Exact Lexicographic Scheduling and Approximate Rescheduling

Dimitrios Letsios, Ruth Misener

[a]Department of Computing; Imperial College London; South Kensington SW7 2AZ; UK

Abstract

In industrial scheduling, an initial planning phase may solve the nominal problem and a subsequent recovery phase may intervene to repair inefficiencies and infeasibilities, e.g. due to machine failures and job processing time variations. This work investigates the minimum makespan scheduling problem with job and machine perturbations and shows that the recovery problem is strongly NP-hard, at least as hard as solving the problem with full input knowledge. We explore recovery strategies with respect to the (i) planning decisions and (ii) permitted deviations from the original schedule. The resulting performance guarantees are parameterized by the degree of uncertainty. The analysis derives from the optimal substructure imposed by lexicographic optimality, so lexicographic optimization enables more efficient re-optimization. We revisit state-of-the-art exact lexicographic optimization methods and propose a lexicographic optimization approach based on branch-and-bound. Numerical analysis using standard commercial solvers substantiates the method.

Keywords: Scheduling, Lexicographic Optimization, Exact MILP Methods, Robust Optimization, Price of Robustness

1. Introduction

Scheduling, i.e. the ubiquitous process of efficiently allocating resources to guarantee the system operability, requires optimization under uncertainty [56]. A machine may unexpectedly fail, a client may suddenly cancel a job, or a machine may complete a job significantly earlier than expected. *Robust optimization* is a major approach for scheduling under uncertainty assuming deterministic uncertainty sets

*{d.letsios, r.misener}@imperial.ac.uk; Tel: +44 (0) 20759 48315

[5, 8, 26]. Static robust optimization may impose hard constraints forbidding solutions which are highly likely to become infeasible [54], but produces conservative solutions compared to nominal ones obtained with full input knowledge.

Two-stage robust optimization mitigates conservatism by developing contingency plans and chooses one when the uncertainty is realized [6, 36, 9, 29, 10]. We investigate two-stage robust optimization with recovery which consists of first-stage and second-stage decisions revealed before and after uncertainty realization, respectively. Here, the recourse action is specified by solving a second-stage optimization problem. As illustrated in Figure 1, we compute an initial solution to a nominal scenario and modify the solution once the uncertainty is realized, i.e. after the disturbances occur and the input parameters are revealed. Figure 2 illustrates the setting: (i) an initial planning phase solves a nominal scheduling problem instance I_{init} and produces a solution S_{init}, then (ii) a subsequent recovery phase solves instance I_{new} by repairing inefficiencies and infeasibilities, e.g. from machine failures and job processing time variations. In particular, we consider the two-stage robust makespan scheduling problem [27, 35, 14] under uncertainty with a set $\mathcal{J}$ of jobs, where job $j \in \mathcal{J}$ is associated with processing time $p_j > 0$, that have to be assigned on a set $\mathcal{M}$ of parallel identical machines and the objective is minimizing $C_{\max} = \max_{i \in \mathcal{M}}\{C_i\}$, i.e. the maximum machine completion time a.k.a. makespan. This fundamental combinatorial optimization problem is strongly $\mathcal{NP}$-hard.

Two-stage robust optimization with recovery allows more flexible recourse and the extreme case optimizes the problem from scratch without using the first-stage optimization problem decisions. Significantly modifying the nominal solution may be prohibitive, e.g. resource-consuming file retransmission in distributed computing [57]. We therefore introduce *binding* and *free* optimization decisions. Binding decisions are variable evaluations determined from the initial solution after uncertainty realization. Free decisions are variable evaluations that cannot be determined from the initial solution, and are essential to ensure feasibility. For instance, assigning a job with a modified processing time is a binding decision because the planning phase specifies an assignment. Assigning a new job after uncertainty realization is a free decision because the planning phase specifies no assignment.

We focus on rescheduling strategies admitting limited binding decision modi-

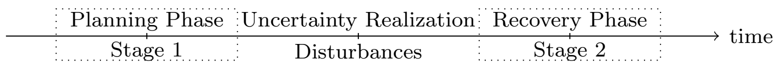


Figure 1: Recoverable robustness model

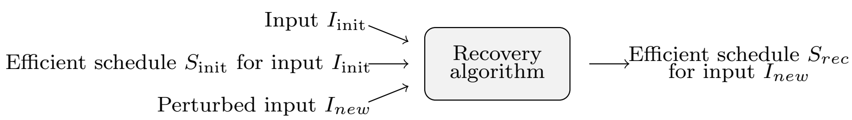


Figure 2: Makespan recovery problem

fications and thereby stay close to the planning phase solution. So, the *original makespan problem* is a standard optimization problem while the *makespan recovery scheduling problem* transforms an initial solution to a new solution with a bounded number of modifications. Because we allow limited decision modifications, first-stage decisions remain critical. Moreover, recovery flexibility allows efficient recourse with free decisions such as variable additions, which is not applicable in classical two-stage robust optimization.

A two-stage robust optimization method for solving a problem under uncertainty requires (i) an exact algorithm producing the initial solution, and (ii) a recovery strategy restoring the initial solution, after uncertainty realization. Analyzing a two-stage robust optimization method necessitates defining (i) the uncertainty set of the problem, and (ii) the investigated performance guarantee.

Uncertainty Set. A robust optimization problem may be harder than its deterministic counterpart [33]. But well-motivated uncertainty parameterizations enable tractability, e.g. bounded uncertainty where the final parameter values $\hat{p}_j$, e.g. processing times, vary in an interval $[p_j^L, p_j^U]$ and at most k parameters deviate from their initial, nominal values [11]. In this box uncertainty setting, the robust counterpart belongs to the same complexity class as its deterministic version for important problems. For analysis purposes, we generalize to an uncertainty model defined by a pair (k, f), separating between *stable* and *unstable* input parameters. Here, k is the number of unstable parameters with respect to perturbation factor $f > 1$. A parameter $p_j > 0$ is stable if $p_j/f \leq \hat{p}_j \leq f p_j$ and unstable, otherwise.

Performance Guarantee. Theoretical performance guarantees are important for determining when robust optimization is efficient [8, 26]. The *price of robustness* quantifies the quality of a robust solution or the performance of a robust algorithm, i.e. the ratio between the robust solution objective value and the optimal solution value obtained with full input knowledge [12]. Denote by $C(I_{new})$ the cost, e.g. makespan, of a robust solution obtained by some algorithm and by $C^*(I_{new})$ the cost of a nominal optimal solution obtained with full input knowledge [40]. We seek the tight, worst-case performance guarantee $\rho = \max_{I_{new} \in \mathcal{I}}(C(I_{new})/C^*(I_{new}))$.

Lexicographic Optimization. LexOpt is a subclass of multiobjective optimization [20, 45]. W.l.o.g., LexOpt minimizes m objective functions $F_1, \ldots, F_m : \mathcal{S} \to \mathbb{R}_0^+$, in decreasing priority order. In other words, LexOpt optimizes the highest-rank objective F_1, then the second most important objective F_2, then the third F_3, etc.:

$$\operatorname{lex}\min\{F_1(S), \ldots, F_m(S) : S \in \mathcal{S}\}. \tag{LexOpt}$$

There are indications that LexOpt is useful in optimization under uncertainty. LexOpt helps maintain a good approximate schedule when jobs are added and deleted dynamically, by performing reassignments [48, 53]. LexOpt is also useful for cryptographic systems against different types of attacks [58]. We consider the makespan problem generalization $\operatorname{lex}\min\{C_1(S), \ldots, C_m(S) : S \in \mathcal{S}\}$ of computing a schedule S with lexicographically minimal machine completion times and we show that it enables more efficient two-stage robust scheduling. That is, we identify robust scheduling as a new LexOpt application.

Apart from optimization under uncertainty, the design of efficient LexOpt methods is motivated by other LexOpt applications: equitable allocation of a divisible resource [37, 25], fairness [13], and selecting strategies that exploit the opponent mistakes optimally in a game theoretic context [50, 42]. Solution strategies include sequential, weighting, and highest-rank objective methods [18, 51, 52, 15, 44, 20, 45]. There is also work characterizing the convex hull of important LexOpt problems [41, 1, 28]. Logic-based methods are also applicable [38].

Contributions. We study makespan scheduling under uncertainty. Section 2.1 defines the minimum makespan problem, which schedules a set of jobs on parallel machines. Section 2.2 defines LexOpt scheduling. We therefore consider existing, exact LexOpt methods and Section 3 develops a novel LexOpt branch-and-bound method. We also propose a recovery strategy with positive performance guarantees. On the contrary, an arbitrary optimal planning solution, which is not LexOpt, has poor worst-case performance.

We show that the makespan recovery problem is strongly $\mathcal{NP}$-hard, at least as hard as solving the problem with full input knowledge. Thus, combining planning and recovery does not mitigate the problem's computational complexity. Section 4.1 develops a recovery strategy that enforces all available binding decisions and performs only essential actions to regain feasibility. Section 4.2 shows that every recovered solution is a weak approximation if planning produces an arbitrary optimal solution. But if the initial optimal solution is LexOpt, Sections 4.2 and 4.3 prove positive performance guarantees for the recovered solution which is efficient in bounded uncertainty settings. For a single perturbation, planning using LexOpt ensures a 2 performance guarantee. For multiple perturbations, the initial solution may be weakly reoptimizable, but we obtain an asymptotically tight performance guarantee which is efficient under the previously-mentioned uncertainty set. This result theoretically justifies that efficient reoptimization requires a well-structured initial schedule. Section 5 presents numerical results. Section 7 concludes. The omitted proofs and a notation table are deferred to a supplementary material.

2. Problem Definitions

This section defines the makespan problem (Section 2.1), the LexOpt scheduling problem (Section 2.2), and discusses the investigated perturbations (Section 2.3).

2.1. Makespan Problem

A makespan problem instance I, e.g. [27, 35, 14], is a pair $(m, \mathcal{J})$, where $\mathcal{J} = \{J_1, \ldots, J_n\}$ is a set of n *independent* jobs, with a processing time vector $\vec{p} = \{p_1, \ldots, p_n\}$, to be executed by a set $\mathcal{M} = \{M_1, \ldots, M_m\}$ of m parallel identical machines. Job $J_j \in \mathcal{J}$ must be processed by exactly one machine $M_i \in \mathcal{M}$ for p_j units of time non-preemptively, i.e. in a single continuous interval without interruptions. Each machine processes at most one job per time. The objective is to minimize the last machine completion time. Given a schedule S, let $C_{\max}(S)$ and $C_i(S)$ be the makespan and the completion time of machine $M_i \in \mathcal{M}$, respectively, in S. Then, $C_{\max}(S) = \max_{1\leq i\leq m}\{C_i(S)\}$. In the following mixed-integer linear optimization (MILP) formulation, binary variable $x_{i,j}$ is 1 if job $J_j \in \mathcal{J}$ is executed by machine $M_i \in \mathcal{M}$ and 0, otherwise.

$$\min_{C_{\max}, C_i, x_{i,j}} \quad C_{\max} \tag{1a}$$

$$C_{\max} \geq C_i \qquad M_i \in \mathcal{M} \tag{1b}$$

$$C_i = \sum_{j=1}^{n} x_{i,j} \cdot p_j \qquad M_i \in \mathcal{M} \tag{1c}$$

$$\sum_{i=1}^{m} x_{i,j} = 1 \qquad J_j \in \mathcal{J} \tag{1d}$$

$$x_{i,j} \in \{0,1\} \qquad J_j \in \mathcal{J}, M_i \in \mathcal{M}. \tag{1e}$$

Expression (1a) minimizes makespan. Constraints (1b) are the makespan definition. Constraints (1c) allow a machine to execute at most one job per time. Constraints (1d) assign each job to exactly one machine.

2.2. LexOpt Scheduling Problem

LexOpt minimizes m objective functions $F_1, \ldots, F_m : \mathcal{S} \to \mathbb{R}_0^+$ over a set $\mathcal{S}$ of feasible solutions. The functions are sorted in decreasing priority order, i.e. F_i is more important than $F_{i'}$, for each $1 \leq i < i' \leq m$. Formally, an optimal algorithm for $\operatorname{lex}\min\{F_1(S), \ldots, F_m(S) : S \in \mathcal{S}\}$ computes a solution $S^* \in \mathcal{S}$ such that $F_1(S^*) = v_1^* = \min\{F_1(S) : S \in \mathcal{S}\}$ and $F_i(S^*) = v_i^* = \min\{F_i(S) : S \in \mathcal{S}, F_1(S) = v_1^*, \ldots, F_{i-1}(S) = v_{i-1}^*\}$, for $i = 2, \ldots, m$.

Consider two solutions S and S' to a LexOpt problem $\operatorname{lex}\min\{F_1(S), \ldots, F_m(S) : S \in \mathcal{S}\}$. S and S' are *lexicographically distinct* if there is at least one $q \in \{1, \ldots, m\}$ such that $F_q(S) \neq F_q(S')$. Further, S is *lexicographically smaller* than S', i.e. $S <_{\text{lex}} S'$ or $\vec{F}(S) <_{\text{lex}} \vec{F}(S')$, if (i) S and S' are lexicographically distinct and (ii) $F_q(S) < F_q(S')$, where q is the smallest component in which they differ, i.e. $q = \min\{i : F_i(S) \neq F_i(S'), 1 \leq i \leq m\}$. S is *lexicographically not greater* than S', i.e. $S \leq_{\text{lex}} S'$ or $\vec{F}(S) \leq_{\text{lex}} \vec{F}(S')$, if either S and S' are lexicographically equal, i.e. not lexicographically distinct, or $S <_{\text{lex}} S'$. The LexOpt problem $\operatorname{lex}\min\{F_1(S), \dots, F_m(S) : S \in \mathcal{S}\}$ computes a solution S^* such that $\vec{F}(S^*) \leq_{\text{lex}} \vec{F}(S)$, for all $S \in \mathcal{S}$.

An instance $I = (m, \mathcal{J})$ of the LexOpt scheduling problem minimizes m objective functions $F_1, \dots, F_m$ lexicographically, where F_q is the distinct q-th greatest machine completion time, for $q = 1, \dots, m$, and produces a feasible schedule $S = (\vec{x}, \vec{C})$. This description motivates a LexOpt formulation with an exponential number of constraints. We reformulate to a polynomial number of variables and constraints in Equations (2a) - (2g). Lemma 1 orders the machine completion times in a LexOpt schedule and derives valid inequalities.

Lemma 1. *There exists an optimal solution to the LexOpt scheduling problem such that:*

1. $C_i \geq C_{i+1}$, *for* $i = 1, \dots, m-1$,
2. $\left[\sum_{q=1}^{i-1} C_q\right] + (m-i+1) \cdot C_i \geq \sum_{j=1}^{n} p_j$ *and* $i \cdot C_i + \left[\sum_{q=i+1}^{m} C_q\right] \leq \sum_{j=1}^{n} p_j$, $\forall\ i = 1, \dots, m$.

Reformulating the LexOpt scheduling problem, Lemma 1 implies the objective (2a) and constraints (2b) - (2d). Constraints (2e) and (2f) enforce feasibility.

$$\operatorname*{lex\,min}_{C_i, x_{i,j}} \quad C_1, \dots, C_m \tag{2a}$$

$$C_i \geq C_{i+1} \qquad M_i \in \mathcal{M} \setminus \{M_m\} \tag{2b}$$

$$\sum_{q=1}^{i-1} C_q + (m-i+1) \cdot C_i \geq \sum_{j=1}^{n} p_j \qquad M_i \in \mathcal{M} \tag{2c}$$

$$i \cdot C_i + \sum_{q=i+1}^{m} C_q \leq \sum_{j=1}^{n} p_j \qquad M_i \in \mathcal{M} \tag{2d}$$

$$C_i = \sum_{j=1}^{n} x_{i,j} \cdot p_j \qquad M_i \in \mathcal{M} \tag{2e}$$

$$\sum_{i=1}^{m} x_{i,j} = 1 \qquad J_j \in \mathcal{J} \tag{2f}$$

$$x_{i,j} \in \{0, 1\} \qquad J_j \in \mathcal{J}, M_i \in \mathcal{M}. \tag{2g}$$

2.3. Perturbations

A two-stage makespan scheduling problem is specified by an initial makespan problem instance $I_{init} = (m, \mathcal{J})$ and a perturbed problem instance $I_{new} = (\hat{m}, \hat{\mathcal{J}})$.

Let $\mathcal{M}$ and $\hat{\mathcal{M}}$ be the set of machines in I_{init} and I_{new}, respectively. Similarly, for a job $J_j \in \mathcal{J} \cap \hat{\mathcal{J}}$, denote by p_j and $\hat{p}_j$ the corresponding processing times in I_{init} and I_{new}. With uncertainty realization, instance I_{init} is transformed to I_{new}. This manuscript investigates the two-stage makespan problem in the case of (i) a single perturbation, and (ii) multiple perturbations. In the former case, the effect of uncertainty realization is one of the following perturbations:

1. *[Processing time reduction]* The processing time p_j of job $J_j \in \mathcal{J}$ is decreased and becomes $\hat{p}_j = p_j/f_j$, for some $f_j > 1$.
2. *[Processing time augmentation]* The processing time p_j of job $J_j \in \mathcal{J}$ is increased and becomes $\hat{p}_j = f_j p_j$, for some $f_j > 1$.
3. *[Job cancellation]* Job $J_j \in \mathcal{J}$ is removed, i.e. $\hat{\mathcal{J}} = \mathcal{J} \setminus \{J_j\}$.
4. *[Job arrival]* New job $J_j \notin \mathcal{J}$ arrives, i.e. $\hat{\mathcal{J}} = \mathcal{J} \cup \{J_j\}$.
5. *[Machine failure]* Machine $M_i \in \mathcal{M}$ fails, i.e. $\hat{\mathcal{M}} = \mathcal{M} \setminus \{M_i\}$.
6. *[Machine activation]* New machine $M_i \notin \mathcal{M}$ is added, i.e. $\hat{\mathcal{M}} = \mathcal{M} \cup \{M_i\}$.

These perturbations are frequently encountered by scheduling practitioners and consist an active topic of research in scheduling under uncertainty [32]. In the case of multiple perturbations, I_{new} is obtained from I_{init} by applying a series of the above type perturbations. Certain perturbations may be considered equivalent. Specifically, in some manuscript proofs: (i) cancelling job $J_j \in \mathcal{J}$ is identical to reducing p_j to zero, i.e. $f_j \to \infty$, (ii) machine $M_i \in \mathcal{M}$ failure is equivalent to new arrivals of the jobs in $\mathcal{J}_i$, where $\mathcal{J}_i$ is the set of jobs assigned to machine M_i in schedule S_{init} for I_{init}, (iii) job arrivals are treated similarly to processing time augmentations. We denote by $f = f_j$ the *perturbation factor* with respect to job $J_j \in \mathcal{J}$, by $k = |\{J_{j'} \in \mathcal{J} : f_{j'} > f\}|$ the number of *unstable jobs*, and by $\delta = \hat{m} - m$ the number of *surplus machines* after uncertainty realization.

3. Exact LexOpt Branch-and-Bound Algorithm (Stage 1)

This section introduces a LexOpt branch-and-bound method using vectorial bounds. Supplementary material describes the sequential [18, 15], weighting [51, 52], and highest-rank objective [44] methods for LexOpt scheduling.

The branch-and-bound algorithm idea is to explore the problem's search tree by bounding all objective functions in each node and eliminating subtrees that cannot lexicographically dominate the incumbent, similar to the *ideal point* in multiobjective optimization [20]. In the LexOpt scheduling problem, computing a vectorial lower bound component is equivalent to approximating a multiprocessor scheduling problem with rejections generalizing the makespan problem. Using this relation, we propose packing-based algorithms for simultaneously computing a vectorial lower and upper bound. The remainder of the section presents (i) the definition of a vectorial bound, (ii) the branch-and-bound method description, (iii) algorithms computing vectorial bounds, and (iv) the branch-and-bound optimality proof.

Definition 1 (Vectorial Bound). *Suppose $\vec{C}(S) = (C_1(S), \ldots, C_m(S))$ is the non-increasing vector of machine completion times in a feasible schedule S of the LexOpt scheduling problem. Vector $\vec{L} = (L_1, \ldots, L_m)$ is a* **vectorial lower bound** *of S if $L_i \le C_i(S)$, for each $1 \le i \le m$. A* **vectorial upper bound** *$\vec{U} = (U_1, \ldots, U_m)$ of S has $U_i \ge C_i(S)$, for each $1 \le i \le m$.*

3.1. Branch-and-Bound Description

Initially, the branch-and-bound method sorts the jobs in non-increasing processing time order, i.e. $p_1 \ge p_2 \ge \ldots \ge p_n$. The search space is a full tree with $n+1$ levels. The root node is located in level 0. The set of leaves is the set $\mathcal{S}$ of all possible m^n possible solutions. Each node except for the leaves has exactly m children. For each $\ell \in \{0, 1, \ldots, n-1\}$, a node v in the ℓ-th tree level represents a fixed assignment of jobs $J_1, \ldots, J_\ell$ to the m machines and jobs $J_{\ell+1}, \ldots, J_n$ remain to be assigned. The m children of node v correspond to every possible assignment of job $J_{\ell+1}$ to the m machines. Denote by $\mathcal{S}(v)$ the set of all schedules in the subtree rooted at node v. The primal heuristic applied in each node is the longest processing time first (LPT) algorithm [27]. In each schedule S obtained by primal heuristic LPT, the algorithm reorders the machines so that $C_1(S) \ge \ldots \ge C_m(S)$. Lexicographic order may not apply at the partial schedule associated with level $\ell < n$ where only jobs $J_1, \ldots, J_\ell$ have been assigned to the m machines. The vectorial lower bound $\vec{L}$ on the LexOpt schedule $S^* \in \mathcal{S}(v)$ below node v is computed in each node using the Section 3.2 algorithms.

The branch-and-bound method explores the search tree via *depth-first search*. Stack Q stores the set of explored nodes. Variable I stores the *incumbent*, i.e.

the lexicographically smallest solution found thus far. At each step, the algorithm picks the node u on top of Q and explores its m children $children(u)$. For each $v \in children(u)$, if the primal heuristic finds a solution S_v such that $\vec{C}(S_v) <_{\text{lex}} \vec{C}(I)$, the incumbent updates. If v is not a leaf, Algorithm 1 computes a vectorial lower bound $\vec{L}$ of the lexicographically best solution in $\mathcal{S}(v)$. When $\vec{C}(I) \leq_{\text{lex}} \vec{L}$, the set $\mathcal{S}(v)$ does not contain any solution lexicographically better than I and the subtree rooted at v is *fathomed*. Otherwise, v is pushed onto stack Q. After completing Algorithm 1, the incumbent is optimal because every other solution has been rejected as not lexicographically smaller than the incumbent.

3.2. Vectorial Bound Computation

Consider node v located in the ℓ-th search tree level of branch-and-bound. We simultaneously compute vectorial lower bound $\vec{L} = (L_1, \ldots, L_m)$ and vectorial upper bound $\vec{U} = (U_1, \ldots, U_m)$ on the LexOpt schedule $S^* \in \mathcal{S}(v)$ below node v. The algorithm performs m iterations. For iteration $i \in \{1, \ldots, m\}$, it calculates a lower bound L_i (Algorithm 1) and an upper bound U_i (Algorithm 2) on the i-th machine completion time using the bounds $U_1, \ldots, U_{i-1}$ and $L_1, \ldots, L_{i-1}$, respectively. Recall that the jobs are sorted, so $p_1 \geq \ldots \geq p_n$. W.l.o.g., each machine executes all jobs with index $\leq \ell$ before any job with index $> \ell$. Therefore, for each schedule in $\mathcal{S}(v)$, a unique vector $\vec{t} = (t_1, \ldots, t_m)$ specifies the machine completion times by considering only jobs $J_1, \ldots, J_\ell$ and ignoring the remaining ones. Further, no job J_j with $\ell + 1 \leq j \leq n$ is executed before time t_q on machine M_q, for $1 \leq q \leq m$. Appendix E proves the correctness of Algorithms 1 and 2. We interpret both computations as approximating a scheduling problem with job rejections.

Vectorial lower bound component L_i. The two-phase computation is equivalent to constructing a *pseudo-schedule* $\widetilde{S}$ which is feasible except that some jobs are scheduled fractionally. First, Algorithm 1 assigns fractionally the jobs $J_{\ell+1}, \ldots, J_h$ to machines $M_1, \ldots, M_{i-1}$, where h is the smallest index such that $\sum_{j=\ell+1}^{h} p_j \geq \sum_{q=1}^{i-1}(U_q - t_q)$. For each $q = 1 \ldots i-1$, machine M_q is assigned sufficiently large job pieces so that its completion time is greater than or equal to U_q. In the second phase, Algorithm 1 assigns the remaining load $\lambda = \sum_{j=h+1}^{n} p_j$ of jobs $J_{h+1}, \ldots, J_n$ fractionally to machines $M_i, \ldots, M_m$. This assignment minimizes

Algorithm 1 Computation of the i-th vectorial lower bound component

1: Select job index $\min\{h : \sum_{j=\ell+1}^{h} p_j \geq \sum_{q=1}^{i-1}(U_q - t_q)\}$.

2: Compute remaining load $\lambda = \sum_{j=h+1}^{n} p_j$.

3: Set $\tau = \max_{i \leq q \leq m}\{t_q\}$.

4: Return the maximum among:

- $\min_{i \leq q \leq m}\{t_q\} + p_{h+1}$, and
- $\max_{i \leq q \leq m}\{t_q\} + \max\left\{\frac{1}{m-i+1}\left(\lambda - \sum_{q=i+1}^{m}(\tau - t_q)\right), 0\right\}$.

the i-th greatest completion time of the resulting fractional schedule. Assuming $p_{n+1} = 0$, the value L_i is the maximum among $\min_{i \leq q \leq m}\{t_q\} + p_{h+1}$ and $\max_{i \leq q \leq m}\{t_q\} + \max\left\{\frac{1}{m-i+1}\left(\lambda - \sum_{q=i+1}^{m}(\tau - t_q)\right), 0\right\}$, where $\tau = \max_{i \leq q \leq m}\{t_q\}$.

Lemma 2. *Consider a node v of the search tree and a machine index $i \in \{1, \ldots, m\}$. Algorithm 1 produces a value $L_i \leq C_i(S)$ for each feasible schedule $S \in \mathcal{S}(v)$ below v such that $C_q(S) \leq U_q$, $\forall\ q = 1, \ldots, i-1$.*

Vectorial upper bound component U_i. Like L_i, the U_i computation requires two phases that may be interpreted as constructing a fractional pseudo-schedule $\widetilde{S}$. Additionally, Algorithm 2 uses the incumbent I. Schedule $\tilde{S}$ combines the partial schedule for jobs $J_1, \ldots, J_\ell$ associated with node v and the pseudo-schedule of the remaining jobs $J_{\ell+1}, \ldots, J_n$ computed by Algorithm 2. Initially, Algorithm 2 assigns a total load $\sum_{q=1}^{i-1}(L_q - t_q)$ of the smallest jobs to machines $M_1, \ldots, M_{i-1}$ so that the completion time of M_q becomes exactly equal to L_q, for $q = 1, \ldots, i-1$. That is, a piece $\tilde{p}_h$ of job J_h and jobs $J_{h+1}, J_{h+2}, \ldots, J_n$ are assigned fractionally to machines $M_1, \ldots, M_{i-1}$ so that $\tilde{p}_h + \sum_{j=h+1}^{n} p_j = \sum_{q=1}^{i-1}(L_q - t_q)$. Next, Algorithm 2 assigns the remaining load $\lambda = \sum_{j=\ell+1}^{h-1} p_j + (p_h - \tilde{p}_h)$ of jobs $J_{\ell+1}, \ldots, J_h$ fractionally and uniformly to the least loaded machines among $M_i, \ldots, M_m$ as follows. Initially, the partial completion times are sorted so that $t_i \leq \ldots \leq t_m$. This sorting occurs only in computing the vectorial upper bound and does not affect any branch-and-boun partial schedule. Let μ be the minimum machine index such that (i) the remaining load λ may be fractionally scheduled to machines $M_i, \ldots, M_\mu$ so that they end up with a common completion time $\tau = \frac{1}{\mu-i+1}\left(\sum_{q=i}^{\mu} t_q + \lambda\right)$, and (ii) the partial completion time t_q of any other machine among $M_{\mu+1}, \ldots, M_m$ is at

Algorithm 2 Computation of the i-th vectorial upper bound component

1: Compute remaining load $\lambda = \sum_{j=\ell}^{n} p_j - \sum_{q=1}^{i-1}(L_q - t_q)$.
2: Sort the machines $M_i, \ldots, M_m$ so that $t_i \leq \ldots \leq t_m$.
3: Select machine index $\min\left\{\mu : \frac{1}{\mu-i+1}\left(\sum_{q=i}^{\mu} t_q + \lambda\right) \leq t_{\mu+1},\ \ i \leq \mu \leq m\right\}$.
4: Return the minimum among $\max\left\{\frac{1}{\mu-i+1}\left(\sum_{q=i}^{\mu} t_q + \lambda\right) + p_\ell,\ \ t_m\right\}$ and $C_i(I)$.

least τ, i.e. $t_{\mu+1} \geq \tau$. Bound U_i is the minimum of $\max\{\tau + p_\ell, t_m\}$ and $C_i(I)$.

Lemma 3. *Consider a node v of the search tree and a machine index $i \in \{1, \ldots, m\}$. Algorithm 2 produces a value $U_i \geq C_i(S)$ for each feasible schedule $S \in \mathcal{S}(v)$ below v such that $C_q(S) \geq L_q$, $\forall\ q = 1, \ldots, i-1$.*

3.3. Branch-and-Bound Optimality Proof

Theorem 1 shows the correctness of our branch-and-bound Algorithm.

Theorem 1. *The branch-and-bound method computes a LexOpt solution.*

Proof:
Consider tree node v. Let $\vec{L} = (L_1, \ldots, L_m)$ and I be the computed vectorial lower bound and the incumbent, when our branch-and-bound algorithm explores v. We show the following invariant: if node v is pruned, then $\vec{C}(S) \geq_{\text{lex}} \vec{C}(I)$, for every schedule $S \in \mathcal{S}(v)$. Node v is pruned when $L \geq_{\text{lex}} \vec{C}(I)$, i.e. one of the following cases: (i) $L_1 > C_1(I)$, (ii) $L_q = C_q(I)\ \forall\ q = 1, \ldots, i-1$ and $L_i > C_i(I)$, for some $i \in \{2, \ldots, m-1\}$, or (iii) $L_i = C_i(I)\ \forall\ i = 1, \ldots, m$. In case (i), because $C_1(S) \geq L_1$, it holds that $\vec{C}(S) >_{\text{lex}} \vec{C}(I)\ \forall\ S \in \mathcal{S}(v)$. In case (ii), either $C_1(S) > L_1$, or $C_1(S) = L_1\ \forall\ S \in \mathcal{S}(v)$. Let $\mathcal{S}_1(v) \subseteq \mathcal{S}(v)$ be the subset of schedules satisfying $C_1(S) = L_1 = C_1(I)$. Algorithm 2 computes $U_1 = C_1(I)$. By Lemma 2, either $C_2(S) > L_2$, or $C_2(S) = L_2$, for each $S \in \mathcal{S}_1(v)$. Let $\mathcal{S}_2(v) \subseteq \mathcal{S}_1(v)$ be the subset of schedules with $C_2(S) = L_2$. We define similarly all sets $\mathcal{S}_1(v), \ldots, \mathcal{S}_{i-1}(v)$. By Lemma 2, for any schedule in $\mathcal{S}_{i-1}(v)$, it holds that $C_q(S) = L_q = C_q(I)\ \forall\ q = 1, \ldots, i-1$ and $C_i(S) \geq L_i > C_i(I)$. Thus, for each $S \in \mathcal{S}(v)$, $\vec{C}(S) >_{\text{lex}} \vec{C}(I)$. Finally, in case (iii), for each $S \in \mathcal{S}_{m-1}(v)$, $C_q(S) = C_q(I)\ \forall\ q = 1, \ldots, m$ and $\vec{C}(S) = \vec{C}(I)$. ■

4. Approximate Recovery Algorithm with Binding Decisions (Stage 2)

This section discusses reoptimizing the makespan problem. Section 4.1 presents our recovery approach. Sections 4.2 and 4.3 analyze the proposed recovery algorithm for a single perturbation and multiple perturbations, respectively. We highlight the importance of LexOpt even in settings with limited severe disturbances.

4.1. Recovery Algorithm Description

This section presents our recovery strategy, Algorithm 3. Reoptimization aims to exploit the initial optimal solution S_{init} for solving the perturbed instance I_{new}. Definition 2 formalizes the notions of *binding* and *free* optimization decisions.

Definition 2. *Consider a makespan recovery problem instance* $(I_{init}, S_{init}, I_{new})$ *with* $I_{init} = (\mathcal{M}, \mathcal{J})$ *and* $I_{new} = (\hat{\mathcal{M}}, \hat{\mathcal{J}})$*.*

- ***Binding decisions*** $\{x_{i,j} : (x_{i,j}(S_{init}) = 1) \wedge (i \in \hat{\mathcal{M}} \cap \mathcal{M}) \wedge (j \in \hat{\mathcal{J}} \cap \mathcal{J})\}$ *are variable evaluations attainable from* S_{init} *in the recovery process.*
- ***Free decisions*** $\{x_{i,j} : (j \in \hat{\mathcal{J}}) \wedge (\nexists i' \in \mathcal{M} \cap \hat{\mathcal{M}} : x_{i',j}(S_{init}) = 1)\}$ *are variable evaluations that cannot be determined from* S_{init} *but are needed to recover feasibility.*

In the makespan scheduling problem, binding decisions are all job assignments in S_{init} which remain valid for the perturbed instance I_{new}. Free decisions are assignments of new jobs or of jobs originally assigned to failed machines.

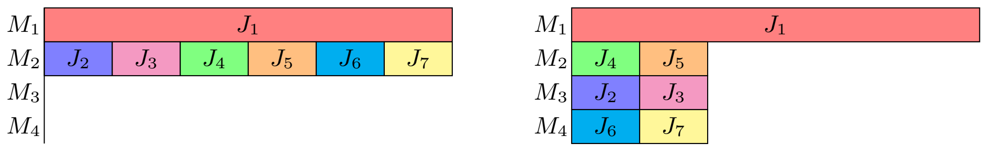
Our recovery strategy maintains all binding decisions and assigns free decisions with the LPT heuristic which takes the lexicographically best decision in each iteration. Algorithm 3 accepts all available binding decisions because: (i) theoretically, using the binding decisions exploits all relevant information provided by S_{init} to solve the perturbed instance I_{new} and quantifies the benefit of staying close to S_{init} and (ii) practically, modifying S_{init} may be associated with transformation costs, e.g. [57], and Algorithm 3 mitigates this overhead. The supplement discusses more flexible recovery with a bounded number of binding decision modifications.

4.2. Single Perturbation

This section designs and analyzes single perturbation recovery algorithms, the first step towards effective reoptimization. Theorem 2 shows that reoptimization

Algorithm 3 Recovery Strategy

Perform all binding decisions (job assignments) with respect to schedule S_{init}.
Schedule free (unassigned) jobs using Longest Processing Time first (LPT).



(a) Weakly recoverable optimal S_{init}. (b) Efficiently recoverable LexOpt S_{init}.

Figure 3: Illustration of the benefit obtained by LexOpt for schedules S_{init}.

beginning with an arbitrary optimal initial solution is ineffective and produces a weak, non-constant approximation factor recovery, e.g. under makespan degeneracy conditions. But reoptimization is more efficient when the initial schedule is a LexOpt solution. Theorem 3 shows that Algorithm 3 for a LexOpt solution has a constant performance guarantee for any single perturbation. Effectively, the LexOpt structure allows reoptimization to overcome the makespan degeneracies.

Theorem 2. *For the makespan recovery problem with a single perturbation, Algorithm 3 produces an $\Omega(m)$-approximate solution if S_{init} is an arbitrary optimal schedule.*

Before proving the positive performance guarantee obtained with LexOpt in Theorem 3, we derive Lemma 4. We denote by $C^*_{\max}(m, \mathcal{J})$ the minimum makespan for a problem instance $(m, \mathcal{J})$. Lemma 4 formalizes the importance of lexicographic ordering in a solution S_{init} when machine M_ℓ is perturbed $\forall \ell \in \mathcal{M}$: the subschedule specified by S_{init} on the remaining $m-1$ machines $\mathcal{M} \setminus \{M_\ell\}$ is a minimum makespan schedule for the subset $\mathcal{J}'$. Furthermore, Lemma 4 relates two minimum makespans for the same $\mathcal{J}$ but a different number of machines.

Lemma 4. *Consider a makespan problem instance $(m, \mathcal{J})$ and let S be LexOpt schedule. Given an arbitrary machine $M_\ell \in \mathcal{M}$, denote by $\mathcal{J}'$ the subset of all jobs assigned to the machines in $\mathcal{M} \setminus \{M_\ell\}$ by S. Then, it holds that:*

1. $\max_{M_i \in \mathcal{M} \setminus \{M_\ell\}} \{C_i(S)\} = C^*_{\max}(m-1, \mathcal{J}')$, *and*
2. $C^*_{\max}(m-1, \mathcal{J}) \leq 2 \cdot C^*_{\max}(m, \mathcal{J})$.

Theorem 3 shows a significantly improved, positive performance guarantee for the makespan recovery problem for a LexOpt initial solution, e.g. Figure 3b.

Theorem 3. *For the makespan recovery problem with a single perturbation, Algorithm 3 produces a tight 2-approximate solution, if S_{init} is LexOpt.*

Proof:

Consider the Section 2.3 perturbation types. Perturbations in the same group are analyzed with similar arguments. The sequel proves a job reduction (type 1 perturbation) using LexOpt. For any perturbation, we cannot achieve a better performance guarantee without modifying binding decisions.

Consider a LexOpt schedule S_{init} for instance I_{init} and suppose that job J_j processing time decreases by $\delta \in (0, p_j]$, i.e. $p_j \leftarrow p_j - \delta$. Cancelling job $J_j \in \mathcal{J}$ is equivalent to reducing p_j to zero. Let M_ℓ be the machine executing J_j in S_{init}. Without loss of generality, job J_j completes last among all jobs assigned to M_ℓ. Algorithm 3 maintains the S_{init} job assignments and returns the recovered schedule S_{rec} obtained by decreasing p_j and $C_\ell(S_{init})$ by δ. Suppose S_{new} is an optimal schedule for the perturbed instance I_{new}. We distinguish two cases depending on whether M_ℓ completes last in S_{rec}.

First, suppose $C_\ell(S_{rec}) < C_{\max}(S_{rec})$. W.l.o.g., $\ell \neq 1$ and $C_1(S_{rec}) = C_{\max}(S_{rec})$, i.e. M_1 completes last in S_{rec}. Let $\mathcal{J}' \subseteq \mathcal{J}$ be the jobs executed by all machines $\mathcal{M} \setminus M_\ell$. Then,

$$
\begin{aligned}
C_{\max}(S_{rec}) &= C^*_{\max}(m-1, \mathcal{J}') && \text{[Lemma 4.1]}, \\
&\leq C^*_{\max}(m-1, \mathcal{J} \setminus \{J_n\}) && [\mathcal{J}' \subseteq \mathcal{J} \setminus \{J_n\}], \\
&\leq 2 \cdot C^*_{\max}(m, \mathcal{J} \setminus \{J_n\}) && \text{[Lemma 4.2]}, \\
&= 2 \cdot C_{\max}(S_{new}) && \text{[Definition]}.
\end{aligned}
$$

Subsequently, consider $C_\ell(S_{rec}) = C_{\max}(S_{rec})$. In this case, $C_{\max}(S_{rec}) = C_{\max}(S_{init}) - \delta$. We claim that S_{rec} is a minimum makespan schedule for I_{new}. Assume for contradiction the existence of an optimal schedule S_{new} for I_{new} such that $C_{\max}(S_{new}) < C_{\max}(S_{init}) - \delta$. Starting from S_{new}, we add δ extra units of time on job J_j and we obtain a feasible schedule $\widetilde{S}$ for I_{init} such that $C_{\max}(\widetilde{S}) < C_{\max}(S_{init})$. But this contradicts the fact that S_{init} is optimal for I_{init}. ∎

4.3. Multiple Perturbations

Reoptimization with multiple disturbances can be seen as a two-player game where (i) we solve an initial problem instance, (ii) a malicious adversary generates perturbations, and (iii) we transform the initial solution into an efficient solution for the new instance. Adversarial strategies with multiple perturbations can render the initial optimal solution weakly reoptimizable. But a LexOpt solution can manage a bounded adversary. Algorithm 3 produces efficient solutions in bounded uncertainty settings and attains a positive performance guarantee parameterized by the degree of uncertainty. For analysis purposes, Definition 3 describes uncertainty set $\mathcal{U}(f,k,\delta)$ with three parameters: (i) the factor f indicating the boundary between *stable* and *unstable* job perturbations, (ii) the number k of unstable jobs, and (iii) the number δ of surplus machines. We assume that the number k of unstable jobs is bounded by the number of machines m, i.e. $k < m$.

Definition 3. *For a makespan problem instance $(m,\mathcal{J})$ with processing time vector $\vec{p}=(p_1,\dots,p_n)$, the* ***uncertainty set*** *$\mathcal{U}(f,k,\delta)$ contains every instance $(\hat{m},\hat{\mathcal{J}})$ with processing time vector $\hat{p}=(\hat{p}_1,\dots,\hat{p}_n)$ such that:*

- ***Stability/unstability boundary.*** *$\hat{\mathcal{J}}$ can be partitioned into the set $\hat{\mathcal{J}}^s$ of stable jobs and the set $\hat{\mathcal{J}}^u$ of unstable jobs, where $p_j/f \le \hat{p}_j \le p_j \cdot f \ \forall\ J_j \in \hat{\mathcal{J}}^s$,*
- ***Bounded number of unstable jobs.*** *$|\hat{\mathcal{J}}^u| \le k$, we assume $k < m$,*
- ***Bounded surplus machines availability.*** *$\hat{m} - m \le \delta$.*

Similarly to Section 4.2, suppose $C^*_{\max}(m,\mathcal{J})$ is the optimal objective value for the makespan problem instance $(m,\mathcal{J})$. Lemma 5 (i) formalizes the optimal substructure imposed by LexOpt, (ii) bounds pairwise machine completion time differences in LexOpt schedules, (iii) quantifies the optimal objective's sensitivity with respect to the number of machines, and (iv) quantifies the objective value sensitivity with respect to processing times.

Lemma 5. *Let $(m,\mathcal{J})$ be a makespan problem instance with LexOpt schedule S.*

1. *Given the subset $\mathcal{J}' \subseteq \mathcal{J}$ of jobs scheduled on the subset $\mathcal{M}' \subseteq \mathcal{M}$ of machines, where $|\mathcal{M}'| = m'$, the sub-schedule of S on $\mathcal{M}'$ is optimal for $(m'\mathcal{J}')$, i.e. $\max_{M_i \in \mathcal{M}'}\{C_i(S)\} = C^*_{\max}(m',\mathcal{J}')$.*
2. *Assuming that $M_i, M_\ell \in \mathcal{M}$ are two different machines and that job $J_j \in \mathcal{J}$ is assigned to machine M_i in S, then $C_\ell(S) \ge C_i(S) - p_j$.*

Type	Perturbation type (Section 2.3)	Performance guarantee
Type 1	Job cancellations, Processing time reductions	$2f \cdot (1 + \lceil \frac{k}{m-k} \rceil)$
Type 2	Processing time augmentations	$f + k$
Type 3	Machine activations	$(1 + \lceil \delta/m \rceil)$
Type 4	Job arrivals, Machine failures	$\max\{2, \rho\}$

Table 1: Algorithm 3 performance guarantees for the perturbation types, considering the (i) perturbation factor f, (ii) number $k < m$ of unstable jobs, and (iii) number δ surplus machines. The term ρ is the product of the performance guarantees obtained for Types 1-3.

3. *It holds that* $C^*_{\max}(\mathcal{J}, m-\ell) \leq \left(1 + \left\lceil \frac{\ell}{m-\ell} \right\rceil\right) \cdot C^*_{\max}(\mathcal{J}, m) \; \forall \, \ell \in \{1, \ldots, m-1\}$.
4. *Let* $(m, \hat{\mathcal{J}})$ *be a makespan problem instance such that* $\frac{1}{f} \cdot \hat{p}_j \leq p_j \leq \hat{p}_j$, *where* $\mathbf{p}$ *and* $\hat{\mathbf{p}}$ *are the processing times in* $(m, \mathcal{J})$ *and* $(m, \hat{\mathcal{J}})$, *respectively. Then,* $\frac{1}{f} \cdot C^*_{\max}(m, \hat{\mathcal{J}}) \leq C^*_{\max}(m, \mathcal{J}) \leq C^*_{\max}(m, \hat{\mathcal{J}})$.

Table 1 summarizes the performance guarantees for the Algorithm 3 recovery strategy with respect to the four Section 2.3 perturbation types. Lemma 6 proves Type 1 and Supplementary material presents Types 2-4. The proofs also show that our analysis is asymptotically tight. Distinguishing the arguments required for each perturbation type, yields a global, tight performance guarantee by propagating the solution degradation with respect to the above sequence. Considering the perturbations in this series of steps is an assumption only for analysis purposes and does not restrict the uncertainty model. Theoretically, LexOpt is essential only for bounding the solution degradation due to job removals and processing time reductions. But practically, the optimal substructure derived by LexOpt, as stated in Lemma 5, is beneficial in an integrated setting with all possible perturbations. Section 5 complements the theoretical analysis with experiments highlighting reoptimization's significance in the recovered solution quality.

Lemma 6. *For the makespan recovery problem with only job cancellations and processing time reductions (Type 1), in the* $\mathcal{U}(f, k, \delta)$ *uncertainty set, Algorithm 3 produces a* $2f \cdot (1 + \lceil \frac{k}{m-k} \rceil)$*-approximate solution and this performance guarantee is asymptotically tight,* $\forall \, k < m$.

Proof:

Processing time reductions are only recovered using binding decisions. Job cancellation is equivalent to reducing the processing time to zero. Considering the recovered schedule S_{rec}, partition the machines $\mathcal{M}$ into the sets $\mathcal{M}^s$ of *stable machines* which are not assigned unstable jobs and $\mathcal{M}^u$ of *unstable machines* which are assigned unstable jobs. That is, $C_i(S_{rec}) \geq \frac{1}{f} \cdot C_i(S_{init})$, for each $M_i \in \mathcal{M}^s$, and $m^s = |M^s| \geq m-k$. Note $\mathcal{M}^u = \mathcal{M} \setminus \mathcal{M}^s$ and $m^u = |\mathcal{M}^u| \leq k$. Denote $M_i \in \mathcal{M}$ as a *critical machine*, if it completes last in schedule S_{rec}, i.e. $C_i(S_{rec}) = C_{\max}(S_{rec})$. There are two cases: $\mathcal{M}^s$ may contain a critical machine, or not.

Case 1: $\mathcal{M}^s$ contains a critical machine.. Let $\mathcal{J}^s_{new}$ be the subset of jobs assigned to the machines $\mathcal{M}^s$ by S_{rec}. Each job in $\mathcal{J}^s_{new}$ has been perturbed by a factor of at most f. Denote by $\mathcal{J}^s_{init}$ the same jobs before uncertainty realization. The jobs in $\mathcal{J}^s_{init}$ are exactly those executed on $\mathcal{M}^s$ in S_{init} and appear in $\mathcal{J}^s_{new}$ with one-by-one smaller processing times. Then,

$$\begin{aligned}
C_{\max}(S_{rec}) &= \max_{M_i \in \mathcal{M}^s} \{C_i(S_{rec})\} && [\mathcal{M}^s \text{ contains a critical machine}],\\
&\leq \max_{M_i \in \mathcal{M}^s} \{C_i(S_{init})\} && [\text{Processing time reduction}],\\
&= C^*_{\max}(m^s, \mathcal{J}^s_{init}) && [\text{Lemma 5.1}],\\
&\leq f \cdot C^*_{\max}(m^s, \mathcal{J}^s_{new}) && [\text{Lemma 5.4}],\\
&\leq f \cdot C^*_{\max}(m^s, \mathcal{J}_{new}) && [\mathcal{J}^s_{new} \subseteq \mathcal{J}_{new}],\\
&= f \cdot C^*_{\max}(m - m^u, \mathcal{J}_{new}) && [m^s = m - m^u],\\
&\leq f \cdot \left(1 + \left\lceil \frac{m^u}{m - m^u} \right\rceil\right) \cdot C^*_{\max}(m, \mathcal{J}_{new}) && [\text{Lemma 5.3}],\\
&\leq f \cdot \left(1 + \left\lceil \frac{k}{m - k} \right\rceil\right) \cdot C_{\max}(S_{new}). && [m^u \leq k]
\end{aligned}$$

Case 2: Only $\mathcal{M}^u$ contains critical machines.. Consider an unstable critical machine $M_i \in \mathcal{M}^u$ in S_{rec}, i.e. $C_{\max}(S_{rec}) = C_i(S_{rec})$. If only one job $J_j \in \mathcal{J}$ has been assigned to M_i, then schedule S_{rec} is optimal, i.e. $C_{\max}(S_{rec}) = C_{\max}(S_{new})$. Now consider M_i that has been assigned at least two jobs in S_{rec}. This perturbation type reduces processing time, so $C_i(S_{rec}) \leq C_i(S_{init})$. Since $k < m$, there is at least one machine $M_\ell \in \mathcal{M}^s$. Furthermore, since S_{init} is LexOpt, Lemma 5.2

requires that $C_\ell(S_{init}) \geq C_i(S_{init}) - p_j$, for each job $J_j \in \mathcal{J}$ assigned to M_i by S_{init}. Since S_{init} contains at least two jobs, there exists a job J_j assigned to M_i by S_{init} such that $p_j \leq \frac{1}{2} \cdot C_i(S_{init})$. Hence, $C_i(S_{init}) \leq 2 \cdot C_\ell(S_{init})$. We conclude that $C_{\max}(S_{rec}) \leq 2 \cdot C_\ell(S_{init})$. Because $M_\ell \in \mathcal{M}^s$, using our analysis for case 1, we derive that

$$C_{\max}(S_{rec}) \leq 2f \cdot \left(1 + \left\lceil \frac{k}{m-k} \right\rceil\right) \cdot C_{\max}(S_{new}).$$

The supplementary material shows the tightness of our analysis. ■

Theorem 4 is a direct corollary of the performance guarantee analysis. Suppose that f_r, k_r are the perturbation factor and the number of unstable processing time reductions including job cancellations, f_a, k_a are the perturbation factor and the number of unstable processing time augmentations, and δ^+ is the increase to the number of machines after uncertainty realization. That is, $f = f_r + f_a$, $k = k_r + k_a$ and $\delta^+ = \max\{\delta, 0\}$.

Theorem 4. *For the makespan recovery problem within a $\mathcal{U}(f, k, \delta)$ uncertainty set, Algorithm 3 achieves the following tight performance guarantee:*

$$2f_r \cdot \left(1 + \left\lceil \frac{k_r}{m-k_r} \right\rceil\right) \cdot (f_a + k_a) \cdot \left(1 + \left\lceil \frac{\delta^+}{m} \right\rceil\right).$$

5. Numerical Results

Section 5.1 describes the system specifications and benchmark instances. Section 5.2 evaluates the exact methods. Section 5.3 discusses the perturbed instances. Section 5.4 evaluates the recovery strategies and the impact of LexOpt.

5.1. System Specification and Benchmark Instances

We ran all computations on an Intel Core i7-4790 CPU 3.60GHz with a 15.6 GB RAM running a 64-bit Ubuntu 14.04. Our implementations use Python 2.7.6 and Pyomo 4.4.1 [31, 30] and solve the MILP models with CPLEX 12.6.3 and Gurobi 6.5.2. The source code and test cases are available [34]. We have generated random LexOpt scheduling instances. *Well-formed instances* admit an optimal

Instances	**m**	**n**	**q**
Moderate	$3, 4, 5, 6$	$20, 30, 40, 50$	$100, 1000$
Hard	$10, 15, 20, 25$	$200, 300, 400, 500$	$10000, 100000$

Table 2: Well-formed Instance Sizes

solution close to a *perfect solution* which has all machine completion times equal, i.e. $C_i = C_{i'}$ for each $i, i' \in \mathcal{M}$. *Degenerate instances* have a less-balanced optimal solution. This section investigates well-formed instances. The supplementary material presents extended numerical results including degenerate instances.

The well-formed instances depend on 3 parameters: (i) the number m of machines, (ii) the number n of jobs, and (iii) a processing time seed q. This test set, summarized in Table 3a, consists of *moderate* and *hard* instances. For each combination of m, n and q, we generate 3 instances by selecting $\vec{p}$ using 3 distributions parameterized by q. Each processing time is rounded to the nearest integer. *Uniform distribution* selects $p_j \sim \mathcal{U}(\{1, \ldots, q\})$. *Normal distribution* chooses $p_j \sim \mathcal{N}(q, q/3)$ and guarantees that 99.7% of the values lie in interval $[0, 2q]$. *Symmetric of normal distribution* samples $p \sim \mathcal{N}(q, q/3)$ and selects $p_j = q - p$ if $p \in [0, q]$, or $p_j = 2q - (p - q)$ if $p_j \in (q, 2q]$. Normal and symmetric normal processing times outside $[0, 2q]$ are rounded to the nearest of 0 and $2q$.

5.2. LexOpt Scheduling

This section numerically evaluates the LexOpt methods. The sequential, highest-rank objective, and weighting methods solve MILP instances. We use MILP termination criteria: (i) 10^3 CPU seconds, and (ii) 10^{-4} relative error tolerance, where the relative gap $(Ub - Lb)/Ub$ is computed using the best-found incumbent Ub and the lower bound Lb. The sequential method solves a sequence of MILP models, each with 10^{-4} relative error tolerance. The simultaneous method solves one minimum makespan MILP model with 10^{-4} makespan error tolerance and populates the solution pool with 2000 solutions. The weighting method and our branch-and-bound method terminate with 10^{-4} weighted value error tolerance, where Ub is the weighted value $W(S) = \sum_{i=1}^{m} B^{m-i} \cdot C_i(S)$ of the returned schedule S and $B = 2$. We compute Lb by similarly weighting the global vectorial lower bound.

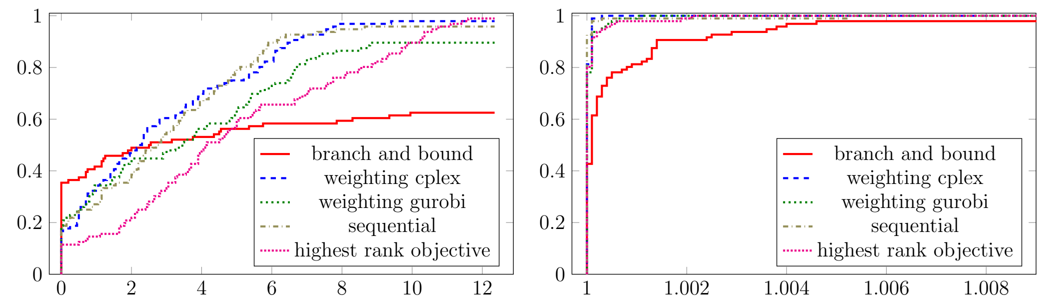
The *sequential method* solves m MILP instances with repeated CPLEX calls, each computing one objective value. We use the CPLEX reoptimize feature to exploit information obtained from solving higher-ranked objectives. If the method exceeds 10^3 CPU seconds in total, it terminates when the ongoing MILP run is completed. We implement the *weighting method* using Pyomo and solve the MILP with CPLEX and Gurobi. We use weighting parameter $B = 2$. The *highest-rank objective method* uses the CPLEX solution pool feature (capacity = 2000) in two phases. The first phase solves the standard makespan MILP model. The second phase continues the tree exploration and generates a pool of solutions.

The Figure 4 *performance profiles* compare the LexOpt methods with respect to elapsed times and best found solutions on the well-formed instances [19]. In terms of running time and number of solved instances, sequential method performs similarly to weighting method on moderate instances and slightly better on hard instances. But the sequential method produces slightly worse feasible solutions than weighting method since lower-ranked objectives are not optimized in case of a sequential method timeout. The highest-rank objective method has worse running times than sequential and weighting methods on moderate instances since the solution pool populate time is large compared to the overall solution time. On hard test cases, populating the solution pool is only a fraction of the global solution time and highest-rank objective method attains significantly better running times than sequential and weighting methods. The highest-rank objective method does not prove global optimality: it only generates 2000 solutions. But, the highest-rank objective method produces the best heuristic results for most test cases.

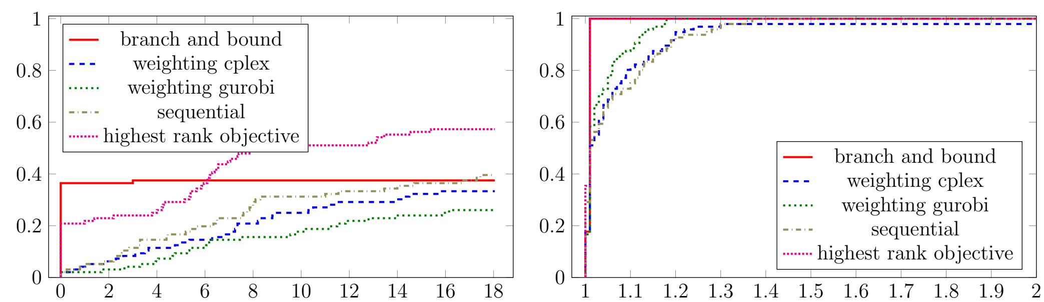
The branch-and-bound method with vectorial lower bounds, obtains good LPT heuristic solutions without populating the entire solution pool. Figure 4 shows that it guarantees global optimality more quickly than the other approaches for test cases where it converges. Branch-and-bound converges for $> 60\%$ of the moderate test cases, and $> 30\%$ hard instances. Branch-and-bound consistently produces a good heuristic, i.e. better than sequential and weighting methods, in hard instances.

5.3. Generation of Initial Solutions and Perturbed Instances

This section describes generating the benchmark instances for the makespan recovery problem. An instance is specified by: (i) an initial makespan problem



(a) Moderate instances: time (s) on $\log_2$ scale (left), upper bounds on $[1, 1.009]$ (right).



(b) Hard instances: time (s) on $\log_2$ scale (left), upper bounds on $[1, 2]$ (right).

Figure 4: Performance profiles for the *well-formed test set* with 10^3 s timeout.

instance I_{init}, (ii) an initial solution S_{init} to I_{init}, and (iii) a perturbed instance I_{new}. Recall that the recovery problem transforms solution S_{init} to a feasible solution S_{new} for instance I_{new} using the recovery strategies.

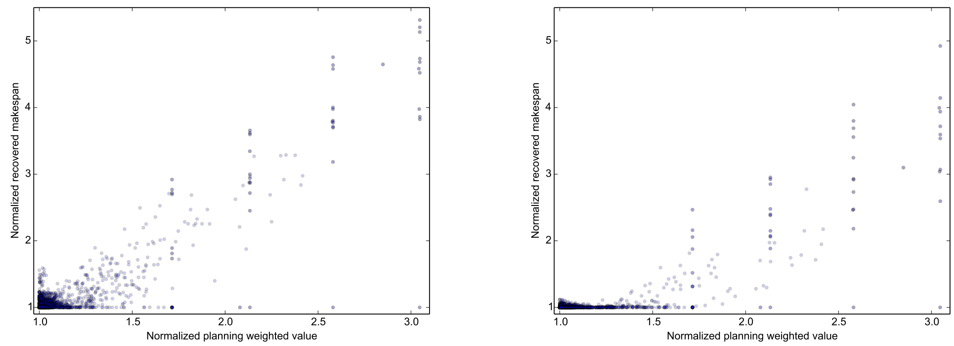
The initial makespan problem instances are the Section M.1 instances. For each instance I_{init}, we generate a set $\mathcal{S}(I_{init})$ of at least 50 diverse solutions by solving I_{init} using the CPLEX solution pool feature and the Section M.2 termination criteria. A key property is that the obtained solutions have, in general, different weighted values, i.e. for many pairs of solutions $S_1, S_2 \in \mathcal{S}(I_{init})$, $W(S_1) \neq W(S_2)$, where $W(S) = \sum_{i=1}^{m} B^{m-i} \cdot C_i(S)$. Using the weighted value as a distance measure from the LexOpt solution, we evaluate a recovered solution's quality as a function of the initial solution distance from LexOpt.

For each makespan problem instance I_{init}, we construct a perturbed instance I_{new} by generating random disturbances. A *job disturbance* is (i) a new job arrival, (ii) a job cancellation, (iii) a processing time augmentation, or (iv) a processing time reduction. A *machine disturbance* is (i) a new machine activation, or (ii) a machine failure. To achieve a bounded degree of uncertainty, i.e. a bounded number k of unstable jobs and number δ of additional machines in the uncertainty set $\mathcal{U}(f, k, \delta)$, we generate $d_n = \lceil 0.2 \cdot n \rceil$ job disturbances and $d_m = \lceil 0.2 \cdot m \rceil$ machine perturbations. To obtain a different range of perturbation factor values, we disturb job processing times randomly. The type of each *job disturbance* is chosen uniformly at random among the four options (i) - (iv). A new job arrival chooses the new job processing time according to $\mathcal{U}(\{1, \ldots, q\})$, where q is the processing time parameter used for generating the original instance. A job cancellation deletes one among the existing jobs chosen uniformly at random. A processing time augmentation of job $J_j \in \mathcal{J}$ chooses a new processing time uniformly at random with respect to $\mathcal{U}(\{p_j + 1, \ldots, 2 \cdot q\})$. Analogously, a processing time reduction of job $J_j \in \mathcal{J}$ chooses a new processing time at random with respect to $\mathcal{U}(\{1, 2, \ldots p_j - 1\})$. The type of a *machine disturbance* is chosen uniformly at random among options (i) - (ii). A new machine activation increases the number of available machines by one. A machine cancellation deletes an existing machine chosen uniformly at random.

5.4. Rescheduling

This section compares the recovered solution quality to the LexOpt using the Section M.3 initial solutions and perturbed instances. Recall that weighted value $W(S) = \sum_{i=1}^{m} B^{m-i} \cdot C_i(S)$ measures the distance of schedule S from LexOpt. For each instance I_{init}, we recover every solution $S_{init} \in \mathcal{S}(I_{init})$ by applying both binding and flexible recovery strategies from Sections 4.1 and G, respectively. For flexible recovery, we set $g = 0.1n$, i.e. at most 10% of the binding decisions may be modified. The flexible recovery MILP model is run with termination criteria of: (i) 100 CPU seconds timeout, and (ii) 10^{-4} relative error tolerance.

The Figure 5a and 5b scatter plots correlate the recovered solution quality to the initial solution distance from the LexOpt solution for the binding and flexible recovery strategy, respectively. We specify each scatter plot point by the normalized weighted value of an initial solution $S_{init} \in \mathcal{S}(I_{init})$ and the normalized makespan of the corresponding recovered solution S_{rec}. The normalized weighted value of S_{init} is $W^N(S_{init}) = \frac{W(S_{init})}{W^*(I_{init})}$, where $W^*(I_{init})$ is the best weighted value in the CPLEX solution pool for instance I_{init}. Similarly, the normalized makespan of S_{rec}



(a) Binding Recovery

(b) Flexible Recovery

Figure 5: Well-formed instances scatter plots illustrating the recovered solution makespan with respect to the initial solution weighted value.

is $C^N(S_{rec}) = \frac{C_{\max}(S_{rec})}{C^*_{\max}(I_{new})}$, where $C^*_{\max}(I_{new})$ is the makespan of the best binding or flexibly recovered schedule for instance I_{new}. For each initial instance and solution pool generates at least 50 diverse solutions, so there is significant computational overhead in recovering all $\approx 2 \times 10^4$ solutions.

Figure 5a indicates that LexOpt facilitates the Algorithm 3 binding recovery strategy, i.e. the expected recovered solution improves if the initial schedule weighted value decreases. This trend is also verified in Figure 5b related to the flexible recovery strategy. Flexible decisions accomplish more efficient recovery. These findings highlight the importance of LexOpt towards more efficient reoptimization. They also motivate efficient solution methods for scheduling with uncertainty where the planning and recovery phases are investigated together.

6. Discussion

LexOpt is useful in various areas including game theory and fairness. Our work initiates the usage of LexOpt in two-stage robust scheduling. Specifically, (i) we investigate the performance of LexOpt methods (sequential, weighting, highest-rank objective) simultaneously in the context of mixed-integer programming, (ii) we identify a common drawback of existing methods as the lack of strong lower bounding techniques, (iii) we propose a new bounding approach for LexOpt problems.

Our novel bounding approach requires (i) defining vectorial bounds by introducing *vectorial bounds* which bound all objectives simultaneously, similar to an *ideal point* in multiobjective optimization [20], (ii) proving that the branch-and-bound algorithm is optimal with respect to the new definition, i.e. it prunes correctly, (iii) computing efficient bounds tighter than weighting and sequential approaches, i.e. the known alternatives for globally bounding LexOpt problems. The branch-and-bound method (i) avoids the iterative MILP of sequential methods, (ii) does not suffer from the precision issues of weighting methods, and (iii) reduces the symmetry of simultaneous methods. Experimentally, our branch-and-bound algorithm proves global optimal optimality fastest for more instances compared to the other investigated LexOpt methods. The numerical results verify a phase transition in the LexOpt scheduling problem, separating more difficult and easier instances.

We provide new insights on the combinatorial structure of robust scheduling. Technically, the analysis of our recovery strategy (i) exploits the lexicographic optimal substructure, (ii) uses sensitivity lemmas quantifying the effect of instance perturbations, (iii) distinguishes two parts based on whether a machine is stable or unstable, and (iv) obtains a global, tight performance guarantee for all perturbations simultaneously by evaluating the effect of each perturbation type individually and propagating the solution degradation. Our experimental two-stage simulation generates multiple initial solutions and verifies that the closest to LexOpt the initial solution is, the better the recovered solution quality we get.

7. Conclusion

Practical scheduling applications frequently require an initial, nominal schedule that is later modified after uncertainty realization. But modifying the nominal schedule may be difficult, e.g. distributed computing file retransmission [57] or university course timetabling changes [46]. We use reoptimization principles to adapt an initial plan to the solution for the final problem instance [2, 7, 16, 49].

Lexicographic ordering is known to expedite the solution of highly-symmetric mixed-integer optimization problems [3, 22, 21, 47], but our results additionally show that the LexOpt solution of the minimum makespan problem enables positive performance guarantees for the recovered solution. Our guarantees are for worst-case schedules, but the computational results show the lexicographic ordering is also useful in randomly-generated instances. Beyond scheduling, the Verschae [55] proofs suggest that this work can be extended to uncertain min-max partitioning problems with generalized cost functions and other applications, e.g. facility location and network communications. Finally, the new branch-and-bound method is broadly relevant to lexicographic optimization.

Acknowledgments

We gratefully acknowledge support from Engineering & Physical Sciences Research Council Research (EPSRC) [grant EP/M028240/1] and a Fellowship to RM [grant EP/P016871/1].

References

[1] Adams, W., Belotti, P., & Shen, R. (2016). Convex hull characterizations of lexicographic orderings. *Journal of Global Optimization*, *66*, 311–329.

[2] Ausiello, G., Bonifaci, V., & Escoffier, B. (2011). Complexity and approximation in reoptimization. In S. B. Cooper, & A. Sorbi (Eds.), *Computability in Context* chapter 4. (pp. 101–129). Imperial College Press.

[3] Balas, E., Fischetti, M., & Zanette, A. (2012). A hard integer program made easy by lexicography. *Mathematical Programming*, *135*, 509–514.

[4] Bauke, H., Mertens, S., & Engel, A. (2003). Phase transition in multiprocessor scheduling. *Physical Review Letters*, *90*, 158701.

[5] Ben-Tal, A., Ghaoui, L. E., & Nemirovski, A. (2009). *Robust optimization*. Princeton University Press.

[6] Ben-Tal, A., Goryashko, A., Guslitzer, E., & Nemirovski, A. (2004). Adjustable robust solutions of uncertain linear programs. *Mathematical Programming*, *99*, 351–376.

[7] Bender, M. A., Farach-Colton, M., Fekete, S. P., Fineman, J. T., & Gilbert, S. (2015). Reallocation problems in scheduling. *Algorithmica*, *73*, 389–409.

[8] Bertsimas, D., Brown, D. B., & Caramanis, C. (2011). Theory and applications of robust optimization. *SIAM Review*, *53*, 464–501.

[9] Bertsimas, D., & Caramanis, C. (2010). Finite adaptability in multistage linear optimization. *IEEE Transactions on Automatic Control*, *55*, 2751–2766.

[10] Bertsimas, D., & Georghiou, A. (2018). Binary decision rules for multistage adaptive mixed-integer optimization. *Mathematical Programming*, *167*, 395–433.

[11] Bertsimas, D., & Sim, M. (2003). Robust discrete optimization and network flows. *Mathematical Programming*, *98*, 49–71.

[12] Bertsimas, D., & Sim, M. (2004). The price of robustness. *Operations Research*, *52*, 35–53.

[13] Bouveret, S., & Lemaître, M. (2009). Computing leximin-optimal solutions in constraint networks. *Artificial Intelligence*, *173*, 343–364.

[14] Brucker, P. (2007). *Scheduling algorithms (5th Ed.)*. Springer.

[15] Burkard, R. E., & Rendl, F. (1991). Lexicographic bottleneck problems. *Operations Research Letters*, *10*, 303–308.

[16] Chassein, A., & Goerigk, M. (2016). On the recoverable robust traveling salesman problem. *Optimization Letters*, *10*, 1479–1492.

[17] Chassein, A., Goerigk, M., Kasperski, A., & Zieliński, P. (2018). On recoverable and two-stage robust selection problems with budgeted uncertainty. *European Journal of Operational Research*, *265*, 423 – 436.

[18] Cramer, J., & Pollatschek, M. A. (1979). Candidate to job allocation problem with a lexicographic objective. *Management Science*, *25*, 466–473.

[19] Dolan, E. D., & Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming*, *91*, 201–213.

[20] Ehrgott, M. (2006). *Multicriteria optimization*. Springer Science & Business Media.

[21] Fischetti, M., Lodi, A., & Salvagnin, D. (2009). Just MIP it! In *Matheuristics* (pp. 39–70). Springer.

[22] Fischetti, M., & Toth, P. (1988). A new dominance procedure for combinatorial optimization problems. *Operations Research Letters*, *7*, 181–187.

[23] Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

[24] Gent, I. P., & Walsh, T. (1996). The TSP phase transition. *Artificial Intelligence*, *88*, 349–358.

[25] Georgiadis, L., Georgatsos, P., Floros, K., & Sartzetakis, S. (2002). Lexicographically optimal balanced networks. *IEEE/ACM Transactions on Networking*, *10*, 818–829.

[26] Goerigk, M., & Schöbel, A. (2016). Algorithm engineering in robust optimization. In L. Kliemann, & P. Sanders (Eds.), *Algorithm Engineering - Selected Results and Surveys* (pp. 245–279). Springer International Publishing volume 9220 of *Lecture Notes in Computer Science*.

[27] Graham, R. L. (1969). Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, *17*, 416–429.

[28] Gupte, A. (2016). Convex hulls of superincreasing knapsacks and lexicographic orderings. *Discrete Applied Mathematics*, *201*, 150–163.

[29] Hanasusanto, G. A., Kuhn, D., & Wiesemann, W. (2015). K-adaptability in two-stage robust binary programming. *Operations Research*, *63*, 877–891.

[30] Hart, W. E., Laird, C. D., Watson, J.-P., Woodruff, D. L., Hackebeil, G. A., Nicholson, B. L., & Siirola, J. D. (2017). *Pyomo–optimization modeling in python*. (2nd ed.). Springer Science & Business Media.

[31] Hart, W. E., Watson, J.-P., & Woodruff, D. L. (2011). Pyomo : modeling and solving mathematical programs in Python. *Mathematical Programming Compututation*, *3*, 219–260.

[32] Kasperski, A., & Zielinski, P. (2014). Minmax (regret) scheduling problems. *Sequencing and scheduling with inaccurate data*, (pp. 159–210).

[33] Kouvelis, P., & Yu, G. (2013). *Robust discrete optimization and its applications* volume 14. Springer Science & Business Media.

[34] Letsios, D., & Misener, R. (2017). Source code. `https://github.com/cog-imperial/two_stage_scheduling`.

[35] Leung, J. Y. (Ed.) (2004). *Handbook of Scheduling - Algorithms, Models, and Performance Analysis*. Chapman and Hall - CRC.

[36] Liebchen, C., Lübbecke, M., Möhring, R., & Stiller, S. (2009). The concept of recoverable robustness, linear programming recovery, and railway applications. In *Robust and online large-scale optimization* (pp. 1–27). Springer.

[37] Luss, H. (1999). On equitable resource allocation problems: A lexicographic minimax approach. *Operations Research*, *47*, 361–378.

[38] Mistry, M., DIddio, A. C., Huth, M., & Misener, R. (2018). Satisfiability modulo theories for process systems engineering. *Computers & Chemical Engineering*, *113*, 98–114.

[39] Mitchell, D. G., Selman, B., & Levesque, H. J. (1992). Hard and easy distributions of SAT problems. In *Proceedings of the 10th National Conference on Artificial Intelligence. San Jose, CA, July 12-16, 1992.* (pp. 459–465).

[40] Monaci, M., & Pferschy, U. (2013). On the robust knapsack problem. *SIAM Journal on Optimization*, *23*, 1956–1982.

[41] Muldoon, F. M., Adams, W. P., & Sherali, H. D. (2013). Ideal representations of lexicographic orderings and base-2 expansions of integer variables. *Operations Research Letters*, *41*, 32–39.

[42] Nace, D., & Orlin, J. B. (2007). Lexicographically minimum and maximum load linear programming problems. *Operations Research*, *55*, 182–187.

[43] Nasrabadi, E., & Orlin, J. B. (2013). Robust optimization with incremental recourse. *Preprint arXiv:1312.4075*, .

[44] Ogryczak, W. (1997). On the lexicographic minimax approach to location problems. *European Journal of Operational Research*, *100*, 566–585.

[45] Pardalos, P. M., Žilinskas, A., & Žilinskas, J. (2016). *Non-convex multi-objective optimization*. Springer.

[46] Phillips, A. E., Walker, C. G., Ehrgott, M., & Ryan, D. M. (2017). Integer programming for minimal perturbation problems in university course timetabling. *Annals of Operations Research*, *252*, 283–304.

[47] Salvagnin, D. (2005). A dominance procedure for integer programming. *Master's thesis, University of Padua*, .

[48] Sanders, P., Sivadasan, N., & Skutella, M. (2009). Online scheduling with bounded migration. *Mathematics of Operations Research*, *34*, 481–498.

[49] Schieber, B., Shachnai, H., Tamir, G., & Tamir, T. (2017). A theory and algorithms for combinatorial reoptimization. *Algorithmica*, .

[50] Schmeidler, D. (1969). The nucleolus of a characteristic function game. *SIAM Journal on Applied Mathematics*, *17*, 1163–1170.

[51] Sherali, H. D. (1982). Equivalent weights for lexicographic multi-objective programs: Characterizations and computations. *European Journal of Operational Research*, *11*, 367–379.

[52] Sherali, H. D., & Soyster, A. L. (1983). Preemptive and non-preemptive multiobjective programming: relationships and counter examples. *Journal of Optimization Theory and Applications*, *39*, 173–186.

[53] Skutella, M., & Verschae, J. (2016). Robust polynomial-time approximation schemes for parallel machine scheduling with job arrivals and departures. *Mathematics of Operations Research*, *41*, 991–1021.

[54] Soyster, A. L. (1973). Convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations Research*, *21*, 1154–1157.

[55] Verschae, J. (2012). *The Power of Recourse in Online Optimization*. Ph.D. thesis Technische Universität Berlin, Germany.

[56] Wiebe, J., Cecílio, I., & Misener, R. (2018). Data-driven optimization of processes with degrading equipment. *Preprint arXiv:1810.09289*, .

[57] Yu, Z., & Shi, W. (2007). An adaptive rescheduling strategy for grid workflow applications. In *International Parallel and Distributed Processing Symposium (IPDPS)* (pp. 1–8). IEEE.

[58] Zufiria, P. J., & Álvarez-Cubero, J. A. (2017). Generalized lexicographic multiobjective combinatorial optimization. Application to cryptography. *SIAM Journal on Optimization*, *27*, 2182–2201.

Supplementary Material

Dimitrios Letsios, Ruth Misener

[b] *Department of Computing; Imperial College London; South Kensington SW7 2AZ; UK*

Appendix A LexOpt Reformulation Lemma

Lemma 1. *There exists an optimal solution to the LexOpt scheduling problem such that:*

1. $C_i \geq C_{i+1}$, *for* $i = 1, \ldots, m-1$,
2. $\left[\sum_{q=1}^{i-1} C_q\right] + (m-i+1) \cdot C_i \geq \sum_{j=1}^{n} p_j$ *and* $i \cdot C_i + \left[\sum_{q=i+1}^{m} C_q\right] \leq \sum_{j=1}^{n} p_j$, $\forall\ i = 1, \ldots, m$.

Proof:

A LexOpt schedule with non-increasing order of machine completion times is straightforward. The machines are identical, so they may be rearranged in any feasible solution to satisfy the proposed order. For the bounds, observe that $\sum_{i=1}^{m} C_i = \sum_{j=1}^{n} p_j$ because all jobs are feasibly executed. Since $C_i \geq C_{i+1} \geq \ldots \geq C_m$, see that $\sum_{q=1}^{i-1} C_q + (m-i+1) \cdot C_i \geq \sum_{j=1}^{n} p_j$. Similarly, as $C_1 \geq C_2 \geq \ldots \geq C_i$, we conclude that $i \cdot C_i + \sum_{q=i+1}^{m} C_q \leq \sum_{j=1}^{n} p_j$. ∎

Appendix B State-of-the-Art LexOpt Methods

This section describes the sequential, weighting, and highest-rank objective methods for solving LexOpt.

B.1 Sequential Method

A sequential method minimizes the objective functions iteratively with respect to their priorities [18, 15]. In each step, a sequential method optimizes the next objective function. Algorithm 4 computes the optimal vector of values $v_1^*, \ldots, v_{i-1}^*$

* {d.letsios, r.misener}@imperial.ac.uk; Tel: +44 (0) 20759 48315

Algorithm 4 Sequential Method

1: $v_1^* = \min\{C_1 : (\vec{x}, \vec{C}) \in \mathcal{S}\}$.
2: **for** $i = 2, \ldots, m$ **do**
3: $\quad v_i^* = \min\{C_i : x \in S, C_1 = v_1^*, \ldots, C_{i\text{-}1} = v_{i\text{-}1}^*\}$
4: Return the solution computed in the last iteration.

Algorithm 5 Weighting Method

1: Select big-M parameter $B = 2$.
2: **for** $i = 2, \ldots, m$ **do**
3: $\quad$ Set machine weight $w_i = B^{m-i}$.
4: Solve $\min\{\sum_{i=1}^{m} w_i \cdot C_i : (\vec{x}, \vec{C}) \in \mathcal{S}\}$.

from the feasible solutions $\mathcal{S}$ to MILP (2). Note that the i-th step requires all values $v_1^*, \ldots, v_{i-1}^*$. Warm-starting the i-th step using the solution of the $(i-1)$-th step improves the sequential method's efficiency.

B.2 Weighting Method

As shown in Algorithm 5, weighting methods select appropriate weights formulate LexOpt as minimizing a weighted sum of the objectives [51]. The standard way of picking the weights is $w_i = B^{m-i}$, $\forall\ i = 1, 2, \ldots, m$, where the *big-M parameter* B is a sufficiently large constant [52]. The highest-rank objectives are associated with the largest weights. Section M.2 applies the weighting method with the big-M parameter $B = 2$, i.e. the smallest integer greater than 1. This big-M choice $B = 2$ produces a LexOpt solution for the tested instances solved within the specified time limit. The weighted sum also measures the distance of any solution from the LexOpt solution. Figures 5, 12 and 13 use this weighted normalization approach with $B = 2$.

B.3 Highest-Rank Objective Method

A LexOpt solution is also an optimal solution for the mono-objective problem of minimizing the highest-rank objective function C_1. This method enumerates all optimal solutions of the highest-rank objective problem and selects the lexicographically smallest [44], e.g. using the *solution pool* feature in CPLEX or Gurobi to collect non-dominated solutions. The highest-rank objective method is useful

Algorithm 6 Highest-Rank Objective Method

Solve $v_1^* = \min\{C_1 : (\vec{x}, \vec{C}) \in \mathcal{S}\}$.
Compute the solution pool $\mathcal{P} = \{(\vec{x}, \vec{C}) \in \mathcal{S} : C_1 = v_1^*\}$.
Return $\operatorname{lex\,min}\{\vec{C} : (\vec{x}, \vec{C}) \in \mathcal{P}\}$.

Algorithm 7 LexOpt Branch-and-Bound Method using Vectorial Bounds

Q: empty stack
r: root node
push(Q, r)
$I = \{+\infty\}^m$
while $Q \neq \emptyset$ **do**
    $u = \text{top}(Q)$
    for $v \in \text{children}(u)$ **do**
        if v is leaf **then**
            S: schedule of v
            $I = \operatorname{lex\,min}\{I, S\}$
        else
            S: heuristic schedule computed via LPT
            $I = \operatorname{lex\,min}\{I, S\}$
            $\vec{L}$: vectorial lower bound of node v
            if $\vec{L} \leq_{\text{lex}} \vec{C}(I)$ **then**
                push(Q, v)

when (i) solution pool is relatively small, or (ii) exact methods cannot handle the LexOpt problem and the solution pool can be efficiently approximated.

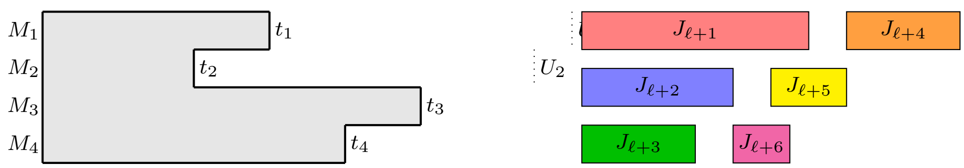
In Algorithm 6, the highest-rank objective problem $v_1^* = \min\{C_1 : (\vec{x}, \vec{C}) \in \mathcal{S}\}$ is the makespan problem, where $\mathcal{S}$ is the set of solutions satisfying (2b) - (2g). Algorithm 6 (i) identifies the solution pool $\mathcal{P}$, and (ii) computes the lexicographically best solution in $\mathcal{P}$, i.e. $\operatorname{lex\,min}\{\vec{C}(S) : S \in \mathcal{P}\}$. In LexOpt, maintaining a single solution in the pool is sufficient if the current solution is always replaced with a lexicographically smaller solution. A simple greedy lexicographic comparison algorithm checks when such an update is essential.

Appendix C Branch-and-Bound Algorithm Pseudocode

This section formally describes the Section 3 branch-and-bound algorithm (Algorithm 7).

Algorithm 8 Longest Processing Time First (LPT) at level ℓ

$\vec{t}$: Initial machine completion times
for $j = \ell + 1, \dots, n$ **do**
  $i = \arg\min_{M_q \in \mathcal{M}} \{t_q\}$
  $C_i \leftarrow t_i + p_j$
Sort the machines so that $C_1 \geq \dots \geq C_m$.



(a) Partial schedule associated with node v.   (b) Remaining jobs.

Figure 6: Computing vectorial lower bound component L_i at node v in the ℓ-th search tree level, by scheduling jobs $J_{\ell+1}, \dots, J_n$ in the partial schedule of v. Jobs $J_{\ell+1}, \dots, J_h$ are rejected in the intervals $[t_q, U_q]$, for $q = 1, \dots, i-1$, and L_i is computed by scheduling jobs $J_{h+1}, \dots, J_m$ on machines $M_i, \dots, M_m$, fractionally, and lower bounding the completion time of machine M_i.

Appendix D Longest Processing Time First Heuristic

This section presents the *Longest Processing Time First (LPT)* (formally described by Algorithm 8) applied in each branch-and-bound tree node v by Algorithm 7 [27]. If node v is located in level ℓ, LPT maintains the assignment for jobs $J_1, \dots, J_\ell$ and greedily assigns the remaining $J_{\ell+1}, \dots, J_n$ jobs with the ordering $p_{\ell+1} \geq \dots \geq p_n$. At each step, LPT assigns the next job to the least-loaded machine. LPT is a powerful heuristic for the LexOpt makespan problem because it (i) produces a 4/3-approximate schedule [27] and (ii) makes the lexicographically best decision in each step.

Appendix E Vectorial Bounds Correctness

This section shows the correctness of Algorithms 1 - 2 for computing vectorial bounds within our branch-and-bound algorithm.

E.1 Vectorial Lower Bound

Lemma 2. *Consider a node v of the search tree and a machine index $i \in \{1, \dots, m\}$. Algorithm 1 produces a value $L_i \leq C_i(S)$ for each feasible schedule $S \in \mathcal{S}(v)$ below* *v such that* $C_q(S) \leq U_q$, $\forall\ q = 1, \dots, i-1$.

Proof:

In schedule S and the pseudo-schedule $\widetilde{S}$ constructed by Algorithm 1, jobs $J_1, \dots, J_\ell$ are assigned to the m machines equivalently and the vector $\vec{t} = (t_1, \dots, t_m)$ specifies the machine completion times with respect to these jobs. All remaining jobs $\mathcal{R} = \{J_{\ell+1}, \dots, J_n\}$ are scheduled differently in $\widetilde{S}$ and S. Schedule $\widetilde{S}$, i.e. Algorithm 1, assigns fractionally the jobs in $\widetilde{\mathcal{R}} = \{J_{\ell+1}, \dots, J_h\}$ to machines $M_1, \dots, M_{i-1}$ and the jobs in $\mathcal{R} \setminus \widetilde{\mathcal{R}} = \{J_{h+1}, \dots, J_n\}$ to machines $M_i, \dots, M_m$. Denote by $\mathcal{R}' \subseteq \mathcal{R}$ the corresponding subset of jobs assigned to machines $M_1, \dots, M_{i-1}$, in schedule S. That is, the jobs in $\mathcal{R} \setminus \mathcal{R}'$ are assigned to machines $M_i, \dots, M_m$.

Initially, observe that $\sum_{J_j \in \mathcal{R}'} p_j = \sum_{q=1}^{i-1} (C_q(S) - t_q) \leq \sum_{q=1}^{i-1} (U_q - t_q) \leq \sum_{J_j \in \widetilde{\mathcal{R}}} p_j$, where the first equality holds by definition, the first inequality is based on the assumption that $C_q(S) \leq U_q$, for each $q = 1, 2, \dots, i-1$, and the second inequality is true because Algorithm 1 fits machines $M_1, \dots, M_{i-1}$ at least up to their respective upper bounds, in the first phase. Thus, $\sum_{J_j \in \mathcal{R} \setminus \mathcal{R}'} p_j \geq \sum_{J_j \in \mathcal{R} \setminus \widetilde{\mathcal{R}}} p_j$.

Next, we claim that $\max_{J_j \in \mathcal{R} \setminus \mathcal{R}'} \{p_j\} \geq \max_{J_j \in \mathcal{R} \setminus \widetilde{\mathcal{R}}} \{p_j\}$. Recall $\max_{J_j \in \mathcal{R} \setminus \widetilde{\mathcal{R}}} \{p_j\} = p_{h+1}$ and $\widetilde{\mathcal{R}}$ consists of jobs $J_{\ell+1}, \dots, J_h$. Assume for contradiction that $\max_{J_j \in \mathcal{R} \setminus \mathcal{R}'} \{p_j\} < p_{h+1}$. Then, $\mathcal{R}'$ must contain all jobs $J_{\ell+1}, \dots, J_{h+1}$. Hence, $\sum_{J_j \in \mathcal{R}'} p_j \geq \sum_{j=\ell+1}^{h+1} p_j > \sum_{J_j \in \widetilde{\mathcal{R}}} p_j$, which is a contradiction.

Because schedule S is feasible, it holds that $C_1(S) \geq \ldots \geq C_m(S)$. We show that $L_i \leq C_i(S)$ by considering three cases. First, since schedule S assigns a job of processing time $\max_{J_j \in \mathcal{R} \setminus \mathcal{R}'} \{p_j\}$ to a machine in $M_i, \dots, M_m$,

$$C_i(S) \geq \min_{i \leq q \leq m} \{t_q\} + \max_{J_j \in \mathcal{R} \setminus \mathcal{R}'} \{p_j\} \geq \min_{i \leq q \leq m} \{t_i\} + p_{h+1}.$$

Second, it is clear that, $C_i(S) \geq \max_{i \leq q \leq m} \{t_i\}$. Finally, based on a standard packing argument and the fact that $\sum_{J_j \in \mathcal{R} \setminus \mathcal{R}'} p_j \geq \sum_{J_j \in \mathcal{R} \setminus \widetilde{\mathcal{R}}} p_j$, if the quantity $\Lambda = \sum_{j=h+1}^{n} p_j - \sum_{q=i}^{m} (\tau - t_q)$ is positive, where $\tau = \max_{i \leq q \leq m} \{t_q\}$, then

$$C_i(S) \geq \max_{i \leq q \leq m} \{t_q\} + \frac{\Lambda}{m - i + 1}.$$

■

E.2 Vectorial Upper Bound

Lemma 7. *Consider a node v of the search tree and a machine index $i \in \{1, \ldots, m\}$. Algorithm 2 produces a value $U_i \geq C_i(S)$ for each feasible schedule $S \in \mathcal{S}(v)$ below v such that $C_q(S) \geq L_q$, $\forall\ q = 1, \ldots, i-1$.*

Proof:

Recall that jobs $J_1, \ldots, J_\ell$ are assigned equivalently in the Algorithm 2 pseudo-schedule $\tilde{S}$ and schedule S. Moreover, vector $\vec{t} = (t_1, \ldots, t_m)$ specifies the identical machine completion times of $\tilde{S}$ and S with respect to these jobs. Let $\mathcal{R} = \{J_{\ell+1}, \ldots, J_n\}$ be the set of remaining jobs. Denote by $\widetilde{\mathcal{R}} = \{J_{\ell+1}, \ldots, J_h\} \subseteq \mathcal{R}$ (considering only the appropriate piece of J_h) the subset of jobs assigned to machines $M_i, \ldots, M_m$ in $\widetilde{S}$ and by $\mathcal{R}' \subseteq \mathcal{R}$ the corresponding subset of jobs assigned to these machines by schedule S. Arguing similarly to the Lemma 2 proof, note that $\sum_{J_j \in \mathcal{R} \setminus \widetilde{\mathcal{R}}} p_j \geq \sum_{J_j \in \mathcal{R} \setminus \mathcal{R}'} p_j$. Additionally, $\max_{J_j \in \mathcal{R} \setminus \widetilde{\mathcal{R}}} \{p_j\} \geq \max_{J_j \in \mathcal{R} \setminus \mathcal{R}'} \{p_j\}$.

Since schedule S is feasible, $C_1(S) \geq \ldots \geq C_m(S)$. Further, the total load assigned to machines $M_i, \ldots, M_m$ among jobs $J_{\ell+1}, \ldots, J_n$ is clearly $\sum_{J_j \in \mathcal{R} \setminus \mathcal{R}'} p_j \leq \lambda = \sum_{j=\ell+1}^{n} p_j - \sum_{q=1}^{i-1} (L_q - t_q)$. To compute U_i, Algorithm 2 assigns an amount of load λ fractionally and uniformly to the least loaded machines among $M_i, \ldots, M_m$. In particular, it sorts these machines so that $t_i \leq \ldots \leq t_m$, and it assigns λ units of processing time to machines $M_i, \ldots, M_\mu$ so that end up having the same completion time $\tau = \frac{1}{\mu - i + 1}\left(\sum_{q=1}^{\mu} t_q + \lambda\right)$. Recall that $\max_{J_j \in \mathcal{R}} \{p_j\} = p_\ell$. Using a simple packing argument, $C_i(S) \leq \max\{\tau + p_\ell, t_m\}$. ∎

Appendix F Makespan Recovery $\mathcal{NP}$-Hardness Proof

This section shows the $\mathcal{NP}$-hardness of the makespan recovery problem even for a single perturbation. That is, an optimal solution for a neighboring instance does not mitigate the computational complexity and recovery remains at least as hard as the original makespan problem.

A solution S_{init} to makespan problem instance I_{init} partitions the set $\mathcal{J}$ of jobs into m components $\mathcal{K}_1, \ldots, \mathcal{K}_m$, where $\mathcal{K}_i$ corresponds to the jobs S_{init} assigns to machine $M_i \in \mathcal{M}$. Define the *critical component* as $\mathcal{K}^* = \arg\max\{F(\mathcal{K}_i) : 1 \leq$

$i \leq m\}$, where $F(\mathcal{K}_i) = \sum_{J_j \in \mathcal{K}_i} p_j$, for each $i = 1, \ldots, m$. The Theorem 5 $\mathcal{NP}$-hardness proof stems observing that the makespan objective inherently depends on the critical components, i.e. the objective value of solution S_{init}. Intuitively, Theorem 5 starts from problem instance I and constructs instance I_{init} with all jobs of I and additional dummy jobs with sufficiently large processing time so that at least one dummy job belongs to the critical component in S_{init}. Then, we perturb I_{init} to balance the effect of the dummy jobs on the new objective value in I_{new}. Solving I_{new} becomes equivalent to solving the computationally intractable instance I.

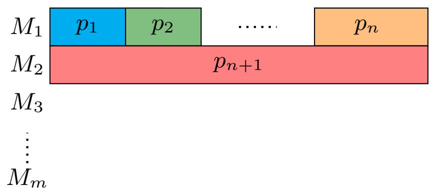
Theorem 5. *Makespan recovery scheduling with a single perturbation is strongly $\mathcal{NP}$-hard.*

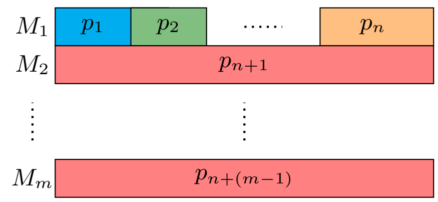
Proof:

Consider the four perturbation types in Section 2.3. We derive an $\mathcal{NP}$-hardness reduction for each perturbation type individually: the reduction uses the same initial instance I_{init} and schedule S_{init} for each type. These reductions start from the strongly $\mathcal{NP}$-hard makespan problem [23] and are based on makespan problem degeneracy. Given an instance $I = (m, \mathcal{J})$ and a target makespan T, the decision version of the makespan problem asks if feasible schedule S for I with makespan $C_{\max}(S) \leq T$ exists. This proof considers instance $I = (m, \mathcal{J})$ and constructs a makespan recovery problem instance $(I_{init}, S_{init}, I_{new})$ with target makespan T_{new}, see Figure 7.

Type 1: Job removal, processing time reduction.

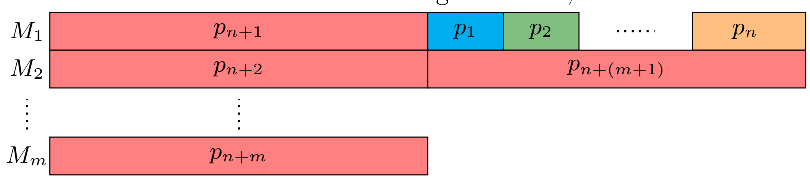
Initial instance I_{init} consists of m machines, the n original jobs, and a dummy job with processing time $p_{n+1} = \sum_{j=1}^{n} p_j$. Schedule S_{init} assigns all jobs $J_1, \ldots, J_n$ to machine M_1, job J_{n+1} to machine M_2, while all machines $M_3, \ldots, M_n$ are empty. Clearly, S_{init} is optimal for I_{init} because any optimal schedule S_{init} for I_{init} has makespan $C_{\max}(S_{init}) \geq p_{n+1}$. Instance I_{new} is obtained from I_{init} by removing job J_{n+1}. We set $T_{new} = T$. Since I_{new} consists only of the jobs in I, I_{new} admits a feasible schedule of makespan T_{new} iff I admits a schedule of makespan T. The processing time reduction case is treated similarly, assuming that p_{n+1} is decreased from $\sum_{j=1}^{n} p_j$ down to 0.





(a) Type 1: Job cancellation, processing time reduction

(b) Types 2 & 4: Job arrival, processing time augmentation, machine failure



(c) Type 3: Machine activation

Figure 7: Constructed instance I_{init} and schedule S_{init} in our makespan recovery $\mathcal{NP}$-hardness reductions for different perturbation types. Jobs $J_1, \ldots, J_n$ are derived from the makespan problem instance $I = (m, \mathcal{J})$ with $\vec{p} = (p_1, \ldots, p_n)$, while each dummy job $J_{n+1}, \ldots, J_{n+(m+1)}$ has processing time $\sum_{j=1}^{n} p_j$.

Types 2 & 4: Job arrival, processing time augmentation, machine failure.

Construct an initial instance I_{init} with m machines, all original n jobs and $m-1$ additional dummy jobs $J_{n+1}, \ldots, J_{n+m-1}$, each one of processing time $p_k = \sum_{j=1}^{n} p_j$, for $k = n+1, \ldots, m-1$. The initial schedule S_{init} assigns all original jobs $J_1, \ldots, J_n$ to machine M_1 and a dummy job to each other machine $M_2, \ldots, M_m$. Clearly, S_{init} is optimal for I_{init} as it is perfectly balanced. Perturb the initial instance I_{init} by adding job J_{n+m} with processing time $p_{n+m} = \sum_{j=1}^{n} p_j$. That is, $I_{new} = (m, n+m, \vec{p}_{new})$ with $\vec{p}_{new} = (p_1, \ldots, p_{n+m})$. Furthermore, we set $T_{new} = \sum_{j=1}^{n} p_j + T$. In the constructed makespan recovery problem instance we ask the existence of a feasible schedule S_{new} with makespan $C_{\max}(S_{new}) \leq T_{new}$. Since $T < \sum_{j=1}^{n} p_j$, if such a schedule exists, every pair of dummy jobs must executed by a different machine. Thus, I and T is a yes-instance of the makespan problem iff I_{new} and T_{new} is yes-instance for the makespan recovery problem.

The processing time augmentation case uses the same arguments assuming an extra dummy job J_{n+m} with $p_{n+m} = 0$ in I_{init}, whose processing time becomes

$\sum_{j=1}^{n} p_j$ in I_{new}. Finally, in the machine removal case, instance I_{new} is perturbed removing machine M_1.

Type 3: Machine activation

Construct the initial instance I_{init} with m machines, all n original jobs, and $m+1$ dummy jobs $J_{n+1}, \ldots, J_{n+(m+1)}$ such that $p_\ell = \sum_{j=1}^{n} p_j$, for $\ell = n+1, \ldots, n+(m+1)$. The initial schedule S_{init} schedules a dummy job and all n original jobs on machine M_1, two dummy jobs on machine M_2 and one dummy job on each machine $M_3, \ldots, M_m$. Any feasible schedule assigns at least two dummy jobs on one machine and has makespan at least $2 \cdot \sum_{j=1}^{n} p_j$, so S_{init} is optimal. Now perturb instance I_{init} by adding a new machine and set $T_{new} = \sum_{j=1}^{n} p_j + T$. As $T < \sum_{j=1}^{n} p_j$, any feasible schedule for I_{new} of length T_{new} must assign one dummy job on every machine. There is a feasible schedule of makespan T_{new} for I_{new} iff there is a feasible schedule of makespan T for I. ∎

Appendix G Flexible Recovery Algorithm

This section presents an alternative, more flexible recovery strategy to Algorithm 3 that allows modifying a bounded number of binding decisions, e.g. as in [43, 17]. Section M.4 numerically investigates the importance of LexOpt for more flexible recovery. The sequel formulates the makespan recovery problem with a bounded number of allowable binding decision adaptations as an MILP.

Let $\mathcal{J}^B = \{J_j \in \mathcal{J}_{init} \cap \mathcal{J}_{new} : \exists i \text{ with } x_{i,j}(S_{\text{init}}) = 1\}$ be the subset of binding jobs which appear in both I_{init}, I_{new} and schedule S_{init} assigns these jobs to common machines of I_{init}, I_{new}. Algorithm 3 maintains the assignment of jobs $\mathcal{J}^B$ as in S_{init} and greedily schedules the free jobs $\mathcal{J}^F = \mathcal{J} \setminus \mathcal{J}^B$. A more flexible recovery strategy allows migrating a bounded number b of binding jobs in $\mathcal{J}^B$. These migrations incur better solution quality at the price extra computational effort and higher transformation cost. In S_{init}, denote by μ_j the machine index which job $J_j \in \mathcal{J}^B$ is initially assigned to and by $\mathcal{J}_i^B \subseteq \mathcal{J}^B$ the subset of binding jobs initially assigned to machine M_i. To formulate the makespan recovery problem with a bounded number g of allowable transformations as an MILP, we extend the minimum makespan MILP (1) by adding the following constraint:

$$\sum_{J_j \in \mathcal{J}^B} \sum_{M_i \in \mathcal{M} \setminus \{M_{\mu_j}\}} x_{i,j} \leq g \tag{3}$$

Appendix H Single Perturbation Negative Performance Guarantee

Theorem 6. *For the makespan recovery problem with a single perturbation, Algorithm 3 produces an $\Omega(m)$-approximate solution if S_{init} is an arbitrary optimal schedule.*

Proof:

We consider job J_n cancellation, but processing time reduction with large decrease of p_n is equivalent. The proof develops a makespan recovery instance where Algorithm 3 produces a schedule S_{rec} with makespan $\Omega(m)$ times far from the makespan of schedule S_{new} for I_{new}.

Figure 3a depicts the structure of instance I_{init} and schedule S_{init}. Instance I_{init} has m machines and $n+1$ jobs with processing times $\vec{p} = (p_1, \ldots, p_n, \sum_{j=1}^{n} p_j)$. Set $p_j = p$, for $j = 1, \ldots, n$ and $p > 0$, and $n = k \cdot m$ for some integer $k \in \mathbb{Z}^+$. Initial optimal schedule S_{init} assigns jobs $J_1, J_2, \ldots, J_n$ to machine M_1, job J_{n+1} to machine M_2 and keeps the other machines $M_3, \ldots, M_m$ empty. Since $C_{\max}(S_{init}) = p_{n+1}$, schedule S_{init} is optimal for I_{init}. Instance I_{new} arises when J_{n+1} cancels. Algorithm 3 returns solution S_{rec} with makespan $C_{\max}(S_{rec}) = \sum_{j=1}^{n} p_j$. But an optimal schedule S_{new} for I_{new} has makespan $C_{\max}(S_{new}) = \frac{1}{m} \sum_{j=1}^{n} p_j$. ∎

Appendix I Recovery Strategy Analysis Omitted Lemma Proofs

This section proves Lemmas 4 and 5.

Lemma 4. *Consider a makespan problem instance $(m, \mathcal{J})$ and let S be LexOpt schedule. Given a machine $M_\ell \in \mathcal{M}$, denote by $\mathcal{J}'$ the subset of all jobs assigned to the machines in $\mathcal{M} \setminus \{M_\ell\}$ by S. Then, it holds that:*

1. $\max_{M_i \in \mathcal{M} \setminus \{M_\ell\}} \{C_i(S)\} = C^*_{\max}(m-1, \mathcal{J}')$, *and*
2. $C^*_{\max}(m-1, \mathcal{J}) \leq 2 \cdot C^*_{\max}(m, \mathcal{J})$.

Proof:

Suppose for contradiction that $\max_{M_i \in \mathcal{M} \setminus \{M_\ell\}} \{C_i(S)\} > C^*_{\max}(m-1, \mathcal{J}')$. Let S^* be an optimal schedule for makespan problem instance $(m-1, \mathcal{J}')$, i.e. $C_{\max}(S^*) = C^*_{\max}(m-1, \mathcal{J}')$. Construct schedule $\widetilde{S}$ by scheduling the jobs in $\mathcal{J}'$ as in S^* and maintaining the assignments of the jobs in $\mathcal{J} \setminus \mathcal{J}'$ to machine M_ℓ as in the LexOpt schedule S for $(m, \mathcal{J})$. Schedule $\widetilde{S}$ is feasible for $(m, \mathcal{J})$ and $\widetilde{S} <_{\text{lex}} S$, which is a contradiction.

Starting from a minimum makespan schedule S^* for the job set $\mathcal{J}$ on m machines, produce a new schedule $\widetilde{S}$ by moving all jobs assigned to machine M_m to the end of machine M_{m-1} without modifying the schedule of the remaining jobs. Clearly, $\widetilde{S}$ is a feasible schedule for $\mathcal{J}$ on $m-1$ machines and the makespan has at most doubled with respect to S^*. Hence, $C^*_{\max}(m-1, \mathcal{J}) \leq C_{\max}(\widetilde{S}) \leq 2 \cdot C_{\max}(S^*) = 2 \cdot C^*_{\max}(m, \mathcal{J})$. ∎

Lemma 5. *Consider a makespan problem instance $(m, \mathcal{J})$ and let S be a LexOpt schedule for it.*

1. *Given the subset $\mathcal{J}' \subseteq \mathcal{J}$ of jobs scheduled on the subset $\mathcal{M}' \subseteq \mathcal{M}$ of machines, where $|\mathcal{M}'| = m'$, the sub-schedule of S on $\mathcal{M}'$ is optimal for $(m' \mathcal{J}')$, i.e.* $\max_{M_i \in \mathcal{M}'} \{C_i(S)\} = C^*_{\max}(m', \mathcal{J}')$.
2. *Assuming that $M_i, M_\ell \in \mathcal{M}$ are two different machines and that job $J_j \in \mathcal{J}$ is assigned to machine M_i in S, then $C_\ell(S) \geq C_i(S) - p_j$.*
3. *It holds that $C^*_{\max}(\mathcal{J}, m-\ell) \leq \left(1 + \left\lceil \frac{\ell}{m-\ell} \right\rceil\right) \cdot C^*_{\max}(\mathcal{J}, m) \; \forall \, \ell \in \{1, \ldots, m-1\}$.*
4. *Let $(m, \hat{\mathcal{J}})$ be a makespan problem instance such that $\frac{1}{f} \cdot \hat{p}_j \leq p_j \leq \hat{p}_j$, where $\mathbf{p}$ and $\hat{\mathbf{p}}$ are the processing times in $(m, \mathcal{J})$ and $(m, \hat{\mathcal{J}})$, respectively. Then,* $\frac{1}{f} \cdot C^*_{\max}(m, \hat{\mathcal{J}}) \leq C^*_{\max}(m, \mathcal{J}) \leq C^*_{\max}(m, \hat{\mathcal{J}})$.

Proof:

1. Assume for contradiction that $\max_{M_i \in \mathcal{M}'} \{C_i(S)\} > C^*_{\max}(m', \mathcal{J}')$. Let S^* be an optimal schedule for instance $(m', \mathcal{J}')$, i.e. $C_{\max}(S^*) = C^*_{\max}(m', \mathcal{J}')$. Construct schedule $\widetilde{S}$ by scheduling the jobs in $\mathcal{J}'$ as in S^* and maintaining the assignments of the jobs in $\mathcal{J} \setminus \mathcal{J}'$ as in S. Schedule $\widetilde{S}$ is feasible for $(m, \mathcal{J})$ and $\widetilde{S} <_{\text{lex}} S$, which contradicts the fact that S is LexOpt.

2. We use an exchange argument. Assume for contradiction that $C_\ell(S) <$

$C_i(S) - p_j$. Then,

$$C_\ell(S) < \max\{C_\ell(S) + p_j, C_i(S) - p_j\} < C_i(S).$$

Consider schedule $\widetilde{S}$ obtained from S by only moving J_j to machine M_ℓ. Schedule $\widetilde{S}$ has $C_i(\widetilde{S}) = C_i(S) - p_j$, $C_\ell(\widetilde{S}) = C_\ell(S) + p_j$, and $C_{i'}(\widetilde{S}) = C_{i'}(S)$, for $M_{i'} \in \mathcal{M} \setminus \{M_i, M_\ell\}$. These inequalities imply that $\widetilde{S} <_{lex} S$ which contradicts the fact that S is LexOpt.

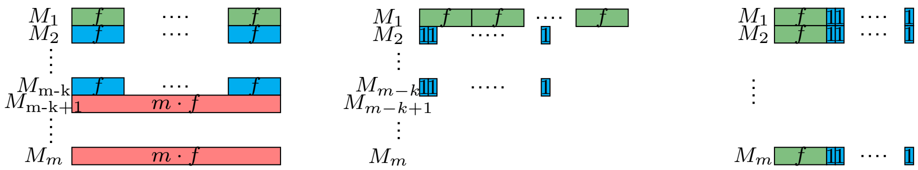
3. Starting from a minimum makespan schedule S^* for $(m, \mathcal{J})$, produce a new schedule $\widetilde{S}$ by moving all jobs scheduled on machines $M_{m-\ell+1}, \ldots, M_m$ to the remaining machines via round-robin. Specifically, for $i = 1, 2, \ldots, \ell$, all content of machine $M_{m-\ell+i}$ is moved to machine $M_{i \bmod (m-\ell)}$, where M_0 corresponds to machine $M_{m-\ell}$. The jobs of the ℓ greatest indexed machines are moved to the $m - \ell$ smallest indexed machines. Machine $M_i \in \{M_1, \ldots, M_{m-\ell}\}$ receives jobs from at most $\lceil \ell/(m-\ell) \rceil$ machines. Clearly, $\widetilde{S}$ is a feasible schedule for $\mathcal{J}$ on $m-\ell$ machines and the makespan has at most multiplied by $1 + \lceil \frac{\ell}{m-\ell} \rceil$ with respect to S^*. Hence,

$$C^*_{\max}(\mathcal{J}, m-\ell) \leq C_{\max}(\widetilde{S}) \leq \left(1 + \left\lceil \frac{\ell}{m-\ell} \right\rceil\right) \cdot C_{\max}(S^*) = \left(1 + \left\lceil \frac{\ell}{m-\ell} \right\rceil\right) \cdot C^*_{\max}(\mathcal{J}, m)$$

4. Let S^* be an optimal schedule for $(m, \mathcal{J})$ with $C_{\max}(S^*) = C^*_{\max}(m, \mathcal{J})$ and construct the feasible schedule $\hat{S}$ for $(m, \hat{\mathcal{J}})$ with equivalent assignments as those in S^*. If job with processing time p_j is executed by machine $M_i \in \mathcal{M}$ in S^*, then job with processing time $\hat{p}_j$ is also executed by machine M_i in $\hat{S}$. Since $p_j \geq \frac{1}{f} \cdot \hat{p}_j$, we have $C_i(S^*) \geq \frac{1}{f} \cdot C_i(\hat{S})$, for each machine M_i. Given that $C_i(\hat{S}) \geq C^*_{\max}(m, \hat{\mathcal{J}})$, the claim follows. The second inequality holds because the processing times in $\mathcal{J}$ are one-by-one smaller than the processing times in $\hat{\mathcal{J}}$. ∎

Appendix J Multiple Perturbations Analysis Lemma 6 Tightness

Construct a family of makespan recovery problems instances depicted in Figure 8. Consider a makespan problem instance I_{init} with m machines, $(m-k) \cdot m$ jobs of length f and k jobs of length $m \cdot f$. We suppose that f and k are asymptotically lower than m, i.e. $f, k = o(m)$. Schedule S_{init} schedules m jobs of length f on each



(a) I_{init}: LexOpt schedule S_{init}. (b) I_{new}: recovered schedule S_{rec}. (c) I_{new}: optimal schedule S_{new}.

Figure 8: Makespan recovery instance showing the tightness of the performance guarantee $O(f \cdot (1 + \lceil \frac{k}{m-k} \rceil))$ for job cancellations and processing time reductions.

of the first $m-k$ machines, one job of length $m \cdot f$ on each of the remaining k machines, and has makespan $m \cdot f$. Instance I_{init} is perturbed as follows: (i) the jobs on machines $M_2, \ldots, M_{m-k}$ are decreased by a factor f and become of length 1, and (ii) every job assigned to the last k machines is cancelled. The recovered schedule has makespan $C_{\max}(S_{rec}) = m \cdot f$. But in an optimal schedule S_{new}, every machine executes a job of length f and $m-k-1$ jobs of unit length. Therefore, given that $k, f = o(m)$ the performance ratio for such an instance is

$$\frac{C_{\max}(S_{rec})}{C_{\max}(S_{new})} = \frac{m \cdot f}{(m-k-1)+f} = \frac{m \cdot f}{(m-k)\left(1 + \frac{f-1}{m-k}\right)}$$

Instance I_{init} is constructed so that $f, k = o(m)$. Therefore, $\frac{f-1}{m-k}$ tends asymptotically to zero and $\frac{C_{\max}(S_{rec})}{C_{\max}(S_{new})}$ is asymptotically equal to $\frac{mf}{m-k} = (1 + \frac{k}{m-k})f$.

Appendix K Single Perturbation Analysis for Type 2-4 Perturbations and Tightness

This section proves the rest of Theorem 3 by analyzing Algorithm 3 for a single perturbation with (i) job arrival, processing time augmentation, and (ii) machine activation, machine failure. We also show that the Theorem 3 performance guarantee is tight for any perturbation type.

K.1 Job arrival, processing time augmentation

Proposition 1. *A solution obtained by Algorithm 3 is 2-approximate in the case of either a new job arrival, or a processing time augmentation.*

Proof:

We prove the proposition only for the case of a new job arrival. The case of job J_j processing time augmentation holds with the same arguments by treating the extra piece of J_j as a new job assigned to the same machine with the one of J_j in S_{init}.

Let $I_{init} = (m, \mathcal{J})$ be an initial makespan problem instance and S_{init} be the initially computed optimal schedule for I_{init}. Next, assume that I_{init} is perturbed by the arrival of new job J_{n+1} with processing time p_{n+1}. Algorithm 3 keeps identical assignments with the ones in S_{init} for jobs $J_1, \dots, J_n$ and job J_{n+1} is assigned to the machine with the minimum completion time in S_{init}. Suppose that Algorithm 3 schedules the new job J_{n+1} on machine $M_\ell \in \mathcal{M}$. That is, $M_\ell = \arg\min_{M_i \in \mathcal{M}}\{C_i(S_{init})\}$. Let S_{rec} and S_{new} the recovered schedule and an optimal schedule, respectively, for the perturbed instance I_{new}. For every machine $M_i \in \mathcal{M}$, it clearly holds that $C_{\max}(S_{new}) \geq C_i(S_{new})$. Since J_{n+1} is executed by a single machine in S_{new}, $C_{\max}(S_{new}) \geq p_{n+1}$. Consider the auxiliary schedule $\widetilde{S}$ obtained from S_{new} by removing job J_{n+1} and maintaining the remaining job assignments. Schedule $\widetilde{S}$ is feasible for I_{init}. Therefore, $C_{\max}(S_{new}) \geq C_{\max}(\widetilde{S}) \geq C_{\max}(S_{init})$. Then,

$$C_{\max}(S_{rec}) = \max\{C_\ell(S_{init})+p_{n+1}, C_{\max}(S_{init})\} \leq C_{\max}(S_{init})+p_{n+1} \leq 2\cdot C_{\max}(S_{new}).$$

■

K.2 Machine activation, machine failure

Proposition 2. *A solution obtained by Algorithm 3 is 2-approximate in the case of either a machine failure, or a machine activation.*

Proof:

Initially, consider an optimal schedule S_{init} for instance $I_{init} = (m, \mathcal{J})$ and suppose, without loss of generality, that machine M_m fails. Let $\mathcal{J}'$ be the subset of jobs assigned to M_m in S_{init}. Clearly, $C_m(S_{init}) = \sum_{J_j \in \mathcal{J}'} p_j \leq C_{\max}(S_{init})$. Algorithm 3 maintains the binding assignments obtained from S_{init}, for the jobs in $\mathcal{J} \setminus \mathcal{J}'$, and assigns the jobs in $\mathcal{J}'$ to machines $M_1, \dots, M_{m-1}$ using the LPT algorithm.

So, it produces schedule S_{rec} with $C_{\max}(S_{rec}) \leq C_{\max}(S_{init}) + \sum_{j \in \mathcal{J}'} p_j \leq 2 \cdot C_{\max}(S_{init})$. Let S_{new} be an optimal schedule for I_{new}. Since I_{new} has a smaller number of machines and the same jobs compared to I_{init}, it must be the case that $C_{\max}(S_{init}) \leq C_{\max}(S_{new})$. Hence, $C_{\max}(S_{rec}) \leq 2 \cdot C_{\max}(S_{new})$.

Now consider an optimal schedule S_{init} for instance $I_{init} = (m, \mathcal{J})$ and suppose that a new machine M_{m+1} is activated. Algorithm 3 keeps identical assignments with the ones in S_{init} and machine M_{m+1} empty. Let S_{rec} and S_{new} be the algorithm's schedule and a minimum makespan schedule, respectively, for I_{new}. By Lemma 4.2, $C_{\max}(S_{rec}) = C^*_{\max}(m, \mathcal{J}) \leq 2 \cdot C^*_{\max}(m+1, \mathcal{J}) = C_{\max}(S_{new})$. ∎

K.3 Tightness

We show the tightness of our analysis, for each perturbation type. Initially, consider an initial makespan problem instance $I_{init} = (m, \mathcal{J})$ with $n = m + 1$ jobs of equal processing time p. In a LexOpt schedule S_{init}, machine M_1 executes jobs J_1 and J_2 while machine M_i executes job J_{i+1}, for $i = 2, 3, \ldots, m$. That is, $C_{\max}(S_{init}) = 2p$. Next, consider that I_{init} is disturbed by one the following perturbations: (i) job J_n is removed, (ii) processing time p_n is decreased down to zero, or (iii) new machine M_{m+1} is activated. In all cases, the algorithm's schedule has makespan $C_{\max}(S_{rec}) = 2p$. However, an optimal schedule S_{new} for I_{new}, assigns exactly one job of length p to each machine and has makespan $C_{\max}(S_{new}) = p$.

Subsequently, consider an instance I_{init} with one job of length $p_1 = m$ and $(m-1) \cdot m$ jobs of unit length $p_j = 1$, for $j = 2, \ldots, n$, where $n = 1 + (m-1) \cdot m$. LexOpt schedule S_{init} assigns the long job J_1 to machine M_1 and exactly m unit jobs to each machine among $M_2, \ldots, M_m$. That is, $C_{\max}(S_{init}) = m$. Suppose that one of the following perturbations occurs: (i) arrival of job J_{n+1} with processing time $p_{n+1} = m$, (ii) processing time augmentation of p_n up to $m + 1$, or (iii) failure of machine M_1. In all cases, Algorithm 3 returns schedule S_{rec} of makespan $C_{\max}(S_{rec}) = 2m$. But in an optimal schedule S_{new}, a long job is assigned to the same machine with at most one unit job, while every other machine contains exactly $m + 1$ unit jobs. Hence, $C_{\max}(S_{new}) = m + 1$.

Appendix L Multiple Perturbations Analysis for Type 2-4 Perturbations

This section completes the proof of Theorem 6. We prove a positive performance guarantee for the cases where the initial input I_{init} is perturbed by (i) processing time augmentations, (ii) machine activations, and (iii) job arrivals, machine failures.

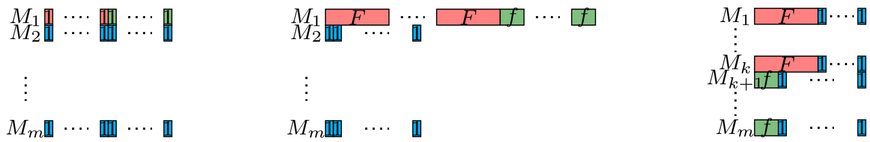
L.1 Type 2: Processing time augmentations

Proposition 3. *Consider a ρ-approximate schedule S_{init} for makespan problem instance I_{init} which is perturbed by processing time augmentations, where at most k processing times are perturbed by a factor more than f. Then, Algorithm 3 produces an $(f+k)$-approximate schedule for I_{new} and this performance guarantee is tight.*

Proof:

Let S_{rec} and S_{new} be the algorithm's recovered schedule and a minimum makespan schedule, respectively, for I_{new}. For each machine $M_i \in \mathcal{M}$, we show that $C_i(S_{rec}) \leq (f+k) \cdot C_{\max}(S_{new})$. Schedule S_{rec} keeps identical assignments with the ones in S_{init} and augmented processing times. We distinguish between *unstable jobs* augmented by a factor more that f and *stable jobs* whose processing time increases by a factor no more than f. In order to bound unstable processing time augmentations, we denote by F be the maximum processing time in I_{new}. We assume that $F > f$ because otherwise our analysis holds by only bounding stable jobs. Since I_{new} belongs to the $\mathcal{U}(f,k,\delta)$ uncertainty set of I_{init}, in the transition from S_{init} to S_{rec}, the schedule of machine M_i is modified as follows: (i) the processing time of at most k jobs increases up to F, and (ii) all remaining jobs are augmented by a factor at most f. Therefore, $C_i(S_{rec}) \leq f \cdot C_i(S_{init}) + k \cdot F$. Next, consider schedule S_{new}. It clearly holds that $C_{\max}(S_{new}) \geq C_{\max}(S_{init})$ because the processing times in I_{new} are one by one greater with respect to I_{init}. Furthermore, given that at least one job of length F is executed by one machine, $C_{\max}(S_{new}) \geq F$. Hence, $C_{\max}(S_{rec}) \leq (f+k) \cdot C_{\max}(S_{new})$.

For the tightness of our analysis, consider an initial makespan problem instance I_{init} with m machines and $n = m^2$ unit-length jobs. An optimal schedule S_{init} attains makespan $C_{\max}(S_{init}) = m$ and each machine executes exactly m jobs.



(a) I_{init}: LexOpt schedule S_{init}. (b) I_{new}: recovered schedule S_{rec}. (c) I_{new}: optimal schedule S_{new}.

Figure 9: Makespan recovery instance which shows that the performance guarantee $O(f+k)$ is asymptotically tight in the case of processing time augmentations.

Suppose that the processing time of k among the jobs assigned to M_1 becomes F and that the processing time of every remaining job assigned to M_1 becomes f. Furthermore, no other processing time is augmented. Schedule S_{rec} performs the same job assignments with the ones in S_{init} and attains makespan $C_{\max}(S_{rec}) = k \cdot F + (m-k) \cdot f$. We consider a family of makespan recovery problem instances such that $F = f + m$, $F = \Theta(m)$, $f = o(m)$ and $k = o(m)$. The perturbed instance I_{new} consists of k jobs of length F, $m-k$ jobs of length f and $m(m-1)$ jobs of unit length. An optimal schedule S_{new} for I_{new} assigns one job of length F and k unit length jobs on machines $M_1, \ldots, M_k$. Moreover, S_{new} assigns one job of length f and $m+k$ unit length jobs one machines $M_{k+1}, \ldots, M_m$. Therefore, $C_{\max}(S_{new}) = F + k$, or $C_{\max}(S_{new}) = f + m + k$. Then, we have that

$$\frac{C_{\max}(S_{rec})}{C_{\max}(S_{new})} = \frac{k \cdot F}{F+k} + \frac{(m-k) \cdot f}{f+m+k} = k \cdot \frac{1}{1+\frac{k}{F}} + f \cdot \frac{1}{1+\frac{f+2k}{m-k}}$$

Since $k = o(F)$ and $k, f = o(m)$, the performance guarantee $k+f$ of our recovery strategy is asymptotically tight. ■

L.2 Type 3: Machine activations

Proposition 4. *Consider a ρ-approximate schedule S_{init} for makespan problem instance I_{init} which is perturbed by k new machine activations. Let I_{new} be the perturbed instance. Algorithm 3 produces a $(1 + \lceil k/m \rceil)$-approximate schedule for I_{new} and this performance guarantee is tight.*

Proof:

Consider the schedule S_{rec} produced by the algorithm. We denote by $\mathcal{M}^s$ the set of *stable machines* which are available in S_{init} and by $\mathcal{M}^u$ the set new activated machines. Our recovery strategy keeps the schedule S_{init} for the machines in $\mathcal{M}^s$ and leaves the machines $\mathcal{M}^u$. That is, $C_{\max}(S_{rec}) = C^*_{\max}(m, \mathcal{J})$. By definition, in an optimal schedule S_{new} for I_{new}, it must be the case that $C_{\max}(S_{new}) = C^*_{\max}(m+k, \mathcal{J})$. By Lemma 5.3, we conclude that $C_{\max}(S_{rec}) \leq (1 + \lceil k/m \rceil) \cdot C_{\max}(S_{new})$.

For the tightness of the analysis, we consider an initial makespan problem instance with m machines and $n = m \cdot (m+k)$ unit jobs. Initial optimal schedule S_{init} executes $m+k$ unit jobs on each machine and has makespan $C_{\max}(S_{init}) = m+k$. Since we only consider machine activations, there are no free decisions and the recovered schedule S_{rec} is the same with S_{init} except that there are k additional empty machines. That is, $C_{\max}(S_{rec}) = m+k$. But an optimal schedule S_{new} for I_{new} assigns exactly m jobs on each machine and has makespan $C_{\max}(S_{new}) = m$. Thus, $\frac{C_{\max}(S_{rec})}{C_{\max}(S_{init})} = 1 + \frac{k}{m}$. ■

L.3 Type 4: Job arrivals & machine failures

Proposition 5. *Consider a ρ-approximate schedule S_{init} for makespan problem instance I_{init} which is perturbed by new job arrivals and machine failures. Algorithm 3 produces a $\max\{2, \rho\}$-approximate schedule for the perturbed instance I_{new} and this performance guarantee is tight.*

Proof:

Initial schedule S_{init} does not provide any assignment for the jobs whose machine has failed in I_{new}. Therefore, these assignments together with the ones of the newly arrived jobs are treated as free decisions and are performed by our recovery strategy using Longest Processing Time (LPT) first.

Consider the recovered schedule S_{rec} by Algorithm 3. We partition the machines into the set $\mathcal{M}^s$ of *stable machines*, which are not assigned any free jobs, and the set $\mathcal{M}^u$ of *unstable machines*, which are assigned at least one new free job. We distinguish two cases based on whether there is a critical machine in $\mathcal{M}^s$, or not.

In the former case, the makespan of schedule S_{rec} is equal to the one of the initial schedule S_{init}, i.e. $C_{\max}(S_{rec}) = C_{\max}(S_{init})$. Let S_{new} be an optimal schedule for the perturbed instance I_{new}. Because S_{init} is ρ-approximate for I_{init} and I_{new} contains all jobs in I_{init} as well as some additional new jobs and the same number of machines, it must be the case that $C_{\max}(S_{init}) \leq \rho \cdot C_{\max}(S_{new})$. Thus, $C_{\max}(S_{rec}) \leq \rho \cdot C_{\max}(S_{new})$.

In the latter case, we proceed similarly to the standard list scheduling analysis by [27]. More specifically, let J_j be a job which completes last in S_{rec}. Due to our hypothesis, it must be the case that J_j is among the incoming jobs and is assigned to a machine using LPT. Since J_j was not scheduled earlier than its begin time b_j, all processors are occupied until b_j in S_{rec}. Thus, $C_{\max}(S_{rec}) = b_j + p_j \leq \frac{1}{m}\sum_{J_{j'} \in \mathcal{J}_{new}} p_{j'} + p_j \leq 2 \cdot C_{\max}(S_{new})$.

The tightness of our analysis is derived based on two observations. In the case where $\rho > 2$, we may design a makespan problem instance such that the arrival of a new job with a small processing time has the effect that the recovered schedule remains ρ-approximate. In the case where $\rho \leq 2$, Algorithm 3 cannot be better than 2-approximate as LPT is known to be tightly 2-approximate when the initial instance I_{init} is empty without jobs [27]. ∎

L.4 Tightness

For the tightness of our analysis, we pick an instance with k jobs of processing time p and $(m-k) \cdot p$ unit-length jobs. The fraction $(m-k)/m$ is integer. In schedule S_n^*, the machines $M_1, M_2, \ldots, M_{m-k}$ contain p unit-length jobs each one while every other machine contains exactly one job of processing time p. That is, $C_{\max}(S_n^*) = p$. Then, assume that the k jobs of processing time p are cancelled. Clearly, the algorithm's schedule has makespan $C_{\max}(S_{alg}) = p$. On the other hand, in S_{n-k}^*, each machine $M_i \in \mathcal{M}$ is assigned exactly $(\frac{m-k}{m}) \cdot p$ unit jobs.

For the tightness of the analysis, we consider an initial job set with m jobs; one job of processing time p and $m-1$ unit-length jobs. Then, k new jobs arrive so that exactly k/m jobs are assigned to each machine. Among all k jobs, k/m have processing time p (the jobs J_{n+1}, J_{n+m+1},$J_{n+2m+1}, \ldots$) and $(m-1)(k/m)$ jobs are of unit length. In the algorithm's schedule, all $1 + k/m$ jobs of processing time p are executed by machine M_1, while every other processor executes $1 + k/m$ unit jobs. That is, $C_{\max}(S_{alg}) = (1 + k/m)p$. In the optimal schedule S_{n+k}^*, each machine executes exactly one job of processing time p and $m-1$ unit-length jobs. Therefore, its makespan is $C_{\max}(S^*_{n+k}) = p + (m-1)$ which implies that the algorithm's performance ratio is $\Omega(k/m)$ for appropriate values of k and p.

Appendix M Extended Numerical Results

Section M.1 describes the system specifications and benchmark instances. Section M.2 evaluates the exact methods. Section M.3 discusses generating the perturbed instances. Section M.4 evaluates the recovery strategies and the impact of LexOpt on the recovered solution quality.

M.1 System Specification and Benchmark Instances

We ran all computations on an Intel Core i7-4790 CPU 3.60GHz with a 15.6 GB RAM running a 64-bit Ubuntu 14.04. Our implementations use Python 2.7.6 and Pyomo 4.4.1 [31, 30] and solve the MILP models with CPLEX 12.6.3 and Gurobi 6.5.2. The source code and test cases are available [34]. We have generated random LexOpt scheduling instances. *Well-formed instances* admit an optimal solution close to a *perfect solution* which has all machine completion times equal, i.e. $C_i = C_{i'}$ for each $i, i' \in \mathcal{M}$. *Degenerate instances* have a less-balanced optimal solution.

For randomly-generated makespan scheduling with b-bit integers, instances with small $\kappa = b/n$ values are easier to solve than instances with larger κ values [4]. The *phase transition* from "easy" to "hard" instances becomes sharper as n increases and occurs at threshold value $\kappa^* = \frac{\log_2 m}{m-1}$. Small κ exhibits exponentially many perfect solutions, but for κ larger than the critical value κ^*, the expected number of perfect solutions becomes exponentially small. Similar phase transitions exist, e.g. in satisfiability [39] and the traveling salesman problem [24], where instances near the threshold tend to be the most difficult.

We generate well-formed instances by varying three parameters: (i) the number m of machines, (ii) the number n of jobs, and (iii) a processing time seed q. The well-formed test set, summarized in Table 3a, consists of *moderate*, *intermediate*, and *hard* instances. For each combination of m, n and q, we generate three instances by selecting $\vec{p}$ using three distributions parameterized by q. Each processing time is rounded to the closest integer. *Uniform distribution* selects

Instances	m	n	q
Moderate	$3, 4, 5, 6$	$20, 30, 40, 50$	$100, 1000$
Intermediate	$10, 12, 14, 16$	$100, 200, 300, 400$	$10000, 100000$
Hard	$10, 15, 20, 25$	$200, 300, 400, 500$	$10000, 100000$

(a) Well-formed Instances

Instances	m	n	q
Moderate	3	$20, 25, 30, 35$	$2^{15}, 2^{19}, 2^{23}, 2^{27}$
	4	$25, 30, 35, 40$	$2^{16}, 2^{20}, 2^{23}, 2^{26}$
	5	$30, 35, 40, 45$	$2^{17}, 2^{20}, 2^{23}, 2^{26}$
	6	$35, 40, 45, 50$	$2^{18}, 2^{20}, 2^{23}, 2^{25}$
Intermediate	10	$40, 50, 60, 70$	$2^{14}, 2^{18}, 2^{22}, 2^{25}$
	12	$45, 55, 65, 75$	$2^{14}, 2^{17}, 2^{21}, 2^{24}$
	14	$55, 65, 75, 85$	$2^{16}, 2^{19}, 2^{21}, 2^{24}$
	16	$60, 70, 80, 90$	$2^{16}, 2^{18}, 2^{21}, 2^{24}$

(b) Degenerate Instances

Table 3: Instance Sizes

$p_j \sim \mathcal{U}(\{1, \ldots, q\})$. *Normal distribution* chooses $p_j \sim \mathcal{N}(q, q/3)$ and guarantees that 99.7% of the values lie in interval $[0, 2q]$. *Symmetric of normal distribution* samples $p \sim \mathcal{N}(q, q/3)$ and selects $p_j = q - p$ if $p \in [0, q]$, or $p_j = 2q - (p - q)$ if $p_j \in (q, 2q]$. Normal and symmetric normal processing times outside $[0, 2q]$ are rounded to the closest of 0 and $2q$.

Following [4], we produce the degenerate instances, summarized in Table 3b, by choosing the processing times randomly from $\{1, \ldots, q\}$, where $q = 2^{\lfloor \kappa(m) \cdot n \rfloor}$ and $\kappa(m) = (\log_2 m)/(m-1)$. We select the number of jobs so that the processing time parameter $q = 2^{\lfloor \kappa(m) \cdot n \rfloor}$ does not lead to CPLEX precision issues (the 64-bit CPLEX version stores 32-bit integers). For each combination of m and n with the corresponding $q = 2^{\lfloor \kappa(m) \cdot n \rfloor}$, we generate three instances similarly to the well-formed test set using the uniform, normal and symmetric of normal distributions.

M.2 LexOpt Scheduling

This section discusses implementing the Section 3 LexOpt methods and evaluates them numerically. The sequential, highest-rank objective, and weighting methods require solving MILP instances. We run Algorithms 4-6 and our branch-and-bound algorithm with termination criteria: (i) 10^3 CPU seconds, and (ii) 10^{-4} relative error tolerance, where the relative gap $(Ub - Lb)/Ub$ is computed using the best-found incumbent Ub and the lower bound Lb. The sequential method solves a sequence of minimum makespan MILP models each one with 10^{-4} makespan relative error tolerance. The simultaneous method solves one minimum makespan MILP model with 10^{-4} makespan error tolerance and populates the solution pool with 2000 solutions. The weighting method and our branch-and-bound method terminate with 10^{-4} weighted value error tolerance, where the upper bound Ub is the weighted value $W(S) = \sum_{i=1}^{m} B^{m-i} \cdot C_i(S)$ of the returned schedule S. Here $B = 2$, see Section B.2. We compute Lb by similarly weighting the global vectorial lower bound.

The *sequential method* solves m MILP instances, each computing one objective value in the LexOpt solution. We implement the method with repeated CPLEX calls and the CPLEX reoptimize feature which exploits information obtained from solving higher ranked objectives. If the method exceeds 10^3 CPU seconds in total, it terminates when the ongoing MILP run is completed. Each individual MILP is run with the termination criteria mentioned earlier. We implement the *weighting method* using Pyomo and solve the MILP with CPLEX and Gurobi. As discussed in Section B.2, the method sets parameter $B = 2$. The *highest-rank objective method* uses the CPLEX solution pool feature in two phases. The first phase solves the standard makespan MILP model. The second continues the tree exploration and generates solutions using information stored in the initial phase. We set the solution pool capacity to 2000 and choose as replacement strategy removing the solution with the worst objective value.

The Figure 10 *performance profiles* compare the LexOpt methods with respect to elapsed times and best found solutions on the well-formed instances [19]. In terms of running time and number of solved instances, sequential method performs similarly to weighting method on moderate and intermediate instances, and slightly better on hard instances. But the sequential method produces slightly worse feasible solutions than weighting method since lower-ranked objectives are not optimized in case of a sequential method timeout.

The highest-rank objective method has worse running times than sequential and weighting methods on moderate instances since the solution pool populate time is large compared to the overall solution time. On intermediate and hard test cases, populating the solution pool is only a fraction of the global solution time and highest-rank objective method attains significantly better running times than sequential and weighting methods. The highest-rank objective method does not prove global optimality: it only generates 2000 solutions. But, in terms of solution quality, the highest-rank objective method produces the best heuristic results for most test cases.

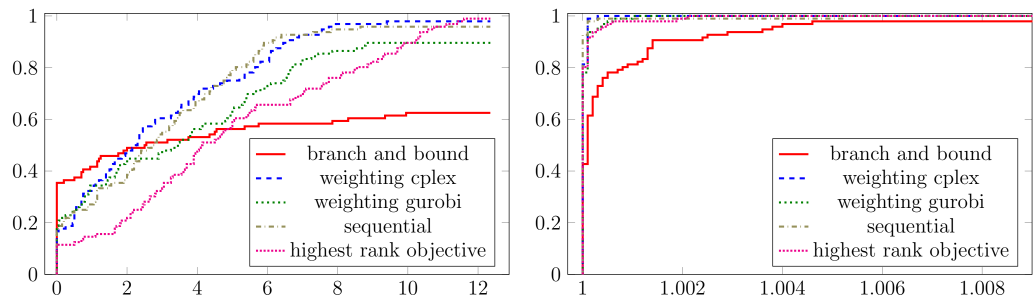
The branch-and-bound method with vectorial lower bounds, obtains good solutions with the LPT heuristic, and avoids populating the entire solution pool. Figure 10 shows that our method guarantees global optimality more quickly than the other approaches for test cases where it converges. Branch-and-bound converges for $> 60\%$ of the moderate test cases, and $> 30\%$ of intermediate and hard instances. Branch-and-bound consistently produces a good heuristic, i.e. better than sequential and weighting methods, in intermediate and hard instances.

Figure 11 compares the LexOpt methods on the degenerate instances. These instances are indeed significantly harder to solve than well-formed instances of identical size. No solver converges for any intermediate degenerate instance, while every solver converges for $> 30\%$ of the intermediate well-formed instances. The solvers also struggle on moderate degenerate instances. In terms of solver comparison, we derive similar results those obtained for the well-formed instances. The weighting method slightly dominates the sequential method. The highest-rank objective method produces the best heuristic results. Our branch-and-bound method produces the second best heuristic result in the majority of degenerate test cases and converges to global optimality quickly for instances where it converges.

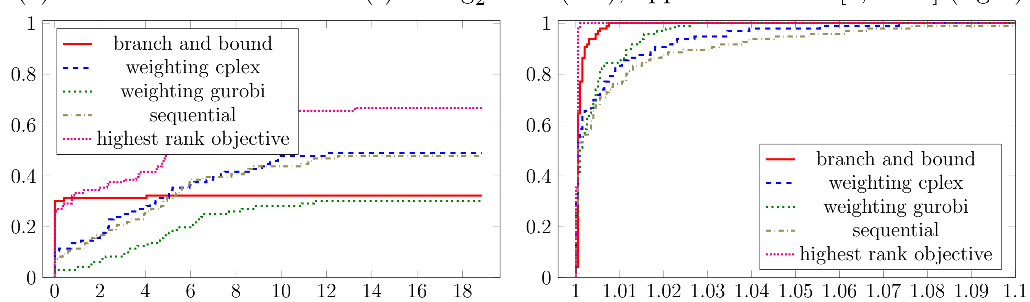
M.3 Generation of Initial Solutions and Perturbed Instances

This section describes generating the benchmark instances for the makespan recovery problem. An instance is specified by: (i) an initial makespan problem instance I_{init}, (ii) an initial solution S_{init} to I_{init}, and (iii) a perturbed instance I_{new}. Recall that the recovery problem transforms solution S_{init} to a feasible solution S_{new} for instance I_{new} using the recovery strategies.

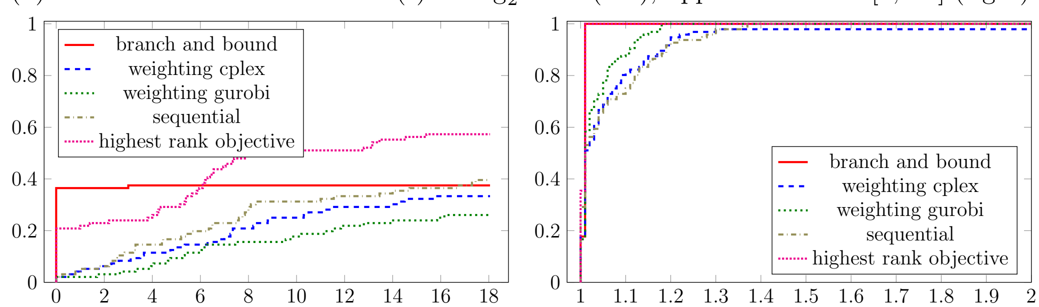
The initial makespan problem instances are the 384 Section M.1 instances. For each instance I_{init}, we generate a set $\mathcal{S}(I_{init})$ of at least 50 diverse solutions by solving I_{init} using the CPLEX solution pool feature and the Section M.2 termination criteria. A key property is that the obtained solutions have, in general, different weighted values, i.e. for many pairs of solutions $S_1, S_2 \in \mathcal{S}(I_{init})$, $W(S_1) \neq W(S_2)$,



(a) Moderate instances: time (s) on $\log_2$ scale (left), upper bounds on $[1, 1.009]$ (right).

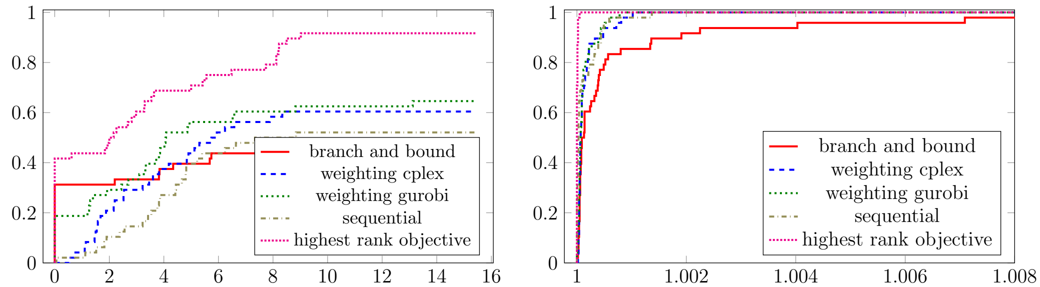


(b) Intermediate instances: time (s) on $\log_2$ scale (left), upper bounds on $[1, 1.1]$ (right).

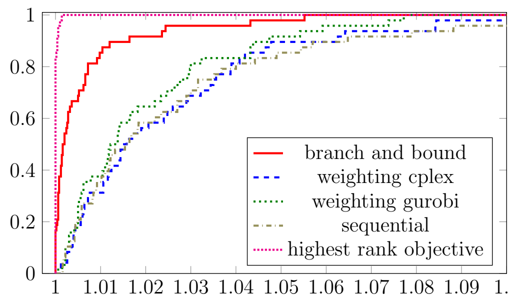


(c) Hard instances: time (s) on $\log_2$ scale (left), upper bounds on $[1, 2]$ (right).

Figure 10: Performance profiles for the *well-formed test set*, 10^3 s timeout.



(a) Moderate instances: time (s) on $\log_2$ scale (left), upper bounds on $[1, 1.008]$ (right).



(b) Intermediate instances: upper bounds on $[1, 1.1]$. *No solver converges for any intermediate degenerate instance within the specified time limit.*

Figure 11: Performance profiles for the *degenerate test set* with 10^3 s timeout.

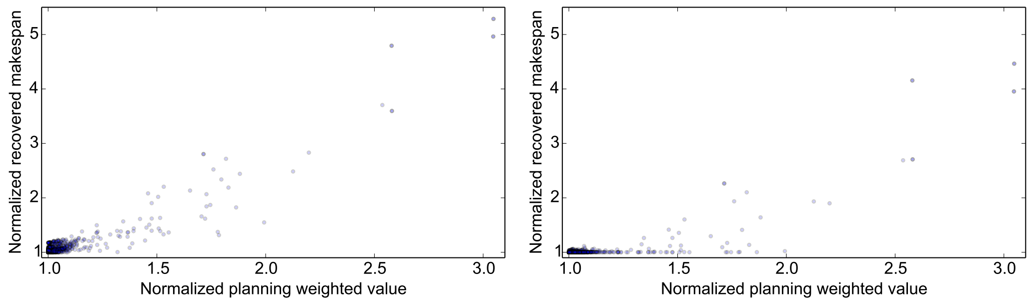
where $W(S) = \sum_{i=1}^{m} B^{m-i} \cdot C_i(S)$. Using the weighted value as a distance measure from the LexOpt solution, we evaluate a recovered solution's quality as a function of the initial solution distance from LexOpt.

For each makespan problem instance I_{init}, we construct a perturbed instance I_{new} by generating random disturbances. A *job disturbance* is (i) a new job arrival, (ii) a job cancellation, (iii) a processing time augmentation, or (iv) a processing time reduction. A *machine disturbance* is (i) a new machine activation, or (ii) a machine failure. To achieve a bounded degree of uncertainty, i.e. a bounded number k of unstable jobs and number δ of additional machines in the uncertainty set $\mathcal{U}(f, k, \delta)$, we generate $d_n = \lceil 0.2 \cdot n \rceil$ job disturbances and $d_m = \lceil 0.2 \cdot m \rceil$ machine perturbations. To obtain a different range of perturbation factor values, we disturb job processing times randomly. The type of each *job disturbance* is chosen uniformly at random among the four options (i) - (iv). A new job arrival chooses the new job processing time according to $\mathcal{U}(\{1, \ldots, q\})$, where q is the processing time parameter used for generating the original instance. A job cancellation deletes one among the existing jobs chosen uniformly at random. A processing time augmentation of job $J_j \in \mathcal{J}$ chooses a new processing time uniformly at random with respect to $\mathcal{U}(\{p_j + 1, \ldots, 2 \cdot q\})$. Analogously, a processing time reduction of job $J_j \in \mathcal{J}$ chooses a new processing time at random with respect to $\mathcal{U}(\{1, 2, \ldots p_j - 1\})$. The type of a *machine disturbance* is chosen uniformly at random among options (i) - (ii). A new machine activation increases the number of available machines by one. A machine cancellation deletes an existing machine chosen uniformly at random.

M.4 Rescheduling

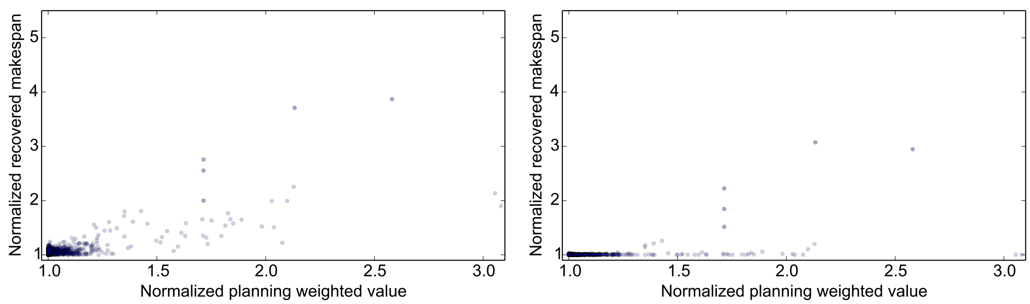
This section compares the recovered solution quality to the LexOpt using the Section M.3 initial solutions and perturbed instances. Recall that weighted value $W(S) = \sum_{i=1}^{m} B^{m-i} \cdot C_i(S)$ measures the distance of schedule S from LexOpt. For each instance I_{init}, we recover every solution $S_{init} \in \mathcal{S}(I_{init})$ by applying both binding and flexible recovery strategies from Sections 4.1 and G, respectively. For flexible recovery, we set $g = 0.1n$, i.e. at most 10% of the binding decisions may be modified. The flexible recovery MILP model is run with termination criteria of: (i) 100 CPU seconds timeout, and (ii) 10^{-4} relative error tolerance.

The Figure 12a and 13a scatter plots correlate the binding recovered solution quality to the initial solution distance from the LexOpt solution on well-formed and degenerate instances. Figures 12b and 13b are the corresponding scatter plots of the flexible recovery strategy. We specify each scatter plot point by the normalized weighted value of an initial solution $S_{init} \in \mathcal{S}(I_{init})$ and the normalized makespan of the corresponding recovered solution S_{rec}. The normalized weighted value of S_{init} is $W^N(S_{init}) = \frac{W(S_{init})}{W^*(I_{init})}$, where $W^*(I_{init})$ is the best weighted value in the CPLEX solution pool for instance I_{init}. Similarly, the normalized makespan of S_{rec} is $C^N(S_{rec}) = \frac{C_{\max}(S_{rec})}{C^*_{\max}(I_{new})}$, where $C^*_{\max}(I_{new})$ is the makespan of the best binding or flexibly recovered schedule for instance I_{new}. There are 384 initial instances and solution pool generates at least 50 diverse solutions, so there is significant computational overhead in recovering all $\approx 2 \times 10^4$ solutions. Figures 12a and 13a plot 5 randomly-selected instances of each among the classes: well-formed moderate, well-formed intermediate, degenerate moderate, degenerate intermediate and we recover all their initial solutions.



(a) Binding Recovery (b) Flexible Recovery

Figure 12: Well-formed instances scatter plots illustrating the recovered solution makespan with respect to the initial solution weighted value.



(a) Binding Recovery (b) Flexible Recovery

Figure 13: Degenerate instances scatter plots illustrating the recovered solution makespan with respect to the initial solution weighted value.

Figures 12a and 13a indicate that LexOpt facilitates the Algorithm 3 binding recovery strategy, i.e. the expected recovered solution improves if the initial schedule weighted value decreases. This trend is also verified in Figures 12b and 13b related to the flexible recovery strategy. Flexible decisions accomplish more efficient recovery. These findings highlight the importance of LexOpt towards more efficient reoptimization. They also motivate efficient solution methods for scheduling with uncertainty where the planning and recovery phases are investigated together.

Appendix N Table of Notation

Name	Description
Indices	
i, q, μ	Machine indices (q, μ typically used as auxiliary machine indices)

j, h, ℓ	Job indices (h, ℓ typically used as auxiliary job indices)
Makespan problem input	
$I = (m, \mathcal{J})$	Makespan problem instance
m	Number of machines
$M_i \in \mathcal{M}$	Set $\mathcal{M} = \{M_1, \dots, M_m\}$ of machines contains each machine M_i indexed by i
n	Number of jobs
$J_j \in \mathcal{J}$	Set $\mathcal{J} \in \{J_1, \dots, J_n\}$ of jobs with processing times $\vec{p}$ contains each job J_j indexed by j
$\vec{p}, p_j$	Vector $\vec{p} = (p_1, \dots, p_n)$ of job processing times contains processing times p_j of job J_j
Makespan problem variables	
$C_{\max}$	Makespan
C_i	Variable corresponding to machine M_i completion time
$x_{i,j}$	Binary variable indicating assignment of job J_j to machine M_i
$\mathcal{K}_i$	Subset of jobs assigned to machine M_i (schedule component)
$\mathcal{K}^*$	Critical component attaining the makespan
Further scheduling notation	
$S = (\vec{y}, \vec{C}), S', \widetilde{S}$	Schedules ($S', \widetilde{S}$ typically used as auxiliary schedules)
S^*	LexOpt schedule
$\mathcal{S}$	Set of all feasible schedules
LexOpt scheduling problem	
$\leq_{\text{lex}}$	Lexicographic comparison operator
F_i	Objective function indexed by i, i.e. i-th greatest completion time
$\vec{F}$	Vector $(F_1, \dots, F_m)$ of objective values
v_i^*	Value of F_i in LexOpt schedule S^*
$\vec{v}^*$	Vector of objective values in LexOpt schedule S^*
$\mathcal{T}_q$	Set of tuples $(i_1, \dots, i_q)$ with q pairwise disjoint machine indices
w_i	Weight of objective function F_i in weighting method
$\mathcal{P}$	Solution pool in highest-rank objective method
Branch-and-Bound	
Q	Stack with visited, unexplored nodes
I	Incumbent, i.e. lexicographically best-found solution
u, v, r	Branch-and-bound tree nodes (r typically used as the root node)
$\mathcal{S}(u)$	Branch-and-bound feasible solutions below node u
ℓ	Branch-and-bound tree level, i.e. job index
t_i	Partial machine M_i completion time at a branch-and-bound node
$\mathcal{R}$	Subset of jobs scheduled below a branch-and-bound node
$\vec{L}, L_i$	Vectorial lower bound $\vec{L} = (L_1, \dots, L_m)$ contains each component L_i
$\vec{U}, U_i$	Vectorial upper bound $\vec{U} = (U_1, \dots, U_m)$ contains each component U_i
τ	Time point
$\tilde{p}_j$	Piece of job J_j
λ, Λ	Amount of processing time load
Recoverable robustness model and rescheduling problem	
I_{init}	Initial instance $(m_{init}, \mathcal{J}_{init})$
I_{new}	Perturbed instance $(m_{new}, \mathcal{J}_{new})$
S_{init}	Initial optimal schedule for I_{init}
S_{rec}	Recovered schedule for I_{new}
S_{new}	Optimal schedule for I_{new}
ρ	Approximation ratio
Uncertainty modeling	

$\mathcal{U}(f, k, \delta)$	Uncertainty set
f	Perturbation factor
k	Number of unstable jobs
δ	Number of new machines
$C^*_{\max}(m, \mathcal{J})$	Optimal objective value of makespan problem instance $(m, \mathcal{J})$
f_a, f_r	Perturbation factor of processing time augmentations (f_a) and reductions (f_r)
k_a, k_r	Number of unstable processing time augmentations (k_a) and reductions (k_r)
δ^+	$\max\{\delta, 0\}$
Recovery with binding decisions	
T	Target makespan for makespan problem instance I
T_{new}	Target makespan for perturbed makespan problem instance I_{new}
$\mathcal{M}'$	Subset of machines
m'	Number of machines in M'
$\mathcal{J}'$	Subset of jobs
η	Processing time decrease of job J_j
$(\hat{m}, \hat{\mathcal{J}})$	Neighboring instance of $(m, \mathcal{J})$
$\hat{p}_j$	Job J_j processing time in $(\hat{m}, \hat{\mathcal{J}})$
$\mathcal{M}^s, \mathcal{M}^u$	Stable machines $\mathcal{M}^s$, i.e. assigned only stable jobs, unstable machines $\mathcal{M}^u = \mathcal{M} \setminus \mathcal{M}^s$
m^s, m^u	Number of stable (m^s) and unstable (m^u) machines
$\mathcal{J}^s_{init}$	Subset of stable jobs in I_{init}
$\mathcal{J}^s_{new}$	Subset of stable jobs in I_{new}
F	Maximum processing time in I_{new}
Flexible recovery	
$\mathcal{J}^B_i$	Binding jobs originally assigned to machine M_i
$\mathcal{J}^B, \mathcal{J}^F$	Subset of binding ($\mathcal{J}^B$) and free ($\mathcal{J}^F = \mathcal{J} \setminus \mathcal{J}^B$) jobs
μ_j	Machine executing job J_j in S_{init}
g	Limit on binding job migrations
Numerical results	
κ	Phase transition parameter
κ^*	Phase transition parameter critical value
b	Number of bits for processing time generation
q	Processing time parameter
$\mathcal{U}$	Discrete uniform distribution
$\mathcal{N}$	Normal distribution
W	Weighted value, i.e. weighted sum of objective functions
Ub, Lb	Best-found incumbent (Ub) and lower bound (Lb)
d_m, d_n	Number of machine (d_m) and job (d_n) disturbances
W^N	Normalized weighted value
W^*	Best computed weighted value
$C^N_{\max}$	Normalized makespan
$C^*_{\max}$	Best recovered makespan

Table 4: Nomenclature