Constraint Programming Approach to a Bilevel Scheduling Problem

András Kovács, Tamás Kis
Computer and Automation Research Institute
Hungarian Academy of Sciences
E-mail addresses: {andras.kovacs,tamas.kis}@sztaki.hu

March 4, 2010

Abstract

Bilevel optimization problems involve two decision makers who make their choices sequentially, either one according to its own objective function. Many problems arising in economy and management science can be modeled as bilevel optimization problems. Several special cases of bilevel problem have been studied in the literature, e.g., linear bilevel problems. However, up to now, very little is known about solution techniques of discrete bilevel problems. In this paper we show that constraint programming can be used to model and solve such problems. We demonstrate our first results on a simple bilevel scheduling problem.

Keywords: Scheduling, bilevel programming, constraint modeling, QCSP

1 Introduction

Bilevel programming deals with decision and optimization problems whose outcome is determined by the interplay of two self-interested decision makers who decide sequentially. First, the decision maker called the *leader* makes its choice. Then, in view of the leader's decision, the *follower* chooses its response. Either decision maker aims at minimizing (maximizing) its own objective function. In the general case, the objective values mutually depend on the choices of the other party. Technically, the follower's role can be seen as solving a parametric optimization problem, whose parameters are determined by the leader. The particularly interesting situation is that of the leader, who is assumed to have a complete knowledge of the follower's constraints, objective, and input data. He endeavors to find his best choice subject to the response that he can expect from the self-interested follower. In the optimistic (pessimistic) case the leader assumes that the follower chooses from the set of its optimal responses the one that is the most (least) favorable for the leader.

Formally, the set of all variables in the problem is partitioned into two sets: the leader's variables X, and the follower's variables Y. The leader can assign values to X, while the follower decides about Y, and it is assumed that all variables have finite domains. The leader aims at minimizing f subject to the constraint set C and the follower's optimality condition, which states that the follower will minimize g subject to D. Also, the leader must avoid the values of X for which the follower's response does not satisfy C. Throughout the paper we assume that both the leader and the follower try to *minimize* their objectives, though, the same techniques can be used for maximization or mixed problems as well. Hence, the optimistic bilevel problem can be formulated as:

$$\begin{aligned} \min_{X,Y} \quad & f(X,Y) & (1)\\ \text{subject to} \quad & & \\ & C(X,Y) & (2)\\ & Y \in \arg\min_{Y'}(g(X,Y') \mid D(X,Y')) & (3) \end{aligned}$$

In formula (3), the operator *arg min* refers to the set of all optimal solutions of the problem at hand. Moreover, the pessimistic case of the problem is described as:

$$\begin{aligned} \min_{X}\max_{Y} \quad & f(X,Y) & (4)\\ \text{subject to} \quad & & \\ & C(X,Y) & (5)\\ & Y \in \arg\min_{Y'}(g(X,Y') \mid D(X,Y')). & (6) \end{aligned}$$

Bilevel programming techniques can be applied to model various decision problems of actors in customer-producer relations, in competition, or at various levels of an organizational hierarchy. Despite this, well-founded theoretical results are known for special cases of bilevel problems only. These include various exact and heuristic approaches to linear bilevel problems (where all constraints and both objective functions are linear expressions over continuous variables), and mostly heuristic methods for other cases, such as bilinear problems [13]. The papers [12, 14] address problems where the follower's variables can take discrete values. For (fully) discrete bilevel problems, which are in the focus of this study, only sporadic application results are available, see [22, 26]. Also, to the best of our knowledge, this paper is the first to investigate the solution of bilevel optimization problems using constraint programming (CP) techniques.

1.1 A motivating example

The classical approach in management science assumes that the different departments of the same company, although have individual decision roles and responsibilities, subsume their interest to the same global objective. This objective is related to maximizing the long-term profit of the company. The reality is often different: the performance of each department is evaluated using, and rewarded based on, a different performance measure. These performance measures are only distantly related to the global objective of the company, and are often conflicting. Hence, a relevant alternative model of the joint operation of several departments is using multilevel programming techniques [4]. A simple case study is presented below.

Consider the bilevel scheduling problem where the management of the company (the leader) is responsible for *order acceptance* and the workshop foreman (the follower) decides on the *execution sequence* of the tasks corresponding to accepted orders. The leader has no direct influence on the sequencing decisions. Formally, there is a set of tasks T, some of which will have to be scheduled on a single unary resource. Task j is characterized by its processing time p_j, release time r_j, and deadline d_j. The difference between the profit if j is executed on time and the loss of reputation if it is rejected is captured by the cost (or task weight) w_j^1 to be paid if the task is rejected. A solution is acceptable for the leader only if all the accepted tasks are completed on time. The leader must select the tasks that will be actually executed: the binary variable x_j is 1 if task j is accepted and 0 if rejected. The objective of the leader is to minimize $\sum_j w_j^1(1-x_j)$ subject to the temporal constraints.

The sequencing decisions are made by the follower, who aims at minimizing the total weighted completion time of the tasks selected by the leader, i.e., $\{j \mid x_j = 1\}$. The start and completion times of tasks j are denoted by S_j and C_j, respectively, and the relation $S_j + p_j = C_j$ holds. The task weights w_j^2 that express the importance of tasks for the follower are independent from the leader's task weights w_j^1. We assume that the follower observes the release times, but the organizational relations within the company are such that the leader cannot force the follower to obey the deadlines. Hence, it might happen that a set of tasks could be scheduled on time, but the follower prefers to execute them in a sequence that violates some deadlines. Such task sets do not lead to feasible solutions of the bilevel problem.

Using the classical three-field scheduling notation [18], the follower's problem corresponds to a parametric version of $1|r_j|\sum_j w_j^2 C_j$. The first field of the notation specifies the machine environment; in our case number 1 stands for a single machine problem. The second field defines the constraints on activities; they are subject to individual release dates (r_j) in this problem. Finally, the third field states that the optimization criterion is the total weighted completion time of the tasks ($\sum_j w_j^2 C_j$). In our bilevel problem, this widely studied problem is parameterized with variables x_j, which decide the set of tasks to be considered by the follower.

This sample problem is a special type of bilevel problems where the leader's objective depends only on the leader's variables. However, the feasibility of a solution depends on the follower's response as well. For further examples from the scheduling domain, see Section 2.

1.2 Structure of this paper

The remainder of this paper is organized as follows. First, we review the related literature. After making the necessary definitions and presenting some basic theoretical results (Section 3), we introduce a generic CP approach to discrete bilevel optimization problems (Section 4). In Section 5 we illustrate the use of those techniques on the sample scheduling problem. Finally, we present experimental results in Section 6, and then conclude the paper.

2 Related literature

A number of different approaches in optimization deal with situations where the decision maker has only limited control of the problem at hand. Stochastic programming [31] considers random events occurring with known probability, and aims at optimizing the expected performance. Quantified problem solving looks at finding strategies for all possible actions of an adversary. In contrast, bilevel programming assumes a self-interested adversary with completely known objectives, and wishes to find a solution with the assumption that the adversary acts rationally.

2.1 Applications of bilevel programming

Probably the earliest example and a motivation of bilevel optimization problems came from economic game theory. In a two-player *Stackelberg game* two competing firms, the market leader and a follower company, for example a new entrant, produce equivalent goods. The firms decide their production quantities sequentially, which together determine the market price, with the aim of maximizing their own profit [13].

The application of bilevel programming to the coordination of multi-divisional organizations has been proposed in [4]. The approach is illustrated on a case study of three divisions of a paper company. The divisions are responsible for different stages of processing the paper, hence, the end product of one division serves as raw material for another division. Each division can decide to buy or sell on the outside market or from/to another division. The objective of the corporate unit is to set the internal transfer prices in such a way that the optimal decisions on the divisional level coincide with the corporate optimum. This problem can be encoded into a linear bilevel problem, and solved by known algorithms from the literature.

There exist a few application areas of discrete bilevel problems, and especially bilevel scheduling. In [26], the production planning problem of a pharmaceutical company is considered, while [22] studies a bilevel problem that may arise in flow shop scheduling. These papers take a relatively straightforward solution approach: they enumerate (a part of) the leader's possible choices, and for each choice, compute the follower's response. Brown et al. [8] investigate a bilevel project scheduling problem where the objective of the decision maker is to cause maximal delay of its adversary's project, which is given as a PERT network. The interdictor can buy delays, while the project owner can buy speed-ups on some arcs of the network from their limited budget. In [23], we present basic complexity and algorithmic results for bilevel scheduling problems.

Various other bilevel optimization problems arise naturally in economy and management science. Perhaps the most widely discussed example is the *toll setting* problem in a network, e.g., in a system of regional highways [24]. The owner of the network (the leader) seeks for the optimal pricing of each link in the network so as to maximize its profit. The follower corresponds to the ensemble of the users of the network. A fixed amount of users belong to each origin-destination pair, and each user selects the path that minimizes his costs, composed of the travel time and the tolls to pay. Many variations of this basic problem have been investigated, including problems where tolls or traffic signs are set by the local authorities who wish to control the movement of hazardous materials or consider other environmental effects [27]. Another typical application is the optimization of chemical processes. Here, the follower's optimality condition describes that the steady-state result of a chemical reaction is an equilibrium where the reacting substances reach their energy minimum [10].

2.2 Related problems in CP

In constraint programming, a problem class strongly related to bilevel programming is the class of *quantified constraint satisfaction problems* (QCSPs), and their optimization versions, *quantified constraint optimization problems* (QCOPs) [5, 17]. While a classical constraint program corresponds to evaluating a formula that contains existentially quantified variables only (e.g., $\exists x \exists y\ C(x, y)$), in QCSP it is allowed to have universally quantified variables as well (e.g., $\exists x \forall y\ C(x, y)$). In papers [5] and [7], the basic QCSP and QCOP language has been extended with restricted quantification, resulting in the QCSP+ and QCOP+ languages. A sample QCSP+ formula is $\exists x \forall y[L(x, y)]\ C(x, y)$, which contains the restricted quantifier $\forall y[L(x, y)]$. This reads *"for all y such that $L(x, y)$ it holds that..."*. It is easy to show that a QCSP+ formula can be translated into a QCSP formula with negation and disjunction.

A number of QCSP solvers have been proposed in the literature, including the open-source QCOP+ solver called QeCode by Benedetti at al. [2], built on the top of Gecode [1]; QCSP-Solve, a solver partly motivated by ideas from QBF-solving by Gent et al. [16]; and the bottom-up solver BlockSolve by Verger & Bessière [36].

We are aware of two applications of QCSP to scheduling problems. Benedetti et al. [6] present a QCSP+ model of a scheduling game in which an adversary can change some task parameters–e.g., the resource requirement of some tasks subject to a limit on the overall increase of requirements. The objective is to find a robust schedule that remains feasible whatever actions the adversary takes. Nightingale [29] presents a QCSP model for job-shop scheduling with the risk of machine breakdowns. We will investigate the relation of bilevel programming to QCSP in detail in the next section.

Another related problem is the class of *adversarial constraint satisfaction* *problems* (ACSP) [9]. ACSP can be used to model games played by n agents with potentially conflicting interests that consist of a fixed number of rounds. The number of rounds equals the number of variables in the ACSP, which is typically much larger that the number of agents. The main difference between ACSP and bilevel programming is that in ACSP in each round the forthcoming agent is free to choose an arbitrary variable to instantiate, i.e., variables are not assigned to agents a priori. Also, in ACSP, all agents must satisfy the same set of constraints, although in theory it is possible to incorporate a measure of constraint violations into the optimization criteria of each agent.

2.3 Bilevel problems in game theory

The presence of two self-interested decision makers make bilevel programming interesting from the game theoretical point of view as well. The optimal solution of an optimistic (pessimistic) bilevel program corresponds to a weak (strong) Stackelberg equilibrium [34]. As opposed to the classical Nash equilibrium for continuous games, Stackelberg equilibrium refers to games with turns. Also, Stackelberg equilibrium is different from subgame perfect equilibrium (SPE) [37], since SPE requires the strategy to cover all possible moves of the opponent, not only its optimal ones. The original concept of Stackelberg has been extended to an oligopolistic market with one leader and N followers by Sherali et al [33]. In that model the followers reach an equilibrium solution in the market of a single homogeneous product and the leader, supplying the same product without any collusion with the other firms, sets the production levels in an optimal (profit maximizing) fashion by explicitly considering the reaction of the other firms to its output variations. This model leads to a mathematical program with equilibrium constraints, and the latter area has a rich literature.

2.4 Solution techniques applied

In our work we rely on known techniques of constraint programming and operations research, and adapt these to bilevel problems. For a detailed presentation of the applied constraint propagation algorithms and search techniques the reader is referred to [3, 32].

We note that the applied decomposition to a master problem and a subproblem resembles the (logic-based) Benders decomposition approach to single-level problems [19]. However, a substantial difference is that in the single level Benders case one is free to choose the separation of the master and the subproblem as it is the most efficient computationally, whereas in the bilevel case the separation comes from the problem definition. This implies that for bilevel problems it can be rather challenging to feedback strong cuts (or constraints) from the subproblem to the master problem.

3 Basic properties of discrete bilevel problems

In this section we analyze basic properties of discrete bilevel problems. First, we give a closer look at the potential definitions of the follower's optimality condition. Then, we demonstrate that bilevel problems differ substantially from single level problems with a single or multiple objectives, whereas they are more related to quantified constraint satisfaction problems. Finally, we investigate how the bilevel problem can be relaxed to a single level problem, and address the computational complexity of bilevel problems.

3.1 On the optimistic and pessimistic cases

The optimistic and pessimistic formulations of the bilevel problem, shown in formulae (1-3) and (4-6), respectively, capture two different standpoints of the leader. This difference is relevant in problems where the follower can have several optimal solutions, and these differ essentially from the viewpoint of the leader: only some of them satisfy C or they incur different costs f. The *optimistic* formulation assumes that the leader is allowed to choose one from the follower's optimal solutions, or, equivalently, the follower is friendly enough to choose an optimal response that satisfies C and minimizes f, if there exists one.

In contrast, in the *pessimistic* case, the leader wishes to safeguard against the risks of an unfavorable follower response by assuming that the follower selects its optimal response that is the least favorable for the leader. There are two possible interpretations of the pessimistic formulation (4-6) used in the literature. Both approaches consider a response from the follower's optimal set that maximizes f. However, the first interpretation allows the follower to choose a response that violates C [35], whereas the second interpretation assumes that the follower must select a response satisfying C, if there exists one [13]. In the first case, which we call the *hard pessimistic* formulation, the leader must select values for X in such a way that *all* optimal solutions of the follower satisfies C. This is not necessary in the second, so-called *soft pessimistic* formulation. It must be noted that in most of the existing applications of bilevel programming the constraint set C is empty, and therefore the two pessimistic cases are equivalent.

In the core of this paper, we focus on the optimistic case. At the same time, we note that the similar techniques can be used for the pessimistic case. The necessary, minor changes in the algorithmic details will be discussed in Section 4.5.

3.2 Bilevel versus single level problems

Below we demonstrate the difference of the single level and the bilevel problems on an instance of our sample problem presented in Figure 1. In the single level case, the leader could accept all the four tasks and process them, e.g., in the order $(1, 2, 3, 4)$. In the bilevel case, the leader only chooses the tasks to process, but the follower sequences them. If the leader selects all tasks, then the follower's response is the solution of the corresponding $1|r_j|\sum_j w_j^2 C_j$ problem, i.e., the sequence $(4, 3, 2, 1)$. This solution is infeasible, because task 1 violates its deadline. In fact, the optimal bilevel solution is selecting the tasks $\{1, 2, 3\}$, and processing them in the order $(1, 3, 2)$, which respects all deadlines. An interesting, seemingly paradoxical situation is that the strictly smaller set of tasks $\{1, 2\}$ cannot be scheduled, because the follower's response, $(2, 1)$, violates the deadline of task 1. This also warns us that inference methods that work for the single level case might not generalize to the bilevel problem.

Task j	p_j	r_j	d_j	w_j^1	w_j^2
1	1	0	1	2	1
2	2	0	100	2	4
3	1	1	100	2	20
4	1	0	100	1	5

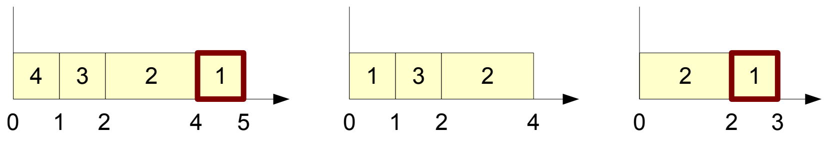


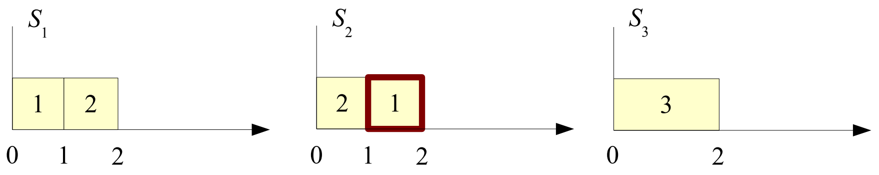
Figure 1: A bilevel problem instance and the follower's response for various choices of the leader. The tasks marked with a thick frame in the schedules violate their deadlines.

3.3 Bilevel versus bicriteria approaches

Although both bilevel programming and single level bicriteria approaches seek for solutions that are attractive w.r.t. two different objective functions, the two approaches differ essentially. They model two different situations: bicriteria optimization looks for the best compromise in a centralized way, while bilevel optimization follows a simple, hierarchical protocol with two autonomous partners, each interested in optimizing its own objective value. Indeed, the optimal solution of the bilevel problem might not be Pareto optimal for the corresponding single level bicriteria problem, and vice versa. Below we illustrate this phenomenon on our sample problem.

Consider the problem instance presented in Figure 2. The candidate task sets to be scheduled are $\{1\}$, $\{2\}$, $\{3\}$, and $\{1, 2\}$, and it is easy to see that no other task set can be scheduled to meet the deadlines. A Pareto optimal schedule is $(1, 2)$, denoted by S_1, which leads to objective values $f = 3$ and $g = 7$. Observe that schedule S_1 is not a feasible solution of the bilevel problem: if the leader decided to accept tasks $\{1, 2\}$, then the follower would sequence these according to $(2, 1)$, resulting in schedule S_2. However, S_2 violates the leader's deadline constraint on task 1, and hence, it is not a feasible solution of the bilevel problem. In fact, the optimal bilevel solution is the schedule (3), called S_3, which has $f = 4$ and $g = 20$. The leader prefers S_3 to $\{1\}$ and $\{2\}$ as

Task j	p_j	r_j	d_j	w_j^1	w_j^2
1	1	0	1	2	1
2	1	0	2	2	3
3	2	0	2	3	10



Schedule	f	g	Feasibility	Optimality
S_1	3	7	Feasible	Pareto optimal
S_2	3	5	Infeasible, task 1 violates deadline	-
S_3	4	20	Feasible	Bilevel optimal

Figure 2: Difference of the bilevel and the bicriteria Pareto optimal solutions. The figure presents a problem instance and its three different solutions.

well. Note that S_1 Pareto dominates S_3, which means that the bilevel optimal solution is Pareto dominated.

3.4 Bilevel programming versus QCSP

As it has been described above, the main difference between QCSP and bilevel programming is that in QCSP, one wishes to find a strategy that covers all possible actions of the adversary, whereas in bilevel programming we assume that the follower will act rationally according to its known objectives. Now we show that bilevel programs can be translated into a QCOP+ with a single pair of quantifiers $\exists\diamond\forall\diamond$ and vice versa. We assume that the function symbols f and g and the relation $\leq$ is available in the constraint language.

First, note that the optimistic bilevel problem corresponds to the QCOP+

$$\min_{X,Y}\{f(X,Y) \mid C(X,Y) \ \wedge \ D(X,Y) \ \wedge \\ \forall Y'[D(X,Y')] \ g(X,Y) \leq g(X,Y')\}.$$

Here, the first line of the formula describes that $\langle X,Y\rangle$ is a feasible solution, while the second line states that $\langle X,Y\rangle$ is an optimal response of the follower, because all the alternative responses Y' would result in a greater or equal value of g. Furthermore, the hard pessimistic bilevel problem can be rewritten as

$$\begin{aligned}\min_{X,Y}\{f(X,Y) \mid C(X,Y) \ \wedge \ D(X,Y) \wedge \\ &\forall Y'[D(X,Y')]\ g(X,Y) \leq g(X,Y')\wedge \\ &\forall Y''[D(X,Y'') \wedge g(X,Y) = g(X,Y'')] \\ &\qquad C(X,Y'') \wedge f(X,Y) \geq f(X,Y'')\}.\end{aligned}$$

Similarly to the optimistic case, the first and second lines describe that the solution $\langle X, Y\rangle$ is feasible and optimal for the follower. The third and fourth lines encode the hard pessimistic assumption, i.e., that all the optimal responses of the follower Y'' must satisfy C and result in a value of f not worse than $f(X,Y)$. Note that the QCOP+ equivalent of a soft pessimistic bilevel program can be derived from the above formula by omitting $C(X,Y'')$ from the last line.

Although translation to QCOP+ is a theoretically sound approach to solving bilevel problems, it can be rather inefficient. The main deficiency of the approach is that the computed strategy must cover all possible decisions of the follower explicitly. To verify these claims, we have implemented our sample problem in QeCode. Experimental results are presented in Section 6.

3.5 The single level relaxation

Various components of the solution algorithms for bilevel problems rely on well understood techniques for single level problems. Therefore, it seems natural to look for relations between bilevel and single level problems. The simplest way of reduction is to let the leader decide on every variable, and completely disregard the existence of the follower. The resulting problem will be called the *single level relaxation* of the bilevel problem, and its solution value is obviously a lower bound on the bilevel solution cost:

Definition 1 *The single level relaxation of a bilevel program is, using the set of all variables $X' = (X,Y)$, the problem* $\min\{f(X') \mid C(X') \wedge D(X')\}$.

3.6 Computational complexity

Bilevel problems are complex optimization problems, they often belong to a higher complexity class than their corresponding single level relaxations. For example, linear bilevel problems (where both the single level relaxation and the follower's subproblem is a linear program) are known to be NP-complete [13]. Here, we focus on the complexity of decision versions of discrete bilevel problems, especially in the case where the (decision version of the) single level relaxation is NP-hard. It is easy to observe that a bilevel problem is–except for degenerate cases–at least as complex as its single level relaxation, hence, NP-hard. On the other hand, discrete bilevel problems are in PSPACE, because all instantiations of the variables can be enumerated and evaluated in polynomial space using a recursive algorithm, similarly to the algorithm defined for solving quantified boolean formulae in [15].

Now, a discrete bilevel program may or may not belong to NP. It is easy to define discrete bilevel problems with an NP-compete follower's subproblem that are outside NP. Consider an unconventional, but valid bilevel scheduling problem where all variables belong to the follower. Both the leader and the follower aim at sequencing the set of tasks on a single machine subject to release times r_j and strict deadlines d_j. However, the leader would be interested in minimizing $\sum C_j$, whereas the follower minimizes the weighted earliness penalty $\sum w_j(d_j - C_j)$. This problem is NP-hard, because with follower weights $w_j \equiv 0$, all solutions are equivalent for the follower, and hence, the optimistic bilevel problem corresponds to the classical $1|r_j, d_j|\sum C_j$ problem, which is NP-hard [25]. On the other hand, it is also co-NP-hard, because with follower weights $w_j \equiv 1$ the two criteria are the negatives of each other. Hence, verifying the feasibility of a bilevel solution is equivalent to proving that no better solution exists for the follower's NP-hard subproblem, $1|r_j, d_j|\sum w_j(d_j - C_j)$. Since the bilevel problem is both NP-hard and co-NP-hard, it is outside NP (unless P=NP).

The above complexity results indicate that no direct encoding of discrete bilevel problems into CP or MIP can be expected. For the case of our main sample problem, we were not able to prove that it is outside NP, but we conjecture that it is, since no trivial certificate seems to exist for a positive answer.

4 Modeling and solving bilevel problems by CP

In this section we first present a generic approach to solving discrete bilevel optimization problems by CP. Then, we introduce several algorithmic techniques to improve the efficiency of the solver. Each of the subsections has a counterpart in the Section 5, where the use of the given technique is illustrated on the sample scheduling problem. Unless stated otherwise, we consider the optimistic bilevel problem.

We use the notation $\text{Dom}(X)$ for the current domain of a CP variable Z. Furthermore, the minimum and maximum values in $\text{Dom}(X)$ will be denoted by $\check{Z}$ and $\hat{Z}$, respectively.

4.1 The basic constraint model

Given the discrete optimistic bilevel problem as described in formulae (1-3), let us define an equivalent constraint program that encodes the problem from the leader's point of view. We also call it the *master problem.* The decision variables are both X and Y. They are subject to constraints C and C_{Opt}, where C is the set of the leader's constraints, and C_{Opt} describes the follower's optimality condition:

$$\min_{X,Y}\{\ f(X,Y) \mid C \wedge C_{\text{Opt}}\},$$

where

$$C_{\text{Opt}}: \quad Y \in \arg\min_{Y'}\{g(X,Y') \mid D(X,Y')\}.$$

The optimization problem contained in C_{Opt} is called the *follower's subproblem.* We assume that C is a set of classical constraints over finite-domain variables, which have appropriate propagation algorithms defined in the literature. In contrast, in the generic case, constraint C_{Opt} contains the parametric version of an NP-hard discrete optimization problem. Hence, there is little hope that algorithms readily available in the literature can be applied to propagate it, or generalized arc-consistency can be achieved efficiently. Therefore, we propose to settle for a generate-and-test approach for propagating C_{Opt}, i.e., to propagate only when all of the leader's variables X become bound. The pseudo-code of the propagation algorithm is presented in Figure 3. The algorithm first determines the follower's minimum cost, g^* (line 3), or returns the symbol `'no_solution'` if no feasible solution exists. Then, it computes the follower's response according to the optimistic assumptions, Y^+ (line 6). Both of these steps require solving the follower's subproblem with known parameters X. Exact solution approach, e.g., CP search must be used, since the bilevel problem formulation requires finding exact optimum. When solving the follower's subproblem in lines 3 and 6, the domains of Y in the master problem must be ignored, since those domains are corrupted by the propagators of C, i.e., constraints that the follower disregards.

```
PROCEDURE Propagate_C_Opt()
1   IF $X$ is not bound
2     RETURN
3   LET $g^* := \min_{Y'}\{g(X,Y') \mid D(X,Y')\}$
4   IF $g^* =$ 'no_solution'
5     Fail
6   LET $Y^+ := \arg\min_{Y'}\{f(X,Y') \mid g(X,Y') = g^* \wedge C(X,Y') \wedge D(X,Y')\}$
7   IF $Y^+ =$ 'no_solution'
8     Fail
9   Instantiate $Y \leftarrow Y^+$
```

Figure 3: Algorithm for propagating constraint C_{Opt}.

Regarding *search techniques* for the master problem, we propose to perform any kind of search, exact or non-exact, in the space of the instantiations of the leader's variables. It is not necessary to consider the follower's variables, since values will be assigned to them by constraint C_{Opt}.

4.2 Lifting the follower's constraints and dominance rules into the master problem

The above basic CP formulation can be strengthened by adding redundant constraints that propagate even when a part of the variables X is not bound. First, observe that the constraint set D can be added to the basic model, because it reduces the search space to values of X that have at least one feasible follower response. Furthermore, assume that there are weak dominance rules known for the follower's sub-problem, i.e., properties that *all* optimal solutions of the sub-problem must satisfy. Then, the conjunct of these rules can be encoded into a constraint C_{Dom}, and added to the CP model as a redundant constraint. Hence, the following CP model is a sound representation of the bilevel problem, and it leads to stronger propagation than the basic model:

$$\min_{X,Y}\{\ f(X,Y) \mid C \wedge D \wedge C_{\text{Opt}} \wedge C_{\text{Dom}}\}.$$

4.3 Bounds on the follower's cost

Below we present a novel technique that prunes the search tree based on the difference of the constraint sets that the leader and the follower must satisfy. We will characterize the values that g can take in solutions that are feasible for the leader, as well as the values that g can take in optimal responses of the follower. Clearly, if the two ranges do not overlap, then there is no feasible bilevel solution in the current branch of the search tree.

Let UB denote the value of the best known solution, and let us characterize the current branch of the search tree by the domain of X, denoted by $\text{Dom}(X)$. Any feasible improving solution of the master problem must obey C, D, and $f < UB$. Hence, a valid lower bound g^{Lmin} on the values g in the solutions that are acceptable for the leader in the current branch is:

$$g^{Lmin} = \min_{X' \in \text{Dom}(X)} \min_{Y}\{g(X',Y) \mid \\ C(X',Y) \ \wedge\ D(X',Y) \ \wedge\ f(X',Y) < UB\}.$$

On the other hand, for any fixed leader's choice in this branch, the follower will return a response that minimizes g subject to D. By taking the maximum of these minimum values, we get an upper bound g^{Fmax} on g in the solutions that are acceptable for the follower in the current branch:

$$g^{Fmax} = \max_{X' \in \text{Dom}(X)} \min_{Y}\{g(X',Y) \mid D(X',Y)\}.$$

Note that the constraints in the definition of g^{Fmax} are a subset of the constraints for g^{Lmin}, and therefore $g^{Fmax} < g^{Lmin}$ can occur. This means that no solution in the current branch of the search tree is both feasible for the leader and optimal for the follower. At the same time $g^{Fmax} \geq g^{Lmin}$ is also possible, since g^{Fmax} is a maximin, whereas g^{Lmin} is a minimum.

Lemma 1 *If $g^{Fmax} < g^{Lmin}$, then the current search branch contains no feasible improving bilevel solution, and therefore it can be fathomed.*

In general, it is difficult to compute the exact values of g^{Fmax} and g^{Lmin}. Instead, an upper estimate of g^{Fmax}, denote by $\hat{g}^{Fmax}$ can be used, and similarly, a lower estimate of g^{Lmin}, denoted by $\check{g}^{Lmin}$ can be applied.

Finally, note that the application of the above bounds makes sense in the leaves of the search tree as well, since they may prove the infeasibility of the leaf faster than solving the follower's subproblem in an exact way.

4.4 Lower bounds on the leader's cost

In theory, it is straightforward to apply the classical lower bounding technique of operations research to bilevel problems: let $\check{f}$ be a lower bound and $\hat{f}$ an upper bound, typically the value of the best known solution. Now, if $\check{f} \geq \hat{f}$ then the current branch of the search tree does not contain a feasible improving solution. In practice, the effective use of this technique is challenging, because good lower bounds for bilevel problems are rarely available from the literature. A possible approach is using the single level relaxation, whose solution imposes a lower bound on the bilevel problem. If the single level relaxation is still intractable, then it can be relaxed further. We note that the value of the single level relaxation is often far from the optimal solution of the bilevel problem.

4.5 Extension to the pessimistic case

The models and algorithms for the optimistic case can be extended easily to the pessimistic case. The extension requires modifying the propagator of C_{Opt}. In the soft pessimistic case, the function *min* in line 6 of Figure 3 must be replaced with *max* as follows:

```
6    LET $Y^+ := \arg\max_{Y'}\{f(X,Y') \mid g(X,Y') = g^* \wedge C(X,Y') \wedge D(X,Y')\}$
```

In the hard pessimistic case, in addition to the above change, the following lines must be inserted after line 5 (outside the IF-THEN branch of lines 4-5) to ensure that the follower does not have an optimal solution that violates C:

```
5a   IF there exists an $Y'$ such that $\{g(X,Y') = g^* \wedge \bar{C}(X,Y') \wedge D(X,Y')\}$
5b     Fail
```

In line 5a, the expression $\bar{C}$ stands for the negation of C, i.e., $\bar{C} = \vee_{c \in C} \neg c$. Hence, $\bar{C}$ corresponds to a reified constraint. We note that the use of reified constraints in $\bar{C}$ makes the CP approach substantially less efficient for hard pessimistic problems, and may limit the applicable constraints depending on the solver used. The enhancements presented in Sections 4.2-4.4 can be applied without any change.

5 Modeling and solving the scheduling problem

5.1 The basic constraint model

The basic constraint model of our scheduling problem contains n binary variables x_j to denote if task j is scheduled, and n optional activities with start and end time variables. The activities are subject to a unary resource and time window constraints, and the follower's optimality constraint. The objective function is expressed as $f = \sum_j w_j^1(1 - x_j)$ using a weighted sum constraint. Our search strategy selects in each node the task j whose x_j variable is unbound and has the greatest w_j^1. Then, it creates two children of the node according to $x_j = 1$ (left branch) or $x_j = 0$ (right branch).

The follower's optimality constraint is a custom developed constraint, which embeds a constraint-based solver for the $1|r_j|\sum w_j^2 C_j$ problem. The naive constraint model with a unary resource constraint, release time constraint, and the cost expressed using a weighted sum constraint is used. A classical chronological schedule-or-postpone search strategy (called *setTimes* in Ilog) is used, and the subproblem solver also includes dominance rules from [20].

5.2 Lifting the follower's dominance rules into the master problem

A number of efficient dominance rules are known for the $1|r_j|\sum w_j^2 C_j$ problem, see, e.g., [20]. However, the condition side of most of these rules is too complex to fire when only the x_j variables are bound, and very little is known about the task start times or the order. We lifted the following simple dominance rule to the master problem. Without loss of generality we can assume that tasks are indexed in the *weighted shortest processing time* order (WSPT, non-increasing w_j^2/p_j), with ties broken by *earliest due date* (EDD, non-decreasing d_j).

Lemma 2 *For any fixed leader's choice, i.e., assignment of the variables x_j, there exists an optimal follower's response according to the optimistic bilevel assumption such the tasks that start after $r_{\max} = \max_{\{j|x_j=1\}} r_j$ are ordered by the above defined task index.*

Proof: The proof essentially matches the proofs of the optimality of the WSPT order for the $1||\sum_j w_j C_j$ problem and the EDD order for the feasibility problem subject to deadlines, but with uniform release times. Let j and k be two subsequent tasks, both starting after $r_{\max}$ in any feasible bilevel solution, i.e., in a schedule that is both optimal for the follower and feasible for the leader. Now, $w_j^2/p_j \geq w_k^2/p_k$ holds, because otherwise the schedule would be suboptimal for the follower: swapping j and k would decrease $\sum_j w_j^2 C_j$. Furthermore, if $w_j^2/p_j = w_k^2/p_k$ and $d_j > d_k$, then swapping j and k preserves the optimality of the follower's response and does not cause infeasibility for the leader. Repeating the swaps until no further such task pairs exists results in an optimal feasible response according to the optimistic bilevel assumption. □

```
PROCEDURE Propagate_Dominance()
1  LET $r_{\max} := \max_{\{j|1\in\mathrm{Dom}(x_j)\}} r_j$
2  $s := 0$
3  FORALL $j \in \{1,...,n\}$ IN INCREASING ORDER
4    IF $x_j$ is bound to 1 AND $\check{S}_j \geq r_{\max}$
5      Update $\check{S}'_j := \max(\check{S}_j, s)$
6      $s := \check{S}'_j + p_j$
7    ELSE IF $s > \hat{S}_j$
8      Update $\hat{S}'_j := \max(r_{\max} - 1, r_j)$
9  $e := \infty$
10 FORALL $j \in \{1,...,n\}$ IN DECREASING ORDER
11   IF $x_j$ is bound to 1 AND $\check{S}_j \geq r_{\max}$
12     Update $\hat{C}'_j := \min(\hat{C}_j, e)$
13     $e := \hat{C}'_j - p_j$
14   ELSE IF $e < \check{C}_j$
15     Update $\hat{S}'_j := \max(r_{\max} - 1, r_j)$
```

Figure 4: Algorithm for propagating the dominance rule.

In theory, it would be possible to encode the above dominance rule directly into $\frac{1}{2}n(n-1)$ reified constraints, i.e., one constraint for each pair of tasks. However, to achieve more efficient propagation, we implemented a new algorithm for propagating the above dominance rule as a single global constraint. The algorithm, displayed in Figure 4, tightens the bounds of the domains of start and end time variables. Recall that the minimum and maximum start (end) times are denoted by $\check{S}_j$ and $\hat{S}_j$ ($\check{C}_j$ and $\hat{C}_j$), respectively. Tightening the bounds of a start time variable S_j means updating its initial domain of $[\check{S}_j, \hat{S}_j]$ to $[\check{S}'_j, \hat{S}'_j]$, where $\check{S}'_j \geq \check{S}_j$ and $\hat{S}'_j \leq \hat{S}_j$ hold. The constraint requires an initialization step, which sorts the tasks according to the above defined order in $O(n \log n)$ time. Then, each propagation run takes $O(n)$ time. The algorithm first computes an upper bound of $r_{\max}$ (line 1). Then, it considers the tasks j in the above order, and maintains an earliest start time s of the task under consideration (lines 3-8). If j must be scheduled after $r_{\max}$, then it must start after s (lines 4-6). Otherwise, if j cannot start after s, then it cannot be scheduled after $r_{\max}$ (lines 7-8). In the latter case, one of the following conditions hold: either j is not scheduled, in which case $\hat{S}'_j$ can be set to r_j; or j starts strictly before $r_{\max}$. Overall, $\hat{S}'_j$ can be set to $\max(r_{\max} - 1, r_j)$. Finally, the algorithm repeats the same type of inference for the latest finish times, considering the tasks in the reverse order (lines 10-15).

5.3 Bounds on the follower's cost

5.3.1 The upper bound

Our follower's subproblem is $1|r_j|\sum w_j^2 C_j$. Let $g(T_1)$ denote the follower's minimal cost when the leader accepts the task set T_1. It is obvious that if $T_1 \subseteq T_2$ then $g(T_1) \leq g(T_2)$. Therefore, the cost of any heuristic solution to the follower's problem with task set $T_{max} = \{j \mid 1 \in \mathrm{Dom}(x_j)\}$ can be used as $\hat{g}^{Fmax}$. In our solver, we have implemented the constructive heuristic called *CPRTWT* with the *makeBetter* improvement step after the insertion of each task to the schedule, originally introduced in [21].

5.3.2 The lower bound

Computing $\check{g}^{Lmin}$ requires obtaining a lower bound on a $\sum w_j^2 C_j$ problem subject to release times, deadlines, and optional activities. Our lower bound (LB) is based on the model of [30] for a similar problem, though, without optional activities:

$$
\begin{array}{rll}
\displaystyle\min_{z_j, C_j} & \displaystyle\sum_j w_j^2 C_j & (7) \\
\text{subject to} & & \\
\forall j & (r'_j + p_j) z_j \leq C_j & (8) \\
\forall j & C_j \leq d'_j z_j & (9) \\
\forall i, j \ (i \neq j) & (C_i \leq C_j - p_j z_j) \ \vee \ (C_j \leq C_i - p_i z_i) & (10) \\
 & \displaystyle\sum_j w_j^1 z_j \geq W & (11) \\
\forall j & z_j \in \mathrm{Dom}(x_j) & (12)
\end{array}
$$

In this formulation, variables z_j indicate if task j is processed ($z_j = 1$) or not ($z_j = 0$). Variables C_j denote the completion times. Constraints (8) and (9) specify the release times and deadlines of the task, and also ensure that C_j is 0 if j is not scheduled. Note that the time windows taken from the CP model can be used, since these are strengthened by propagation compared to the original values. Line (10) defines the unary resource constraint. Constraint (11) states that the total weight of the tasks selected for processing must be at least $W = \sum_j w_j^1 - UB + 1$ in order to achieve an improving solution for the leader. In a given search node, it can already be known for some tasks if they are already selected for processing by the leader or not, while it is still an open question for the rest (12). Note that $\mathrm{Dom}(x_j) \subseteq \{0, 1\}$. Now, by dualizing (8) and (9) with multipliers a and b, we receive the following Lagrangian relaxation (LR):

$$
\begin{aligned}
\min_{z_j, C_j} \quad & \sum_j w_j^2 C_j \; + \; \sum_j [a_j((r'_j + p_j)z_j - C_j) + b_j(C_j - d'_j z_j)] \\
= \quad & \sum_j (w_j^2 - a_j + b_j) C_j \; + \; \sum_j [a_j(r'_j + p_j) - b_j d'_j] z_j \qquad (13)
\end{aligned}
$$

subject to

$$
\begin{aligned}
\forall i, j \; (i \neq j) \quad & (C_i \leq C_j - p_j z_j) \; \vee \; (C_j \leq C_i - p_i z_i) \qquad (14) \\
& \sum_j w_j^1 z_j \geq W \qquad (15) \\
\forall j \quad & z_j \in \mathrm{Dom}(x_j) \qquad (16)
\end{aligned}
$$

Next, we present how the LR problem can be solved to optimality for fixed non-negative Lagrangian multipliers a and b. We exploit that the first component of the objective function corresponds to $\sum_{\{j \mid z_j = 1\}} w'_j C_j$ with $w'_j = w_j^2 - a_j + b_j$, while the second component does not contain completion time variables C_j. Therefore, for any fixed z, the optimal solution is a no-delay schedule containing the selected tasks in WSPT order.

Computing the LB in leaves of the search tree We have developed two different methods for computing the optimal solution of LR: one to be used in the leaves of the search tree, and another for internal search nodes. In leaves we exploit that there are no optional tasks, i.e., the variables z_j are fixed. In this case, $\breve{g}^{Lmin}$ equals the objective value of the WSPT schedule, which can be computed in $O(n \log n)$ time.

Computing the LB in internal search nodes In internal search nodes, the LR problem with a fixed choice of multipliers a and b can be solved by the following dynamic program (DP). As an initialization step, we sort the tasks j that may be scheduled ($1 \in \mathrm{Dom}(x_j)$) by non-increasing w'_j/p_j, which corresponds to the WSPT order according to the modified weights w'_j. Tasks that cannot be scheduled ($1 \notin \mathrm{Dom}(x_j)$) are completely ignored by the algorithm.

The DP fills in a 3 dimensional table whose cells are indexed by parameters k, v, and t. The content of each cell characterizes the optimal solution of a subproblem of LR, received by applying the following restrictions to LR:

- The task selected for processing are a subset of $\{1, ..., k\}$;
- The total leader's weight of the selected task must be equal to v;
- The schedule must end at time t.

The optimal schedule in cell (k, v, t) will be denoted by $\sigma(k, v, t)$, and its cost by $u(k, v, t)$. It is sufficient to store the cost $u(k, v, t)$ in the table, while we use $\sigma(k, v, t)$ only to show the correctness of the algorithm below. For combinations of parameters k, v, and t that do not lead to a feasible schedule, we consider $u(k, v, t) = \infty$.

The DP fills in the table layer-by-layer, using induction over the different values of k. The first layer ($k = 1$) contains two finite values only, which characterize the two trivial solutions: $u(1, w_1^1, p_1) = w_1^2 p_1$ for a schedule that contains task 1 only, and $u(1, 0, 0) = 0$ for the empty schedule.

Each of the values $u(k, v, t)$ in the subsequent layers, $k \geq 2$, can be derived in one of the following two ways. If task k is contained in schedule $\sigma(k, v, t)$, then k is the last task in this schedule due to the optimality of the WSPT order. Consequently, $\sigma(k, v, t)$ is the concatenation of the optimal schedule for the subproblem without task k, $\sigma(k - 1, v - w_k^1, t - p_k)$, and task k itself. Alternatively, if task k is not contained in $\sigma(k, v, t)$, then $\sigma(k, v, t)$ is identical to the schedule computed in the previous layer, $\sigma(k - 1, v, t)$. From these two candidate schedules, the one that leads to the lowest cost is selected. Hence, the value of $u(k, v, t)$ can be computed as:

$$u(k, v, t) = \min(u(k - 1, v - w_k^1, t - p_k) + t w_k^2,\ u(k - 1, v, t))$$

Note that a cell corresponds to a feasible solution of LR if and only if it has $v \geq W$, i.e., it satisfies constraint (15). All other constraints of LR are respected by all cells by definition. Hence, the optimal solution of LR can be retrieved from the last layer of the table, denoted as layer $k_{\max}$, as follows:

$$\check{g}^{Lmin} = \min_{v,t} \{u(k_{\max}, v, t) | v \geq W\}$$

The DP runs in pseudo-polynomial time and space: its complexity is $O(nVP)$, where $V = \sum_j w_j^1$ and $P = \sum_j p_j$.

Setting the Lagrangian multipliers The above methods result in an optimal solution of LR for any fixed non-negative multipliers a and b, assuming $w'_j = w_j^2 - a_j + b_j \geq 0$. To find the multipliers that provide the strongest LB, we embedded the above methods into a loop, and adjusted the multipliers after each cycle as follows. If task j violates its release time in the current optimal solution of LR, then its weight is decreased in order to move it later in the WSPT order. Namely, we set $w'_j = \frac{w'_k p_j}{p_k} - \varepsilon$, where k is the successor task of j in the schedule. Similarly, if j violates its deadline, then its weight is increased to $w'_j = \frac{w'_k p_j}{p_k} + \varepsilon$, where k is the predecessor of j. If task j respects both its release time and deadline then its weight is not changed. Note that tasks cannot violate their release time and deadline at the same time. The method was initialized with $a_j = b_j = 0$.

The best run times were achieved with the number of cycles set to 15 in leaves, and not using this method in internal search nodes (c.f. the experimental results for further details).

5.4 Lower bounds

The single level relaxation (SLR) of our problem is the $1|r_j|\sum w_j^1 U_j$ scheduling problem, where the optimization criterion $\sum w_j^1 U_j$ stands for the weighted number of late tasks. This problem is NP-complete. Nevertheless, various solution techniques and polynomial lower bounds are available from the literature. The current best algorithm for the SLR is the branch-and-bound of [28]. We have implemented the mixed-integer programming (MIP) formulation of the single level problem proposed in this paper, and solved its linear relaxation in each node of the search tree. The parameters r_j and d_j were updated in each node by the tighter time windows taken from the CP model.

The gap between the lower bound and the bilevel solution originates from two sources: solving the SLR instead of the bilevel problem, and the further linear relaxation of the SLR. In preliminary experiment we have found that over 75% of the gap is due to taking the SLR, and only 25% originates from solving the linear relaxation. Overall, this lower bound was not sufficiently tight to be used for pruning the search tree efficiently.

5.5 An enhanced propagator for $C_{\mathbf{Opt}}$

A basic propagator for the follower's optimality constraint C_{Opt} can be built based on the generic scheme presented in Section 4.1. Below we present an enhanced algorithm that fully exploits the follower's lower and upper bounds during the exact solution of the follower's subproblem. This algorithm can be applied in bilevel problems where the leader's objective, f, does not depend on the follower's response.

The pseudo-code of the algorithm is shown in Figure 5. In lines (3-5), the algorithm checks if the computed bounds on the follower's cost allow the existence of a solution. Afterwards, it solves the follower's subproblem *with* the leader's deadline constraints, resulting in solution Y^+ and cost g^+ (line 6). Then, the follower's subproblem is solved *without* the leader's deadline constraints, leading to cost g^* (line 10). Search can be aborted when a solution with $g^* < g^+$ is reached, because this solution, as well as any potential improving solution, will lead to failure in lines (11-12) anyway. The solution Y^+ is optimal for the leader if and only if $g^+ = g^*$ (note that $g^+ \geq g^*$ always holds). Observe that the order of lines (6) and (10) is reversed w.r.t. the basic version of the propagator, which is advantageous because the problem faced in line (6) is tighter and generally easier-to-solve than the problem of line (10). This latter step exploits that f does not depend on the follower's response.

6 Computational experiments

6.1 Experimentation of the proposed solution techniques

In this section we report computational results achieved on the sample scheduling problem. The solver was implemented in such a way that each inference

```
PROCEDURE Propagate_C_Opt2()
1  IF $X$ is not bound
2    RETURN
3  Compute $\hat{g}^{Fmax}$ and $\check{g}^{Lmin}$
4  IF $\hat{g}^{Fmax} < \check{g}^{Lmin}$
5    Fail
6  LET $Y^+ := \arg\min_{Y'}\{g(X,Y') \mid C(X,Y') \wedge D(X,Y') \wedge g(X,Y') <= \hat{g}^{Fmax}\}$
7  IF $Y^+ =$ 'no_solution'
8    Fail
9  LET $g^+ := g(X,Y^+)$
10 LET $g^* := \min_{Y'}\{g(X,Y') \mid D(X,Y') \wedge g(X,Y') <= \hat{g}^{Fmax}\}$
11 IF $g^+ > g^*$
12   Fail
13 Instantiate $Y \leftarrow Y^+$
```

Figure 5: Enhanced algorithm for propagating constraint C_{Opt}.

technique presented in a separate section above could be switched on or off individually. Moreover, the bounds on the follower's cost (see Section 5.3) could be computed independently in the internal search nodes of the master problem using the DP, or in the leaves using the WSPT schedule. Preliminary experiments showed that all the presented techniques contribute to pruning the search tree, but it does not pay off in terms of search time to use the leader's lower bound or to compare the follower's bounds in internal search nodes. Therefore we decided to switch off these two components in the main experiments, even in solver version V1 that we will consider the complete solver in the sequel. Further versions, V2 and V3, were created by gradually switching of the introduced inference techniques. The last version, V4, used only the basic constraint model of Section 5.1 without any enhancements. The techniques used in the different versions are summarized in Table 1.

	V1	V2	V3	V4
Basic constraint model (Section 5.1)	+	+	+	+
Enhanced propagator for C_{Opt} (Section 5.5)	+	+	+	
Follower's dominance rules in master problem (Section 5.2)	+	+		
Bounds on the follower's cost, leaves only (Section 5.3)	+			

Table 1: Comparison of the tested versions of the solver.

The solver was implemented in C++ using ILOG Solver and Scheduler, both for the bilevel master problem and the follower's subproblem solver. ILOG Cplex was used for the computation of the LP lower bound. The experiments were run on a 1.86 GHz Intel Xeon computer with 2 GB of RAM under Windows Server 2003. The time limit was set to 600 seconds per problem instance.

Problem instances have been generated similarly to the instances for the single level problem of minimizing the weighted number of late jobs in [11] and [28], with the only difference that we have also added the follower's weights w_j^2. The parameters of the generator are the number of tasks n, the range of release times, k_R (a larger value means a greater variance of the release times), and the tightness of the deadlines, k_D (the larger the value, the wider the time windows). Parameter n varied between 20 and 50 with increments of 5, while k_R and k_D were chosen from the set $\{1, 5, 10, 20\}$. Generating 10 instances with all possible combinations of the 3 parameters resulted in 1120 instances altogether. Processing times were generated using $U[1, 100]$, release times from $U[0, k_R n]$, deadlines from $U[r_j + p_j, r_j + p_j + k_D n]$, while weights w_j^1 and w_j^2 from $U[1, 10]$, where $U[a, b]$ denotes the discrete uniform distribution over integers from the interval $[a, b]$.

The comparison of the results achieved with the four solver versions is displayed in Table 2, while Table 3 provides further statistics about runs of the propagator of C_{Opt} in the complete version, V1. In both tables, each row contains combined result for the instances with a given value of N and k_D. Column *Opt* displays the number of instances that could be solved to proven optimality out of 40, while column *Best* shows the number of instances on which the solver found the best solution known for the instance. *Time* contains the average computation time in seconds or 600 for instances where the time limit was hit. Column *Nodes* shows the average number of search nodes. The additional columns in Table 3 contain the number of times the propagator of C_{Opt} reached the different steps of computation, as well as the total time of these computation steps: calculating the follower's bounds (*Step (1)*), the follower's minimum cost when the leader's constraints are respected (*Step (2)*), and the minimum cost when the leader's constraints are ignored (*Step (3)*). These steps corresponds to lines 3-5, 6-9, and 10-12 of the pseudo-code in Figure 5.

The results show that the stronger versions of the solver were able to solve instances with up to 20-25 tasks to optimality, whereas the naive version, V4, started to have difficulties even with some 20-task instances. On the whole, the complete version, V1, solved 6.6%, 9.9%, and 32.3% more instances to optimality than versions V2, V3, V4, respectively. The difference becomes slightly more significant as the problem size increases, and the comparison of average computation times brings roughly the same result. The two versions V1 and V2 (and the other two versions V3 and V4 likewise) generate the same number of search nodes for all instances that they could solve on time. This happened because the two versions differ only in the way of processing the leaves. For some larger problems it happened that the solvers did not find any solutions at all. This was the case, e.g., for parameters $N = 50$ and $k_D = 20$, where no feasible solution has been found by any solver version. Except for these cases, V1 always found the best solution among the four solver versions.

Smaller values of k_D made the problems easier to solve for all versions, because then the leader had a smaller choice of task sets to accept, and those sets are identified relatively efficiently without the follower's optimality condition, too. This is made apparent especially by the low number of calls to the propagator of C_{Opt} with $k_D = 1$ (column *Step (1)* in Table 3). The results also depend on k_R (small k_R makes them easier to solve), but much less than on k_D or N.

The analysis of the runs of the enhanced propagator in Table 3 shows that follower's bounds computation inferred the infeasibility of the leaf in 67% of the cases. The exact CP solver had to be called with the leader's constraints in the remaining 33% of the runs (*Step (2)*), and without the leader's constraint in only 0.5% of the runs (*Step (3)*). On the one hand, this low percentage is an excellent result, since the last step of the algorithm is the most time consuming. On the other hand, it also shows that at least 99.5% of the leaves did not contain a solution that is both feasible for the leader and optimal for the follower. Hence, future research should address the efficient propagation of the follower's optimality constraint C_{Opt} also in the internal search nodes. Overall, 89% of the total computation time was spent in the propagator of C_{Opt} in the leaves (23%, 62%, and 4% in steps (1), (2), and (3), respectively).

6.2 Results with translation to QCSP

To verify our contributions, we have implemented our scheduling problem in QeCode 2.0 using two different encodings to QCOP+. Our first QCOP+ model is based on the rewriting presented in Section 3.4, with one pair of existential and universal quantifiers. This model proved to be rather inefficient and memory consuming, since all possible decisions of the follower had to be enumerated explicitly in the computed strategy as possible values for the universally quantified variables. For this reason, instances with at most 4 tasks were trackable only, which is nearly an order of magnitude smaller than the instances we were interested in.

We have also implemented an alternative QCOP+ model with two existential quantifiers, and two different optimization criteria in the different quantifier scopes. The outer scope corresponds to the leader's choice, while the inner scope encodes the follower's subproblem. Note however that modeling tricks were required to overcome two shortcomings of the QCOP+ formalism. First, since QeCode does not allow to state a constraint set C different from D, we had to embed a measure of violations of C into f. Furthermore, to ensure that a solution according to the optimistic assumption is computed, we added $\varepsilon \cdot f$ as tiebreaker to g. Using this model, we were able to solve instances with at most 5 tasks of our scheduling problem.

The above results show that even the naive version, V4, of our bilevel solver outperforms the approach of translation to QCOP+. This occurs because the techniques proposed in this paper exploit the bilevel problem structure, which is typically not present in general QCSP problems. Hence, these results provide justification for research on specialized solution techniques for discrete bilevel problems.

7 Conclusions

This paper introduced novel CP-based modeling and solution techniques for discrete bilevel optimization problems. Since bilevel problems are computationally difficult–they are often outside NP–, techniques that improve the efficiency of the solver are of key importance. Hence, we have presented how classical techniques of operations research, such as dominance rules or lower bounds, can be applied to bilevel problems. New algorithms for propagating the follower's optimality constraint and computing bounds on the follower's cost were proposed. These techniques were illustrated on a bilevel scheduling problem and evaluated in computational experiments.

We think that an interesting direction for future research is the development of new inference techniques for discrete bilevel problems. Depending on the specific problem, these can include the filtering of the leader's variable domains based on inference from the follower's optimality condition, or the re-use of the follower's response computed in earlier visited leaves.

Acknowledgements

We thank the authors of QeCode for making their software freely available and for the discussions on QCSP. 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] Gecode: Generic constraint development environment, version 3.1, 2009. `http://www.gecode.org/`.

[2] QeCode: An open QCSP solver, version 2.0, 2009. `www.univ-orleans.fr/lifo/members/vautard/qecode`.

[3] P. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-based Scheduling.* Kluwer Academic Publishers, 2001.

[4] J. F. Bard. Coordination of a multidivisional organization through two levels of management. *Omega*, 11:457–468, 1983.

[5] M. Benedetti, A. Lallouet, and J. Vautard. QCSP made practical by virtue of restricted quantification. In *International Joint Conference on Artificial Intelligence*, pages 38–43, 2007.

[6] M. Benedetti, A. Lallouet, and J. Vautard. Modeling adversary scheduling with QCSP+. In *Proc. of the 2008 ACM symposium on Applied computing*, pages 151–155, 2008.

[7] M. Benedetti, A. Lallouet, and J. Vautard. Quantified constraint optimization. In *CP2008, Principles and Practice of Constraint Programming (Springer LNCS 5202)*, pages 463–477, 2008.

[8] G. G. Brown, W. M. Carlyle, J. Royset, and R. K. Wood. *On the Complexity of Delaying an Adversary's Project*, volume 29 of *Operations Research/Computer Science Interfaces*, chapter 1. Springer, 2005.

[9] K. N. Brown, J. Little, P. J. Creed, and E. C. Freuder. Adversarial constraint satisfaction by game-tree search. In *Proc. of ECAI 2004*, pages 151–155, 2004.

[10] P.A. Clark and A. Westerberg. Bilevel programming for steady-state chemical process design. I: Fundamentals and algorithms. *Computers and Chemical Engineering*, 14(1):87–97, 1990.

[11] S. Dauzère-Pérès and M. Sevaux. Using Lagrangean relaxation to minimize the weighted number of late jobs on a single machine. *Naval Research Logistics*, 50(3):273–288, 2003.

[12] S. Dempe. Discrete bilevel optimization problems. Technical report, Universität Leipzig, 2001.

[13] S. Dempe. *Foundations of Bilevel Programming.* Kluwer, 2002.

[14] D. Fanghänel and S. Dempe. Bilevel programming with discrete lower level problems. *Optimization: A Journal of Mathematical Programming and Operations Research*, 58(8):1029–1047, 2009.

[15] M.R. Garey and D.S. Johnson. *Computers and Intractability - A Guide to the Theory of NP-completeness.* Freeman, 1979.

[16] I. Gent, P. Nightingale, and K. Stergiou. QCSP-Solve: A solver for quantified constraint satisfaction problems. In *Proceedings of IJCAI-2005*, pages 138–143, 2005.

[17] I. P. Gent, P. Nightingale, A. Rowley, and K. Stergiou. Solving quantified constraint satisfaction problems. *Artificial Intelligence*, 172:738–771, 2008.

[18] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Operations Research*, 5:287–326, 1979.

[19] J. N. Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96:33–60, 2003.

[20] A. Jouglet, P. Baptiste, and J. Carlier. Branch-and-bound algorithms for total weighted tardiness. In *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter 13. Chapman & Hall / CRC, 2004.

[21] A. Jouglet, D. Savourey, J. Carlier, and P. Baptiste. Dominance-based heuristics for one-machine total cost scheduling problems. *European Journal of Operational Research*, 184:879–899, 2008.

[22] J. K. Karlof and W. Wang. Bilevel programming applied to the flow shop scheduling problem. *Computers and Operations Research*, 23(5):443–451, 1996.

[23] T. Kis and A. Kovács. On bilevel machine scheduling problems. *Submitted to OR Spektrum*, 2009.

[24] M. Labbé, P. Marcotte, and G. Savard. A bilevel model of taxation and its application to optimal highway pricing. *Management Science*, 44(12):1608–1622, 1998.

[25] J. K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[26] Z. Lukač, K. Šorić, and V. Vojvodić Rosenzweig. Production planning problem with sequence dependent setups as a bilevel programming problem. *European Journal of Operational Research*, 187:1504–1512, 2008.

[27] P. Marcotte, A. Mercier, G. Savard, and V. Verter. Toll policies for mitigating hazardous materials transport risk. *Transportation Science*, 43(2):228–243, 2009.

[28] R. M'Hallah and R.L. Bulfin. Minimizing the weighted number of tardy jobs on a single machine with release dates. *European Journal of Operational Research*, 176:727–744, 2007.

[29] P. Nightingale. Non-binary quantified CSP: Algorithms and modelling. *Constraints*, 14(4):539–581, 2009.

[30] Y. Pan and L. Shi. Dual constrained single machine sequencing to minimize total weighted completion time. *IEEE Transactions on Automation Science and Engineering*, 2(4):344–357, 2005.

[31] A. Prékopa. *Stochastic Programming.* Kluwer Academic Publishers, 1995.

[32] F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming.* Elsevier, 2006.

[33] H. D. Sherali, A. L. Soyster, and F. H. Murphy. Stackelberg-nash-cournot equilibria: Characterizations and computations. *Operations Research*, 31:253–276, 1983.

[34] H. Stackelberg. *Marktform and Gleichgewicht.* Julius Springer, Vienna, 1934.

[35] A. Tsoukalas, W. Wiesemann, and B. Rustem. Global optimisation of pessimistic bi-level problems. In P. M. Pardalos and T. F. Coleman, editors, *Lectures on Global Optimization*, pages 215–243. American Mathematical Society, 2009.

[36] G. Verger and C. Bessière. Blocksolve: a bottom-up approach for solving quantified CSPs. In *CP2006, Principles and Practice of Constraint Programming (Springer LNCS 4204)*, pages 635–649, 2006.

[37] B. von Stengel. Equilibrium computation for two-player games in strategic and extensive form. In N. Nisan, T. Roughgarden, É. Tardos, and V.V. Vazirani, editors, *Algorithmic Game Theory*, chapter 3. Cambridge University Press, 2007.

		V1				V2				V3				V4			
N	k_D	Opt	Best	Nodes	Time	Opt	Best	Nodes	Time	Opt	Best	Nodes	Time	Opt	Best	Nodes	Time
20	1	40	40	43	0.02	40	40	43	0.02	40	40	43	0.02	40	40	43	0.02
	5	40	40	240	0.05	40	40	240	0.14	40	40	309	0.18	40	40	309	0.29
	10	40	40	503	0.16	40	40	503	0.43	40	40	688	0.52	40	40	688	4.02
	20	40	40	4385	6.53	40	40	4385	20.06	40	40	11542	24.51	26	29	7681	270.53
25	1	40	40	97	0.02	40	40	97	0.03	40	40	97	0.03	40	40	97	0.03
	5	40	40	1670	0.43	40	40	1670	2.38	40	40	1942	2.56	40	40	1942	5.16
	10	40	40	6974	4.98	40	40	6974	20.24	40	40	10568	24.26	35	38	8607	142.98
	20	29	37	35507	230.72	21	24	18151	307.42	21	24	71127	330.87	6	9	13132	546.85
30	1	40	40	284	0.05	40	40	284	0.10	40	40	285	0.10	40	40	285	0.11
	5	38	39	30690	33.10	38	38	11624	41.45	38	38	13328	42.23	37	37	7276	70.56
	10	37	40	44316	83.90	31	33	38476	177.01	30	33	85007	203.11	11	15	25908	502.53
	20	13	24	133946	442.32	12	18	65989	453.52	9	12	228712	521.80	0	3	11701	600.00
35	1	40	40	701	0.12	40	40	701	0.24	40	40	706	0.24	40	40	706	0.27
	5	38	38	58298	43.34	38	38	38762	72.39	36	36	56082	95.30	28	30	21472	255.52
	10	26	35	229963	312.59	16	18	119492	396.13	15	17	251134	419.90	3	5	21570	575.97
	20	5	8	178155	561.73	2	4	70353	581.79	1	3	200364	593.90	0	0	1612	600.00
40	1	39	39	11571	15.21	39	39	3792	15.29	39	39	3814	15.28	39	39	3436	15.31
	5	34	36	276036	154.51	30	31	156687	246.98	28	30	222036	286.20	15	16	57854	467.86
	10	12	22	433949	512.41	10	15	316517	535.60	2	12	403749	598.32	0	1	8168	600.00
	20	0	3	152912	600.00	0	1	52706	600.00	0	0	207786	600.00	0	0	310	600.00
45	1	39	39	13618	16.22	39	39	6799	18.59	39	39	6637	18.65	39	39	6371	19.34
	5	21	32	500967	357.07	16	21	232003	395.01	14	19	270896	419.36	4	6	22899	555.90
	10	2	15	445267	586.29	0	9	238465	600.00	0	4	260543	600.00	0	0	4566	600.00
	20	0	2	127072	600.00	0	2	35495	600.00	0	1	160533	600.00	0	0	497	600.00
50	1	39	39	26054	19.24	39	39	22030	27.36	39	39	22283	28.01	39	39	21934	32.47
	5	13	27	683993	484.13	8	15	373070	518.58	7	13	412258	549.67	1	2	18848	586.40
	10	0	7	239020	600.00	0	3	108710	600.00	0	1	231011	600.00	0	0	1348	600.00
	20	0	0	58843	600.00	0	0	19428	600.00	0	0	156308	600.00	0	0	108	600.00
$\sum$		745	842			699	747			678	720			563	588		

Table 2: Comparison of four different versions of the solver.

N	k_D	Opt	Best	Nodes	Time	Step (1)		Step (2)		Step (3)	
						Runs	Time	Runs	Time	Runs	Time
20	1	40	40	43	0.02	6	0.00	3	0.00	3	0.00
	5	40	40	240	0.05	51	0.00	14	0.00	4	0.00
	10	40	40	503	0.16	104	0.00	38	0.00	4	0.00
	20	40	40	4385	6.53	2687	1.10	1291	3.93	9	0.23
25	1	40	40	97	0.02	6	0.00	3	0.00	3	0.00
	5	40	40	1670	0.43	868	0.20	37	0.00	4	0.00
	10	40	40	6974	4.98	2914	2.15	1074	1.03	10	0.35
	20	29	37	35507	230.72	19492	20.13	13263	205.25	19	2.33
30	1	40	40	284	0.05	20	0.00	6	0.00	4	0.00
	5	38	39	30690	33.10	24704	24.03	8891	5.38	163	0.53
	10	37	40	44316	83.90	11786	15.35	6130	57.75	25	2.88
	20	13	24	133946	442.32	75825	62.48	21176	350.13	13	11.35
35	1	40	40	701	0.12	35	0.00	19	0.00	5	0.00
	5	38	38	58298	43.34	28561	27.18	9223	6.90	75	0.50
	10	26	35	229963	312.59	63536	75.03	31118	172.68	400	27.60
	20	5	8	178155	561.73	102183	111.03	16730	411.23	6	12.95
40	1	39	39	11571	15.21	9525	10.63	4860	3.70	5	0.00
	5	34	36	276036	154.51	36983	54.15	16676	31.55	692	25.30
	10	12	22	433949	512.41	82944	134.20	38362	264.23	541	28.60
	20	0	3	152912	600.00	88775	106.93	11535	468.90	5	0.00
45	1	39	39	13618	16.22	8641	11.48	3391	3.03	17	0.05
	5	21	32	500967	357.07	69785	128.40	38407	93.08	1338	40.50
	10	2	15	445267	586.29	104149	136.25	38077	335.18	121	20.63
	20	0	2	127072	600.00	66531	137.60	4918	436.50	0	0.00
50	1	39	39	26054	19.24	7199	11.95	2245	2.75	170	0.83
	5	13	27	683993	484.13	80435	150.78	34328	141.30	723	36.55
	10	0	7	239020	600.00	76213	121.50	27487	399.50	56	20.63
	20	0	0	58843	600.00	34352	101.43	3939	482.95	0	0.00
Avg.						35654	51.57	11901	138.46	158	8.28

Table 3: Detailed results achieved with the complete version, V1, of the solver.