Solution Algorithms for Minimizing the Total Tardiness with Budgeted Processing Time Uncertainty

Marco Silva, Michael Poss, Nelson Maculan

Solution Algorithms for Minimizing the Total Tardiness with Budgeted Processing Time Uncertainty

Marco Silva[a,*], Michael Poss[b], Nelson Maculan[c]

[a] *LIA, Université d'Avignon et des Pays des Vaucluse, Avignon, France*
[b] *UMR CNRS 5506 LIRMM, Université de Montpellier, Montpellier, France*
[c] *PESC, Universidade Federal do Rio de Janeiro, Brazil*

Abstract

We investigate algorithms that solve exactly the robust single machine scheduling problem that minimizes the total tardiness. We model the processing times as uncertain and let them take any value in a budgeted uncertainty set. Therefore, the objective seeks to minimize the worst-case tardiness over all possible values. We compare, through computational experiments, two types of solution algorithms. The first combines classical MILP formulations with row-and-column generation algorithms. The second generalizes the classical branch-and-bound algorithms to the robust context, extending state-of-the-art concepts used for the deterministic version of the problem. By generalizing the classical branch-and-bound algorithm we are able to assemble and discuss good algorithmic decisions steps that once put together make our robust branch-and-bound case attractive. For example, we extend and adapt dominance rules to our uncertain problem, making them an important component of our robust algorithms. We assess our algorithms on instances inspired by the scientific literature and identify under what conditions an algorithm has better performance than others. We introduce a new classifying parameter to group our instances, also extending existing methods for the deterministic problem case.

Keywords:
Robust Optimization, Scheduling Problem, Row-and-column generation, Integer programming, Branch-and-bound

1. Introduction

Scheduling is an important and diverse topic in combinatorial optimization. It has applications in many different industries, ranging from production and manufacturing systems to transportation and logistics systems. The scheduling problem considered in this study considers a single machine. Hence, we are

*Corresponding author
Email addresses: marco.costa-da-silva@univ-avignon.fr (Marco Silva), michael.poss@lirmm.fr (Michael Poss), maculan@cos.ufrj.br (Nelson Maculan)

given a set of n jobs, denoted $N = \{1, 2, \ldots, n\}$, and any feasible solution for the problem consists of a permutation of N, represented by σ where $\sigma(k)$ denotes the job that occupies position k for each $k \in N$. The objective function considered herein supposes that each job $i \in N$ has a given due date d_i and a processing time p_i. Assuming that the processing times are known with precision, we can define the completion time of job i for σ as

$$C_i(\sigma) = \sum_{k=1}^{\sigma^{-1}(i)} p_{\sigma(k)}, \tag{1}$$

and the tardiness of job i can be defined as $T_i(\sigma) = \max(C_i(\sigma) - d_i, 0)$. Denoting by X the set of all permutations of N, the problem that minimizes the tardiness can be formally stated as

$$\min_{\sigma \in X} \sum_{i=1}^{n} T_i(\sigma). \tag{2}$$

Notice that problem (2) is often denoted as $1||\sum_j T_j$ in the literature, where 1 means that a single machine is used and $\sum_j T_j$ represents the objective function employed.

In real applications, the processing times are hardly known with precision (see the examples below). Hence, it is more realistic to assume that p can take any value within a given set U that contains plausible values for p. Then, the robust counterpart of (2) is

$$\min_{\sigma \in X} \max_{p \in U} \sum_{i=1}^{n} T_i(p, \sigma), \tag{3}$$

where we introduced p in the definition of T_i to emphasize that its value depends on $p \in U$. It is similar for the completion time of job i that will be noted as $C_i(p, \sigma)$.

In this study, we let U be the budgeted uncertainty set proposed by [7]. Hence, we consider a parameter Γ and, for each job i, a nominal value $\bar{p}_i$ and a deviation $\hat{p}_i$. Then, the uncertainty set is defined as

$$U \equiv \left\{ p : p_i = \bar{p}_i + \hat{p}_i \xi_i,\ i = 1, \ldots, n; 0 \leq \xi_i \leq 1, \sum_{i=1}^{n} \xi_i \leq \Gamma \right\}, \tag{4}$$

Note that we have eliminated downward deviations from the original uncertainty set because they are not used by worst-case scenarios of our problem. The motivation behind definition U is two-fold. First, the set is easy to characterize and build from historical data, as only lower and upper bounds on p are required, while the value of Γ let us model the risk-averseness of the decision maker. Specifically, higher values of Γ lead to larger uncertainty sets and thus, more conservative solutions. For instance $\Gamma = 0$ means that $U = \{\bar{p}\}$ while $\Gamma = n$ means that U is the box $[\bar{p}, \bar{p} + \hat{p}]$. Second, the set has a nice combinatorial structure that can be exploited by combinatorial algorithms to provide more

efficient solution algorithms than using arbitrary uncertainty sets. In fact, these reasons have made (4) extremely popular in the robust combinatorial optimization and integer programming literature. In particular, recent papers on robust scheduling have provided polynomial algorithms for robust problems that would have been NP-hard using arbitrary uncertainty sets [8, 33].

The effect of uncertainty on scheduling has been studied under many different perspectives. For a more recent survey on different perspectives to scheduling under uncertainty one can see, for example, [9].

Applications of robust approaches to the total tardiness problem have been implemented in different industries. Typically, these applications solve problems where there are penalties associated with not fulfilling due dates and the decision maker has a conservative attitude in trying to minimize these penalties no matter the realization of uncertainty. In [10] the authors study project scheduling at a large IT services delivery center in which there are unpredictable delays, which are known to belong to a bounded set. They apply robust optimization to minimize tardiness while informing the customer of a reasonable worst-case completion time. They decompose the problem into a master problem and a subproblem. The subproblem is solved using constraint programming concepts in an algorithm called Logic-Based Benders Decomposition. The subproblems generate valid cuts to be used by the master problem in successive iterations. In [25] the authors consider a scheduling problem in which manufacturing companies with large energy demand must comply with total energy consumption limits in specified time intervals and have to deal with the fact that in reality the production schedules are not executed exactly as planned due to unexpected disturbances such as machine breakdowns or material unavailability. The goal is to find a robust schedule which minimizes total tardiness and guarantees that the energy consumption limits are not violated if the start times of operations are arbitrary delayed within a given limit. A robust total weighted tardiness approach is implemented in [1] for operation room planning in a hospital with uncertain surgery duration. The problem aims at minimizing a measure of waiting time of the patients and a tardiness function is created weighted by a patient urgency parameter.

In this work, we shall develop two types of exact solution algorithms to solve (3). The first type of algorithms combines the integer programming formulations available for $1||\sum_j T_j$ with the decomposition algorithm in the line of [37] and [4]. More specifically, our integer programming formulations will turn the static scheduling problem into an adjustable robust optimization problems by linearizing the convex function $\sum_j T_j$. This approach is classical in robust lot-sizing (e.g. [6]) and it has been studied for more general robust convex constraints in [31] and [38]. The second type of algorithms takes the (combinatorial) branch-and-bound algorithms developed for $1||\sum_j T_j$ and extend them to the robust counterpart (3).

Most exact solution algorithms for $1||\sum_j T_j$ are strongly based on dominance conditions and decomposition principles applied to the sequencing of jobs. These conditions and principles define ordering and positioning restrictions for the jobs in an optimal sequence and are used as a base to develop branch-

and-bound and dynamic programming algorithms. For branch-and-bound algorithms different lower bound propositions were developed, including even the absence of lower bounds and relying only on the power of decomposition (for examples see [12, 30, 20, 26, 32]). Decomposition principles were initially developed in [21] and establish conditions by which a single machine scheduling problem can be decomposed into subproblems that can be solved independently. For the robust scheduling case, due to the added complexity of having to deal with correlations of uncertainty between subproblems, we do not consider the decomposition principles introduced there.

Many integer formulation approaches can also be found in the literature for deterministic single machine problems in general (see [19] for review and comparisons). The formulations are based on the type of variables used to define the sequencing of jobs. Some effort has been made in order to improve performance of this approach by defining strong valid inequalities through polyhedral studies (see [29] for example). Six different integer programming formulations of the single machine total tardiness problem are compared in [5]. They verify that a generic integer programming approach does not compete with state-of-the-art branch-and-bound tardiness algorithms. They conclude that the sequence-position formulation provides most computationally effective solutions and note that, although the attention paid to time-index formulations may be justified as they provide insight into theoretical results, they leave something to be desired when it comes to computational experience.

We review next some recent works on robust scheduling, which originated in the seminal paper from [11]. From a robust optimization perspective, in [11], the authors work the concept of uncertainty defined by scenarios or interval data. They introduce three types of robust objective defined as absolute robustness, as used in this paper, deviation robustness, meaning minimizing the maximum deviation from optimality, and relative robustness defined as the one that exhibits the best worst case percentage deviation from optimality. The authors develop solutions with a objective performance of total flow time. The objective is based on deviation robustness criterion. Processing times are uncertain and defined by interval data, where a finite interval of equally possible values for processing times for each job is given. They introduce a MILP formulation for the problem, develop dominance rules and propose a branch-and-bound algorithm where the lower bounds are given as a surrogate relaxation of the MILP formulation. Heuristics are also presented. This MILP formulation is further improved in [24], using classical robust dual reformulation. The author also introduces additional restrictions to the formulation based on the dominance rules developed by [11]. He compares performance between his pure MILP solution using CPLEX with the branch-and-bound algorithm proposed by [11] and conclude that, for the instances used, MILP solution performs better.

That seminal work has also been followed by a sequence of works on the theoretical aspects of robust scheduling [e.g. 3, 23, 36] where the authors have shown that very simple scheduling problems become NP-hard as soon as the uncertainty set contains more than one scenario. Some authors have also studied the complexity of simple robust scheduling problems under budgeted uncer-

tainty, see [8, 33]. We also refer to [34] for a recent survey on robust scheduling.

From a numerical viewpoint, the closest work from the current manuscript has been carried out in [13] where the authors study the optimal allocation of surgery blocks to operating rooms. Therein the authors introduce different models, including a robust model that is similar to the problem here studied. However, a fundamental difference is that we let T_i be different for each $p \in U$, which is underlined by the notation $T_i(p, \sigma)$ used in (3), while [13] consider a static-model where T_i must be fixed independently from $p \in U$. While the static model is easier to handle computationally, it is also more conservative, as we shall illustrate briefly in our numerical experiments.

Our main contributions with respect to the literature are summarized below.

- We introduce an alternative approach to the existing literature of robust single machine tardiness scheduling, where we let T_i be different for each $p \in U$.
- We extend dominance rules developed for deterministic single machine total tardiness problem to our robust version. In our case, dominance rules are new constraints added to reduce the domains of the defined variables and, as a consequence, the solution space of our problem.
- We develop a branch-and-bound algorithm based on the dominance rules above and a defined lower bound. We generalize the classical branch-and-bound algorithms to the robust context, extending state-of-the-art concepts used for the deterministic version of the problem. By generalizing the classical branch-and-bound algorithm we are able to assemble and discuss good algorithmic decisions steps that once put together make our robust branch-and-bound case attractive.
- We introduce different MILP formulations for our problem, and define dynamic programming routines to solve separation problems within a given decomposition algorithm. We combine classical MILP formulations with row-and-column generation algorithms.
- We compare the solution performance of our different MILP formulations and branch-and-bound algorithms. We assess our algorithms on instances inspired by the scientific literature and identify under what conditions an algorithm has better performance than others. We introduce a new classifying parameter to group our instances, also extending existing methods for the deterministic problem case.

Outline We formally present the problem with MILP formulations in Section 2. There, we also present the row-and-column generation algorithm utilized to solve it. Next, in Section 3, we present the branch-and-bound algorithm developed to solve our problem. Sections 4 and 5 detail key concepts used in the algorithms presented. In Section 6 we present computational results.

2. MILP formulations

In [5] the authors summarize different MILP formulations for the deterministic single machine total tardiness problem. Based on the variables defined and their deterministic formulations, we present here the equivalent robust counterpart formulations utilized in our experiment. The formulations are based on the type of variables defined to represent the sequencing of jobs (see [5]). We have utilized sequence-positions variables and linear ordering variables.

Sequence position variables are variables x such that $x_{ik} = 1$ if job i is in position k and $x_{ik} = 0$ otherwise. Linear ordering variables are variables y such that $y_{ij} = 1$ if job i precedes job j and $y_{ij} = 0$ otherwise. These two sets of variables do not depend on the uncertain parameters as they describe the jobs ordering, which is fixed independently from the values taken by the processing times. In contrast, our robust formulations consider that tardiness, t, and job starting time, s, are only defined after realization of uncertainty, so that in a generic form there are infinite number of variables and constraints, each one representing a realization of uncertainty. We represent this in our formulation by introducing an index set W for our uncertainty set U, possibly uncountable, and utilizing a superscript x^w for any given variable or data x, meaning there is one for each $w \in W$. We show later in Subsection 2.3 that only a finite subset of W is needed, for a suitable W, to solve the problem.

We disregard here the time-indexed formulation (e.g. [28]) in which binary variables indicate when jobs are completed. While advanced decomposition algorithms (called SSDP in [35]) have solved efficiently the deterministic counterpart of $1||\sum_j T_j$, the robust counterpart cannot benefit from these techniques because the binary variables would depend on each vector $p \in U$. Specifically, the resulting formulation would have a pseudo-polynomial number of variables *for each* $p \in U$, making SSDP completely unrealistic [14]. Finally, notice that we also disregard the disjunctive constraints formulation (see [5]). Initial tests of our algorithms revealed that this formulation algorithm performed much worse, among all instances, when compared to others algorithms, so that we eliminated it from our analyses.

2.1. Sequence-position formulation

The robust counterpart of the sequence-position formulation is given by the following:

$$\begin{aligned}
\min \quad & z \\
\text{s.t.} \quad & \sum_{k=1}^{n} t_k^w \leq z && \forall\, w \in W && (5)\\
& \sum_{i=1}^{n} p_i^w \sum_{u=1}^{k} x_{iu} - \sum_{i=1}^{n} d_i x_{ik} \leq t_k^w && \forall\, k \in N\,, \forall\, w \in W && (6)\\
& \sum_{i=1}^{n} x_{ik} = 1 && \forall\, k \in N && (7)\\
& \sum_{k=1}^{n} x_{ik} = 1 && \forall\, i \in N && (8)\\
& x_{ik} \in \{0,1\},\, t_k^w \geq 0,\, z \geq 0 && \forall\, k, i \in N\,, \forall\, w \in W, && (9)
\end{aligned}$$

where variable z is the overall objective and t_k is tardiness of job in position k. Constraint (6) calculates tardiness for job in position k while constraints (7) and (8) guarantee that each position is occupied by only one job and each job occupies only one position.

If dominance rules are introduced to the above formulation, the information of precedence of jobs can be used to reduce the number of sequence-position variables. For this we define A_i to be the set of jobs known to follow job i and B_i as the set of jobs known to precede job i and use the cardinality of sets B_i and A_i to restrict possible positions k. Job i cannot occupy the first $|B_i|$ positions and the last $|A_i|$ positions. We can, therefore, reduce the the number of variables x_{ik} by limiting the variation of k as $|B_i| + 1 \leq k \leq n - |A_i|$ and constraints (6), (7) and (8) are substituted for:

$$\sum_{i=1}^{n} p_i^w \sum_{u=|B_i|}^{\min(k,n-|A_i|)} x_{iu} \; - \sum_{i:|B_i|<k\leq n-|A_i|} d_i x_{ik} \leq t_k^w \quad \forall\, k \in N\,, \forall\, w \in W \tag{10}$$

$$\sum_{k=|B_i|+1}^{n-|A_i|} x_{ik} = 1 \qquad \forall\, i \in N \tag{11}$$

$$\sum_{i:|B_i|<k\leq n-|A_i|} x_{ik} = 1 \qquad \forall\, k \in N \tag{12}$$

We add a new set of constraints to guarantee precedence between jobs:

$$\sum_{k=|B_j|+1}^{n-|A_j|} k x_{jk} \geq \sum_{k=|B_i|+1}^{n-|A_i|} k x_{ik} + 1 \qquad \forall\, i \in N,\, j \in A_i \tag{13}$$

Constraints (13) define that the position occupied by job j has to be at least one position greater than job i position.

2.2. Linear ordering formulation

The robust counterpart of the linear ordering formulation is given by the following:

$$\begin{aligned}
\min \quad & z \\
\text{s.t.} \quad & \sum_{j=1}^{n} t_j^w \leq z && \forall\, w \in Wr && (14)\\
& p_j^w + \sum_{\substack{i=1 \\ i \neq j}}^{n} p_i^w y_{ij} - d_j \leq t_j^w && \forall\, j \in N, \forall\, w \in W && (15)\\
& y_{ij} + y_{ji} = 1 && \forall\, i, j \in N,\, i < j && (16)\\
& y_{ij} + y_{jk} + y_{ki} \leq 2 && \forall\, i, j, k \in N\, i \neq j, j \neq k, i \neq k && (17)\\
& y_{ij} \in \{0,1\},\, t_j^w \geq 0,\, z \geq 0 && \forall\, i, j \in N,\, i \neq j, \forall\, w \in W, && (18)
\end{aligned}$$

where variable z is the overall objective and t_j is tardiness of job j. Constraint (15) calculates tardiness for job j while constraints (16) and (17) guarantee the right ordering of jobs.

In case dominance rules are considered the precedence of jobs information can be used and a new set of constraints is added:

$$y_{ij} = 1 \qquad \forall\, i \in N\,, j \in A_i \qquad (19)$$

2.3. Row-and-column generation algorithm

To solve our robust integer formulation we use the decomposition algorithm proposed by [37]. We relax the problem into a master problem where each robust constraint is written only for a finite subset $U^0 \subseteq U$. Given a feasible solution to the master problem, we check whether the solution is feasible for each robust constraint by solving adversarial separation problems. In case one or more robust constraints are infeasible, we expand U^0 by one or more vectors and solve the augmented master problem under a row-and-column generation approach.

It turns out that our adversarial separation problem is in fact the problem of, given a sequence of jobs, finding the worst-case realization of uncertainty value and verifying if that value is greater than the one provided by the master problem. We can solve that by the methods described in Section 4. Since our uncertainty set U is polyhedral, there will be a finite number of extreme solutions to search and the algorithm terminates. See [37, Proposition 2] for more details on this previous statement.

3. Branch-and-bound

The total tardiness problem has been studied through many methods of implicit enumeration, notably of branch-and-bound type. We develop below a branch-and-bound algorithm to solve our robust problem to optimality. The key elements of our branch-and-bound algorithm are: branching, node selection and bound and pruning. We describe each one of them in what follows.

3.1. Branching

We create a search tree with no jobs scheduled at the root node. From the root node, n branches lead to n nodes on the first level, each of which corresponds to a particular job being scheduled in the n-th position. Generally, each node at level l in a tree corresponds to a set $J_l \subseteq \{1, \ldots, n\}$ filling the last l positions in a given order. This is called branching by backward sequencing. By successively placing each job j ($j \in N \backslash J_l$) in the $|N \backslash J_l|$-th position, $|N \backslash J_l|$ new nodes are created. This is reasonable because all sequences of jobs are feasible in our problem.

In the literature both branching by backward sequencing(from last to first position) and forward sequencing(from first to last position) have been implemented. See for instance [30] where the authors implement backward sequencing as we have done. The decision of what method to use has been made by leveraging other existing properties. In our case we use backward sequencing in order to facilitate calculation of the lower bound, as described in item 3.3.2. There we show that modified due dates are calculated and dependent on the completion date of each job and by backward sequencing that does not involve jobs with positions already defined at each node (set J_l). That reduces computation time specially in nodes that are more distant from the root.

3.2. Node selection

We make use of a best bound or a depth first approach as search strategy for node selection. For best bound the node selected is the one, among unprocessed nodes, with minimum lower bound. This way we never branch any node whose lower bound is larger than the optimal value. For depth first the node selected is the one, among unprocessed nodes with maximum depth in the search tree. This way we navigate the tree prioritizing the search of new incumbent values.

3.3. Bound and pruning

After each branching we prune nodes based on dominance rules described in Section 5, on lower bounds when it is greater than the best upper bound calculated at that stage and on optimality conditions when lower bound matches upper bound of a given node.

3.3.1. Upper bound

An initial upper bound, as incumbent solution, is calculated at the root node based on a constructive heuristic algorithm (see Algorithm 1 for its pseudocode). The general idea behind this algorithm is that we shall assign jobs from last position to first. At each position we assign the job that provides the smallest ratio of the least tardiness and the greatest maximum processing time. By doing so, we leave jobs with grescheduleater tardiness or less maximum processing times to be assigned later in the sequencing, where their value of tardiness will decrease or their contribution to partial makespan of that position will be less than other jobs already assigned.

We leverage the calculated dominance rules to create sets of allowable jobs for each position k, $ALJ[k]$. Let SJ be the sequence of jobs already selected, from position n to position $k+1$. Then, we define $ALJ[k] = \{i \in N \backslash SJ \mid A_i \subseteq SJ\}$. This set is never empty, since if all A_i, $i \in N \backslash SJ$, have elements outside SJ, every element in $N \backslash SJ$ is followed by another element in $N \backslash SJ$ which creates a subcycle and that is prevented by construction when calculating the dominance rules (for example see [30] where these cycles are avoided by constructing the transitive closure of the set of known precedence relations immediately after a new relation has been found and by only examining pairs of jobs that are not yet related).

We assign the job i with the least ratio $T_i / \max_{p \in U} p_i$ to each position. This ratio for job i is given by the formula $\frac{\max(0, \max_{p \in U} \sum_{j \in N \backslash SJ} p_j - d_i)}{\bar{p}_i + \hat{p}_i}$. We solve the worst-case evaluation problem (Section 4), using the sequence assigned by our heuristic, to calculate the initial upper bound.

An upper bound is also calculated at the last level of the search tree using the sequence of jobs defined for that level, and solving the worst-case evaluation problem.

3.3.2. Lower bound

We detail in this subsection the lower bound that will be used to cut part of the branch-and-bound tree.

Here we introduce counterparts of (2), $P_{det}(p, \delta)$, and (3), $P_{rob}(\delta)$, where parameters processing time, $p \in U$, and due date, denoted $\delta \in \mathbb{R}^n_+$, are made explicit. In the same way we denote their optimal solution costs by $z_{det}(p, \delta) = \min_{\sigma \in X} \sum_{i=1}^{n} T_i(p, \sigma, \delta)$ and $z_{rob}(\delta) = \min_{\sigma \in X} \max_{p \in U} \sum_{i=1}^{n} T_i(p, \sigma, \delta)$, where now we emphasize dependence of T_i on δ values.

The first step to provide a lower bound to the original robust problem (3) comes from the idea of relaxing the due dates, d_i, of each job $i \in N$.

Proposition 1. *Define $d_i^{'} \geq d_i, \forall i \in N$, as the relaxed due date of job i. Therefore, $z_{det}(p, d^{'}) \leq z_{det}(p, d), \forall p \in U$.*

Proof. First we verify for each job $i \in N$ and schedule $\sigma \in X$ and a given $p \in U$:

$$d_i^{'} \geq d_i \implies T_i(p,\sigma,d^{'}) \leq T_i(p,\sigma,d) \implies \sum_{i=1}^{n} T_i(p,\sigma,d^{'}) \leq \sum_{i=1}^{n} T_i(p,\sigma,d). \tag{20}$$

Now, supposing σ_1 is an optimal solution for $P_{det}(p,d)$ and σ_2 is an optimal solution for $P_{det}(p,d^{'})$, we have:

$$z_{det}(p,d^{'}) = \sum_{i=1}^{n} T_i(p,\sigma_2,d^{'}) \leq \sum_{i=1}^{n} T_i(p,\sigma_1,d^{'}) \leq \tag{21}$$
$$\sum_{i=1}^{n} T_i(p,\sigma_1,d) = z_{det}(p,d).$$

□

The second step to provide a lower bound is to modify the due dates, d, in such a way that the optimal schedule becomes easy to provide. For this step we extend the work previously done in [32] to our robust case. Specifically, in [32], the authors apply the following properties to problem (2):

Proposition 2.

1. *The EDD schedule (ordering jobs by non decreasing due dates) is optimal if $C_{\sigma(k)}(p,\sigma) \leq d_{\sigma(k)} + p_{\sigma(k)}$ for all positions k [17, Corollary 2.2].*
2. *The statement above can be relaxed to be true only for positions k where $p_{\sigma(k)} < \max(p_{\sigma(1)},\ldots,p_{\sigma(k-1)})$, because on the other hand if $p_{\sigma(k)} \geq \max(p_{\sigma(1)},\ldots,p_{\sigma(k-1)})$ other arguments in [17] assure that jobs $\{\sigma(1),\ldots,\sigma(k-1)\}$ precede job $\sigma(k)$ [32, Property 5].*

The Proposition below extends the above properties to our robust case (3).

Proposition 3. *Given an EDD schedule σ^*, if $\max_{p\in U} C_{\sigma^*(k-1)}(p,\sigma^*) \leq d_{\sigma^*(k)}$ for each position $k > 1$ for which $\min_{p\in U} p_{\sigma^*(k)} < \max(\max_{p\in U}(p_{\sigma^*(1)}),\ldots,$ $\max_{p\in U}(p_{\sigma^*(k-1)}))$ then it is robust optimal.*

Proof. We first show that if $\sigma^* \in \arg\min_{\sigma\in X} \sum_{i=1}^{n} T_i(p,\sigma,d), \forall p \in U$, then $\sigma^* \in \arg\min_{\sigma\in X}\max_{p\in U} \sum_{i=1}^{n} T_i(p,\sigma,d)$. Suppose $\sigma^* \notin \arg\min_{\sigma\in X}\max_{p\in U} \sum_{i=1}^{n} T_i(p,\sigma,d)$ and $\exists\, \sigma^{**} \neq \sigma^*$, $\sigma^{**} \in \arg\min_{\sigma\in X}\max_{p\in U} \sum_{i=1}^{n} T_i(p,\sigma,d)$. Then, by definition of the robust optimal solution, $\max_{p\in U} \sum_{i=1}^{n} T_i(p,\sigma^*,d) > \max_{p\in U} \sum_{i=1}^{n} T_i(p,\sigma^{**},d)$. This is a contradiction since by hypothesis $\sum_{i=1}^{n} T_i(p,\sigma^*,d) \leq \sum_{i=1}^{n} T_i(p,\sigma^{**},d), \forall p \in U$.

Now suppose we have for $k > 1$, $\max_{p \in U} C_{\sigma^*(k-1)}(p, \sigma^*) \leq d_{\sigma^*(k)}$. Therefore $C_{\sigma^*(k-1)}(p, \sigma^*) \leq d_{\sigma^*(k)}, \forall p \in U$, and by Proposition 2.1,

$$\sigma^* \in \arg\min_{\sigma \in X} \sum_{i=1}^{n} T_i(p, \sigma, d), \forall p \in U.$$

Therefore, as showed above, $\sigma^* \in \arg\min_{\sigma \in X} \max_{p \in U} \sum_{i=1}^{n} T_i(p, \sigma, d)$.

On the other hand, if for a given position $k > 1$,

$$\min_{p \in U} p_{\sigma^*(k)} \geq \max(\max_{p \in U}(p_{\sigma^*(1)}), \ldots, \max_{p \in U}(p_{\sigma^*(k-1)})),$$

then we have $p_{\sigma^*(k)} \geq \max(p_{\sigma^*(1)}, \ldots, p_{\sigma^*(k-1)}), \forall p \in U$, and by Proposition 2.2, we can then relax the application of Proposition 2.1 for position $k, \forall p \in U$ and σ^* is robust optimal regardless the application of Proposition 2.1.

□

Proposition 4. *Given the EDD schedule σ^*, and defining $d'_{\sigma^*(k)} = \max(d_{\sigma^*(k)}, \max_{p \in U} C_{\sigma^*(k-1)}(p, \sigma^*))$ for positions k as in Proposition 3, $z_{rob}(d') \leq z_{rob}(d)$. Also, $\sigma^* \in \arg\min_{\sigma \in X} \max_{p \in U} \sum_{i=1}^{n} T_i(p, \sigma, d')$.*

Proof. We first show that the EDD ordering does not change by modifying the due dates according to Proposition 3.

Consider any two jobs l and j, so that $d_l \leq d_j$ and let $d'_i = \max(d_i, s_i)$, where s_i is the worst-case starting time of job $i, \forall i \in N$. We have two cases:

- $d'_l = d_l \leq d_j \leq d'_j$,
- $d'_l = s_l \leq s_j \leq d'_j$,

and $\forall l, j \in N \mid d_l \leq d_j \implies d'_l \leq d'_j$.

We conclude that the EDD schedule is unchanged and it will satisfy Proposition 3, therefore $\sigma^* \in \arg\min_{\sigma \in X} \max_{p \in U} \sum_{i=1}^{n} T_i(p, \sigma, d')$.

Now consider that we do not adjust due dates for jobs in EDD positions $k > 1$ for which $\min_{p \in U} p_{\sigma^*(k)} \geq \max(\max_{p \in U}(p_{\sigma^*(1)}), \ldots, \max_{p \in U}(p_{\sigma^*(k-1)}))$. As in Proposition 3, other arguments in [17] assure that jobs $\{\sigma^*(1), \ldots, \sigma^*(k-1)\}$ precede job $\sigma^*(k)$ and the EDD sequence is still optimal.

We show in the sequence that $z_{rob}(d') \leq z_{rob}(d)$. We have:

$$z_{rob}(d) = \min_{\sigma \in X} \max_{p \in U} \sum_{i=1}^{n} T_i(p, \sigma, d) \geq \max_{p \in U} \min_{\sigma \in X} \sum_{i=1}^{n} T_i(p, \sigma, d) = \max_{p \in U} z_{det}(p, d).$$

where the inequality follows from swapping the min and max terms. Additionally, by Proposition 1:

$$z_{rob}(d) \geq \max_{p \in U} z_{det}(p, d) \geq \max_{p \in U} z_{det}(p, d'). \tag{22}$$

Now consider that by definition $z_{rob}(d^{'}) = \max_{p \in U} \sum_{i=1}^{n} T_i(p, \sigma^*, d^{'})$ and by hypothesis $z_{det}(p, d^{'}) = \sum_{i=1}^{n} T_i(p, \sigma^*, d^{'}), \forall p \in U$, then we have:

$$z_{rob}(d^{'}) = \max_{p \in U} z_{det}(p, d^{'}). \tag{23}$$

Finally from (22) and (23):

$$z_{rob}(d^{'}) \leq z_{rob}(d). \tag{24}$$

□

As already seen, each node level l of our search tree is composed of a set J_l filling the last l positions in a given order. Since the order of the last l positions is defined, we concentrate on an EDD sequencing for the first $n - l$ positions. We order by due date the first $n-l$ positions. We relax their due dates using the rules defined above. We concatenate the solution above, using modified relaxed due dates, with the other sequenced jobs of the node, using original due dates and use the worst-case evaluation methods presented in Section 4 to find the associated lower bound.

The pseudocode used for our branch-and-bound algorithm is presented in Algorithm 2.

Algorithm 1 Upper bound heuristic

Input // Γ and vectors $\bar{p}, \hat{p}, d$
$SJ \leftarrow$ void // Sequence of selected jobs
for $k = n$ to 1 **do**
  $ALJ[k] = \{i \in N \backslash SJ \mid A_i \subseteq SJ\}$ // Allowable jobs for position k
  $JR_i = \frac{\max(0, \max_{p \in U} \sum_{j \in N \backslash SJ} p_j - d_i)}{\bar{p}_i + \hat{p}_i}, i \in ALJ[k]$ // Store ratio job i
  if $\min_i JR_i = 0$ **then** $SJ \leftarrow SJ \cup \arg\max_{i \in ALJ[k], JR_i = 0} \bar{p}_i + \hat{p}_i$
  else $SJ \leftarrow SJ \cup \arg\min_{i \in ALJ[k]} JR_i$ // job selected for position k
Calculate maximum total tardiness for sequence SJ //Worst-case solution
Return solution - sequence of jobs SJ and maximum total tardiness value

4. Worst-case evaluation

In this section we discuss how to perform an evaluation of the worst-case realization of the uncertainty set given a sequence of jobs. Recall that it is used to solve the adversarial separation problem in the row-and-column generation algorithm and also used in our branch-and-bound algorithm. This problem, for a given sequence of jobs $\sigma = \{\sigma(k), k = 1, \ldots, n\}$, where k represents a

Algorithm 2 Branch-and-bound algorithm

```
Input                                          // Γ and vectors p̄, p̂, d
Initialize
//Nodes list ← root node, Incumbent solution ← void, Lower bound ← −∞
while There are still nodes to be branched in the Nodes list do
    Node Select                                // Select node based on search criteria
    Prune                                      // by lower bound
    Update Incumbent solution                  // use best upper bound so far
    Branch node
    Prune                                      // by dominance rules
    Calculate lower and upper bound            // of new nodes
    Update Nodes List
Return optimal solution - sequence of jobs and total tardiness value
```

position and $\sigma(k)$ represents the job that occupies that position, is defined by the worst-case total tardiness, T^*_σ, associated with this sequence and given by:

$$T^*_\sigma = \max_{p \in U} \left(\sum_{k=1}^{n} \max \left(0, \sum_{k'=1}^{k} p_{\sigma(k')} - d_{\sigma(k)} \right) \right) \tag{25}$$

We assume that maximal deviations of processing times of our budgeted uncertainty set are integers. Also, for the sake of simplicity, we consider in what follows that Γ is a non negative integer. As we will see, the budgeted uncertainty set allows us to explore some properties that simplify complexity of our algorithms.

Using these assumptions, statement (25) reflects a problem with a convex function being maximized over a polytope defined by uncertainty set U. Hence, to define the worst-case robust realization of uncertainty we only have to take into account specific realizations of the uncertainty set given by:

- Extreme points of the polytope. For each job i, we only consider values $\bar{p}_i$ and $\bar{p}_i + \hat{p}_i$
- It is clear that any worst-case realization will use as much budget of uncertainty as possible. Hence we can assume $\sum_i \xi_i = \Gamma$

We work two alternative methods to evaluate a worst-case solution value: one based on dynamic programming and another based on a simple heuristic.

4.1. Dynamic programming

We adopt a dynamic programming algorithm developed by [2] for a general class of robust optimization problems. The complexity of this algorithm is $O(n\Gamma\bar{\Phi})$, where $\bar{\Phi}$ is the maximum cumulative processing time deviation allowed, that is $\bar{\Phi} = \max_{S \subseteq N : |S| = \Gamma} \sum_{i \in S} \hat{p}_i$. It is favored when processing time deviations

are small. We verify that the optimal solution for the adversarial problem only depends on the cumulative deviations of the job processing times. Let us define a value-function $\alpha(k, \gamma, \Phi)$, where $1 \leq k \leq n$, $0 \leq \gamma \leq \Gamma$, $0 \leq \Phi \leq \bar{\Phi}$, as the optimal value of the restricted problem for a set of jobs positions $\{1, \ldots, k\}$ with at most γ deviations and a cumulative deviation Φ. The optimal value of the adversarial problem is defined by $T^*_\sigma = \max_{\Phi \in \{0,\ldots,\bar{\Phi}\}} \alpha(n, \Gamma, \Phi)$. Furthermore, we see that the value-function satisfies the recursion:

$$\alpha(k,\gamma,\Phi) = \begin{cases} \max(0,\, \Phi + \sum_{k'=1}^{k} \bar{p}_{\sigma(k')} - d_{\sigma(k)}) + \alpha(k-1,\gamma,\Phi), & \text{if } \Phi - \hat{p}_{\sigma(k)} < 0 \\ \max(0,\, \Phi + \sum_{k'=1}^{k} \bar{p}_{\sigma(k')} - d_{\sigma(k)}) + \\ \qquad \max(\alpha(k-1,\gamma,\Phi),\, \alpha(k-1,\gamma-1,\Phi-\hat{p}_{\sigma(k)})), & \text{if } \Phi - \hat{p}_{\sigma(k)} \geq 0 \end{cases}$$

Also, the following statements are used to initialize the dynamic programming table:

$$\begin{aligned} &\alpha(1,0,0) = \max(0,\, \bar{p}_{\sigma(1)} - d_{\sigma(1)}) \\ &\alpha(1,\gamma,\hat{p}_{\sigma(1)}) = \max(0,\, \hat{p}_{\sigma(1)} + \bar{p}_{\sigma(1)} - d_{\sigma(1)}), \quad 1 \leq \gamma \leq \Gamma \\ &\alpha(1,\gamma,\Phi) = -\infty, \text{ for remaining cases} \\ &\alpha(k,0,0) = \max(0,\, \sum_{k'=1}^{k} \bar{p}_{\sigma(k')} - d_{\sigma(k)}) + \alpha(k-1,0,0), \quad 2 \leq k \leq n \\ &\alpha(k,0,\Phi) = -\infty, \quad 2 \leq k \leq n,\, 1 \leq \Phi \leq \bar{\Phi} \end{aligned}$$

In [2] the authors show that their dynamic programming approach outperforms the rather traditional approach of using a MILP formulation to solve the worst-case evaluation subproblem. For our problem, we have also performed comparisons between the two approaches and the overall results were favorable to the dynamic programming approach, so that we discarded the MILP approach from our study.

4.2. Heuristic

We implement a simple greedy algorithm of complexity $O(\Gamma n^2)$ to obtain a lower bound on T^*_σ. The pseudocode for this heuristic is presented in Algorithm 3. Given a sequence of jobs, we execute an algorithm with Γ iterations. At each iteration we define a job to have its processing time deviated to its maximum. Following are the steps executed:

- At each one of the Γ iterations, we verify total tardiness originated by setting each one of the n jobs processing time to its maximum (if not already considered in previous iteration as deviated) and pick the one that provided maximum total tardiness.
- If total tardiness is not augmented in relation to the previous iteration we set the processing time of the first job in the sequence not already deviated to its maximum.

Algorithm 3 Worst-case evaluation heuristic

Input
//Sequence of jobs: $\sigma = \{\sigma(k), k = 1, \ldots, n\}$, Γ and vectors $\bar{p}, \hat{p}, d$
$D \leftarrow$ void // Set of jobs with deviated processing times
$MAXTT = -\infty$ // Worst-case total tardiness
for $g = 1$ to Γ **do**
$TT_i(D) = \sum_{j=1}^{n} \max(0, (\sum_{k \leq \sigma^{-1}(j)} \bar{p}_{\sigma(k)} + \sum_{\substack{k \leq \sigma^{-1}(j) \\ \sigma(k) \in D \cup \{i\}}} \hat{p}_{\sigma(k)} - d_j)),$ for $i \in N \backslash D$
if $\max_i \{TT_i(D)\} > MAXTT$ **then** $D \leftarrow \arg\max \{TT_i(D)\}; MAXTT = \max_i \{TT_i(D)\}$
else $D \leftarrow \arg\min_{i \in N \backslash D} \{\sigma^{-1}(i)\}$
Total tardiness $= MAXTT$
Return Total tardiness

It can be used as a lower bound when the performance of an exact solution in the algorithm is an issue.

5. Dominance rules

Dominance rules have been extensively used in the past in combinatorial optimization problems and specially in scheduling problems [18]. A dominance rule is established in order to reduce the solution space either by adding new constraints to the problem, or by writing a procedure that attempts to reduce the domain of the variables, or by building interesting solutions directly.

One of the main theoretical developments for the total tardiness problem were the dominance rules derived by [17]. The author proved three fundamental theorems that helped establish precedence relations among job pairs that must be satisfied in at least one optimal schedule. These dominance rules are a major component of existing state-of-art algorithms. All the three theorems assumes the ordering of jobs by their processing times. Although we can naturally apply these rules to our robust problem for jobs that do not overlap, that is, given two jobs $i, j, \max_{p \in U} p_i < \min_{p \in U} p_j$, there is a tendency to have the applicability of these rules reduced since processing times are uncertain.

Later, [30], working with single machine weighted total tardiness problem, extended these dominance conditions to more general forms where the ordering of processing times are not always necessary, and so, are more adequate to be extended to our robust problem where processing times are uncertain.

To extend dominance conditions of [30] to our robust problem we have to consider definitions that follow, where B_i and A_i are sets of jobs known to precede job i and follow i, respectively. Also for notation purposes, for any set Q, $Q \subseteq N$, the notation $P(Q)$ represents $\sum_{i \in Q} p_i$.

- We order two jobs i, j that do not overlap. That is $i \prec j$ if $\max_{p \in U} p_i < \min_{p \in U} p_j$.

- We define the earliest completion date of a job i, $E_i = \min_{p \in U} (P(B_i) + p_i)$
- We define the latest completion date of a job i, $L_i = \max_{p \in U} P(N \setminus A_i)$

Using definitions above, conditions of [30] can be restated as follows.

Proposition 5. *Job i precedes job j in at least one robust optimal schedule if at least one of the conditions below are satisfied:*

1. $i \prec j$ *and* $d_i \leq \max(E_j, d_j)$
2. $d_j \geq \max(L_i - \min_{p \in U} p_j,\ d_i)$
3. $d_j \geq L_i$

Proof. Extending the dominance rules defined by [30] to our robust problem is based on the idea that if a rule is valid for a predefined worst-case event of uncertainty it will be valid for all realizations of uncertainty. In other words, if a job i precedes job j in a worst-case predefined event, it will precede for all realizations of uncertainty. Hence, this additional restriction can be added to the overall robust problem.

For the sake of completeness we restate below the dominance conditions defined in [30]:

- At least one optimal schedule has job i preceding job j if $d_i \leq \max(P(B_j) + p_j, d_j)$, $\alpha_i \geq \alpha_j$ and $p_i \leq p_j$ [30, Corollary 1].
- At least one optimal schedule has job i preceding job j if $d_j \geq P(N \setminus A_i) - p_j$, $d_j \geq d_i$ and $\alpha_i \geq \alpha_j$ [30, Corollary 2].
- At least one optimal schedule has job i preceding job j if $d_j \geq P(N \setminus A_i)$ [30, Corollary 3].

where α_i, $i \in N$ are the weight coefficients for the weighted tardiness problem that in our case are equal to 1.

For item 1 of Proposition 5 one can verify:

$$\max_{p \in U} p_i < \min_{p \in U} p_j \text{ and } d_i \leq \max(\min_{p \in U} (P(B_j) + p_j), d_j) \implies$$
$$p_i \leq p_j \text{ and } d_i \leq \max(P(B_j) + p_j, d_j), \forall p \in U$$

For item 2 of Proposition 5 one can verify:

$$d_j \geq \max_{p \in U} P(N \setminus A_i) - \min_{p \in U} p_j \text{ and } d_j \geq d_i \implies$$
$$d_j \geq P(N \setminus A_i) - p_j \text{ and } d_j \geq d_i, \forall p \in U$$

For item 3 of Proposition 5 one can verify:

$$d_j \geq \max_{p \in U} P(N \setminus A_i) \implies d_j \geq P(N \setminus A_i), \forall p \in U.$$

This way Corollary 1,2 and 3 in [30], respectively, are satisfied by all realizations of the uncertainty set. Hence, they can be extended to the robust problem as defined in Proposition 5.

□

By applying these rules successively we can populate, for each job i, the set of jobs known to precede i, B_i and follow i, A_i in some optimal sequence. The idea is that as we run rules above, and pair of jobs are ordered, the sets A_i and B_i will grow, favoring new runs.

These dominance conditions can be incorporated in our MILP formulations as precedence constraints or can be used to prune non optimal node sequences in our branch-and-bound algorithm. They can also be applied dynamically, during branching decisions at each node. If, applied for the subproblem of sequencing jobs $j \in N \backslash J_l$, they identify that there is job $i \in N \backslash J_l$ that precedes a job j, then job i can be eliminated and job j considered for the $|N \backslash J_l|$-th position.

Calculating latest and earliest completion times for each subproblem can, though, add non desired computational complexity. Many proposed algorithms avoid this additional complexity by applying at each node only a relaxed version of the third condition of Proposition 5, called Elmaghraby's lemma [16], where the latest completion time of each job $i \in N \backslash J_l$, L_i, is relaxed to makespan of the subproblem. In other words, if there is a job j that has zero tardiness for the last position ($d_j \geq$ makespan), it can be considered for branching and all other jobs $i \in N \backslash J_l$ eliminated (for examples, see [30] and [27]).

We experiment a compromise between these two previous approaches. We relax the latest and earliest completion dates of all jobs $j \in N \backslash J_l$ by considering maximal ($\max_{p \in U} P(N \backslash J_l)$) and minimal ($\min_{p \in U} P(N \backslash J_l)$) makespan, and setting $L_j^{adj} = \min(L_j, \max_{p \in U} P(N \backslash J_l))$ and $E_j^{adj} = \min(E_j, \min_{p \in U} P(N \setminus J_l))$, where L_j^{adj} and E_j^{adj} are the adjusted latest and earliest completion dates for job j, and L_j and E_j are the latest and earliest completion dates for job j calculated for the original set of jobs N. Proposition 5, adjusted, can be used to identify a job $j \in N \backslash J_l$ that succeeds other jobs of set $N \backslash J_l$. If that job j is found, the number of new nodes during branching can be reduced.

Observation 5.1. *If a job $i \in N \backslash J_l$ satisfies one of the conditions below, for some job $j \in N \backslash J_l$, $i \neq j$, then job i can be eliminated during branching at level l of the search tree.*

- $i \prec j$ *and* $d_i \leq \max(E_j^{adj}, d_j)$
- $d_j \geq \max(L_i^{adj} - \min_{p \in U} p_j,\ d_i)$
- $d_j \geq L_i^{adj}$

As a remark, any assignment of positions should be in alignment with precedence constraints already generated.

6. Implementation and results

6.1. Implementation details

Instances We construct instances based on those of the literature, using the same directives as in [27]. We vary hardness of problem solving using parameters R, relative range of due dates and T, average tardiness factor. These parameters are used to define the average and range of variation of due dates. Due date range significantly affects the time performance of algorithms. Due dates widely distributed are easier to solve. The values of R are chosen as $R = \{0.2, 0.6, 1.0\}$, and the values of T are chosen as $T = \{0.2, 0.6, 0.8\}$. With $P = \sum_{i=1}^{n} \bar{p}_i$ and chosen R and T, we generate a non negative integer due date d_i from the uniform distribution $[P(1 - T - R/2), P(1 - T + R/2)]$ for each job i.

We define a new parameter G, to control the relative range of variation of uncertainty. By range of variation of uncertainty we mean the difference in values of Total Tardiness when we change from one realization of uncertainty to another. The parameter G is an artificial measure introduced to control the values of the total allowed deviation, Γ, and the percentage of allowed deviation from nominal processing times for each job. A small Γ and large process deviation time potentially produce larger differences in Total Tardiness when we change from one realization of uncertainty to another. Small G corresponds to small Γ and large process deviation time. The values of G are as $G = \{10, 100\}$. For each job i, an integer processing time $\bar{p}_i$ is generated from the uniform distribution $[1, 100]$ and an integer processing deviation time $\hat{p}_i$ is generated from the uniform distribution $[\bar{p}_i \frac{2}{G}, \bar{p}_i \frac{7}{G}]$. The value of Γ is an integer generated from the uniform distribution $[5 \times 10^{-3} Gn, 9 \times 10^{-3} Gn]$.

Using a sample size of ten for each of the 18 combinations of R and T and G we generate 180 instances for problems with 20, 40, 60 and 100 jobs, giving a total of 720 instances tested.

Algorithms Specification Algorithms are coded in Julia [22] using JuMP package and Cplex 12.7. All algorithms run in an Intel CORE i7 CPU 3770 machine. A limit of 9600 seconds of computing time is given for each instance.

All algorithms are tested on the same set of instances. We test MILP formulations (sequence-position and linear ordering) and branch-and-bound algorithms here developed. MILP formulations run under a row-and-column generation method where the separations problems are solved using the dynamic programming algorithm here presented. Each MILP formulation algorithm runs also with the option of using dominance rules to insert precedence constraints.

The branch-and-bound algorithm runs with the options of best bound or depth first search strategy. Upper bounds are calculated at root level and level $n-1$ of our search tree, as described in Section 3. Lower bounds are calculated using the dynamic programming method only at level 1. At level $n-1$ the lower bound is equal to the upper bound so that we do not have to recalculate. At other levels of the search tree the lower bound is approximated using the heuristic method of Section 4 to improve performance. Dominance rules are used to create precedence between jobs and prune nodes during branching. We

also apply dynamically, at each node, the two last conditions of Observation 5.1.

The name and configuration of each algorithm is presented in Table 1.

6.2. Comparative performance of the algorithms

We first present in Table 2 general results comparing performance of the algorithms for all instances. Algorithm $BB1$ is the one that solves the majority of the instances with best time. $\%Best\ Performance$ measurement indicates that the algorithm $BB1$ solved 61% of instances with best performance, followed by algorithm $SEQ2$, that solved 37%.

Measurement $\%\,Solved\,100$ indicates that all algorithms were not able to solve the majority of the 100 jobs instances within the time limit. The other presented measurements ($\%\,Solved$, T_{median}, $Avg\,\%\,Gap$) favor algorithms $SEQ1$ and $SEQ2$. We present medians in order to mitigate the effect of instances not solved within the time limit.

Figure 1 presents a comparison of all algorithms using a performance profile [15]. In this figure, the vertical axis points out in percentage, for each algorithm, in how many instances the result was not more than x times - horizontal axis - worse than the best algorithm. For $x = 1$, the indicators replicates the best performance indicators presented in Table 2.

Figure 1 evidences that each algorithm has different time variation characteristics. In particular, algorithm $LIN1$ and $SEQ1$ present the highest ascending slopes, indicating that, although they are not best performers as measured by $\%Best\ Performance$, they present the characteristic of less solution time variance among all algorithms.

Table 2 and Figure 1 evidence that algorithm $BB1$ solves the majority of the instances with best time, but it is due to conditions that favor its core characteristics. When these conditions are not satisfied, other algorithms are favored. To analyze these conditions we group our instances and verify the performance of algorithms.

For this we present Table 3 where the same indicators are listed, but now grouped by special instances. We follow concepts defined in [27] and group instances based on their R, relative range of due dates and T, average tardiness factor. Instances with ratio T/R greater than 1 are classified as "High hardness" and classified as "Low hardness" otherwise. In general "Low hardness" instances will be more completely classified by dominance rules as stated in [27]. We also group instances by their G value. Instances with $G = 10$ are classified as "Large uncertainty range" and as "Small uncertainty range" otherwise. "Large uncertainty range" instances are the ones that, in general, have large solution value distance between one realization of uncertainty to others. These groups are not disjoint.

Analysis of Table 3 shows that algorithm $SEQ2$ has best $\%Best\ Performance$ indicator for "High hardness" instances while $BB1$ has best $\%Best\ Performance$ indicator for "Low hardness" instances and "Large uncertainty range" instances.

These results so far are expected since, on one side, the branch-and-bound algorithm relies heavily on dominance conditions to prune nodes and this is favored in "Low hardness" instances. On the other hand, our MILP RCG algorithms are not favored by "Large uncertainty range" instances, since in general it will require more calls to separation problems.

This effect can be better verified in Figure 2 where we present performance profiles for each group of instances. In general, now grouped by special instances, the algorithms with $\%Best\ Performance$ indicator show consistency of performance along the x-axis. It evidences, for "High hardness" instances, that when characteristics as dominance conditions are not favored, the sequence-position formulation is privileged. Algorithm $LIN1$ is privileged for "Small uncertainty range" instances.

These results suggest that, in general, if an instance is not well defined by dominance rules, a "High hardness" instance, $SEQ2$ is favored. If not, if an instance is well defined by dominance rules, a "Low hardness" instance, the characteristic of uncertainty range will define the algorithm to be favored: "Large uncertainty range" instances are favored by $BB1$ and, "Small uncertainty range" instances are favored by $LIN1$.

In Figure 3 we present different performance profiles, now comparing each MILP RCG method among their different configurations. We can observe that both sequence-position and linear ordering formulations had better performance when precedence constraints, based on dominance rules, were added. The effect of precedence constraints was most pronounced in $LIN1$ algorithm. This can be partially explained because the precedence constraints added for the $LIN2$ algorithm (see equation 19) are effective to restrict the search of the CPLEX linear programming algorithm to a reduced set of extreme points. For the sequence-position formulation, although the presence of precedence constraints in $SEQ1$ helped it to present the highest ascending slope among the two, it is $SEQ2$ that is the best performer, although only sligthy above $SEQ1$.

Table 4 presents average time performance of each MILP algorithm. Here we present average as mean measures because we are interested in contributions of all instances (even outliers).We also present standard deviation as a secondary measure. Time performance is split between master and adversarial problems. We also present average number of iterations, split between "Large uncertainty range" and "Small uncertainty range" instances. These results evidence that the critical step for these algorithms is the master problem resolution, even with the addition of precedence constraints. We see also that the number of iterations are greater for "Large uncertainty range" instances, as it was designed to be.

Figure 4 presents performance profile for our two branch-and-bound algorithms. Algorithm $BB1$, based on a depth first search strategy, clearly outperforms algorithm $BB2$, based on a best bound search strategy. To analyze this effect we present in Table 5 detailed results for our branch-and-bound algorithms. It shows that the dominance rules were effective to prune nodes in both strategies. It also shows that the best bound strategy was not successful as the depth first strategy to prune nodes by its lower bound. In fact, by the way our branch-and-bound algorithms were implemented, lower bound values improves

Algorithm Name	Method	Dominance Rules	Best Bound Strategy	Depth First Strategy
Seq1	Sequence MILP RCG	Yes	-	-
Seq2	Sequence MILP RCG	No	-	-
Lin1	Linear MILP RCG	Yes	-	-
Lin2	Linear MILP RCG	No	-	-
BB1	Branch-and-bound	Yes	No	Yes
BB2	Branch-and-bound	Yes	Yes	No

RCG meaning row-and-column generation

Table 1: Algorithms

as the algorithm reaches the leaves of the tree and that favors the depth first strategy. It is also a consequence of our choice of a lower bound algorithm that is easy to calculate but not tight. On the other hand, depth first search strategy is also able to find an upper bound more quickly, which helps to improve performance.

6.3. Assessing the robustness

We assess below the costs provided by different models under different uncertainty sets on two instances with 20 jobs. The six models considered are related to the robust model used and to the value of $\Gamma \in \{0, 5, 10, 20\}$. The model with $\Gamma = 0$ is the deterministic model that ignores uncertainty, while the one with $\Gamma = 20$ is the deterministic model that is completely risk-averse and overestimates all parameters. The other values of Γ model intermediate risk aversions by using either the robust model studied here, or the simpler static model obtained by adding the constraints $T_i(p, \sigma) = T_i(p', \sigma)$ for each $i \in N$ and $p, p' \in U$. Let σ^Γ and σ^Γ_{stat} be the solutions obtained by the robust models for the value $\Gamma \in \{10, 15\}$, and denote similarly the deterministic solutions by σ^0_{det} and σ^{20}_{det}. We report on Figure 5 the costs $T^\Gamma(\sigma) = \max_{p \in U} \sum_{i \in N} T_i(p, \sigma)$ on two instances that illustrate the general patterns that can be observed.

A "Large uncertainty range" instance, where there are large solution value distances between one realization of uncertainty to others is presented. For this instance, varying the level of conservativeness, that is, varying Γ and the number of jobs processing times that can vary, have significant impact on total tardiness. From Figure 5 we see that the nominal solution, as well as static solutions, are suboptimal for $\Gamma \in \{1, \ldots, 7\}$. Specifically, for these values of Γ, σ^5 is roughly 20% cheaper than σ^5_{stat}, and the ratio increased when considering the other models. For larger values of Γ, the cheapest solution is σ^{20}_{det}, with the robust solutions σ^{10} and σ^{10}_{stat} being roughly 5% more expensive. We also see that the deterministic solution σ^0_{det} behaves extremely badly as soon as $\Gamma > 1$. To summarize, analyzing these two figures show that the robust solution σ^5 should be preferred because it is never far from being the cheapest one.

A "Small uncertainty range" instance example is also presented. For this instance, varying levels of conservativeness do not have a great impact on the total tardiness.

Indicator	Seq1	Seq2	Lin1	Lin2	BB1	BB2
% Best Performance	31	37	33	22	61	21
$\%\,Solved$	73	70	63	47	67	41
$\%\,Solved\,100$	44	44	31	31	40	25
$Avg\,\%\,Gap$	8	5	4	10	10	33
T_{median}	37.84	30.93	69.97	9600	41.63	9600

% Best Performance is percentage of total instances where algorithm was best in time performance
% $Solved$ is percentage of total instances solved within time limit
% $Solved\,100$ is percentage of total 100 jobs instances solved within time limit
$Avg\,\%\,Gap$ is average percentage gap of solutions not solved to optimality
T_{median} is median solution time (s)

Table 2: Performance of algorithms for all instances

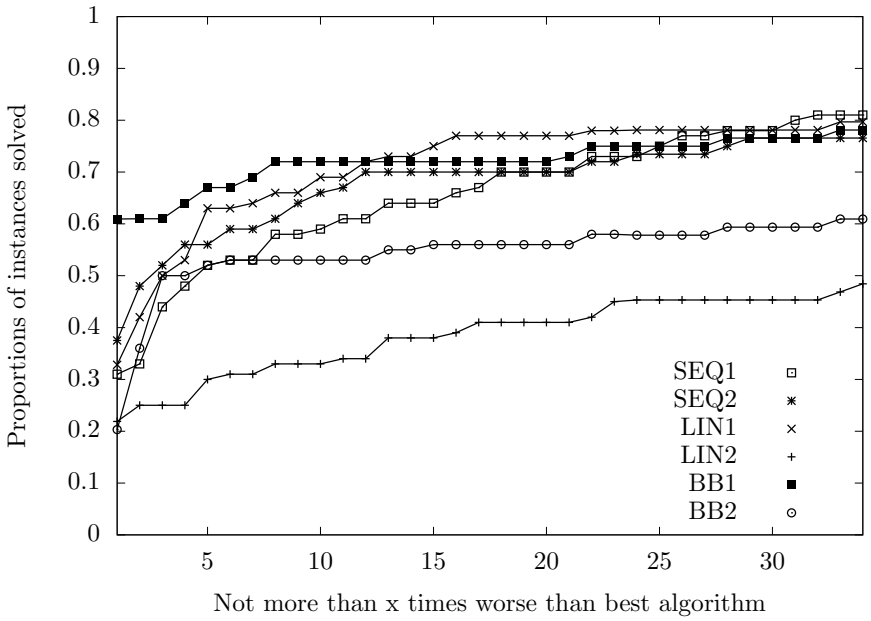

Figure 1: Performance profile among all instances

Instances group	Indicator	Seq1	Seq2	Lin1	Lin2	BB1	BB2
High Hardness	% Best Performance	47	56	47	28	38	28
	$\%\,Solved$	63	63	44	16	43	13
	$\%\,Solved\,100$	38	38	0	0	0	0
	$Avg\,\%\,Gap$	0	0	37	10	9	35
	T_{median}	248.25	270.32	9600	9600	9600	9600
Low Hardness	% Best Performance	16	19	19	16	85	13
	$\%\,Solved$	84	78	81	78	88	69
	$\%\,Solved\,100$	63	50	63	63	63	50
	$Avg\,\%\,Gap$	17	8	15	8	14	29
	T_{median}	26.31	18.45	2.26	14.11	0.62	1.21
Large Uncertainty Range	% Best Performance	53	31	37	31	75	31
	$\%\,Solved$	63	63	50	44	57	34
	$\%\,Solved\,100$	35	35	25	25	30	30
	$Avg\,\%\,Gap$	8	16	42	15	11	31
	T_{median}	105.91	223.78	5074.54	9600	201.79	9600
Small Uncertainty Range	% Best Performance	10	44	28	12	50	10
	$\%\,Solved$	84	78	75	50	63	47
	$\%\,Solved\,100$	63	50	38	38	38	25
	$Avg\,\%\,Gap$	0	0	28	0	11	34
	T_{median}	35.03	6.33	9.56	8400.55	146.45	9600

% Best Performance is percentage of total instances of a group where algorithm was best in time performance
$\%\,Solved$ is percentage of total instances of a group solved within time limit
$\%\,Solved\,100$ is percentage of total 100 jobs instances of a group solved within time limit
$Avg\,\%\,Gap$ is average percentage gap of solutions not solved to optimality
T_{median} is median solution time (s) considering all instances of a group

Table 3: Performance of algorithms, grouping by special instances

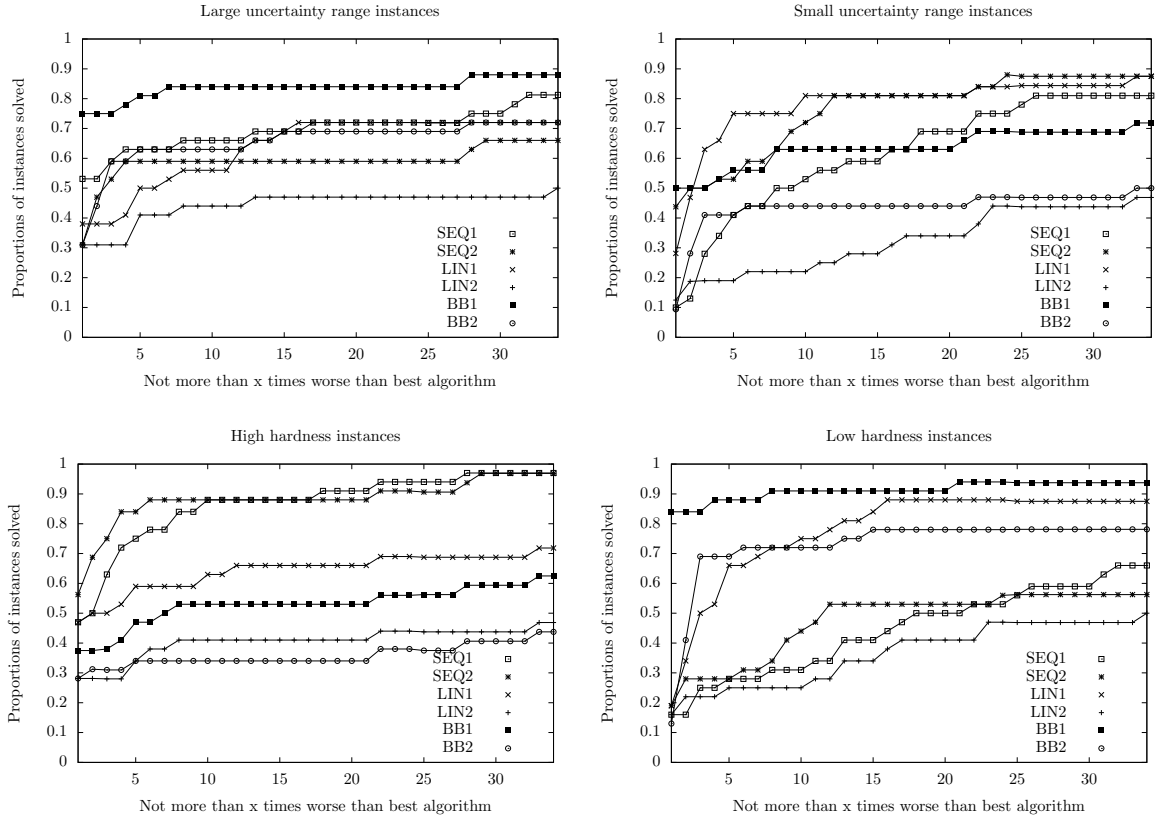

Figure 2: Performance profile of algorithms, grouping by special instances

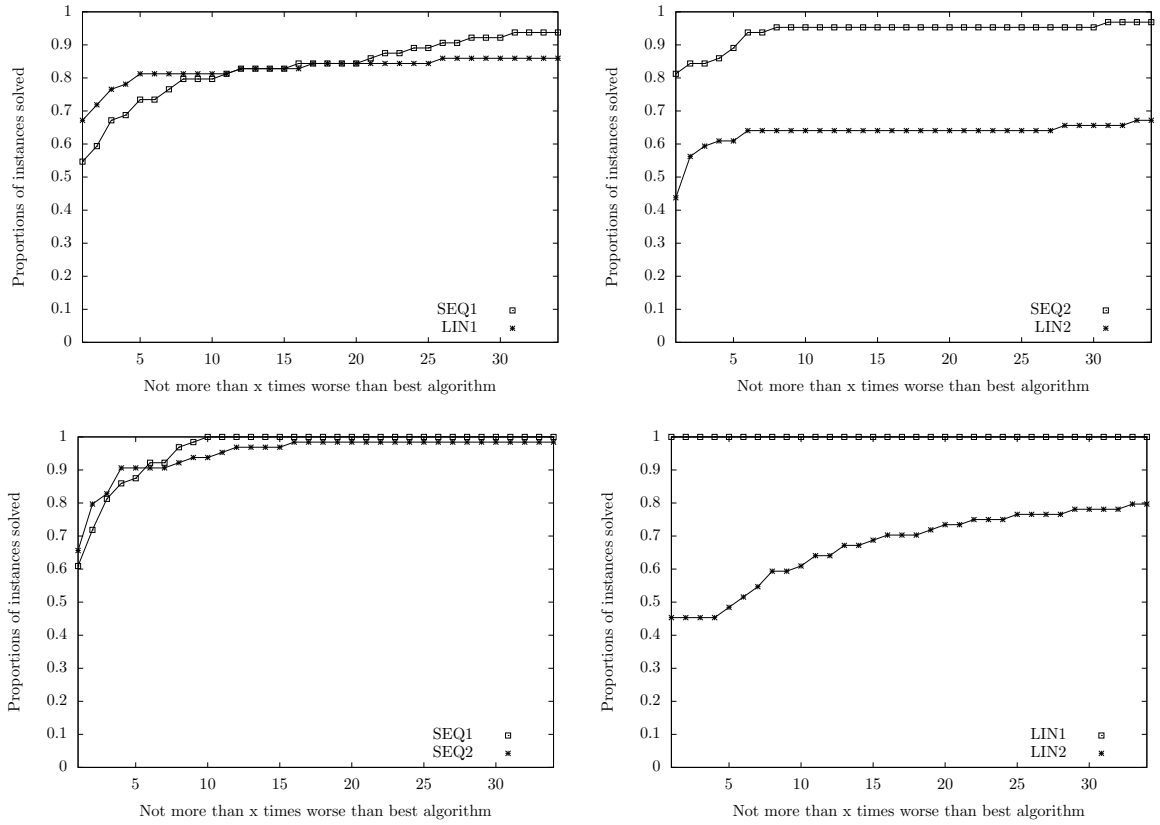

Figure 3: Performance profile MILP methods and configurations

Indicator	Seq1	Seq2	Lin1	Lin2
Master Problem Time*	2862.88/4803.15	3436.01/5166.87	3872.16/5045.17	5138.58/4572.73
Adversarial Problem Time*	82.57/187.91	3.78/7.07	0.47/1.32	0.92/2.15
LUR Number of Iterations* **	5.93/6.89	6.81/6.26	2.28/2.09	3.00/2.19
SUR Number of Iterations* ***	1.75/0.43	1.90/0.50	1.65/0.49	1.78/0.49

** We present average value / standard deviation value.*
*** Large uncertainty range (LUR) instances only.*
**** Small uncertainty range (SUR) instances only.*

Table 4: MILP algorithms - all instances

Indicator	BB1	BB2
Nodes visited*	3511383/11695381	571012/683244
% Nodes pruned by dominance*	69/24	95/16
% Nodes pruned by lower bound*	20/18	0/14
Solution time*	4882.83/4774.04	6724.52/4712.30
% Solution gap**	10	33

** We present average value / standard deviation value.*
*** Average value calculated for non optimal solutions as percentage difference to best solution among all algorithms.*

Table 5: Branch-and-bound algorithms - all instances

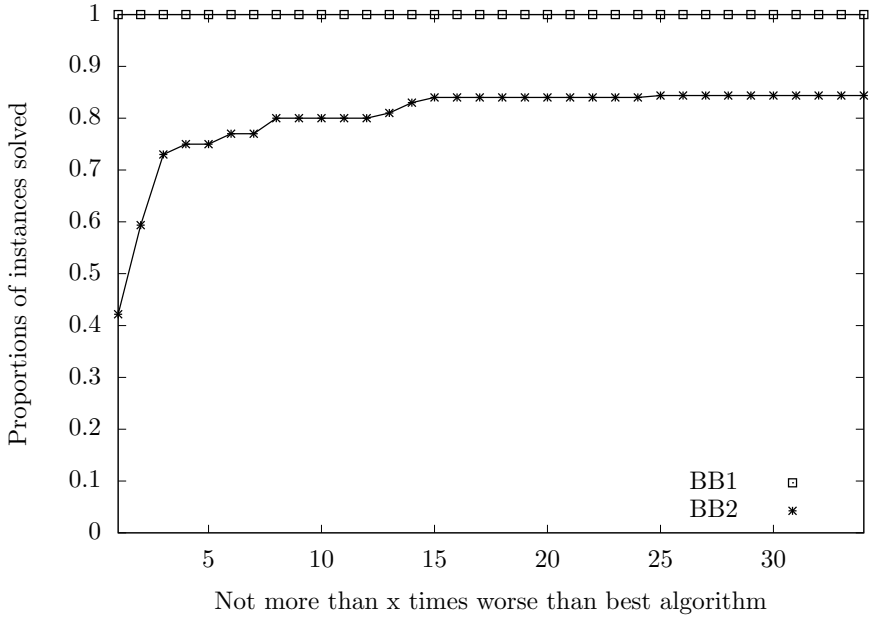

Figure 4: Algorithms BB performance profile among all instances

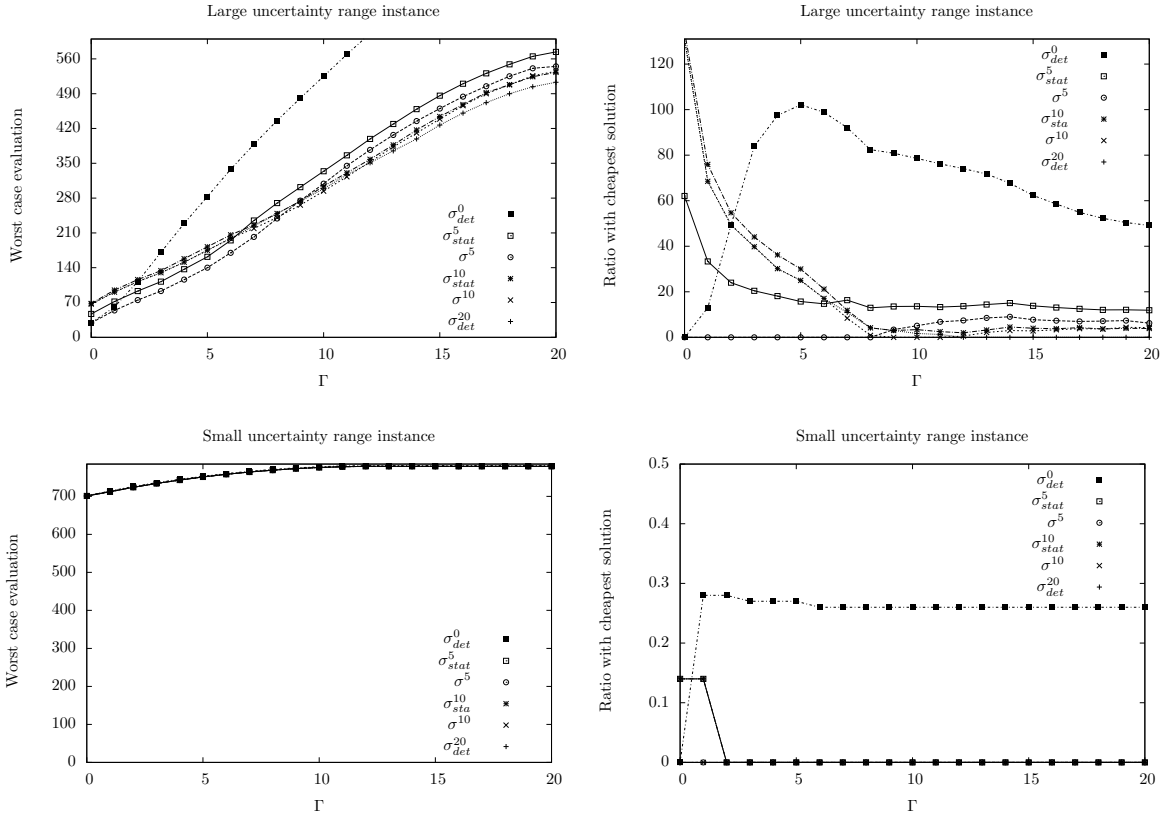

Figure 5: Worst-case evaluation for robust and nominal solutions

7. Conclusions

We develop a new branch-and-bound approach for robust single machine total tardiness problem representing uncertainty by a set, extending dominance rules and lower bounds concepts used for the deterministic case. We define MILP formulations for our problem and apply the dominance rules studied as additional precedence constraints for these formulations. We have experimented different algorithms and presented computational results where we were able to verify in which conditions algorithms better perform. More specifically we were able to identify:

- The dominance rules were fundamental for a good performance of our branch-and-bound algorithms and MILP formulations algorithms.
- We could verify characteristics of our instances that influence performance and suggest the best algorithm, in general, for each type of instance.
- Depth first search strategy was a key feature for our branch-and-bound algorithm.

We have shortly assessed the cost of the solutions returned by the robust model, and compared them to the deterministic solutions. The latter seem to indicate that:

- The deterministic solutions overestimating the parameters should be preferred over those underestimating the latter.
- The robust solution with $\Gamma = n/4$ seems to perform well under all circumstances. In our assessment of robustness, different from other models, it has showed to be never far from being the cheapest one. It also improved roughly 20% over the static model that has been used before in other contexts (e.g. [13]).

8. References

[1] Addis, B., Carello, G., Tànfani, E., 2014. A robust optimization approach for the Advanced Scheduling Problem with uncertain surgery duration in Operating Room Planning - an extended analysis.
URL `https://hal.archives-ouvertes.fr/hal-00936019`

[2] Agra, A., Santos, M. C., Nace, D., Poss, M., 2016. A dynamic programming approach for a class of robust optimization problems. SIAM Journal on Optimization 26 (3), 1799–1823.

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

[4] Ayoub, J., Poss, M., 2016. Decomposition for adjustable robust linear optimization subject to uncertainty polytope. Comput. Manag. Science 13 (2), 219–239.

[5] Baker, K. R., Keller, B., 2010. Solving the single-machine sequencing problem using integer programming. Computers and Industrial Engineering 59 (4), 730 – 735.

[6] Ben-Tal, A., Goryashko, A. P., Guslitzer, E., Nemirovski, A., 2004. Adjustable robust solutions of uncertain linear programs. Math. Program. 99 (2), 351–376.
URL https://doi.org/10.1007/s10107-003-0454-y

[7] Bertsimas, D., Sim, M., 2004. The Price of Robustness. Oper. Res. 52 (1), 35–53.

[8] Bougeret, M., Pessoa, A. A., Poss, M., 2019. Robust scheduling with budgeted uncertainty. Discrete Applied Mathematics 261, 93 – 107.

[9] Chaari, T., Chaabane, S., Aissani, N., Trentesaux, D., 2014. Scheduling under uncertainty: survey and research directions. In: International Conference on Advanced Logistics and Transport, ICALT.

[10] Coban, E., Heching, A., Hooker, J. N., Scheller-Wolf, A., 2016. Robust Scheduling with Logic-Based Benders Decomposition. Springer International Publishing, Cham, pp. 99–105.

[11] Daniels, R. L., Kouvelis, P., 1995. Robust scheduling to hedge against processing time uncertainty in single-stage production. Management Science 41 (2), 363–376.

[12] Della Croce, F., Tadei, R., Baracco, P., Grosso, A., 1998. A new decomposition approach for the single machine total tardiness scheduling problem. Journal of the Operational Research Society 49 (10), 1101–1106.

[13] Denton, B. T., Miller, A. J., Balasubramanian, H. J., Huschka, T. R., 2010. Optimal allocation of surgery blocks to operating rooms under uncertainty. Operations research 58 (4-part-1), 802–816.

[14] Detienne, B., 2018. Private communication.

[15] Dolan, E. D., Moré, J. J., 2001. Benchmarking optimization software with performance profiles. CoRR cs.MS/0102001.
URL http://arxiv.org/abs/cs.MS/0102001

[16] Elmaghraby, S. E., 1968. The one-machine sequencing problem with delay costs. Journal of Industrial Engineering 19, 105–108.

[17] Emmons, H., 1969. One-Machine Sequencing to Minimize Certain Functions of Job Tardiness. Operations Research 17 (4), 701–715.

[18] Jouglet, A., Carlier, J., 2011. Dominance rules in combinatorial optimization problems. European Journal of Operational Research 212 (3), 433–444.

[19] Keha, A. B., Khowala, K., Fowler, J. W., 2009. Mixed integer programming formulations for single machine scheduling problems. Computers & Industrial Engineering 56 (1), 357–367.

[20] Koulamas, C., 2010. The single-machine total tardiness scheduling problem: Review and extensions. European Journal of Operational Research 202 (1), 1–7.

[21] Lawler, E. L., 1977. A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. Annals of Discrete Mathematics 1, 331 – 342.

[22] Lubin, M., Dunning, I., 2013. Computing in Operations Research using Julia. CoRR abs/1312.1431.
URL http://arxiv.org/abs/1312.1431

[23] Mastrolilli, M., Mutsanas, N., Svensson, O., 2008. Approximating single machine scheduling with scenarios. In: Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques. Springer, pp. 153–164.

[24] 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.
URL https://doi.org/10.1007/s10852-006-9044-3

[25] Módos, I., Šůcha, P., Hanzálek, Z., 2016. Robust scheduling for manufacturing with energy consumption limits. In: 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA). pp. 1–8.

[26] Potts, C. N., Wassenhove, L. N. V., 1982. A decomposition algorithm for the single machine total tardiness problem. Operations Research Letters 1 (5), 177–181.

[27] Potts, C. N., Wassenhove, L. N. V., 1985. A branch and bound algorithm for the total weighted tardiness problem. Operations Research 33 (2), 363–377.

[28] Pritsker, A. A. B., Waiters, L. J., Wolfe, P. M., 1969. Multiproject scheduling with limited resources: A zero-one programming approach. Management science 16 (1), 93–108.

[29] Queyranne, M., Schulz, A., 1994. Polyhedral Approaches to Machine Scheduling. Preprint-Reihe Mathematik. TU, Fachbereich 3.

[30] Rinnooy Kan, A. H. G., Lageweg, B. J., Lenstra, J. K., 1975. Minimizing Total Costs in One-Machine Scheduling. Operations Research 23 (5), 908–927.

[31] Roos, E., den Hertog, D., Ben-Tal, A., de Ruiter, F. J., Zhen, J., 2018. Approximation of hard uncertain convex inequalities. Optimization Online URL http://www. optimization-online. org/DB_HTML/2018/06/6679. html.

[32] Szwarc, W., Della Croce, F., Grosso, A., 1999. Solution of the single machine total tardiness problem. Journal of Scheduling 2 (2), 55–71.

[33] Tadayon, B., Smith, J. C., 2015. Algorithms and complexity analysis for robust single-machine scheduling problems. J. Scheduling 18 (6), 575–592.

[34] Tadayon, B., Smith, J. C., 2015. Robust offline single-machine scheduling problems. Wiley Encyclopedia of Operations Research and Management Science.

[35] Tanaka, S., Fujikuma, S., Araki, M., 2009. An exact algorithm for single-machine scheduling without machine idle time. Journal of Scheduling 12 (6), 575–593.

[36] Yang, J., Yu, G., 2002. On the robust single machine scheduling problem. Journal of Combinatorial Optimization 6 (1), 17–33.

[37] Zeng, B., Zhao, L., 2013. Solving two-stage robust optimization problems using a column-and-constraint generation method. Operations Research Letters 41 (5), 457–461.

[38] Zhen, J., D. R. F. J., Den Hertog, D., 2017. Robust optimization for models with uncertain soc and sdp constraints. Optimization Online URL http://www.optimization-online.org/DB_FILE/2017/12/6371.pdf.