C_j$, Sahni [21] describes an $(1+\varepsilon)$-approximation algorithm with $O(n^2/\varepsilon)$ time complexity (an FPTAS). For any fixed number of machines, the FPTAS of Schuurman and Wöginger [23] for $Pm||\sum_j w_j C_j$ can be generalised to our problem. To apply their algorithm to the bilevel total weighted completion time problem, it suffices to note that the FPTAS for $Pm||\sum_j w_j C_j$ is based on a dynamic program which processes the jobs in decreasing w_j/p_j order. In our case, we apply the same algorithm for a job sequence (1) in the optimistic case, and (2) in the pessimistic case. Therefore, we have

Lemma 11 *There is an FPTAS for the bilevel total weighted completion time problem with a fixed number of machines.*

The time complexity of the FPTAS is $O(nL^m)$, where $L = \lceil \log_\Delta p_{sum} \rceil$ with $p_{sum} = \sum_{j=1}^{n} p_j$, and $\Delta = 1 + \frac{\varepsilon}{2n}$, ε being the desired relative error.

4 The bilevel order acceptance problem

In this section we establish complexity results for the bilevel order acceptance problem, and study some special cases.

4.1 Preliminaries

The bilevel order acceptance problem is a generalisation of the single-machine weighted number of late jobs problem. In that problem there are n jobs, each job has a processing time p_j, a due-date d_j, and a weight w_j. A sequence of jobs is sought such that the total weight of those jobs competed after their due-dates is minimal. This problem is

denoted by $1||\sum_j w_j U_j$. In the decision version of the problem, there is also given a constant K and one asks whether there is a feasible solution with total weight of late jobs not greater than K. It is well-known that there always exists an optimal solution such that those jobs completed on time are processed in earliest due-date order (EDD order for short), i.e., if both of the jobs j and k are completed before their due-dates, then j is processed before k, if $d_j \leq d_k$ (cf. [4]).

4.2 Global ordering of jobs

Given the leader's decision about the selection of jobs J' to be completed on time, the follower sequences the jobs in WSPT order (with respect to weights w_j^2) to minimise its objective function. In case of ties, there is a distinction between the optimistic and the pessimistic cases. We define two global orderings of jobs such that the optimal optimistic and pessimistic solutions, respectively, are sub-sequences of the global job orders. In the optimistic case the global order is

$$\text{job } j \text{ precedes job } k \text{ if } w_j^2/p_j > w_k^2/p_k, \text{ or } (\ w_j^2/p_j = w_k^2/p_k \text{ and } d_j < d_k), \quad (14)$$

while in the pessimistic case it is

$$\text{job } j \text{ precedes job } k \text{ if } w_j^2/p_j > w_k^2/p_k, \text{ or } (\ w_j^2/p_j = w_k^2/p_k \text{ and } d_j > d_k). \quad (15)$$

Proposition 5 *Both the optimistic and the pessimistic order acceptance problem always admits an optimal non-delayed schedule. In particular, in the pessimistic case, no job with $w_j^2 = 0$ can be accepted.*

Proof First notice that it is feasible for the leader not to choose any jobs, so the set of feasible solutions is not empty. Moreover, in both the optimistic and the pessimistic case, the follower schedules those accepted jobs j with $w_j^2 > 0$ without any delay before them. The same holds in the optimistic case for accepted jobs j with $w_j^2 = 0$ as well, since such schedules are the most favourable for the leader.

Now consider the pessimistic case and suppose the leader accepts some jobs with $w_j^2 = 0$ (if such a job exists). Since the follower plays against the leader, it may answer a schedule which delays some of the accepted jobs j with $w_j^2 = 0$ beyond their deadlines. Such a schedule is though optimal for the follower, but it is infeasible for the leader. Therefore, the leader cannot accept any jobs with $w_j^2 = 0$. □

If the follower must choose a non-delayed optimal schedule, then also in the pessimistic case jobs with $w_j^2 = 0$ can be accepted by the leader.

Lemma 12 *There exists an optimal solution for the optimistic bilevel order acceptance problem such that the jobs are sequenced according to (14).*

Proof The existence of an optimal schedule is ensured by Proposition 5. Let J^* be the leader's optimal selection of jobs. The follower schedules the jobs in J^* by the WSPT rule with job-weights w_j^2. In case of ties, i.e., $w_j^2/p_j = w_k^2/p_k$, it can always schedule first the job with smaller due-date, which is expressed in (14). □

Lemma 13 *There exists an optimal solution for the pessimistic bilevel order acceptance problem such that the jobs are sequenced according to (15).*

Proof The existence of an optimal solution is guaranteed by Proposition 5. Let J^* be the leader's optimal selection of jobs. If $w_j^2 > 0$ for all $j \in J^*$, or no job may be delayed unnecessarily, the follower schedules the jobs in J^* by the WSPT rule with job-weights w_j^2. The worst case for the leader is that in case of ties, i.e., $w_j^2/p_j = w_k^2/p_k$, the job with larger due-date is scheduled first, which is expressed in (15). □

Finally, similarly to Lemma 3, one can prove the following:

Lemma 14 *Any instance Π of the bilevel order acceptance problem can be converted into an instance $\bar{\Pi}$ such that the followers' WSPT order is unique, and all the optimal solutions of $\bar{\Pi}$ are optimal for Π as well.*

4.3 Complexity

Given a constant K, in the optimistic case the decision version of the bilevel order acceptance problem asks whether there exists a subset $J' \subseteq J$ of jobs such that

1. $\sum_{j \in J \setminus J'} w_j \leq K$, and
2. there exists a WSPT order $\prec$ of the jobs in J' with $C_j \leq d_j$ for all $j \in J'$, where $C_j = p_j + \sum_{k \in J' : k \prec j} p_k$ is the completion time of job j in the order $\prec$.

In contrast, in the pessimistic case the decision problem asks whether there exists $J' \subseteq J$ such that

1. $\sum_{j \in J \setminus J'} w_j \leq K$, and
2. for every WSPT order $\prec$ of the jobs in J', $C_j \leq d_j$ for all $j \in J'$, where $C_j = p_j + \sum_{k \in J' : k \prec j} p_k$ is the completion time of job j in the order $\prec$.

Notice that if the WSPT order is unique for each subset of jobs, then the optimistic and the pessimistic cases of the problem coincide.

Lemma 15 *The bilevel order acceptance problem is NP-complete both in the optimistic and in the pessimistic cases.*

Proof Membership of NP follows from the fact that the follower's problem can be solved in polynomial time. Concerning NP-hardness, for any instance Π_1 of the single-level $1||\sum_j w_j U_j$ problem, we define an instance of the bilevel order acceptance problem Π_2 with n jobs, processing times p_j, and job weights $w_j^1 = w_j$, and w_j^2 inducing an EDD order, i.e., $w_j^2/p_j \geq w_k^2/p_k$ if and only if $d_j \leq d_k$ (such weights clearly exist). Moreover, w_j^2 can be chosen such that the EDD order is unique. Therefore, the optimistic and the pessimistic cases coincide. Finally, Π_2 has the same constant K as Π_1.

First suppose Π_1 admits a solution with value at most K. Since we may assume that those jobs completed on time are processed in EDD order, this immediately yields a feasible solution for the bilevel scheduling problem with the same objective function value for the leader. Conversely, suppose the bilevel problem instance Π_2 admits a feasible solution with value at most K. Then this solution is also feasible for the single-level problem instance Π_1 with the same value. □

The problem is not strongly NP-hard, since it is solvable in pseudo-polynomial time, see below.

Notice that the above proof shows that if the job weights w_j^2 induce an EDD order, then the bilevel problem becomes equivalent to the single level $1||\sum_j w_j U_j$ problem. But, this is not true in general.

4.4 A dynamic program for the general case

A modified version of the dynamic program proposed by Lawler & Moore [15] for $1||\sum_j w_j U_j$ is applicable for solving our bilevel problem. In that algorithm the jobs are processed in EDD order, i.e., the jobs are reindexed such that $d_1 \leq d_2 \leq \cdots \leq d_n$. Let $F_j(t)$ be the value of the optimal schedule of the problem involving jobs $1, \ldots, j$ that ends at time t. If $0 \leq t \leq d_j$ and job j is on time in the schedule corresponding to $F_j(t)$, then $F_j(t) = F_{j-1}(t - p_j)$. Otherwise, $F_j(t) = F_{j-1}(t) + w_j$. If $t > d_j$, then $F_j(t) = F_{j-1}(t) + w_j$ because job j is late.

We modify the algorithm of Lawler & Moore by processing the jobs in order (14) in the optimistic case, and in order (15) in the pessimistic case, and the jobs are reindexed to reflect the appropriate ordering. Let $T = \sum_{j \in J} p_j$. The recursion from $j-1$ to j is:

$$F_j(t) = \begin{cases} \min\{\ F_{j-1}(t - p_j),\ F_{j-1}(t) + w_j^1\ \} & \text{for } t = 0, \ldots, d_j, \\ F_{j-1}(t) + w_j^1 & \text{for } t = d_j + 1, \ldots, T, \end{cases}$$

with $F_0(0) = 0$, $F_j(t) = \infty$ for $t < 0$, and $j = 0, \ldots, n$; or $j = 0$ and $1 \leq t \leq T$.

The smallest $F_n(t)$ gives the optimum value, and by some book-keeping one also gets the optimal solution. Namely, let $v_j(t) = 0$ if $F_j(t) = F_{j-1}(t - p_j)$, and $v_j(t) = 1$ otherwise. Suppose $F_n(t^*) \leq F_n(t)$ for all t. Starting from $v_n(t^*)$, we can determine the optimal solution by visiting the jobs backward. In the general step, if $v_j(t) = 1$, then job j is rejected, and we proceed with $v_{j-1}(t)$. Otherwise, job j is accepted, and we proceed with $v_{j-1}(t - p_j)$. Repeating this until all the jobs are checked, we get a subset of accepted jobs. Since $v_j(t) = 0$ only if $t \leq d_j$, in the resulting solution all accepted jobs are completed on time. Moreover, the processing order respects the follower's WSPT order by construction.

The time and space complexity of the algorithm is $O(nT)$.

4.5 Polynomial time algorithm for the $w^1 \equiv 1$ special case

This special case can be solved by a modified version of the Moore-Hogdson algorithm [19]. In that algorithm, the jobs are processed in EDD order. Starting with an empty schedule, the jobs are appended to the end of the growing schedule one-by-one. If the appended job j completes late, then the job k in the partial schedule with the largest processing time p_k is rejected and removed from the schedule ($j = k$ is allowed). Note that removing at most one job from the schedule always yields a feasible schedule.

We modify the Moore-Hogdson algorithm by processing the jobs in the order (14) (optimistic case) or (15) (pessimistic case). Our modified algorithm appends jobs to the end of the schedule one-by-one and removes the actual lengthiest job when necessary exactly as the original Moore-Hogdson algorithm. The algorithm runs in $O(n \log n)$ time.

We adapt the proof of [2], page 86, of the soundness of the Moore-Hogdson algorithm to our more general case. In the following proof, $|\sigma|$ denotes the number of jobs in some schedule σ.

Theorem 3 *The Modified Moore-Hogdson algorithm provides an optimal solution to the Bilevel order acceptance problem.*

Proof If all the jobs can be processed on time, then the algorithm obviously finds this optimal solution. Otherwise, the algorithm removes at least one job from the schedule being constructed. Let j be the job that is removed first, in the step where job k is appended to the schedule. This implies that at least one of the jobs $1, ..., k$ is missing from every feasible schedule. Since j has the longest processing time among these jobs, any partial schedule σ' of the first k jobs that includes j but misses job h with $1 \leq h \leq k$, completes not earlier than the partial schedule $\sigma_{jh} = \sigma' \setminus \{j\} \cup \{h\}$ obtained by replacing j with h (the jobs are sequenced in the global order). Moreover, if no job is late in σ', then so is in σ_{jh}, because σ_{jh} schedules a subset of jobs in $\{1, \ldots, k-1\}$ and the global ordering of the latter yields a schedule in which no job is late. Hence, there exists an optimal schedule that does not contain j.

The rest of the proof goes by induction on the number of jobs n. Clearly, the algorithm is sound for $n = 1$. Assume it is correct for all instances involving $n - 1$ jobs. For n jobs, let σ be the schedule constructed by the algorithm and σ^* an optimal schedule with $j \notin \sigma^*$, where j is the first job removed while constructing σ. Observe that when our algorithm is applied to the problem involving jobs $\{1, ..., j-1, j+1, ..., n\}$ only, it constructs σ again, and this schedule is optimal for the reduced problem. Since σ^* is also a feasible solution for the reduced problem, we have $|\sigma| \geq |\sigma^*|$, and hence, σ is optimal for the original problem, too. □

5 Final remarks

In this paper we have derived complexity results and various algorithms for two bilevel machine scheduling problems. In one of them, both the leader and the follower have the same type of objective function, while in the other problem the objective functions of the two levels are different. This can be one way of classifying bilevel machine scheduling problems.

An important aspect is that we have dealt only with problems such that the scheduling problem of the follower is polynomially solvable. A considerably harder case is when the follower's problem is NP-hard. In that case, there is no way to characterise the optimal solutions in a concise way, unless P = NP. However, the short representation of all optimal solutions has been the key in solving some of the special cases discussed in this paper.

Acknowledgements The authors are grateful to the two anonymous referees for constructive comments that helped to improve the paper. The work reported here has been supported by OTKA grant K76810 and NKTH grant OMFB-01638/2009. A. Kovács acknowledges the support of the János Bolyai scholarship No. BO/00138/07.

References

1. Agnetis A, Mirchandani PB, Pacciarelli D, Pacifici A (2004), Scheduling problems with two competing agents, Oper Res 52: 229-242.
2. Brucker P (2007), Scheduling Algorithms, 5. edition, Springer.
3. Cardinal J, Demaine ED, Fiorini S, Joret G, Langerman S, Newman I, Weimann O (2009), The Stackelberg minimum spanning tree game, Algorithmica, in press.
4. Chen B, Potts CN, Wöginger GJ (1998), A review of machine scheduling: Complexity, algorithms and approximability. In: Du D-Z, Pardalos P (eds.), Handbook of Combinatorial Optimization, Kluwer Academic Publishers.
5. Dempe S, Richter K (2000), Bilevel programming with knapsack constraints, Cent Eur J Oper Res 8: 93–107.
6. Dempe S (2002), Foundations of bilevel programming, Kluwer Academic Publishers, Dordrecht.
7. Dempe S (2003), Annotated bibliography on bilevel programming and mathematical programming with equilibrium constraints, Optimization 52: 333–359.
8. DeNegre ST, Ralphs TK (2009), A branch-and-cut algorithms for integer bilevel programs, In: Chinneck JW, Kristjansson B, Saltzman M (eds.), Operations Research and cyber-infrastructure, Springer, pp. 65–78.
9. Graham R L, Lawler E L, Lenstra J K, and Rinnooy Kan A H G (1979), Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Operations Research 5:287–326.
10. Harjunkoski I, Grossmann IE (2002), Decomposition techniques for multistage scheduling problems using mixed-integer and constraint programming methods, Comput Chem Eng 26: 1533-1552.
11. Hoogeveen H (2005), Multicriteria scheduling, Eur J Oper Res 167: 592–623.
12. Karlof JK, Wang W (1996), Bilevel programming applied to the flowshop scheduling problem, Comput Oper Res 23: 443–451.
13. Karp RM (1972), Reducibility among combinatorial problems, In: Miller RE, Thatcher JW (eds.) Complexity of Computer Computations, Plenum Press, New York.
14. Lasserre JB (1992), An integrated model for job-shop planning and scheduling, Manage Sci 38: 1201–1211.
15. Lawler EL, Moore JM (1969), A functional equation and its application to resource allocation and sequencing problems. Manage Sci 16: 77–84.
16. Lenstra JK, unpublished.
17. Lukač Z, Šorić K, Rosenzweig V (2008), Production planning problem with sequence dependent setups as a bilevel programming problem, Eur J Oper Res 187: 1504–1512.
18. Marcotte P, Savard G (2005), Bilevel programming: a combinatorial perspective, In: Avis D, Hertz A, Marcotte O (eds.), Graph Theory and Combinatorial Optimization, Springer, pp. 191–218.
19. Moore J M (1968), An n job, one machine sequencing algorithm for minimizing the number of late jobs. Manage Sci 15: 102–109.
20. Nisan N, Ronen A (2001), Algorithmic mechanism design, Games and Economic Behavior 35: 166–196.
21. Sahni SK (1976), Algorithms for scheduling independent tasks, J ACM 23: 116-127.
22. Tardos É (1986), A strongly polynomial algorithm for solving combinatorial linear programs , Oper Res 34: 250 –256.
23. Schuurman P, Wöginger GJ (2001), Approximation schemes – A tutorial, Research Report Woe-65, CS Department, TU Graz, Austria.