Annals of Operations Research (2020) 284:263–282
https://doi.org/10.1007/s10479-019-03263-6

ORIGINAL RESEARCH

# Robust min–max regret scheduling to minimize the weighted number of late jobs with interval processing times

**Maciej Drwal**[1] · **Jerzy Józefczyk**[1]

Published online: 10 May 2019


**Abstract**
We consider the robust version of single machine scheduling problem with the objective to minimize the weighted number of jobs completed after their due-dates. The jobs have uncertain processing times represented by intervals, and decision-maker must determine their execution sequence that minimizes the maximum regret. We develop an exact solution algorithm based on a specialized branch and bound method, using mixed-integer linear programming formulations for a common due-date and for job-dependent due-dates. Finally, we examine the solution algorithm in a series of computational experiments.

**Keywords** Robust optimization · Uncertainty · Scheduling · Mixed-integer programming

## 1 Introduction

Consider the following basic scheduling problem. Given is a set of $n$ jobs to be processed on a single machine in a sequence. Each job may require a different processing time, cannot be interrupted when started and there are no precedence constraints between jobs. Each job has an associated due-date, and an optimal schedule is such that the maximum number of jobs complete before their due-dates, or, equivalently, the number of late jobs is minimized. In a more general variant, each job may also have an associated weight, and an optimal solution maximizes the total weight of jobs completed on-time.

Assuming that we know prescisely how much of the processing time is required for each job, this problem can be solved fairly easily. If each job has the same due-date, then the shortest processing time first (SPTF) schedule is optimal for the unweighted case, and in the presence of weights the problem is equivalent to knapsack (Karp 1972). In case of arbitrary due-dates for jobs with equal weights an optimal schedule can be computed with a greedy method. In this paper we consider a more realistic setting in practical applications, when processing times

This research was partially supported by National Science Center, Poland, under Grant 2017/26/D/ST6/00423.

✉ Maciej Drwal
maciej.drwal@pwr.edu.pl

[1] Faculty of Computer Science and Management, Wroclaw University of Science and Technology, Wybrzeze Wyspianskiego 27, 50-370 Wroclaw, Poland

are not known precisely (at the time when a complete schedule must be dispatched). One of the standard techniques is to model the processing times as random variables with suitable probability distributions (Pinedo 1983). Then such a problem can be solved by optimizing for the expected value of the objective function. But in practice it is very often the case that no probability distribution of processing times is known either; for example, decision-maker may not have enough data to estimate it. For some critical applications it may be better then to assume the robust approach (Ben-Tal et al. 2009; Goerigk and Schöbel 2016): use a schedule that would perform possibly well even if the processing times turn out to be unfavorable.

We study the scheduling problem under the assumption that the processing times are known only to belong to prespecified intervals. A robust solution would be the one that performs the best in the worst-case scenario (a particular realization of the processing times). Standard formulations in the robust optimization often use min–max objective function (Ben-Tal et al. 2009). However, for the problem in question, the min–max criterion appears to be too restrictive: the worst-case scenario is simply the one consisting of the interval upper bounds. Instead, it is more reasonable to seek a solution that has the value deviating the least from optimality, when faced with its worst-case scenario. This can be accomplished by optimizing for the *maximum regret* of a solution.

In this paper we develop new mixed-integer programming techniques for min–max regret scheduling with interval processing times, with the objective to minimize the weighted number of late jobs. We consider the common-due date variant first, followed by the arbitrary due-dates variant. A branch and bound algorithm is presented and evaluated in a series of computational experiments.

### 1.1 Related work

Minimizing the maximum regret is a common approach to solving discrete optimization problems with uncertain input data (Aissi et al. 2009; Kouvelis and Yu 1997). This criterion is known to satisfy certain desirable properties of solutions when dealing with uncertainty in decision-making, especially as an useful alternative to min–max, Hurwicz and Laplace criteria (see Milnor 1951 for more details). Robust optimization can often be interpreted as seeking a solution in a form of game against the adversary that controls the input data of the deterministic optimization problem.

A substantial part of works concerning robust scheduling with min–max and min–max regret criteria use the assumption that discrete scenario sets are provided as input data. Even for such limited representation, many problems, which are easily solvable when parameters are known exactly, become significantly more difficult to solve (Kasperski and Zieliński 2016). For instance, the minimization of the number of late jobs with 2 processing time scenarios is NP-hard (Aloulou and Della Croce 2008). So is the problem with unit processing times and weighted number of late jobs criterion with 2 weight scenarios , and the 2-machine flow shop problem with 2 processing times scenarios (Kasperski and Zielinski 2014).

However, the interval representation of uncertainty appears to be more practically motivated, especially for modeling concepts of continuous nature, such as time. Although the scenario space in that case is no longer finite or even countable, the structure of such sets of uncertain parameters can sometimes be exploited. For the scheduling with the weighted number of late jobs criterion, a variant with unit processing times and interval weights has been considered within the min–max regret framework (see Kasperski 2008, chapter 14). While it is unknown whether this problem is NP-hard, it is known to be a generalization of the selecting items problem (where all jobs have a common due-date). The latter is proven

to be solvable in polynomial time in case of interval data, but NP-hard for discrete scenarios (Averbakh 2001). Computational experience suggests that most of the scheduling problems with interval data are hard to solve. The problem with total completion time on a single machine with interval weights is weakly NP-hard (Lebedev and Averbakh 2006), while its parallel machine variant is strongly NP-hard (Drwal and Rischke 2016). The problem with the same criterion but with uncertain processing times has been considered in Montemanni (2007) and Sotskov et al. (2009).

The uncertainty of processing times appears to occur very commonly in practical applications of scheduling (Herroelen and Leus 2005; Li and Ierapetritou 2008). Thus it is important to examine it from the robust optimization perspective. Also related to the considered scheduling problem is scheduling with due-window (Biskup and Feldmann 2005; Mosheiov and Sarig 2009), and determination of a common due-date (Gordon et al. 2002). Moreover, a variant of the considered scheduling problem with a common due-date is very similar to the knapsack problem, whose min–max regret version with interval item values has been shown to be $\Sigma_2^p$-hard (Deineko and Woeginger 2010). Heuristic algorithms for this problem (including ones based on integer programming) have been examined (Furini et al. 2015). In this paper, however, the knapsack subproblem would appear in a variant with fixed item values but with interval uncertainty in item sizes.

## 2 Problem formulation

Let us define the considered scheduling problem. Using the Graham classification scheme (Graham et al. 1979) this problem is denoted $1||\sum w_jU_j$. The problem is defined by specifying the set of jobs $J = \{1, 2, \ldots, n\}$. Each job $j \in J$ is described by three parameters: $p_j$, a processing time, $d_j$, a due-date, and $w_j$, the weight or priority (interpreted as the cost of missing a deadline). All the parameters are assumed to be positive rational numbers. Since we assume that each job is ready to be executed immediately, and no precedence constraints are specified, we do not consider schedules with idle times. Once the execution of a job has started, it cannot be preempted. Consequently, a solution (schedule) can be represented as a permutation of job indices $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$, where $\pi(k)$ is the index of job to be scheduled as the $k$th from the beginning. We denote by $\mathcal{P}$ the set of all permutations of $\{1, \ldots, n\}$. We also define a function $C : \mathcal{P} \times J \to \mathbb{R}_+$, where $C(\pi, j) = \sum_{k=1}^{j} p_{\pi(k)}$ is the completion time at which $j$th job finishes processing in the schedule $\pi$.

Given a schedule $\pi \in \mathcal{P}$, we define $U_j(\pi) = 1$ if $j$th job completes after its due-date, and $U_j(\pi) = 0$ otherwise, i.e.:

$$U_j(\pi) = \begin{cases} 0, & C(\pi, j) \le d_j, \\ 1, & \text{otherwise.} \end{cases}$$

We say that a job is *on-time*, if it completes not later than its due-date. Otherwise, we say that the job is *late*.

The objective is to find a schedule $\pi$ that minimizes the total cost of late jobs:

$$F(\pi) = \sum_{j \in J} w_j U_j(\pi). \tag{1}$$

Problem $1||\sum w_jU_j$ is weakly NP-hard (Karp 1972). However, its all unit weights variant, $1||\sum U_j$, can be solved in polynomial time by a greedy algorithm (Brucker 2007). A case with common due date $1|d_j = d|\sum w_jU_j$ can be reduced to the knapsack problem in a

straightforward way, while its unweighted case, $1|d_j = d|\sum U_j$, can be solved by sorting jobs in the order of nondecreasing processing times, i.e., applying the *shortest processing time first* (SPTF) rule (Brucker 2007).

When considering problems with uncertain data, we call the deterministic counterpart (i.e., one with certain parameters) a *nominal* problem. A nominal problem becomes a subproblem in the *robust* formulation.

We now define the uncertain variant of the problem. We assume that due-dates and weights of jobs are certain, but the exact processing times of jobs are only known to belong to the given intervals. For each $j \in J$, we have that the processing time $p_j \in [p_j^-, p_j^+]$, where interval bounds are known positive rational numbers. A vector of processing times will be called a *scenario*. The set of all possible scenarios is defined as:

$$\mathcal{U} = \{\mathbf{p} = (p_1, \ldots, p_n) : \forall_{j\in J} \;\; p_j^- \leq p_j \leq p_j^+\}.$$

The value of objective function in a scenario $\mathbf{p} \in \mathcal{U}$ will be denoted by $F(\pi, \mathbf{p})$.

Given a solution $\pi \in \mathcal{P}$, and a scenario $\mathbf{p} \in \mathcal{U}$, the *regret* is defined as:

$$R(\pi, \mathbf{p}) = F(\pi, \mathbf{p}) - \min_{\sigma\in\mathcal{P}} F(\sigma, \mathbf{p}). \tag{2}$$

We measure the quality of a solution $\pi \in \mathcal{P}$ under interval uncertainty using the notion of maximum regret. The problem of computing the maximum regret for a given $\pi$ is called an *adversarial* problem. It involves solving a nested minimization problem. The solution $\sigma^*$ that minimizes $F(\sigma, \mathbf{p})$ is called an *adversarial* solution. Finally, the robust objective of the problem is:

$$Z(\pi) = \max_{\mathbf{p}\in\mathcal{U}} R(\pi, \mathbf{p}). \tag{3}$$

A scenario $\mathbf{p}$ that maximizes the regret will be called a *worst-case scenario*. Hence we define an optimal solution $\pi^*$ of an uncertain problem as:

$$Z^* = Z(\pi^*) = \min_{\pi\in\mathcal{P}} Z(\pi). \tag{4}$$

A solution $\pi^*$ that minimizes the maximum regret is called a *robust optimal* solution.

Note that the nominal problem (1) can be equivalently defined as maximization of the total weight of on-time jobs, instead of minimization of the total weight of late jobs, since the sum of all weights $W = \sum_{j\in J} w_j$ is constant:

$$F(\pi) = \sum_{j\in J} w_j U_j(\pi) = W - \sum_{j\in J} w_j \left(1 - U_j(\pi)\right) = W - \bar{F}(\pi).$$

Consequently, the regret (2) can expressed as the difference between the total weight of on-time jobs in optimal solution for scenario $\mathbf{p}$, and the total weight of on-time jobs in a schedule $\pi$ under scenario $\mathbf{p}$,

$$R(\pi, \mathbf{p}) = \max_{\sigma\in\mathcal{P}} \bar{F}(\sigma, \mathbf{p}) - \bar{F}(\pi, \mathbf{p}). \tag{5}$$

This form will be used in Sect. 3 in order to formulate mixed-integer programs.

It can be seen that the robust version of the scheduling problem (4) is in fact a variant of bilevel optimization problem (Colson et al. 2007): the evaluation of the objective function (3) involves solving a nested maximization problem that determines the worst-case scenario. This is the reason that standard mathematical programming methods cannot be applied directly, and specialized solution methods need to be developed.

## 3 Maximum regret subproblem

In this section we discuss the subproblem (3), which consists of maximizing the regret function for a given fixed schedule $\pi \in \mathcal{P}$. We will refer to this as the *adversarial problem*; it is natural to interpret the robust problem as a two-player game between a decision-maker, controlling schedule $\pi$, who is faced by an adversary who controls the scenario $\mathbf{p}$ and an alternative schedule $\sigma$. The choice of scenario influences the value of both schedules $\pi$ and $\sigma$.

As already noted, the nominal problem is NP-hard in the presence of weights, so is the problem of maximizing regret (5). The solution to this problem can, however, be computed by solving a mixed-integer program.

Before we proceed, let us discuss how the worst-case scenarios are formed. Provided that not all jobs are on-time in $\pi$, there is $j \in J$, such that all jobs scheduled before $j$ are on-time, while the job $j$, and all jobs scheduled after $j$, are late. Intuitively, the maximum regret (5) is attained when first late job $j$ appears early in the schedule, but late in the alternative schedule provided by adversary. Consequently, the worst-case processing times of jobs preceding $j$ tend to be longer (close to their upper interval bounds). At the same time, jobs following $j$ in the schedule $\pi$ would have their processing times as short as possible (equal to their lower interval bounds), allowing the adversary to reschedule them early.

An *extreme* scenario is defined as a scenario that consists entirely of interval endpoints; in our case, $\mathbf{p}^E = (p_1^E, \ldots, p_n^E)$, where $p_j^E \in \{p_j^-, p_j^+\}$ for all $j \in J$. Many min–max regret combinatorial optimization problems have the property that the worst-case scenario is an extreme scenario. This is true for all min–max regret problems with a nominal problem of the form $\min\{\mathbf{x} \cdot \mathbf{c} : \mathbf{x} \in \mathcal{F}\}$, where $\mathbf{c}$ is a vector of nonnegative real numbers, and $\mathcal{F} \subset \{0, 1\}^n$. This is also true, for example, for a variant of the problem in question with uncertain weights and unit processing times, and for 2-machine flow shop problem with interval processing times (Kasperski 2008), although both of them do not belong to the aforementioned class. This property immediately implies an (exponential time) algorithm for discovering the worst-case scenario for a large class of problems. However, even for the simplest variant of the considered scheduling problem with uncertain processing times, the worst-case scenarios are not necessarily extreme scenarios, as shown in the following:

**Proposition 1** *A worst-case scenario of* INTERVAL MIN- - MAX REGRET $1|d_j = d|\sum U_j$ *is not necessarily an extreme scenario.*

***Proof*** Consider the following example (Fig. 1). Let $n = 4$ jobs be given, each with the same processing time interval: $p_j^- = 2$, $p_j^+ = 5$ (we assume for simplicity of computation that each job has the same weight $w_j = 1$, thus each permutation has the same worst-case scenario). Suppose that due-date is $d_j = 8$, and consider $\pi = (1, 2, 3, 4)$. Given a scenario $\mathbf{p}$, the regret (5) can be written as $R(\pi, \mathbf{p}) = L - K$, where $K$ is the number of on-time jobs in schedule $\pi$, and $L$ is the number of on-time jobs in adversarial schedule $\sigma$. It can be seen that in the worst-case scenario only one job can complete on-time, while the adversary can reschedule so that three jobs are on-time. This follows from the fact that at least one job will be on-time in all scenarios, thus $K \geq 1$. There exists only one scenario when all four jobs can be on-time: the all interval lower-bounds scenario, which obviously is not the worst-case; thus $L \leq 3$ holds for the worst-case one, which implies $R(\pi, \mathbf{p}) \leq 2$. Consequently, the worst-case scenario is: $p_1 = 5$, $p_2 = 3 + a$, $a \in (0, 1]$, $p_3 = 2$, $p_4 = 2$. Such a scenario has the maximum regret equal to 2, while the adversarial schedule is $\sigma = (3, 4, 2, 1)$. Note that this is not an extreme scenario for all $a \in (0, 1]$. □

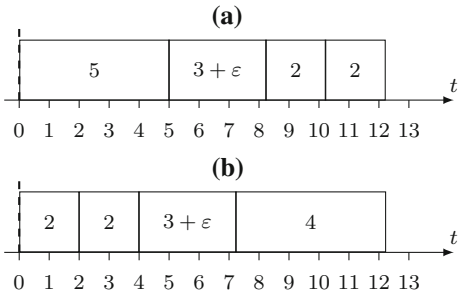


**Fig. 1** Illustration of Proposition 1: **a** schedule (1, 2, 3, 4) in the worst-case scenario, and **b** the worst-case alternative schedule (3, 4, 2, 1)

It can be seen that there are no extreme worst-case scenarios at all for the problem instance considered above. We have also shown that there may exist infinitely many worst-case scenarios. Another example of a min–max regret scheduling problem with the property that worst-case scenario is not always extreme is the minimization of the weighted number of late jobs with interval due-date uncertainty (Drwal 2017, 2018).

### 3.1 Maximum regret MIP for common due-date

Let us first consider the case with a common due-date, $d_j = d$, for $j \in J$. The following mixed-integer program (MIP) allows to compute the worst-case scenario and the value of maximum regret (5) for a given schedule $\pi \in \mathcal{P}$. The program involves binary decision variables $\mathbf{y}$ and $\mathbf{z}$, as well as continuous decision variables $\mathbf{p}$. Variable $y_j$ assumes the value 1 when the job $j$ is on-time in the schedule $\pi$ (i.e., $y_j = 1 - U_j(\pi)$), while $z_j = 1$ if job $j$ is on-time in the worst-case adversarial solution for $\pi$ (i.e., $z_j = 1 - U_j(\sigma)$). Continuous variable $p_j$ denotes the processing time of job $j$. The objective is to:

$$\text{maximize} \quad R(\mathbf{p}, \mathbf{y}, \mathbf{z}) = \sum_{j \in J} w_j (z_j - y_j), \tag{6}$$

subject to:

$$\sum_{j \in J} p_j z_j \leq d, \tag{7}$$

$$\forall_{k=1,\ldots,n} \quad \sum_{i=1}^{k} p_{\pi(i)} \geq d(1 - y_{\pi(k)}) + \varepsilon, \tag{8}$$

$$\forall_{j \in J} \quad p_j^- \leq p_j \leq p_j^+, \tag{9}$$

$$\forall_{j \in J} \quad z_j \in \{0, 1\}, y_j \in \{0, 1\}. \tag{10}$$

The objective function (6) can be seen as a difference between the total weight of on-time jobs in the worst-case adversarial solution $\sigma$, and the total weight of on-time jobs in the schedule $\pi$.

Note that constraint (8) allows to determine whether the job at position $k$ is late in the schedule $\pi$ (given a certain scenario). The objective function increases when variables $y_j$ assume values 0, but for any $j = \pi(k)$ this is possible only when the sum of $k$ first jobs' processing times exceeds $d$. In order to exclude the case when this sum is exactly $d$ we include a constant $0 < \varepsilon < \min_j p_j^-$ at the righthand side of (8).

Vector of processing times $\mathbf{p}^*$ corresponding to optimal solution of MIP is a worst-case scenario of the problem. Note that although constraint (7) is nonlinear, it is straightforward to write an equivalent mixed-integer linear program by introducing auxiliary nonnegative continuous variables $v_j$ in place of mixed terms $p_j z_j$. The following linear constraints are sufficient to replace (7):

$$\sum_{j\in J} v_j \le d, \tag{11}$$

$$\forall_{j\in J} \quad p_j + p_j^+ z_j - v_j \le p_j^+. \tag{12}$$

### 3.2 Maximum regret MIP for arbitrary due-dates

Next, we formulate MIP for the case with job-dependent due-dates. Note that, in the previous case, the order of jobs in the adversarial schedule did not matter (it was enough to determine the set of on-time jobs), and constraint (7) allowed to determine the optimal values of $\mathbf{z}$. This time it is necessary to distinguish between jobs' due-dates, and consequently we use separate constraint for each job, similar to (8). Decision variables $\mathbf{y}$, $\mathbf{z}$ and $\mathbf{p}$ have the same meaning as in MIP (6)–(10). In addition, we introduce permutation variable $\sigma$, interpreted as the worst-case alternative solution. The objective is to:

$$\text{maximize} \quad R(\sigma, \mathbf{p}, \mathbf{y}, \mathbf{z}) = \sum_{j\in J} w_j(z_j - y_j), \tag{13}$$

subject to:

$$\forall_{k=1,\ldots,n} \quad y_{\pi(k)} = 0 \;\Rightarrow\; \sum_{i=1}^{k} p_{\pi(i)} \ge d_{\pi(k)} + \varepsilon, \tag{14}$$

$$\forall_{k=1,\ldots,n} \quad z_{\sigma(k)} = 1 \;\Rightarrow\; \sum_{i=1}^{k} p_{\sigma(i)} \le d_{\sigma(k)}, \tag{15}$$

$$\forall_{j\in J} \quad p_j^- \le p_j \le p_j^+, \tag{16}$$

$$\forall_{j\in J} \quad z_j \in \{0,1\}, y_j \in \{0,1\}. \tag{17}$$

Since vector $\sigma$ is a decision variable, we need to rewrite (15) as a set of equivalent linear constraints. For this we introduce new decision variable $\mathbf{s} \in \{0,1\}^{n\times n}$, a binary matrix that encodes permutation $\sigma$. We have $s_{jk} = 1$ if job $j$ is scheduled at position $k$, and $s_{jk} = 0$ otherwise. There is a single 1 in each row and each column of $\mathbf{s}$, enforced by matching constraints:

$$\forall_{k=1,\ldots,n} \quad \sum_{j\in J} s_{jk} = 1, \tag{18}$$

$$\forall_{j\in J} \quad \sum_{k=1}^{n} s_{jk} = 1. \tag{19}$$

This allows to rewrite the objective function (13) as:

$$R(\mathbf{s}, \mathbf{p}, \mathbf{y}, \mathbf{z}) = \sum_{j\in J}\sum_{k=1}^{n} s_{jk} z_k w_j - \sum_{j\in J} w_j y_j, \tag{20}$$

and constraint (15) as:

$$\forall_{k=1,\ldots,n} \quad z_k = 1 \quad \Rightarrow \quad \sum_{i=1}^{k} \sum_{j \in J} s_{ji} p_j \leq \sum_{j \in J} s_{jk} d_j. \tag{21}$$

Note that variables $\mathbf{z}$ have now the following interpretation: $z_k = 1$ if a job placed at position $k$ in the schedule $\sigma$ completes on-time, and $z_k = 0$ otherwise. Which job it actually is from the set of jobs $J$ is to be determined by solving the program. Moreover, the conditional constraints (14) and (15) can be stated as standard linear constraints, respectively, as:

$$\forall_{k=1,\ldots,n} \quad y_{\pi(k)} M_k + \sum_{i=1}^{k} p_{\pi(i)} \geq d_{\pi(k)} + \varepsilon, \tag{22}$$

and

$$\forall_{k=1,\ldots,n} \quad \sum_{i=1}^{k} \sum_{j \in J} s_{ji} p_j \leq \sum_{j \in J} s_{jk} d_j + (1 - z_k) N_k, \tag{23}$$

where $M_k$ and $N_k$ are sufficiently large constants. The same technique applies also to (27).

Finally, in order to obtain mixed-integer linear program solvable by standard mathematical programming methods, we linearize products $s_{jk} z_k$ in (20) and $s_{jk} p_j$ in (23) in analogous way to (11)–(12) in the common due-date case.

## 4 Branch and bound algorithm

Although the considered problems are computationally difficult, we may ask whether it is still possible to determine optimal robust solutions in practice. In order to avoid the exhaustive search it is desirable to develop a solution method that exploits the problem's structure. We present a method for exploring the search space that is capable of substantially reducing the number of potential candidate solutions to examine. Such scheme could serve both as an exact solution algorithm, as well as a foundation for various heuristic solution methods. The method can be practically effective for moderately sized problem instances and produce robust optimal solutions in realistic timeframes. For a greater number of jobs, due to its worst-case exponential time complexity, it often fails to give an optimal solution, but can still be useful, as it is capable of generating a sequence of gradually better suboptimal schedules.

The algorithm is first presented for the problem variant with common due-date $d_j = d$ for all $j \in J$. Then its extension for the variant with arbitrary due-dates is discussed in Sect. 4.2.

The presented Algorithm 1 explores the solution space, which can be visualized as a search tree, with each node representing a subset of jobs' permutations – candidate solutions. The root of the search tree corresponds to the full set of permutations $\mathcal{P}$, while each leaf corresponds to single permutation (a feasible solution of the robust optimization problem). Each inner node represents a subset of permutations: a node at depth $k$ represents a set of schedules with first $k$ positions fixed (with root corresponding to $k = 0$, and leaves to $k = n$).

We propose a special branching procedure, which subdivides the search space by considering partially fixed schedules. A priority queue of nodes to be explored is maintained during the execution of the algorithm. Initially the queue contains only the root (all entries unfixed). The algorithm processes a node by first checking if it corresponds to fully fixed schedule (in this case the maximum regret can be computed by solving MIP (6)–(10), and the resulting value would be compared with the current best value). Otherwise, the node corresponds to

a partial schedule. The algorithm applies heuristic in order to compute a complete schedule from the partial one (see Step 12). The heuristic assumes interval upper bounds $p_j^+$ for all fixed jobs, and mid-point processing times $p_j^- + \frac{1}{2}(p_j^+ - p_j^-)$ for the remaining jobs in the partial schedule, and solves the nominal problem. For the obtained schedule $\pi'$ the maximum regret is computed by solving MIP (6)–(10) (and similarly, it is checked if a new incumbent has been found).

Subsequently, the branching is performed on the first unoccupied position in the partial schedule. A new node is created for each subsequent assignment of a previously unscheduled job at that position.

Note that the algorithm is capable of exploring whole search space by generating all possible permutations, but in many cases a branch of the search tree may be discarded early. It is possible that a newly created node corresponds to a subset of schedules with definite value of maximum regret, even if not all jobs are fixed (see Step 20). This happens, when the subsequence of fixed jobs at a given node already completes after due-date (even in the most optimistic all-lower bounds scenario $p_j = p_j^-$). Then the order of the remaining jobs is irrelevant to the actual value of the solution: jobs that are already known to be late in any scenario may be scheduled in arbitrary order. Thus we do not need to branch on the positions that determine their order, but we can put them in any order (e.g., lexicographical), and treat the node as a leaf of the search tree. Similarly, it is not necessary to consider all partial schedules which are permutations of subsequences of jobs that complete on-time in the all-upper bounds scenario $p_j = p_j^+$. This is due to the fact that jobs that are always on-time, regardless of the scenario, may be scheduled in arbitrary order (only the choice of the first late job may influence the value of maximum regret). Consequently, only a node with first such subsequence generated would be added to the queue. That subsequence would be stored in memory, and all the following nodes with the same jobs in their partial schedules would be discarded.

By evaluating a lower bound of optimal solution in each node of the search tree and comparing them with the currently best solution value, a large number of candidate solutions may be discarded earlier, reducing the computational effort. For each node we determine the value of lower bound (LB), by considering the maximum regret in a scenario that is easy to compute, but is not necessarily the worst-case one. A typically bad-case (but not worst-case) scenario can be derived from an extreme scenario, in which all jobs that start before due-date in schedule $\pi$ are set to have processing times at their interval upper-bounds, while all other jobs' processing times assume their interval lower-bounds (this is explained in more detail later; see Algorithm 2). If this solution lower bound value is greater or equal to the current upper bound on the optimal solution (i.e., the value of currently best feasible solution), we discard the node from further considerations.

A new node is added to the queue with a priority based on the "permutation distance" from the current incumbent solution. Let $\pi^{-1}(j)$ denote the position of job $j$ in the schedule $\pi$; for a partial schedule, if $j$ is not yet scheduled in $\pi$, then $\pi^{-1}(j) = n + 1$. The permutation distance is defined as:

$$D(\pi_1, \pi_2) = \sum_{j=1}^{n} |\pi_1^{-1}(j) - \pi_2^{-1}(j)|. \tag{24}$$

A distance between schedules (permutations of jobs) is computed as the sum of differences between position indices of each job in both schedules. If a job is not present in one of the schedules (as it happens when one schedule is partial), then value $n + 1$ is added for that job. The lower priority value, the closer to the beginning of the queue the node is placed.

By prioritizing partial schedules that are similar to the current incumbent schedule, the algorithm explores the neighborhood of incumbent solution first, before proceeding to the other areas of the search space. The algorithm can be initialized with an arbitrary schedule, but it is desirable to have a good initial solution. A heuristic algorithm could be used to find one, e.g., based on sampling solutions, or solving nominal problem in various fixed scenarios (e.g., all interval mid-points).

**Algorithm 1** Branch & bound method.

**Require:** Initial incumbent solution $\pi_0$.
**Ensure:** Robust optimal solution $\pi^*$.

1: Let $UB$ be equal to the value of initial solution $\pi_0$. Initialize queue with root node.
2: Take the first *node* off the queue (if the queue is empty the algorithm terminates).
3: **if** the *node* corresponds to fully fixed schedule $\pi$ **then**
4: Compute maximum regret $R$ of $\pi$ .
5: **if** $R < UB$ **then**
6: update incumbent solution to $\pi$, and let $UB \leftarrow R$, go to Step 2
7: **end if**
8: **else**
9: **if** *node*'s $LB \geq UB$ **then**
10: discard *node* and go to Step 2
11: **end if**
12: Run heuristic to compute solution $\mathbf{x}'$ from the *node*'s partial schedule.
13: Compute maximum regret $R'$ of $\mathbf{x}'$ .
14: **if** $R' < UB$ **then**
15: update incumbent solution to $\mathbf{x}'$, and let $UB \leftarrow R'$, go to Step 2
16: **end if**
17: Branching. Let $k$ be the depth of the current *node*.
18: **for** $i = 1, \ldots, n - k$ **do**
19: create new node by fixing $i$th unscheduled job at position $k + 1$,
20: **if** new node corresponds to subset of schedules with a fixed maximum regret **then**
21: fix the remaining jobs in arbitrary order,
22: **else**
23: compute lower bound $LB$ for new node (see Algorithms 2 and 3)
24: **if** $LB \geq UB$ **then**
25: discard the created node and go to Step 2
26: **end if**
27: **end if**
28: add new node to the queue with the priority equal to the distance (24) between incumbent and new node's partial schedule
29: **end for**
30: **end if**

### 4.1 Bounding rule

In Step 23–25 of Algorithm 1 it is possible to cut-off a branch of the search tree after examining a partial schedule with unfixed entries. Let $\mathbf{h} = [h_1, \ldots, h_n]$ represents a partial schedule, i.e., $h_i$ is either equal to the index of job fixed at position $i$, or $h_i = 0$ if the contents of position $i$ is not determined yet. Algorithm 2 allows to compute a lower bound of the value of optimal solution that can be obtained from a node corresponding to the partial schedule. If that lower bound is greater or equal to the value of incumbent solution, the corresponding subset of candidate solutions can be discarded from further search.

Note that Algorithm 2 requires solving the following knapsack problem twice as a subroutine:

$$v(\mathbf{s}, \mathbf{v}, \kappa) = \max \left\{ \sum_{i=1}^{m} x_i v_i : \sum_{i=1}^{m} x_i s_i \leq \kappa, \ \forall_i \ x_i \in \{0, 1\} \right\}, \tag{25}$$

where $\mathbf{s}$ is the vector of item sizes, $\mathbf{v}$ is the vector of item values, and $\kappa$ is the capacity.

**Algorithm 2** Lower bound computation for common due-date variant.

**Require:** Partial schedule $\mathbf{h}$, scenario $\tilde{\mathbf{p}} \in \mathcal{U}$.
**Ensure:** The lower bound value $LB$ of a node corresponding to $\mathbf{h}$.

1: Initialization. Let $t \leftarrow 0$, $i \leftarrow 1$.
2: **while** $t < d$ and $h_i \neq 0$ **do**
3: let $q_{h_i} \leftarrow p^+_{h_i}$,
4: let $t \leftarrow t + p^+_{h_i}$,
5: increment $i \leftarrow i + 1$,
6: **end while**
7: **while** $h_i \neq 0$ **do**
8: let $q_{h_i} \leftarrow p^-_{h_i}$,
9: let $t \leftarrow t + p^-_{h_i}$,
10: increment $i \leftarrow i + 1$,
11: **end while**
12: Let $q_j \leftarrow \tilde{p}_j$ for all $j \in J \backslash H$.
13: Solve the knapsack problem (25) with capacity $C = d - t$ for items with sizes $s_j = q_j$ and values $v_j = w_j$, for $j \in J \backslash H$. Denote the value of solution as $v_K = v(\mathbf{s}, \mathbf{v}, C)$.
14: Solve the knapsack problem (25) with capacity $d$ for items with sizes $s_j = q_j$, and values $v_j = w_j$, for $j \in J$. Denote the value of solution as $v_L = v(\mathbf{s}, \mathbf{v}, d)$.
15: **return** $LB = v_L - v_K - \sum_{j \in H} w_j$

The idea behind this algorithm is to determine a lower bound on the robust optimal solution for a restricted problem: the schedules must agree with a partial schedule $\mathbf{h}$ for a given node, and their maximum regret is computed in a fixed scenario $\tilde{\mathbf{p}}$. The scenario is selected so that it is "close" to worst-case scenario, but is easy to construct: jobs in partial schedule that complete before due-date have maximal processing times $p^+_j$, while jobs starting after due-date have minimal processing times $p^-_j$; all the remaining (unfixed) jobs have randomly selected processing times from within their intervals. In order to estimate the bound on robust optimal solution we consider the total weight of on-time jobs in the restricted schedule, and the total weight of on-time jobs in the adversarial schedule that can be achieved in the considered scenario.

Let us denote by $\mathcal{F}(\mathbf{h})$ the set of schedules that agree with a partial schedule $\mathbf{h}$ on all fixed positions (i.e., except on positions labelled with "0"). We also denote $H = \{j \in J : \exists_i \ h_i = j\}$. Then we have:

**Proposition 2** *Let* $\mathbf{h}$ *be a partial schedule and* $LB(\mathbf{h})$ *be the value computed by Algorithm 2 for* $\mathbf{h}$. *Then* $LB(\mathbf{h}) \leq \min_{\pi \in \mathcal{F}(\mathbf{h})} Z(\pi)$.

***Proof*** Let $\mathcal{U}' \subset \mathcal{U}$. For any $\pi \in \mathcal{P}$ we have:

$$\max_{\mathbf{p} \in \mathcal{U}'} R(\pi, \mathbf{p}) \leq \max_{\mathbf{p} \in \mathcal{U}} R(\pi, \mathbf{p}) = Z(\pi),$$

thus for $\pi^* \in \arg\min_{\pi \in \mathcal{F}(\mathbf{h})} Z(\pi)$ we have that the maximum regret on a restricted uncertainty set $\mathcal{U}'$ is a lower bound on the robust optimal value (at the node corresponding to the partial schedule $\mathbf{h}$).

Moreover, $\max_{\mathbf{p} \in \mathcal{U}'} R(\pi^*, \mathbf{p}) = v_L^* - v_K^*$, where $v_K^*$ is the total weight of on-time jobs in robust optimal solution $\pi^*$, and $v_L^*$ is the total weight of on-time jobs in adversarial solution. The Algorithm 2 returns value $v_L - v_K - \sum_{j \in H} w_j$, which we show is no greater than $v_L^* - v_K^*$.

The algorithm computes maximum regret for a singleton set $\mathcal{U}'$ containing scenario $\mathbf{q}$, where:

(a) $q_j = p_j^+$, for $j \in H$ that complete before due-date in scenario $\mathbf{q}$,
(b) $q_j = p_j^-$, for $j \in H$ that start after due-date in scenario $\mathbf{q}$,
(c) $q_j = \tilde{p}_j$, for $j \in J \backslash H$.

The Algorithm 2 constructs in Step 13 the set of on-time jobs that would fill the positions with undefined contents in a partial schedule $\mathbf{h}$. This is accomplished by assuming scenario $\mathbf{q}$, and solving the knapsack problem, restricted to unfixed jobs $j \in J \backslash H$. The knapsack capacity is equal to the residual capacity $C = d - t$, where $d$ is the due-date, and $t$ is the total processing time taken by fixed jobs in scenario $\mathbf{q}$. The choice of jobs corresponding to the optimal solution of the knapsack problem will result in the largest possible total weight $v_K$ of on-time jobs in all the schedules that agree with $\mathbf{h}$. Note that if $t \geq d$ then the value of on-time jobs is already determined, and $v_K = 0$. This value, plus the total weight of fixed jobs, $\sum_{j \in H} w_j$, gives an upper bound on the total weight of on-time jobs in robust optimal solution $v_K^*$.

On the other hand, the total weight of on-time jobs in adversarial schedule in scenario $\mathbf{q}$ can be computed by simply solving the nominal (knapsack) problem with item sizes equal to processing times $q_j$. This is computed in Step 14, and this value is denoted $v_L$.

Since $v_K + \sum_{j \in H} w_j \geq v_K^*$, and $v_L = v_L^*$, we obtain a lower bound on the maximum regret of a robust optimal solution for a partial schedule $\mathbf{h}$, computed for the restricted uncertainty set $\mathcal{U}' = \{\mathbf{q}\}$, and consequently, for the full uncertainty set $\mathcal{U}$. □

In order to obtain a good lower bound it is suggested to run Algorithm 2 a number of times for different scenarios $\tilde{\mathbf{p}}$, and to use the maximum value returned over the sequence of runs as a final lower bound of a node. One can also observe that typically in practice the worst-case scenarios contain many extreme jobs (i.e., ones with processing time either $p_j^-$ or $p_j^+$), with only few non-extreme jobs. Thus it may be a good strategy to randomize over scenarios with extreme jobs when initializing Algorithm 2 with $\tilde{\mathbf{p}}$.

### 4.2 Arbitrary due-dates variant

The branch and bound method presented in the previous sections can be easily extended to handle the problem variant with arbitrary due-dates for each job. The main difference is that the maximum regret of a schedule, required in Steps 4 and 13 of Algorithm 1, is computed using MIP (13)–(21), and that the optimal solution to the nominal problem $1||\sum w_j U_j$ can be computed by solving the following MIP:

$$\text{maximize} \quad \sum_{j \in J} w_j \sum_{k=1}^{n} s_{jk} z_k, \tag{26}$$

subject to:

$$\forall_{k=1,\ldots,n} \quad z_k = 1 \quad \Rightarrow \quad \sum_{i=1}^{k} \sum_{j \in J} s_{ji} p_j \leq \sum_{j \in J} s_{jk} d_j, \tag{27}$$

$$\forall_{k=1,\ldots,n} \quad \sum_{j \in J} s_{jk} = 1, \tag{28}$$

$$\forall_{j \in J} \quad \sum_{k=1}^{n} s_{jk} = 1, \tag{29}$$

$$\forall_{j \in J} \forall_{k=1,\ldots,n} \quad s_{jk}, z_j \in \{0, 1\}. \tag{30}$$

In order to compute node's lower bound in Step 23 the Algorithm 3 could be used, which is based on the same principle as Algorithm 2. The difference is that nominal problem (13)–(21) needs to be solved twice instead of the knapsack problem (25). The former problem is first solved with additional constraints that fix jobs on positions determined in a partial schedule $\mathbf{h}$ (to get an upper bound $v_K$ on the value of optimal solution that can be derived from $\mathbf{h}$), and then the second time without additional constraints (to get a lower bound $v_L$ on the adversarial solution). Again, a number of runs with different randomly generated scenarios $\tilde{\mathbf{p}}$ could be performed to obtain good lower bound. However, due to the higher complexity of MIP (13)–(21), the computations would typically require significantly more time. In all experimental results presented in Sect. 5 we sampled 5 random scenarios when computing lower bound for each node.

Finally, we remark that the special cases of both problem variants, when all jobs have equal weights ($w_j = 1$ for $j \in J$), can be solved much faster. In the variant with a common due-date, the solution algorithm of the knapsack problem (25) can be replaced by a simple greedy algorithm that selects items in the order of nondecreasing sizes. In the variant with arbitrary due-dates, the solution algorithm of nominal MIP (26)–(30) can be replaced by a polynomial time algorithm for scheduling problem $1||\sum U_j$ (Brucker 2007).

**Algorithm 3** Lower bound computation for arbitrary due-dates variant.

**Require:** Partial schedule $\mathbf{h}$, scenario $\tilde{\mathbf{p}} \in \mathcal{U}$.
**Ensure:** The lower bound value $LB$ of a node corresponding to $\mathbf{h}$.

1: Initialization. Let $t \leftarrow 0$, $i \leftarrow 1$.
2: **while** $h_i \neq 0$ **do**
3: let $q_{h_i} \leftarrow p^+_{h_i}$,
4: let $\tau = t + p^+_{h_i} - d_{h_i}$,
5: **if** $\tau > 0$ **then**
6: $q_{h_i} \leftarrow q_{h_i} - \tau + \varepsilon$
7: **end if**
8: let $t \leftarrow t + q_{h_i}$,
9: increment $i \leftarrow i + 1$
10: **end while**
11: Let $q_j = \tilde{p}_j$ for all $j \in J \backslash H$.
12: Solve the nominal problem (26)–(30) for $\mathbf{p} = \mathbf{q}$, and additional constraints $s_{jk} = 1$, where $h_k = j$, for $j \in H$. Denote the value of solution as $v_K$.
13: Solve the nominal problem (26)–(30) for $\mathbf{p} = \mathbf{q}$. Denote the value of solution as $v_L$.
14: **return** $LB = v_L - v_K$

## 5 Experimental results

In this section we present the results of computational experiments, conducted on different types of randomly generated data sets. We have taken into consideration four experiments, each designed to assess different aspect of the proposed solution method.

***Experiment 1*** In the first experiment we have examined the performance of Algorithm 1 on the problem variant with common due-date. The algorithm was run on three types of sets of random problem instances (A, B and C), with increasing number $n$ of jobs in each set (between 10 and 100). For each problem instance we generated initial incumbent solution by computing maximum regret for mid-point scenario heuristic solution, as well as 1000 random permutations of jobs (taking the best out of these solutions). The Algorithm 1 was run until either termination condition was met (i.e., queue of nodes became empty), or the time limit set to 1 h was reached. Since for a number of larger instances it was often the case that the Algorithm 1 failed to terminate within the assumed time limit, some of the obtained solutions are either suboptimal, or lack the certificate of optimality. Consequently, we compared the obtained results with ones computed using faster solution method, the mid-point scenario heuristic. In this reference method we apply the algorithm for deterministic problem to the scenario consisting of all interval mid-points. The results of this experiment are summarized in Table 2.

***Experiment 2*** In order to assess the quality of solutions computed by the Algorithm 1 for which no certificate of optimality has been obtained within 1 h, we compared the results with solutions obtained by a randomized search heuristic. The heuristic sampled random schedules for 1 h and reported the best solution found. Note that for many instances the Algorithm 1 terminated much faster. Consequently, results for the data sets with instances containing larger number of jobs (when Algorithm 1 tends to run out of time) have been reported. The results of this experiment are summarized in Table 3.

***Experiment 3*** For the problem variant with common due-date we also tested the performance of Algorithm 1 on data set containing instances with only 50% uncertain processing times. This allowed to determine whether the amount of uncertainty influences the performance of the solution method. As in Experiment 1, the results have been compared against solution values returned by mid-point scenario heuristic. The results of this experiment are summarized in Table 4.

***Experiment 4*** In this experiment we have examined the performance of Algorithm 1 on the problem variant with job-dependent due-dates. Three types of data sets have been used (B, C and D). Due to high computational requirements, in this case only data sets with instances containing up to $n = 30$ jobs have been considered. The results of this experiment are summarized in Table 5.

The branch and bound method has been implemented in Python programming language and CPLEX 12.8 solver has been used for solving all MIP subproblems. Computations were performed on machines with Intel Xeon processors with 16 cores at 2.4 GHz and 40 GB of RAM (all cores were available for solving each problem instance).

### 5.1 Construction of data sets

We observe that especially difficult to solve are the problem instances where many potential candidate solutions cannot be easily discarded by the algorithm, since the bounding methods

**Table 1** Description of data sets used in experiments

| Type | $p_{\min}$ | $p_{\max}$ | $p_w$ | $d_{\min}$ | $d_{\max}$ | $w_{\max}$ |
|---|---|---|---|---|---|---|
| A | 10 | 20 | 20 | 20 | 50 | 100 |
| B | 5 | 20 | 30 | 20 | 50 | 100 |
| C | 10 | 20 | 10 | $n$ | $3n$ | 100 |
| D | 5 | 10 | 20 | $n$ | $3n$ | 100 |

(Algorithms 2 and 3) give too weak lower bounds. This often happens when processing time interval lower bounds of many jobs are very small, as in that case a large number of jobs complete before due-date in adversarial schedule. Thus, two main factors that differentiates instance difficulty is the average value of due-date $d_j$ and the minimal value of interval lower-bound $p_j^-$.

Consequently, the data sets used in experiments are divided into four different types, ranging from "A" – easiest, to "D" – hardest (Table 1). Each set of instances is described by six parameters: $p_{\min}, p_{\max}, p_w, d_{\min}, d_{\max}, w_{\max}$, where:

(1) interval lower-bounds $p_j^-$ are uniformly random integers between $p_{\min}$ and $p_{\max}$,
(2) interval widths are uniformly random integers between 1 and $p_w$,
(3) due-dates $d_j$ are uniformly random integers between $d_{\min}$ and $d_{\max}$,
(4) job weights $w_j$ are uniformly random integers between 1 and $w_{\max}$.

### 5.2 Results

Results for problem instances for the variant with common due-date are presented in Tables 2, 3 and 4, while for the variant with job-dependent due-dates in Table 5. Note that each row contains values that are averaged over 10 instances of a given type.

In Tables 2, 4 and 5, the first two columns describe the data set (number of jobs $n$, and type of input data A, B, C or D). The following two columns contain the mean value and standard deviation of computation time, excluding the results from instances when the algorithm did not terminate before the time limit. The limit used in all the experiments was equal to 1 h of wall-clock time. Next two columns contain the mean value and standard deviation of the number of iterations performed by Algorithm 1 (i.e., the number of nodes explored in the search tree, excluding the pruned nodes) before termination. The next column, labeled "opt.", indicates for how many instances out of 10 a certificate of optimality has been found (i.e., the algorithm terminated normally). Note that it is likely that for some instances without such certificates optimal solutions might have been found too. Following are the mean value and standard deviation of solution values obtained by Algorithm 1. The fourth column in the group, denoted "gap%", is the mean value of optimality gap (i.e., the ratio $(UB - LB)/UB$, expressed in percents, where $UB$ is the value of the best feasible solution found and $LB$ is the best lower bound). The mean value and standard deviation of solution values from mid-point scenario heuristic are given in the next two columns. The last column contains the measure of improvement achieved by Algorithm 1 ($v_B$) over the mid-point scenario heuristic ($v_M$), expressed in percents, computed as $(v_M - v_B)/v_M$.

Table 3 contains mean values and standard deviations of best solutions found using randomized search heuristic, which sampled random schedules for 1 h of wall clock time. The heuristic was run on the same data sets as the ones used in Experiment 1, reported in Table 2. The last column of the table contains the measure of improvement achieved by Algorithm 1

**Table 2** Experimental results for common due-date variant

| Data sets | | Time (s) | | Iterations | | Solution value | | | | Midpoint | | Imp. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | Type | Mean | SD | Mean | SD | Opt. | Mean | SD | Gap% | Mean | SD | % |
| 10 | A | 2.8 | 2.6 | 10.2 | 9.9 | 10 | 44.7 | 31.3 | – | 83.4 | 52.1 | 46 |
| 20 | A | 59.0 | 78.8 | 23.2 | 33.7 | 10 | 53.8 | 30.9 | – | 88.0 | 59.5 | 39 |
| 30 | A | 382.0 | 517.1 | 97.3 | 123.7 | 10 | 52.5 | 36.5 | – | 95.5 | 34.6 | 45 |
| 40 | A | 903.8 | 1114.0 | 142.1 | 174.0 | 10 | 78.6 | 33.2 | – | 124.9 | 22.7 | 24 |
| 50 | A | 983.0 | 1086.4 | 101.0 | 122.5 | 10 | 95.4 | 18.8 | – | 137.9 | 31.3 | 31 |
| 60 | A | 430.6 | 302.0 | 56.5 | 94.4 | 9 | 75.4 | 31.2 | $< 1$ | 128.0 | 40.4 | 41 |
| 70 | A | 308.9 | 303.5 | 57.5 | 108.3 | 8 | 49.6 | 40.4 | $< 1$ | 62.8 | 59.1 | 21 |
| 80 | A | 824.6 | 1294.2 | 55.4 | 45.1 | 6 | 89.7 | 39.1 | 1 | 131.8 | 42.6 | 32 |
| 90 | A | 523.5 | 434.4 | 29.8 | 36.0 | 5 | 77.1 | 45.4 | 1 | 128.7 | 65.3 | 40 |
| 100 | A | 150.8 | 102.0 | 36.2 | 35.3 | 5 | 78.5 | 45.7 | 3 | 117.9 | 57.3 | 33 |
| 10 | B | 10.1 | 8.0 | 28.5 | 24.9 | 10 | 77.1 | 52.9 | – | 105.5 | 61.7 | 27 |
| 20 | B | 155.7 | 171.6 | 128.9 | 138.7 | 10 | 146.6 | 20.9 | – | 176.5 | 19.5 | 17 |
| 30 | B | 308.5 | 356.1 | 143.5 | 165.1 | 10 | 144.3 | 80.9 | – | 182.0 | 83.2 | 20 |
| 40 | B | 653.1 | 751.5 | 240.8 | 324.2 | 8 | 135.6 | 53.8 | $< 1$ | 180.4 | 59.9 | 25 |
| 50 | B | 158.2 | 224.6 | 286.4 | 302.4 | 5 | 157.3 | 67.2 | $< 1$ | 202.5 | 61.4 | 22 |
| 60 | B | 941.5 | 725.6 | 269.2 | 199.3 | 4 | 182.5 | 89.6 | $< 1$ | 207.6 | 92.7 | 12 |
| 70 | B | 1355.7 | 1103.2 | 178.6 | 133.2 | 4 | 167.3 | 71.2 | $< 1$ | 214.2 | 70.4 | 22 |
| 80 | B | 1844.1 | 1758.8 | 104.7 | 96.8 | 5 | 150.7 | 60.3 | 1 | 189.2 | 66.5 | 20 |
| 90 | B | 1232.0 | 1293.6 | 86.6 | 82.0 | 3 | 196.9 | 112.2 | 3 | 238.3 | 112.6 | 17 |
| 100 | B | 542.2 | 482.3 | 86.4 | 57.8 | 3 | 191.6 | 60.7 | 4 | 231.4 | 48.4 | 17 |
| 10 | C | 2.2 | 1.5 | 14.4 | 15.4 | 10 | 25.6 | 26.4 | – | 39.4 | 40.1 | 35 |
| 20 | C | 187.2 | 185.7 | 141.4 | 142.3 | 10 | 80.3 | 31.6 | – | 111.0 | 29.9 | 28 |
| 30 | C | 1117.9 | 1346.6 | 888.3 | 914.8 | 8 | 97.1 | 34.7 | $< 1$ | 132.9 | 50.2 | 27 |
| 40 | C | – | – | 437.5 | 442.7 | 0 | 170.4 | 44.3 | 1 | 209.0 | 41.2 | 18 |
| 50 | C | 1969.1 | – | 544.3 | 201.0 | 1 | 200.7 | 55.5 | 1 | 225.4 | 38.9 | 11 |
| 60 | C | – | – | 406.9 | 41.5 | 0 | 248.1 | 68.3 | 2 | 276.1 | 58.3 | 10 |

Comparison of Algorithm 1 and mid-point heuristic

($v_B$) over the heuristic ($v_R$), expressed in percents, computed as $(v_R - v_B)/v_R$. Note that the Algorithm 1 returned better solutions for all instances, even though it often terminated before the 1 h time limit.

### 5.3 Observations

Analyzing the results, we have noticed that the branch and bound method introduced in the previous section gives substantial improvements over solutions obtained both via mid-point scenario heuristic, as well as via randomized search heuristic. In should be noted, however, that the Algorithm 1 should be used as a framework for exact solution methods, that can incorporate various heuristics as subroutines (used an the root and internal nodes). The presented results give insight into the efficiency of the Algorithm 1 used in the variant described in Sect. 5, as compared to two simple heuristic methods. Comparison with mid-

**Table 3** Experimental results for arbitrary due-date variant

| Data sets | | Rnd. search value | | B&B value | | Imp. |
|---|---|---|---|---|---|---|
| *n* | Type | Mean | SD | Mean | SD | % |
| 50 | A | 100.1 | 14.9 | 95.4 | 18.8 | 5 |
| 60 | A | 82.3 | 30.6 | 75.4 | 31.2 | 8 |
| 70 | A | 68.1 | 34.9 | 49.6 | 40.4 | 27 |
| 80 | A | 90.1 | 39.3 | 89.7 | 39.1 | 2 |
| 90 | A | 87.6 | 39.8 | 77.1 | 45.4 | 12 |
| 100 | A | 93.5 | 39.7 | 78.5 | 45.7 | 16 |
| 50 | B | 163.6 | 73.2 | 157.3 | 67.2 | 4 |
| 60 | B | 199.2 | 101.0 | 182.5 | 89.6 | 8 |
| 70 | B | 184.1 | 81.3 | 167.3 | 71.2 | 9 |
| 80 | B | 172.5 | 77.3 | 150.7 | 60.3 | 13 |
| 90 | B | 208.5 | 126.4 | 196.9 | 112.2 | 6 |
| 100 | B | 199.7 | 75.1 | 191.6 | 60.7 | 4 |
| 10 | C | 34.0 | 19.3 | 25.6 | 26.4 | 25 |
| 20 | C | 94.1 | 29.2 | 80.3 | 31.6 | 15 |
| 30 | C | 111.8 | 43.9 | 97.1 | 34.7 | 13 |
| 40 | C | 215.0 | 33.3 | 170.4 | 44.3 | 21 |
| 50 | C | 223.7 | 44.7 | 200.7 | 55.5 | 10 |
| 60 | C | 376.2 | 67.2 | 248.1 | 68.3 | 34 |

Random search heuristic with 1 h time limit compared to Algorithm 1 solution values. Last column reports improvements obtained via Algorithm 1 (see Table 2)

**Table 4** Experimental results for common due-date variant

| Data sets | | Time (s) | | Iterations | | Solution value | | | | Midpoint | | Imp. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *n* | type | Mean | SD | Mean | SD | Opt. | Mean | SD | Gap% | Mean | SD | % |
| 10 | B | 37.3 | 45.2 | 27.6 | 33.2 | 10 | 54.3 | 31.2 | – | 91.0 | 43.7 | 40 |
| 20 | B | 94.1 | 108.8 | 77.3 | 86.1 | 10 | 82.2 | 37.8 | – | 108.3 | 40.1 | 24 |
| 30 | B | 310.5 | 339.3 | 167.0 | 187.2 | 10 | 97.9 | 57.5 | – | 130.8 | 78.8 | 25 |
| 40 | B | 603.8 | 841.0 | 309.4 | 344.9 | 8 | 84.4 | 55.9 | $< 1$ | 126.3 | 72.5 | 33 |
| 50 | B | 669.4 | 1140.1 | 117.5 | 173.0 | 7 | 117.9 | 54.4 | $< 1$ | 168.8 | 65.7 | 30 |
| 60 | B | 1534.9 | 908.2 | 152.8 | 117.2 | 4 | 140.4 | 63.0 | $< 1$ | 183.7 | 94.1 | 23 |
| 70 | B | 1210.9 | 1297.3 | 151.8 | 133.7 | 5 | 131.7 | 73.6 | $< 1$ | 162.5 | 77.0 | 19 |
| 80 | B | 469.1 | 262.7 | 100.8 | 103.0 | 3 | 149.9 | 67.1 | 1 | 189.1 | 55.7 | 21 |
| 90 | B | 463.0 | 394.3 | 70.6 | 76.2 | 5 | 129.1 | 72.5 | 1 | 166.2 | 82.3 | 22 |
| 100 | B | 330.0 | 237.7 | 70.8 | 42.9 | 4 | 164.4 | 73.1 | 2 | 194.7 | 83.7 | 15 |
| 10 | C | 2.5 | 1.5 | 3.3 | 1.0 | 10 | 15.1 | 17.5 | – | 31.1 | 38.3 | 51 |
| 20 | C | 147.7 | 122.8 | 110.1 | 85.5 | 10 | 29.9 | 24.9 | – | 52.5 | 52.3 | 43 |
| 30 | C | 790.5 | 638.0 | 456.5 | 565.1 | 7 | 46.5 | 20.5 | $< 1$ | 114.0 | 15.9 | 59 |
| 40 | C | 772.9 | 564.1 | 435.3 | 388.2 | 5 | 53.7 | 25.1 | 1 | 90.6 | 33.5 | 41 |
| 50 | C | 2602.1 | – | 484.1 | 204.7 | 1 | 98.1 | 19.0 | 1 | 109.0 | 28.0 | 10 |
| 60 | C | 1055.8 | 768.8 | 389.1 | 44.3 | 3 | 104.0 | 58.5 | 2 | 132.0 | 79.5 | 21 |

Comparison of Algorithm 1 and mid-point heuristic on instances with only 50% uncertain jobs

**Table 5** Experimental results for arbitrary due-date variant

| Data sets | | Time (s) | | Iterations | | Solution value | | | | Midpoint | | Imp. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | Type | Mean | SD | Mean | SD | Opt. | Mean | SD | Gap% | Mean | SD | % |
| 10 | B | 33.3 | 31.1 | 55.5 | 59.0 | 10 | 34.8 | 17.5 | – | 83.8 | 37.7 | 58 |
| 15 | B | 626.2 | 730.3 | 100.5 | 129.3 | 8 | 45.6 | 25.7 | 2 | 111.0 | 39.7 | 59 |
| 20 | B | 2407.1 | – | 29.9 | 22.2 | 1 | 81.5 | 28.6 | 8 | 116.5 | 40.9 | 30 |
| 25 | B | 246.6 | – | 12.2 | 7.8 | 1 | 90.2 | 21.2 | 11 | 154.2 | 59.8 | 41 |
| 10 | C | 311.2 | 434.6 | 76.1 | 82.7 | 9 | 69.7 | 38.2 | $< 1$ | 139.1 | 71.7 | 49 |
| 15 | C | 1374.1 | 1227.8 | 64.1 | 40.5 | 5 | 118.8 | 21.3 | 2 | 178.6 | 48.4 | 33 |
| 20 | C | – | – | 13.3 | 2.9 | 0 | 149.2 | 48.1 | 4 | 199.4 | 72.4 | 25 |
| 25 | C | – | – | 17.9 | 3.6 | 0 | 156.1 | 60.9 | 5 | 212.9 | 60.9 | 27 |
| 30 | C | – | – | 7.0 | 0.7 | 0 | 199.5 | 42.9 | 9 | 263.5 | 83.9 | 24 |
| 10 | D | 22.4 | 18.1 | 48.3 | 89.0 | 10 | 32.1 | 23.0 | – | 68.2 | 43.3 | 52 |
| 15 | D | 582.3 | 776.9 | 61.8 | 19.8 | 7 | 49.2 | 28.6 | 8 | 105.2 | 42.8 | 53 |
| 20 | D | 2023.9 | – | 17.6 | 4.0 | 1 | 74.8 | 13.7 | 13 | 128.2 | 43.6 | 42 |
| 25 | D | – | – | 14.0 | 12.3 | 0 | 185.0 | 82.8 | 17 | 228.8 | 86.4 | 19 |
| 30 | D | – | – | 12.3 | 5.1 | 0 | 197.7 | 23.3 | 19 | 222.5 | 37.5 | 11 |

Comparison of Algorithm 1 and mid-point heuristic

point heuristic also provides an estimate of how much more regret would be generated, when stochastic optimization approach (taking expected values of uncertain parameters with entropy-maximizing distribution) would be used instead of robust one.

The drawback of using Algorithm 1, as compared to the mid-point scenario heuristic, is that it often requires considerably long time of computation. It shares the feature of many MIP-based solution techniques, so that optimal solution can be found early, but almost all computation time is spent on finding the proof of its optimality (in terms of exhausting the search tree). However, the branch and bound method is capable of finding consistently better solutions than the randomized search heuristic within 1 h time limit.

The method performs well on moderately sized instances of the problem variant with a common due-date. Unfortunately, the variant with job-dependent due-dates is much more difficult, and already for relatively small instances the exploration of search tree is prohibitively expensive [due to the need of repeatedly solving MIP (13)–(21)]. The method often terminates without a certificate of optimality after 1 h for instances with 20–30 jobs with arbitrary due-dates.

It also appears that the presence of jobs with certain processing times in the data sets improves the performance of the algorithm. More difficult instances with only 50% uncertain processing times took significantly less effort to solve.

## 6 Conclusions

We have addressed the problem of uncertain processing times, represented by intervals, in a basic single machine scheduling problem with due-dates. The min–max regret has been considered as the concept of robustness. The need for modeling uncertainty in the input data has important practical motivation, but it appears that it is very difficult to deal with

in combinatorial optimization problems. Using mixed-integer programming formulations, we have developed an exact branch and bound method, adapted to the min–max regret optimization. This method has been examined in computational experiments, and shown to be superior to various heuristic solution methods (mid-point scenario heuristic and random scenario heuristic). The method can be developed further in several ways. Faster and more tight bounding rules could be applied in place of Algorithms 2 and 3. Problem variants with release dates and precedence constraints could also be solved by a similar approach. Moreover, the branch and bound method and bounding rules introduced in this paper could be modified to handle other robust combinatorial optimization problems with interval data.

**Acknowledgements** The authors are grateful to Wroclaw Centre for Networking and Supercomputing for granting access to the computing infrastructure built in the Projects No. POIG.02.03.00-00-028/08 "PLATON—Science Services Platform".



## References

Aissi, H., Bazgan, C., & Vanderpooten, D. (2009). Min–max and min–max regret versions of combinatorial optimization problems: A survey. *European Journal of Operational Research*, *197*(2), 427–438.

Aloulou, M. A., & Della Croce, F. (2008). Complexity of single machine scheduling problems under scenario-based uncertainty. *Operations Research Letters*, *36*(3), 338–342.

Averbakh, I. (2001). On the complexity of a class of combinatorial optimization problems with uncertainty. *Mathematical Programming*, *90*(2), 263–272.

Ben-Tal, A., El Ghaoui, L., & Nemirovski, A. (2009). *Robust optimization*. Princeton: Princeton University Press.

Biskup, D., & Feldmann, M. (2005). On scheduling around large restrictive common due windows. *European Journal of Operational Research*, *162*(3), 740–761.

Brucker, P. (2007). *Scheduling algorithms*. Berlin: Springer.

Colson, B., Marcotte, P., & Savard, G. (2007). An overview of bilevel optimization. *Annals of Operations Research*, *153*(1), 235–256.

Deineko, V. G., & Woeginger, G. J. (2010). Pinpointing the complexity of the interval min–max regret Knapsack problem. *Discrete Optimization*, *7*(4), 191–196.

Drwal, M. (2017). Min–max regret scheduling to minimize the total weight of late jobs with interval uncertainty. In *International conference on optimization and decision science* (pp. 611–619). Springer.

Drwal, M. (2018). Robust scheduling to minimize the number of late jobs with interval due-date uncertainty. *Computers & Operations Research*, *91*, 13–20.

Drwal, M., & Rischke, R. (2016). Complexity of interval minmax regret scheduling on parallel identical machines with total completion time criterion. *Operations Research Letters*, *44*(3), 354–358.

Furini, F., Iori, M., Martello, S., & Yagiura, M. (2015). Heuristic and exact algorithms for the interval min–max regret Knapsack problem. *INFORMS Journal on Computing*, *27*(2), 392–405.

Goerigk, M., & Schöbel, A. (2016). Algorithm engineering in robust optimization. In L. Kliemann & P. Sanders (Eds.), *Algorithm engineering* (pp. 245–279). Berlin: Springer.

Gordon, V., Proth, J. M., & Chu, C. (2002). A survey of the state-of-the-art of common due date assignment and scheduling research. *European Journal of Operational Research*, *139*(1), 1–25.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, *5*, 287–326.

Herroelen, W., & Leus, R. (2005). Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, *165*(2), 289–306.

Karp, R. M. (1972). Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher & J. D. Bohlinger (Eds.), *Complexity of computer computations* (pp. 85–103). Berlin: Springer.

Kasperski, A. (2008). *Discrete optimization with interval data: Minmax regret and fuzzy approach*. Berlin: Springer.

Kasperski, A., & Zielinski, P. (2014). Minmax (regret) scheduling problems. In Y. Sotskov & F. Werner (Eds.), *Sequencing and scheduling with inaccurate data* (pp. 159–210). New York: Nova Publishers.

Kasperski, A., & Zieliński, P. (2016). Robust discrete optimization under discrete and interval uncertainty: A survey. In M. Doumpos, C. Zopounidis & E. Grigoroudis (Eds.), *Robustness analysis in decision aiding, optimization, and analytics* (pp. 113–143). Berlin: Springer.

Kouvelis, P., & Yu, G. (1997). *Robust discrete optimization and its applications*. Berlin: Springer.

Lebedev, V., & Averbakh, I. (2006). Complexity of minimizing the total flow time with interval data and minmax regret criterion. *Discrete Applied Mathematics*, *154*(15), 2167–2177.

Li, Z., & Ierapetritou, M. (2008). Process scheduling under uncertainty: Review and challenges. *Computers & Chemical Engineering*, *32*(4–5), 715–727.

Milnor, J. (1951). *Games against nature*. Technical report, DTIC Document

Montemanni, R. (2007). A mixed integer programming formulation for the total flow time single machine robust scheduling problem with interval data. *Journal of Mathematical Modelling and Algorithms*, *6*(2), 287–296.

Mosheiov, G., & Sarig, A. (2009). Minmax scheduling problems with a common due-window. *Computers & Operations Research*, *36*(6), 1886–1892.

Pinedo, M. (1983). Stochastic scheduling with release dates and due dates. *Operations Research*, *31*(3), 559–572.

Sotskov, Y. N., Egorova, N. G., & Lai, T. C. (2009). Minimizing total weighted flow time of a set of jobs with interval processing times. *Mathematical and Computer Modelling*, *50*(3), 556–573.