An adaptive robust optimization model for parallel machine scheduling

Cohen, Izack ; Postek, Krzysztof; Shtern, Shimrit

DOI
10.1016/j.ejor.2022.07.018

Publication date
2022

Document Version
Final published version

Published in
European Journal of Operational Research

European Journal of Operational Research 306 (2023) 83–104

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

An adaptive robust optimization model for parallel machine scheduling

Izack Cohen[a,*], Krzysztof Postek[b], Shimrit Shtern[c]

[a] Faculty of Engineering, Bar-Ilan University, Ramat Gan, Israel
[b] Faculty of Electrical Enginnering, Mathematics and Computer Science, Delft University of Technology, Delft, The Netherlands
[c] Faculty of Industrial Engineering and Management, Technion - Israel Institute of Technology, Haifa, Israel

ARTICLE INFO

Article history:
Received 4 October 2021
Accepted 11 July 2022
Available online 16 July 2022

Keywords:
Scheduling
Robust optimization
Parallel machine scheduling
Robust scheduling

ABSTRACT

Real-life parallel machine scheduling problems can be characterized by: (i) limited information about the exact task duration at the scheduling time, and (ii) an opportunity to reschedule the remaining tasks each time a task processing is completed and a machine becomes idle. Robust optimization is the natural methodology to cope with the first characteristic of duration uncertainty, yet the existing literature on robust scheduling does not explicitly consider the second characteristic the possibility to adjust decisions as more information about the tasks duration becomes available, despite that re-optimizing the schedule every time new information emerges is standard practice. In this paper, we develop an adaptive robust optimization scheduling approach that takes into account, at the beginning of the planning horizon, the possibility that scheduling decisions can be adjusted. We demonstrate that the suggested approach can lead to better here-and-now decisions and better makespan guarantees. To that end, we develop the first mixed integer linear programming model for adaptive robust scheduling, and a two-stage approximation heuristic, where we minimize the worst-case makespan. Using this model, we show via a numerical study that adaptive scheduling leads to solutions with better and more stable makespan realizations compared to static approaches.

1. Introduction

Parallel Machine Scheduling (PMS) problems are widely researched owing to their theoretical importance and multiple applications in manufacturing, cloud computing, and project management, among others. Real-life PMS settings involve uncertainty about task duration, which may be characterized by the randomness of each task duration and, possibly, a dependence between task durations.

An ideal scheduling approach should accommodate uncertainty to ensure realistic guarantees on the objective function value and permit adjustments of later-stage scheduling decisions based on different observed task lengths (*e.g.*, different duration realizations of the task scheduled first may result in different allocation decisions of the next tasks).

In the existing literature, commonly, scheduling problems are analyzed using various types of *static schedules*. Static Allocation (SA) is one of the most frequently used static policies, where the tasks are allocated in a certain order to pre-specified machines (e.g., see the exact formulations in Xu, Cui, Lin, & Qian, 2013, for PMS). In such policies, both the decision about the order of tasks on each machine and their allocation to machines are static, and do not change as more information is revealed. Another example are Static List (SL) policies, where the scheduling order of tasks to machines is pre-specified and tasks are allocated, by this order, to the first idle machine (e.g., LPT for PMS makespan minimization, see Gupta & Ruiz-Torres, 2001). These policies can be viewed as having a static order of task allocations but as dynamic and adaptive in the choice of the machine to which each task is allocated. For both types of policies, research has addressed the issue of finding the optimal policy that minimizes the expectation/worst-case value of the objective function over uncertain task durations. For both policy types, however, it holds that the optimal SA and SL policies may change once the duration of some completed tasks is known. We are not aware of works that consider this issue explicitly.

We focus on a makespan minimization objective function that is used for load balancing in PMS and many other scheduling applications. When deciding whether to use the expected value or worst-case value, several factors should be considered. Optimizing

* Corresponding author.
E-mail addresses: izack.cohen@biu.ac.il (I. Cohen), k.s.postek@tudelft.nl (K. Postek), shimrits@technion.ac.il (S. Shtern).

https://doi.org/10.1016/j.ejor.2022.07.018
0377-2217/

over an expectation requires specifying the full probability distribution of task duration, information that is often not readily available or is costly to acquire. Moreover, the makespan of a single realization may significantly differ from the expected value; thus, if the exact scheduling problem is not repeated multiple times, optimizing over the expected value may not be translated into good performance in practice. In contrast, much less information is needed when specifying a set that includes all the reasonable duration realizations, and a worst-case optimization approach provides a guarantee on the performance of any realization in such a set. Therefore, we consider a scheduler who minimizes the worst-possible makespan of a set of tasks over some uncertainty set, which captures all reasonable scenarios within the support of the distribution. This is in line with the paradigm of Robust Optimization (RO), where the best solution is sought under the assumption that the problem's parameters are initially unknown and that, given the decisions, nature picks their worst-possible values from an *uncertainty set* consisting of outcomes that include the true realization with a high probability. A representative real-world PMS example is presented by Xu et al. (2013) who describe a new product development division that needs to manufacture prototypes for multiple newly developed mobile phones. These prototypes include sets of new parts that are being manufactured for the first time. In the absence of historical data regarding processing duration, stochastic models are irrelevant and RO becomes a leading alternative for hedging against schedule delays.

In this mindset, we consider the classical version of PMS, where m identical machines process $n \geqslant m$ tasks that are available at the start of the scheduling horizon. We construct an exact Mixed Integer Linear Optimization (MILO) formulation for minimizing the worst-case makespan, which includes all possible later-stage (re-)scheduling decisions and gives the best-possible Adaptive Robust (AR) policy. Since this formulation scales exponentially in the problem size, we also propose an adaptive heuristic – the Two-stage Static Allocation (2SSA) – where only one re-optimization moment is considered. Next, we compare the optimal scheduling decisions of the adaptive formulation and the corresponding worst-case makespan to those of the optimal SA and SL.

In contrast to the majority of previous works, which compare naive implementations of the SA and SL policies without re-optimization (*i.e.*, re-scheduling) as more information is revealed, we consider the more realistic rolling-horizon implementation of these policies. Under this implementation, whenever one of the machines becomes idle, the scheduler can alter the initial order of tasks by re-solving an optimization problem with the extra information included.

In this paper, we view adaptivity in two distinct ways: (a) the ability of here-and-now decisions to account for future changes to the schedule due to the revealed information, and (b) the flexibility to change future scheduling decisions (in terms of what task is scheduled on which machine) in light of the revealed information. While a rolling-horizon implementation of any policy may provide adaptivity in terms of (b) with regard to future decisions, it does not provide more information regarding to the here-and-now decision. Thus, while AR is fully adaptive, our suggested 2SSA aims to provide partial adaptivity in terms of (a), similarly to SL. Moreover, we show that 2SSA is practically more computationally tractable than SL for the problem sizes we consider.

The main contributions of our research are as follows:

1. We characterize settings in which using adaptive policies may be important. Specifically, we identify settings in which the static and adaptive policies result in the same makespan, and provide performance bounds for the SA policy as well as any rolling-horizon policy with respect to a Perfect Hindsight (PH) policy, which determines the best allocation when the durations are known exactly. The bounds are computed for the popular *budgeted uncertainty set* (Bertsimas & Sim, 2004). Using these bounds, we determine that when the budget is moderate relative to the number of tasks n, and when n is not extremely large, planning for adaptivity may be crucial for good promised and actual performance.
2. We provide a closed-form MILO formulation of the PMS problem with re-scheduling and its heuristic variant. To the best of our knowledge, this is the first such formulation. The heuristic variant, named 2SSA, accounts for one stage of adaptivity. We demonstrate, via experiments, that the way 2SSA partially accounts for adaptivity, balances well between computational efficiency and performance.
3. We demonstrate, through stylised examples and numerical experiments, that both the adaptive robust policy and its heuristic variant can significantly outperform their static alternatives, even when the latter are implemented via a rolling-horizon approach.
4. In terms of managerial insights, the main conclusion of our paper is that, when possible, future re-scheduling of tasks (adaptations) should be taken into account at the planning stage as a way to obtain substantially better makespan guarantees as well as shorter actual makespans compared to non-adaptive approaches.

The remainder of the paper is structured as follows. In Section 2, we review the relevant scheduling and robust optimization literature. Section 3 introduces the notation and definitions used in our formulations and analyses. In Section 4, we discuss structural properties of static and adaptive policies. In Section 5, we introduce the Dynamic Programming (DP) formulations of the adaptive robust scheduling from the scheduler's and adversary's points of view. In Section 6 we develop, via the adversary view, the MILO formulation of the problem, as well as its 2SSA variant. Section 7 demonstrates, through a numerical study, the benefit of using adaptive policies. Section 8 presents managerial insights gained from our investigation of adaptive policies and Section 9 concludes and suggests future research directions.

2. Literature review

We focus on adaptive robust makespan minimization for the classical PMS problem with identical machines. The deterministic version of the problem, which is one of the most studied PMS problems (Ranjbar, Davari, & Leus, 2012), is NP-Hard (Pinedo, 2002). We review the two main approaches for dealing with uncertain durations in the context of PMS: the stochastic optimization approach and the robust optimization approach. The former approach is attractive if probability distributions of task durations are known and the scheduler desires a policy that performs well on average (see, Section 2.1). If, in contrast, the scheduler wants assurance that the policy will perform well for any realization of the durations within a predefined set, or does not have an accurate estimate of the underlying distribution, then a robust (min-max) approach that minimizes the worst-case performance is the best option (see, Sections 2.2–2.3).

Although this paper does not address the stochastic setting, due to the connection between stochastic and robust settings, our literature review will first cover the more studied stochastic models, before transitioning to discuss the RO setting and its adaptive variant.

2.1. Stochastic PMS models

This type of scheduling model optimizes an expected value objective, such as the expected makespan to process n tasks on m

machines. The probability distributions of task durations are assumed to be known or can be inferred based on historical data.

There are only a few known optimal policies for specific probability distributions. For example, the longest expected processing time (LEPT) priority rule – by which tasks are processed according to a non-increasing order of their expected duration – minimizes the expected makespan for exponentially distributed and independent task durations (Cai, Wu, & Zhou, 2014).

Many studies looked at the stochastic PMS under various conditions and assumptions. A partial list of representative publications includes: Möhring, Schulz, & Uetz (1999) who developed non-anticipative scheduling policies via linear programming relaxations to minimize the expected weighted flow time (that is, the sum of expected task completion times). They analyzed the performance of the weighted shortest expected processing time priority rule, which is simple to apply, and found that it is asymptotically optimal; Ranjbar et al. (2012) developed efficient branch-and-bound procedures to maximize the probability that a set of tasks with normally-distributed processing times completes before its due date. Their experiments included up to 20 tasks and five identical machines; Weber (1982) allowed preemptions and developed priority rules based on highest/lowest hazard rates; others solved PMS problems using heuristics such as genetic algorithms and simulated annealing (*e.g.*, Balin, 2011; Yeh, Lai, Lee, & Chuang, 2014).

We mention, for completeness, a scheduling approach which is known as proactive-reactive or predictive-reactive scheduling. Two key characteristics separate this approach from the current paper: 1) Typically, proactive-reactive methods rely on probabilistic knowledge of the uncertain variables to develop efficient schedules, and 2) these methods assume independence between the random variables (e.g., task durations) contrary to the suggested approach that accommodates dependence between the random variables. The main idea of reactive-proactive approaches is to construct a proactive baseline schedule that takes into account possible disruptions (e.g., due to variations of task durations). When schedule disruptions are realized they may be corrected using preselected rules or by resolving the problem. The approach has been applied for different settings such as the weighted parallel machine re-scheduling problem (Tighazoui, Sauvey, & Sauer, 2021), job-shop scheduling (Beck & Wilson, 2007) and resource-constrained project scheduling (Davari & Demeulemeester, 2019). In the latter research, Davari & Demeulemeester solve the proactive and reactive problem jointly by optimizing over numerous realized scenarios while taking into account the baseline project execution schedule and the cost of reactions.

Commonly, when historical data are not accessible or costly to acquire or when the tasks are unique, probability distributions of task durations are not available and schedulers may have to rely on estimations of upper and lower bounds of task durations (Balouka & Cohen, 2019). For non-repetitive tasks, decision makers tend to exhibit a risk-averse behavior that hedges against worst-case scenarios (Daniels & Kouvelis, 1995; Lin & Ng, 2011). In such cases, the use of RO, which we review next, is natural.

2.2. Static robust PMS

To the best of our knowledge, all previous research applying RO to the PMS problem involved static policies that do not consider, when making scheduling decisions, the possibility that decisions could or should be changed later on. We suggest the option to adjust task/machine allocations as new information is uncovered. Below are several studies that use static robust solution approaches.

The closest work to ours is by Xu et al. (2013) who investigated the robust PMS with identical machines and a makespan minimization objective under processing times specified via an interval-type uncertainty. The authors formulated the problem as a MILO, and solved it via exact solution approaches based on iterative relaxation algorithms and several heuristics. A computational experiment with up to five machines and 15 tasks demonstrated that the heuristics' average deviation from the optimal value is smaller than 8%. The current research departs from Xu et al. (2013) by developing an adaptive RO model, which may use convex uncertainty sets including (but not limited to) the conservative interval-type uncertainty set, showing the equivalence between the static and adaptive solution in this case. Bougeret, Jansen, Poss, & Rohwedder (2021); Bougeret, Pessoa, & Poss (2019) focused on developing approximation algorithms for robust PMS with identical machines and a budgeted uncertainty set.

Other studies adopted a min-max regret objective function that minimizes the maximal deviation of a given solution from the optimum across all scenarios, since such objective is considered as less conservative than the traditional robust objective (Aissi, Bazgan, & Vanderpooten, 2009). Conde (2014) formulated a MILO problem to minimize the maximal regret of the flow time for a PMS environment with unrelated machines in which the processing durations are specified via an interval-type uncertainty set. They solved the MILO for problems with up to 40 tasks and 10 machines using a bound on the computation time. Importantly, even a simpler version of this problem with identical machines is NP-Hard (de Ruiter, Brekelmans, & den Hertog, 2016). Xu, Lin, & Cui (2014) who investigated PMS with unrelated machines showed that a solution with the nominal (midpoint) task processing durations is a two-approximation for the min-max regret problem. They suggested an interesting modeling idea by which the original problem is transformed into a robust single machine problem, which yielded an MILO problem with fewer variables and constraints. The authors stressed the importance of researching PMS problems with other objective functions – an idea we adopt by using the makespan minimization objective.

Other papers addressed robust PMS problems with a variety of objectives and constraints such as cost minimization where task processing can be outsourced (Wang & Cui, 2020) and maximization of the probability that all tasks complete by their due dates using a distributionally robust approach (Liu, Liu, Chu, Zheng, & Chu, 2019).

2.3. Adaptive robust optimization

Most RO research in scheduling implements a 'static' approach, as detailed in the previous section. Yet, in problems spanning multiple time periods, most schedulers would adjust their decisions as the actual duration of each task emerges. A common approach for emulating this is the rolling-horizon approach, where the problem is re-solved at selected decision points, taking into account the new information. The downside of this approach is that if a static RO approach is used to solve each of the respective optimization problems, the here-and-now decisions are still optimized as if the later-stage decisions were not to going to change regardless of the uncertainty realization.

A branch of RO research dealing with this issue is Adaptive Robust Optimization (ARO) (Ben-Tal, Goryashko, Guslitzer, & Nemirovski, 2004). Its main idea is to optimize the here-and-now decisions, taking into account all scenarios in which the problem may unfold and the corresponding optimal decisions for each such scenario. Optimal ARO solutions are favorable with respect to static policies since they result in better here-and-now decisions that take into account adaptations in later time stages. At the same time, solving problems via ARO is computationally demanding because of the need to account for the contingent decisions in a large number of scenarios. A particularly difficult case is problems in which later-stage decisions are discrete.

Since solving ARO exactly is often computationally demanding, one may restrict the space of considered policies to obtain a tractable approximation. The most common restriction is using affine decision rules, as introduced by Ben-Tal et al. (2004). There, later-stage decisions are affine functions of the unknown parameters, and the coefficients of these functions are optimized as decision variables. For some problems, affine decision rules are shown to be optimal (Bertsimas, Iancu, & Parrilo, 2010), but this is not true in general. Modeling of continuous variables as affine decision rules may yield computationally efficient solutions as Cohen, Golany, & Shtub (2007) demonstrate for the time-cost tradeoff project scheduling problem.

The four main approaches to approximating ARO in the case of integer decisions, which we deal with in scheduling, are: (i) the K-adaptability approach of Hanasusanto, Kuhn, & Wiesemann (2015), (ii) the iterative partitioning approach of Postek & Hertog (2016) and Bertsimas & Dunning (2016), (iii) cutting-plane-like approaches as used by Zeng & Zhao (2013) for two-stage robust optimization problems with a continuous second-stage decisions, and (iv) decision rule approximations (see, Georghiou, Kuhn, & Wiesemann, 2019, and references therein). All these approaches assume that one knows the time moments at which the values of initially unknown parameters are revealed. This makes them applicable to problems such as unit commitment (Bertsimas, Litvinov, Sun, Zhao, & Zheng, 2012), inventory control (Ben-Tal et al., 2004), or flood protection planning (Postek, Den Hertog, Kind, & Pustjens, 2019). Yanıkoğlu, Gorissen, & den Hertog (2019) gives a broad survey of the adaptive robust optimization methods. In the PMS problem that we focus on, the above mentioned assumption is no longer valid. The time moments at which new information (completion of tasks) becomes available depend on (i) the uncertain durations of the currently running tasks and (ii) previous scheduling decisions. Moreover, all the decisions are discrete schedule-or-not binaries. For this reason, the PMS problem structure is uncharted territory for ARO, and exactly where our research makes a notable contribution.

3. Notation and definitions

We consider minimizing the processing makespan of tasks $i = 1, 2, \ldots, n$ on m identical machines. Tasks and machines are available at time $t = 0$, there are no precedence relations between tasks, and tasks cannot be preempted while processing. The task durations vector $d = (d_1, \ldots, d_n)$ is uncertain and lies within an *uncertainty set* U which can be either finite and discrete or convex, e.g., a polytope. The scheduler aims to minimize the worst-case makespan over this predefined uncertainty set, *i.e.*, assuming that for all scheduling decisions possible, the adversary (nature) will pick the worst-possible duration vector $d \in U$.

To model the problem, we need to describe the possible states of the system. Using the notation summarized in Table 1, in what follows, we develop our modeling framework. To explain the notation for the system state consider a system with $m = 2$ machines, each of which is either processing a task or idle. When a machine completes a task and becomes idle, an unscheduled task, if one exists, has to be immediately allocated to the machine. The system state at such a time moment t is described by $(S, F, D, i, \bar{D}_i)$, where S is an ordered list of started tasks (in the order of starting with arbitrary tie-breaks), F is an ordered list of completed tasks (in the order of finishing with arbitrary tie-breaks), D is an ordered list of realized durations corresponding to the completed tasks in F, i is a task that is currently being processed and its duration d_i is known to be $d_i \geqslant \bar{D}_i$ time units, where $\bar{D}_i$ is its processing time at the instance the state is observed. There are also, however, states in which both machines are busy, and when this occurs, no decision is made. Thus, to keep the state space limited, we only include states in which at least one machine is idle. Fig. 1 and Table 2 demonstrate the states of a system with two machines for a certain realization and schedule.

As the scheduling progresses in time, the up-to-now (total) durations of the running (completed) tasks become known. This new information might reduce the uncertainty about the possible duration of the remaining tasks, eliminating certain parts of the uncertainty set. Therefore, we introduce the notion of state-dependent uncertainty. This notion is formally expressed in the definition of the uncertainty set induced by a system state as

$$U_{S,F,D,i,\bar{D}_i} = \left\{ d \in U : d_k = D_k,\ \forall k \in F,\ d_i \geq \bar{D}_i \right\}.$$

We call a state $(S, F, D, i, \bar{D}_i)$ feasible if $U_{S,F,D,i,\bar{D}_i} \neq \emptyset$. We define the set of feasible states as $\mathcal{S}$.

Note that the notation discussed so far for the case of $m = 2$ can be extended to $m > 2$ by replacing i and $\bar{D}_i$ with I and $\bar{D}$ – the sets of in-process tasks and their processing times until t, respectively.

We now define the notion of a scheduling *policy*. We define a policy P as a mapping from a state to the choice of the next task to be scheduled. That is, $P : \mathcal{S} \to [n]$, where $P(S, F, D, I, \bar{D}) \in [n] \setminus S$. In this work, we compare three types of policies for minimizing the worst-case makespan.

- SA policies, where each task is assigned to a machine according to a predefined order. Thus, given an uncertainty set U, an SA policy amounts to a partition of the tasks to machines $J = (J_1, \ldots, J_m)$ such that $\cup_{j\in[m]} J_j = [n]$ and $J_j \cap J_k = \emptyset$ for all $j \neq k$. Once the decision about the partition J is made, it does not change. Assuming that the tasks in each J_j are given in a certain order (without loss of generality, we assume it is lexicographic), the policy can be explicitly defined as

  $$P^{\mathrm{SA},J}(S, F, D, I, \bar{D}) = \arg\min\{k \in [n] \setminus S : \exists j \in [m],\ k \in J_j, i \notin J_j\ \forall i \in I\}.$$

  Thus, at each decision state, the next task to be scheduled is the first task that has not yet started and has been allocated to the machine that just became idle. Note that this policy is independent of the information gained at each stage about the durations of the tasks that have already begun processing, meaning D and $\bar{D}$, and thus, it does not adapt to this information.
- SL policies, where a task is processed on the first idle machine according to its location in an ordered list. Thus, given an uncertainty set U, an SL policy amounts to a permutation of the tasks: $\pi = (\pi_1, \pi_2 \ldots, \pi_n)$, where for each i_{th} place in the list, $\pi_i \in [n]$ is a specific task, and all tasks must be allocated once, *i.e.*, $\pi_i \neq \pi_j$ for any $i \neq j \in [n]$. Once the decision on the permutation is made, it does not change. Given permutation π, the policy can be explicitly defined as

  $$P^{\mathrm{SL},\pi}(S, F, D, I, \bar{D}) = \pi_i, \quad i = \operatorname{argmin}\{k \in [n] :\ \pi_k \notin S\}.$$

  Thus, at each decision state, the next in order on the list of not-started tasks is scheduled on the idle machine. Note that similarly to the SA policy, the SL policy also does not adapt to the state-dependent information.
- The AR policy, denoted by P^{AR} is the most flexible and considers all possible scenarios for task realizations, choosing the next task according to the actual scenario that was realized. In particular, the scheduling decision for two distinct states with the same completed and processing tasks may be different. Specifically, $(D, \bar{D}) \neq (D^\dagger, \bar{D}^\dagger)$ allows for $P^{\mathrm{AR}}(S, F, D, I, \bar{D}) \neq P^{\mathrm{AR}}(S, F, D^\dagger, I, \bar{D}^\dagger)$.

All three policy types can be applied in a rolling-horizon fashion by solving a new optimization problem once a machine becomes idle, where the uncertainty set is the one induced by the

Table 1
Primary notation.

Indices, parameters and variables	Description
i, j, k	Denote a task or a machine, by context
m	Number of identical machines
n	Number of tasks
t, τ	Denote time from start of processing
d_i, d	Duration of task i and the vector of task durations, respectively
$D_i, \bar{D}_i$	The realized duration of a completed task $i \in F$, and the amount of time task $i \in I$ has been processing at the observation time t, respectively
Sets and lists	
$\mathbb{R}^n_{\geqslant 0}$	An n-dimensional set of non-negative, real numbers
$[n]$	A shorthand for the set $\{1, \ldots, n\}$
S	An ordered list of started tasks
F, D	An ordered list of completed tasks and their realized durations, respectively
$I, \bar{D}$	An ordered list of the in-process tasks at time t and their respective durations till t
U	Uncertainty set for task durations

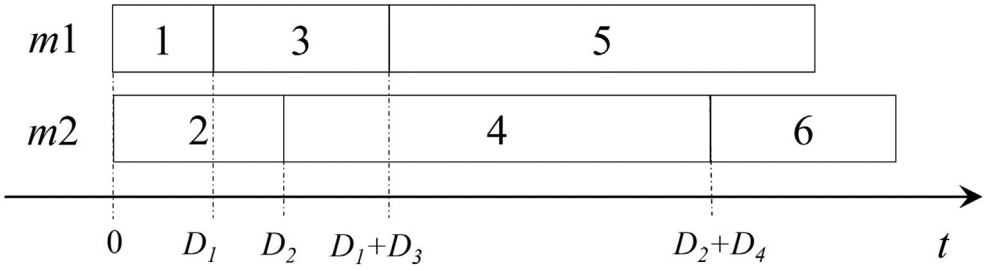

Fig. 1. A schematic representation of an example timeline for a system with two machines.

Table 2
System states for the timeline presented in Figure (1).

Time t	State notation	Decision
0	$(\emptyset, \emptyset, \emptyset, 0, 0)$	Start 1 and 2
D_1	$([1,2],[1],D_1,2,D_1)$	Start 3
D_2	$([1,2,3],[1,2],[D_1,D_2],3,D_2-D_1)$	Start 4
D_1+D_3	$([1,2,3,4],[1,2,3],[D_1,D_2,D_3],4,D_1+D_3-D_2)$	Start 5
D_2+D_4	$([1,2,3,4,5],[1,2,3,4],[D_1,D_2,D_3,D_4],5,D_2+D_4-(D_1+D_3))$	Start 6

revealed state of the system. Nevertheless, only the AR policy takes such implementation into account at $t = 0$. Hence, we expect the optimal AR policy to yield favorable worst-case makespan guarantees due to its improved here-and-now scheduling decisions compared to the optimal SL and SA policies that do not consider adaptations in later-stage decisions.

Finally, to formulate our problem of minimizing the worst-case makespan, we introduce the function $T(S, F, D, i, \bar{D}_i)$ denoting the minimal worst-case duration to process the unfinished tasks (*i.e.*, i and the tasks that have not started), assuming an optimal AR policy is applied. We refer to this function as the *remaining makespan*. Our objective of minimizing the worst-case makespan is equivalent to the function $T(\emptyset, \emptyset, \emptyset, 0, 0)$.

In Section 5, we use the above notation in developing DP formulations for finding the optimal AR policy.

4. Comparing the static and adaptive robust policies

In this section, we characterize and highlight potential differences between the three scheduling policies defined in the previous section. In Section 4.1, we demonstrate that if the uncertain task durations can vary independently of each other, then the best worst-case makespan is identical for all policy types. In Section 4.2, we use toy examples with task duration dependence for showing that an optimal AR policy may yield different first-stage decisions from those of SA and SL. Based on this result, in the remainder of the paper, we focus on situations where task durations are connected to each other via the shape of the uncertainty set. Accordingly, in Section 4.3 we provide performance bounds for the case of two machines under the commonly used budgeted uncertainty set, for which the possible tasks' durations are connected through the budget. Specifically, we bound the performance of the optimal SA policy and of a rolling-horizon implementation of any scheduling policy. These bounds quantify the possible gains from utilizing an adaptive policy as a function of the problem's parameters such as the number of tasks, their durations and the uncertainty set's parameters.

4.1. Equivalence of the scheduling policies under product uncertainty sets

In Section 3, we defined a scheduling policy P as a mapping from the system's states into scheduling decisions. The schedule is created by deciding which tasks to schedule and observing which task completed first and at what time. Thus, for a given policy P and a vector of durations d, the *schedule* can be represented as an ordered partition of the tasks to machines $J = (J_1, \ldots, J_m)$ where its makespan is given by $\max_{j\in[m]} \sum_{i\in J_j} d_i$. Let $\mathcal{M}(P, d)$ denote the function that maps a policy and a set of durations into such a partition.

The optimal SA policy is equivalent to the optimal partitioning of tasks to machines $J^* = (J^*_1, \ldots, J^*_m)$. Thus, for a given duration vector d, we have $\mathcal{M}(P^{\mathrm{SA},J^*}, d) \equiv J^*$, which is independent of d.

Therefore, denoting all sets of partitions of n tasks to m machines as $\mathcal{J}_{n,m}$, an optimal SA policy is obtained by solving the following optimization problem:

$$\min_{J\in\mathcal{J}_{n,m}} \max_{d\in\mathcal{U}} \max_{j\in[m]} \sum_{i\in J_j} d_i.$$

In contrast, the optimal SL policy is equivalent to an optimal permutation π^* of the tasks. Consequently, the allocation of tasks in π^* to machines and the respective makespan depend on the task duration vector d. For illustration, consider two machines, three tasks and the permutation (1,2,3). The schedule will start with 1 and 2 on the two machines; then, if 1 completes first, 3 will be scheduled on its machine and otherwise on the other machine. The worst-case makespan for a *given* SL policy associated with permutation π is, therefore:

$$\max_{d\in\mathcal{U}, J=\mathcal{M}(P^{\mathrm{SL},\pi},d)} \max_{j\in[m]} \sum_{i\in J_j} d_i. \tag{1}$$

We denote Π_n as the set of all permutations of $[n]$. Thus, the optimal SL policy is obtained by minimizing (1) over Π_n:

$$\min_{\pi\in\Pi_n} \max_{d\in\mathcal{U}, J=\mathcal{M}(P^{\mathrm{SL},\pi},d)} \max_{j\in[m]} \sum_{i\in J_j} d_i.$$

Finally, denoting the space of all possible policies as $\mathcal{P}$, the optimal AR policy is given by the solution of the following optimization problem.

$$\min_{P\in\mathcal{P}} \max_{d\in\mathcal{U}, J=\mathcal{M}(P,d)} \max_{j\in[m]} \sum_{i\in J_j} d_i.$$

Product uncertainty sets are a family of uncertainty sets where the duration of each task varies independently of the other tasks, *i.e.*, the uncertainty set is a Cartesian product of individual tasks' duration uncertainty set. A natural candidate for modeling task durations in scheduling settings, is the box uncertainty set, which is an example of a product uncertainty set where each task duration is contained in an interval. The wide use of this uncertainty set underlines the importance of our next result – that the optimal makespan achieved by all three robust policies is equal when using a product uncertainty set. In other words, the here-and-now decisions produced by all three policies are optimal (for brevity, we placed all the proofs in Appendix A).

Proposition 1. *Let $\mathcal{U}$ be a product uncertainty set given by $\mathcal{U} = \mathcal{U}_1 \times \mathcal{U}_2 \times \ldots \times \mathcal{U}_n$, where $\mathcal{U}_i \subseteq \mathbb{R}$ is a closed set for all $i \in [m]$. Then, the optimal SA policy, the optimal SL policy, and the optimal AR policy produce the same worst-case makespan.*

4.2. Different first-stage decisions in RO and AR

Following the results of the previous section, in which we show that for product uncertainty sets the optimal SA, SL and AR policies are equivalent, we focus on uncertainty sets that are not product uncertainty set, and specifically not box shaped. That is, we consider cases where the tasks' durations are connected via the shape of the uncertainty sets. Indeed, we experiment with a variety of restricted uncertainty sets such as discrete, budgeted and ellipsoidal uncertainty sets.

This section demonstrates the superiority of the adaptive AR policy with respect to the alternative SA and SL policies, even if they are implemented in a rolling-horizon fashion; that is, only the here-and-now decisions are implemented and the policies are re-optimized every time a task finishes. Rolling-horizon implementation of scheduling policies is a common practice for adapting to new information across multiple domains such as supply chain management and scheduling, just to name two examples. Specifically, we show via examples that the optimal first-stage decisions of an optimal AR policy may be different than those of optimal SA and SL policies, which implies better performance by the former.

We begin with an example with three tasks in which the optimal here-and-now decisions are the same for the SL and AR policies (thus these two policies are equivalent), but are different for the SA policy. Then, we present an example with four tasks, where the AR policy has different here-and-now optimal decisions from the SL policy. The takeaway is that an AR policy may achieve better objective function values compared to SA and SL policies (by more than 6% in our toy examples).

4.2.1. A three-task example.

For the case of three tasks, we can obtain closed-form short expressions for the worst-case makespan for each policy. Consider first the SA policy in which, without loss of generality, tasks 1 and 2 are processed first, and task 3 is processed after task 1. Thus, $J_1 = (1,3), J_2 = (2)$. The worst-case duration is then

$$\max_{d\in U} \max\{d_1 + d_3, d_2\},$$

which is the maximum of these two terms:

$$\max_{d\in U} d_1 + d_3, \quad \max_{d\in U} d_2.$$

For AR, consider, without loss of generality, a schedule in which tasks 1 and 2 are processed first, and task 3 starts processing as soon as the first of the two tasks is finished. The worst-case duration under AR is

$$\max_{d\in U} \max\{\min\{d_1, d_2\} + d_3, \max\{d_1, d_2\}\},$$

which is the maximum of these three terms:

$$\max_{d\in U} \min\{d_1 + d_3, d_2 + d_3\}, \quad \max_{d\in U} d_1, \quad \max_{d\in U} d_2.$$

To choose the optimal allocation for each policy, we can evaluate the above expressions for every permutation of the three tasks. With respect to SL, we note that for any setting in which $n = m + 1$ ($n = 3$ and $m = 2$ for our example), the optimal SL policy and the optimal AR policy are equivalent.

To make things concrete, we consider a specific setting in which task durations are:

$$d_1 = 0.0580 + 0.95z_1, \ d_2 = 0.1945 + 0.75z_2,$$
$$d_3 = 0.5866 + 0.48z_3,$$

where the uncertain parameters are $(z_1, z_2, z_3) \in Z = \{[0,1]^3, \sum_{i=1}^3 z_i \le 2.5\}$ (*i.e.*, a budgeted uncertainty set). Let us compare the optimal AR (here equivalent to SL) with the static policy of SA. The optimal solutions, respectively, are:

- AR: Start with tasks 1 and 2, and start processing task 3 whenever the first of these tasks has completed. This gives a worst-case duration of 1.83. The worst-case duration is given by the choice $z_1 = 0.7420$, $z_2 = 0.7579$, and $z_3 = 1.0$, leading to $d_1 = 0.7629$, $d_2 = 0.7629$, and $d_3 = 1.0666$, i.e., tasks 1 and 2 finish simultaneously.
- SA: There are two equivalent solutions. (i) Start tasks 1 and 3, and process task 2 after 1 or, (ii) start tasks 2 and 3, and process task 1 after 2. Both options give a worst-case duration of 1.95. Any other option in which the first tasks are 2 and 3 leads to a longer duration. In both cases the worst-case durations are given by $z_1 = z_2 = 1$ and $z_3 = 0.5$.

Thus, even in such a simple setting, the adaptive policy leads to a makespan lower by about 6% compared to SA. Moreover, since the first-stage decisions are different (*i.e.*, 'allocate $\{1,2\}$' by AR compared to 'allocate $\{1,3\}$ or $\{2,3\}$' by SA), a rolling-horizon implementation of the static policy will still be inferior to AR. In fact, if we start with tasks $\{1,3\}$ and schedule task 2 after the first of

Table 3
Specification of the uncertainty set.

Task	Task durations for:				
	Sc. 1	Sc. 2	Sc. 3	Sc. 4	Sc. 5
1	3	4.5	4.75	2.5	0.25
2	2	2	2	3.5	5
3	3	3.5	3	3	3.5
4	5.5	4	4	4	4

them is completed, we will obtain a worst-case makespan of 1.95 (identical to the one in which we did not adapt the decision). If we start with $\{2, 3\}$ and schedule task 1 after the first of them is completed, we obtain a worst-case makespan of 1.88, which is still almost 3% more than that of AR. Indeed, our numerical study presented in Section 7 indicates that AR may be better than SA by up to 30%.

Finally, this example also demonstrates that, contrary to SA, the worst-case durations for AR are not necessarily extreme points of set U. This implies that there is a distinction between convex and continuous uncertainty sets and discrete ones. Thus, even when U is polyhedral, one cannot simply replace the set U with the set of its extreme points. Indeed, when solving this problem for the uncertainty set $\text{ext}(U)$, we would obtain the same optimal initial decision $\{1, 2\}$, albeit with a shorter worst-case makespan 1.77 related to the extreme point $z = (1, 1, 0.5)$. Moreover, in some cases uncertainty set U and $\text{ext}(U)$ may also result in different optimal decisions. Therefore, in this paper we address both discrete and convex uncertainty sets.

4.2.2. A four-task example.

We now illustrate the difference between the three policies, SA, SL and AR, for a setting with $n = 4$ and $m = 2$. We use an uncertainty set that includes the five scenarios outlined in Table 3.

Without providing the (tedious) calculations, we summarize the optimal choices of each of the three policies.

- AR: The optimal AR policy schedules tasks 1 and 4 first. Then, for scenarios 1,2, and 3, task 3 is scheduled to start when the first of 1 and 4 completes, and task 2 is scheduled last. For scenarios 4 and 5, task 2 is scheduled to start immediately when the first of 1 and 4 completes and task 3 is last. The corresponding optimal worst-case makespan value is 7.5.
- SL: There are four optimal list policies that achieve the minimal makespan equal to 8, longer by 6.7% compared to the optimal AR. The policies are: $(1, 2, 4, 3), (1, 4, 2, 3), (2, 3, 4, 1), (2, 4, 1, 3)$. Take for example, (2,3,4,1). It achieves makespans of 7.5,8,7.75,7,7.5 for scenarios 1 to 5, respectively. Thus, the optimal worst-case makespan is 8.
- SA: The static allocation policy partitions the tasks to machines upfront. The optimal partition is $J_1 = (1, 2), J_2 = (3, 4)$. It achieves makespans of 8.5,7.5,7,7,7.5 for scenarios 1 to 5, respectively. Thus, the robust makespan is 8.5. All other partitions lead to longer makespans. The static allocation policy leads to a makespan that is longer by 13.3% compared to AR.

This example underscores the importance of AR in scheduling for gaining better promised makespan guarantees and better actual makespan. Regarding the former aspect, the three policies AR, SL, SA give different 'promised worst-case' makespans, which is important from a managerial standpoint of providing guarantees (*e.g.*, when submitting a contract proposal). Regarding the latter aspect, the AR policy will perform better than both SL and SA even if the latter two were re-optimized every time a task finishes (rolling-horizon). The difference lies in the possibility of making a sub-optimal here-and-now decision in the case of SL and SA. Our numerical study, in Section 7, clearly shows that the performance of a policy deteriorates as the number of its different first-stage decisions with respect to AR increases.

4.3. Performance bounds under the budgeted uncertainty set

In this section, we identify the cases in which taking into account decision adaptivity in the planning stage may lead to a significant makespan improvement compared to static policies and rolling-horizon implementations of policies. We focus on the two-machine case, and establish upper performance bounds for the SA and for policies which do not allow leaving a machine idle. These latter family of policies include SL and a Rolling Horizon (RH) implementation of SA, and are denoted by RH in the formulas below. We compare these policies to the perfect hindsight policy, which we denote as PH – the optimal policy when all the uncertain task durations are assumed to be known in advance. In other words, we wish to bound, from above, the ratios $\frac{\text{SA}}{\text{PH}}$ and $\frac{\text{RH}}{\text{PH}}$. The first ratio provides a bound on the maximum suboptimality of the 'promised' makespan and the second one bounds the maximum suboptimality of the actual rolling-horizon 'execution' of a policy. These two bounds will provide the potential benefit that can be obtained by using AR, and help us to identify regions in which accounting for later adaptivity would be the most useful.

To provide bounds, we need to choose an uncertainty setting. We choose the well known and widely used budgeted uncertainty set, first suggested by Bertsimas & Sim (2004). The structure of the uncertainty set is captured by the following assumption.

Assumption 1. The nominal length of any task i is bounded, *i.e.*, $d_i^0 \in [\underline{a}, \bar{a}]$ for some $0 < \underline{a} \le \bar{a} < \infty$. Moreover, there exists $\alpha > 0$ such that $\bar{d}_i = \alpha d_i^0$ for all $i \in [n]$, and

$$U = \left\{ d \in R^n : d_i = d_i^0 + \bar{d}_i u_i, u_i \in [0, 1], \sum_{i=1}^{n} u_i \le \Gamma \right\}.$$

This uncertainty set implies that in the considered realizations the nominal duration vector d^0 can be augmented by a maximal perturbation $\bar{d}$ proportional to d^0 (can be thought of as a percent of the nominal). However, not all tasks will achieve this maximal perturbation, since the sum of the ratio of the perturbations used for each task cannot exceed a given budget Γ.

4.3.1. Bound on the promised durations of SA versus PH.

We start by bounding the ratio between SA and PH. For the case of two machines, an allocation can be represented as a binary vector $x \in \{0, 1\}^n$ such that

$$x_i = \begin{cases} 1 & \text{if the } i\text{-th task is on machine 1,} \\ 0 & \text{otherwise.} \end{cases}$$

With this notation, the processing time on machine 1 is $x^\top d$ and the processing time on machine 2 is $(e - x)^\top d$ where e is a vector of ones. Thus, the total makespan is $\max\{x^\top d, (e - x)^\top d\}$. Mathematically, the makespans of SA and PH are

$$SA = \min_{x \in \{0,1\}^n} \sup_{d \in U} \max \left\{ x^\top d, (e - x)^\top d \right\},$$

and

$$PH = \sup_{d \in U} \min_{x \in \{0,1\}^n} \max \left\{ x^\top d, (e - x)^\top d \right\}.$$

To bound the $\frac{SA}{PH}$ ratio, we will upper bound SA and lower bound PH. We choose a common allocation to work with, corresponding

to the optimal allocation of PH for $d = d^0$. This allocation, which might be sub-optimal for SA, provides an upper bound on SA that can be found by computing the worst-case $d \in U$. To bound the value of this worst-case, we note that the uncertainty budget will be allocated first to tasks on a single machine before allocating it to the tasks on the other machine.

In order to find a lower bound on PH, we use a specific choice of $d \in U$, in which the budget is allocated equally between the tasks, such that each task is perturbed by Γ/n of its maximal perturbation. Finally, we use the fact that the optimal allocation x for this realization is equal to the nominal allocation.

Below, we summarize the bound obtained by using this approach.

Theorem 1. *Let the number of machines be $m = 2$ and let Assumption 1 hold. Let $\tilde{x}$ be the partition to machines which minimizes the makespan for the deterministic problem where $d = d^0$. Specifically, we denote by $\mathcal{I}$ and $\bar{\mathcal{I}} = [n] \setminus \mathcal{I}$ the set of tasks that $\tilde{x}$ allocates to the first and second machine, respectively. Furthermore, we denote the ordering of d_i^0 for tasks in $\mathcal{I}$ and $[n] \setminus \mathcal{I}$ by $d_{(1)}^{0,\mathcal{I}} \geq \ldots \geq d_{(|\mathcal{I}|)}^{0,\mathcal{I}}$ and $d_{(1)}^{0,\bar{\mathcal{I}}} \geq \ldots \geq d_{(n-|\mathcal{I}|)}^{0,\bar{\mathcal{I}}}$, respectively. Then, the ratio between the worst-case makespan resulting from the SA policy and the PH policy is bounded from above by*

$$\frac{\mathrm{SA}}{\mathrm{PH}} \leq \frac{n}{n+\Gamma\alpha}\left(1+\alpha \max_{\mathcal{S}\in\{\mathcal{I},\bar{\mathcal{I}}\}} \frac{\sum_{k=1}^{\min\{|\mathcal{S}|,\lfloor\Gamma\rfloor\}} d_{(k)}^{0,\mathcal{S}} + \Delta^{\mathcal{S}}(\Gamma) d_{(\min\{|\mathcal{S}|,\lfloor\Gamma\rfloor\}+1)}^{0,\mathcal{S}}}{\sum_{i\in\mathcal{S}} d_i^0}\right),$$

where for any $\mathcal{S} \subseteq [n]$ we have $\Delta^{\mathcal{S}}(\Gamma) = \min\{\Gamma, |\mathcal{S}|\} - \min\{\lfloor\Gamma\rfloor, |\mathcal{S}|\}$.

Indeed, the above theorem shows that for $\Gamma = 0$ and $\Gamma = n$ the ratio is bounded from above by 1, *i.e.*, the value of adaptivity decreases as the budget goes to 0 or n. Thus, for any value of d^0 there exists some $0 < \Gamma < n$ for which this ratio is maximized, and so is the potential value of adaptivity. We also note that as n increases and Γ stays proportional to n, the bound does not necessarily go to 1, indicating that employing the SA without re-optimizing may be extremely suboptimal even for large n.

4.3.2. Bound on the rolling-horizon implementation of any scheduling policy

We now turn to bound the ratio of any RH policy and PH. In this case, we lower bound the worst-case makespan of PH, by assuming that we can perfectly balance the two machines, thus arriving to the bound:

$$PH \geq \sup_{d\in U} \frac{\sum_{i=1}^n d_i}{2}.$$

In order to bound the RH implementation of any scheduling policy, we make the following observation. The difference between the individual makespan of the two machines must be less than or equal to the maximum duration of the last task to finish (note that in an extreme case, this can be an extremely long task, which occupies the entire makespan of one of the machines). The identity of this last task depends on both the policy and the adversary's decision. Regardless, we can bound the makespan of any RH implementation by bounding from above the time at which the last task starts by and adding to it the duration of the last task. For example, if the last task to start is i, the time at which it will start will not be longer than $\sum_{j=1,\ j\neq i}^n d_j/2$. Therefore, we can bound the RH makespan by

$$RH \leq \max_{1\leq i\leq n} \sup_{d\in U} \frac{\sum_{j=1,\ j\neq i}^n d_j}{2} + d_i.$$

Next, we introduce Theorem 2, which uses the above bounds to bound the desired ratio.

Theorem 2. *Let the number of machines be $m = 2$ and let Assumption 1 hold. Then the ratio between the worst-case makespan resulting from the RH policy and the PH policy is bounded from above by*

$$\frac{\mathrm{RH}}{\mathrm{PH}} \leq 1 + \frac{\bar{a}\min\{(1+\alpha),(1+\alpha\Gamma)\}}{\underline{a}(n+\alpha\Gamma)}.$$

The above theorem implies that for any fixed $0 < \underline{a} \leq \bar{a} < \infty, \alpha > 0$ and any $\Gamma > 0$, as $n \to \infty$ any RH implementation gets closer to that of PH. Thus, for very large values of n there will be almost no benefit to computing AR, since one can get good performance by using any arbitrary RH policy.

5. Dynamic programming formulations

In the previous section, we highlighted potential differences between the three types of policies SA, SL, and AR. We now begin to develop a general method to optimize the AR policy. A natural first step is to formulate a scheduler-perspective DP model of the problem – this is the focus of this section. Solving this DP will be computationally intractable, but its formulation lays the foundation of the adversary-perspective DP of Section 5.2 that we solve via an MILO formulation in Section 6. To keep the exposition clear, we consider the two-machine setting here, with the general m-machine cases considered in Appendices B.1 and B.2.

5.1. Scheduler's dynamic programming formulation

At decision points, which occur at (i) the beginning of the planning horizon and (ii) when a task completes, the scheduler allocates to an idle machine a task that minimizes the worst-case makespan from that time point and on (hereafter, remaining makespan). We denote by $T(\cdot)$ the function that outputs the worst-case remaining makespan at a state described by $S, F, D, i, \bar{D}_i$ in which task i is still being processed on a machine and the second machine is idle (*e.g.*, it just completed processing a task). The recursive formulation for T is then given by

$$\begin{aligned} &T(S,F,D,i,\bar{D}_i) \\ &= \min_{k\notin S} \max\Bigg\{ \max_{\substack{d_k: d\in U_{[S,k],F,D,i,\bar{D}_i},\\ d_k\leq d_i-\bar{D}_i}} d_k + T([S,k],[F,k],[D,d_k],i,\bar{D}_i+d_k), \\ &\qquad \max_{\substack{d_i: d\in U_{[S,k],F,D,i,\bar{D}_i},\\ d_k> d_i-\bar{D}_i}} d_i-\bar{D}_i + T([S,k],[F,i],[D,d_i],k,d_i-\bar{D}_i)\Bigg\}. \end{aligned}$$

The objective (outer min) is to allocate a task k that minimizes the worst-case remaining makespan. For each k, there are two possible scenarios from which an adversary chooses the worst of the first max. The first term within the parentheses relates to the possibility that under the worst-case scenario, k completes its processing before the completion of the processed task i. In such a case, the next decision point is due when k completes and i is still running. The second term considers the possibility that under the worst-case scenario, i completes before k, in which case the next decision point is when task i finishes processing.[1]

For the boundary case, given by $|S| = n$, $|F| < n$, when all tasks have already been scheduled, we have that

$$T(S,F,D,i,\bar{D}_i) = \max_{d_i: d\in U_{S,F,D,i,\bar{D}_i}} d_i - \bar{D}_i,$$

[1] In this and later formulations, we seemingly ignore the case where two or more tasks finish processing at exactly the same time. This case can be dealt with explicitly; however, it significantly complicates the presentation of the problems and is, therefore, omitted for clarity in all following formulations.

i.e., the remaining makespan will only be the time until the current task i is completed.

The takeaway from this section is that in order to solve the scheduler-perspective DP for a general uncertainty set U, we may have to optimize over an infinite dimension decision space, *i.e.*, all functions from an infinite number of states for continuous uncertainty sets to scheduling decisions. For that reason, determining the optimal policy is a difficult task, calling for an approach different from the usual DP solution techniques, which we introduce in the next section.

5.2. The adversary dynamic programming formulation

Our approach to determining the optimal schedule takes the perspective of an adversary who seeks the scenario with the worst-possible task durations, taking into account that the scheduler will make the best-possible scheduling decisions at each decision point. In other words, the adversary tries to make the makespan as long as possible given that the scheduler implements an optimal scheduling policy. We will see that for continuous uncertainty sets, while the adversary has an infinite number of possible decisions the dimension of its decision space is finite.

The adversary's decision points are the states in which a certain task has just been scheduled so that either (i) all machines are busy, or (ii) there are no tasks left to schedule. An example of such a state is: $([S,k],F,D,i,\bar{D}_i)$, where task $k \notin S \equiv F \cup \{i\}$ has just been scheduled. To present the recursion, we define the function $T'(\cdot)$ for all the adversary's decision points as the worst-case remaining makespan. Thus, the recursive formula for T' is given by

$$
\begin{aligned}
&T'([S,k],F,D,i,\bar{D}_i) = \\
&\max\Bigg\{ \max_{\substack{d_k : d \in U_{[S,k],F,D,i,\bar{D}_i}, \\ d_k \le d_i - \bar{D}_i}} d_k + \min_{l \notin S \cup \{k\}} T'([S,k,l],[F,k],[D,d_k],i,\bar{D}_i + d_k), \\
&\max_{\substack{d_i : d \in U_{[S,k],F,D,i,\bar{D}_i}, \\ d_k > d_i - \bar{D}_i}} d_i - \bar{D}_i + \min_{l \notin S \cup \{k\}} T'([S,k,l],[F,i],[D,d_i],k,d_i - \bar{D}_i) \Bigg\},
\end{aligned}
$$

as long as $|S| \le n-2$, *i.e.*, not all tasks have started. For the boundary case of $|S| = n-1$, with just one task left being processed and no more tasks to schedule, we have that the worst-case remaining makespan is the maximum of all possible cases in which the newly scheduled task k either completes before the currently processing task i, or after it:

$$
T'([S,k],F,D,i,\bar{D}_i) = \max\left\{ \max_{d_i : d \in U_{[S,k],F,D,i,\bar{D}_i}} d_i - \bar{D}_i, \max_{d_k : d \in U_{[S,k],F,D,i,\bar{D}_i}} d_k \right\}.
$$

An important feature of the adversary-perspective DP, which is missing from the scheduler-perspective problem, is that its decision space can be reduced to one with a finite dimension, and thus can be optimized over. From a mathematical optimization point of view, we can roughly say that the adversary DP problem is a dual representation of the scheduler-perspective DP problem, which, as it turns out, is easier to solve. In the following section, we present an MILO reformulation of this DP.

6. MILO formulation and the 2SSA heuristic

6.1. Mixed-integer problem formulation

In this section, we formulate the adversarial perspective DP of Section 5.2 as an MILO. It is important to note that the adversary optimizes over an entire *s*cenario tree rather than on a *single* vector of task durations d (*i.e.*, a branch). This is because the adversary is also non-anticipative, in the sense that at each point in time, it makes a decision on the duration of the next task to complete while taking into account all possible future decisions of the scheduler.

To construct the scenario tree, we utilize two elements of the state description: S and F – the ordered lists of tasks according to their starting and completion orders, respectively; we denote their combination by $\sigma = (S,F)$. We consider different states in which either (i) one of the tasks has just been completed and the scheduler needs to make a decision (in association with the inner minimum in the DP recursion presented in (12)), or (ii) a new task has just been scheduled and the adversary decides which of the running tasks finishes first and what its remaining duration would be (associated with the outer maximization and inner maximizations in the DP recursion presented in (12)). Before presenting the rigorous notation, let us discuss the main idea behind the scenario tree for a setting with four tasks.

6.1.1. Illustrative case: Two machines and four tasks.

Consider the following *states*:

- $\sigma = (S,F) = (\emptyset,\emptyset)$: Initial state, no tasks are scheduled yet – both machines are idle; a task needs to be scheduled immediately.
- $\sigma = ([1],\emptyset)$: Task 1 has been scheduled on one of the machines – one machine is idle; a task needs to be scheduled.
- $\sigma = ([1,2],\emptyset)$: Task 1 has been scheduled on one machine and task 2 on the other machine – no machine is idle; the scheduler needs to wait until either task 1 or task 2 finish processing.
- $\sigma = ([1,2],[2])$: Having initially scheduled tasks 1 and 2, task 2 finishes first – a machine becomes idle; the scheduler has to schedule another task.
- $\sigma = ([1,2,3],[2])$: Having initially scheduled tasks 1 and 2, task 2 finishes first and task 3 is scheduled immediately – now the scheduler needs to wait until either task 1 or task 3 to finish processing.

The states evolve until both S and F contain all four tasks. To see how $\sigma = (S,F)$ defines the order of events, consider $\sigma = (S,F) = ([1,2,3,4],[1,3,2,4])$, which describes an end state obtained after the following sequence of decisions/events:

- Tasks 1 and 2 are scheduled first,
- task 1 finishes first,
- task 3 is scheduled on the machine where task 1 was processing,
- the next task to finish is task 3 (task 2 is still running),
- Task 4 is scheduled on the newly available machine (where task 3 was processing), and
- task 2 finishes followed by task 4, which finishes last.

Given the order of events as depicted by σ (or more generally, a state within the scenario graph), we can formulate the inequalities that define the sequence of events by comparing the duration of the tasks scheduled on the different machines. Specifically, for the sequence presented above, the corresponding inequalities are:

$$
\begin{aligned}
d_1 &\leqslant d_2 \\
d_1 + d_3 &\leqslant d_2 \\
d_1 + d_3 + d_4 &\geqslant d_2,
\end{aligned}
$$

where the first inequality follows from the fact that tasks 1 and 2 are scheduled at the same moment but the duration of 1 is shorter, the second inequality follows from the fact that task 3 started immediately after task 1 but completed before task 2, and the third inequality follows from task 4 that started immediately after task 3 and completed after task 2. We can present the system of inequalities as:

$$
E_\sigma d \le 0, \quad E_\sigma = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 1 & -1 & 1 & 0 \\ -1 & 1 & -1 & -1 \end{bmatrix}, \quad d = [d_1\ d_2\ d_3\ d_4]^\top.
$$

Thus, for a general $\sigma = (S, F)$ such that $|S| = |F| = 4$, we can define matrix $E_\sigma \in \mathbb{R}^{3\times 4}$, such that its rth row is associated with comparing the sums of scheduled tasks' durations on the two machines when the r-th task, with index F_r, completes. For all tasks started before F_r completes, *i.e.*, $j \in \{S_1, \ldots, S_{r+1}\}$, $(E_\sigma)_{r,j} = 1$ if the task was scheduled to the same machine as F_r, and -1 otherwise. In our example, the second row of E_σ corresponds to task $F_2 = 3$. Thus, task $S_1 = 1$ and $S_3 = 3$ scheduled to the same machine get a coefficient of 1 and task $S_2 = 2$ gets a coefficient of -1.

Given σ, we can also express the total makespan in terms of the duration of different tasks. For example, denoting the makespan represented by σ as t_σ, $t_\sigma = d_1 + d_3 + d_4$, or equivalently

$$t_\sigma = e_\sigma^\top d, \quad e_\sigma = \begin{bmatrix} 1 & 0 & 1 & 1 \end{bmatrix}^\top.$$

In general, $e_\sigma \in \mathbb{R}^4$ is a binary vector encoding which of the tasks' durations constitute the makespan. Since the makespan is determined by the last completed tasks, for each $j \in [4]$, $(e_\sigma)_j = 1$ if j was scheduled to the same machine as F_4, and 0 otherwise. Thus, in our example, $F_4 = 4$, and thus the makespan is determined by tasks 1, 3, and 4 which are scheduled to the same machine.

In a similar way, we can encode all the states within a scenario tree, where its root node represents the initial state $\sigma = (\emptyset, \emptyset)$ from which one proceeds to the next nodes by adding a task to either S or F. Fig. 2 illustrates a part of the tree. Note that we discarded states with $|S| = 1$ when moving from the initial state since it is always optimal to schedule two tasks on the initially idle machines. In other words, state $(\emptyset, \emptyset)$ (initial state) progresses directly to states in which $|S| = 2$ such as $([1, 2], \emptyset)$. The scheduler's last 'real' decision is the one after which a single task is left to be scheduled. After that, the adversary decides on the completion order of the tasks. In our setting of $n = 4$, the last state in which the scheduler makes a decision has two started tasks where one of them already completed. After this decision, three tasks have started and one completed and the adversary makes the final decisions about the completion times of the in-process tasks and the task not yet started (the last remaining task is automatically scheduled when a machine becomes available).

As shown in Fig. 2, task durations corresponding to certain root-to-leaf paths are 'common'. For example, consider the two separate vectors of task duration corresponding to leaves $(S, F) = ([1, 2, 3, 4], [1, 3, 2, 4])$ and $(S, F) = ([1, 2, 3, 4], [1, 3, 4, 2])$. In both vectors, the decision about the duration of task 1 was made before the last scheduling decision was made, and thus the duration of task 1 has to be the same for both paths.

Overall, in Fig. 2, the scheduler aims to progress from the root node to the lowest level in the tree such that

- at a gray node, the scheduler selects the direct children node to go to, and
- at a white node, the adversary selects the direct children nodes to go to.

The adversary's optimization problem is, therefore, to set the arc lengths in such a way that the shortest path for the scheduler is as long as possible. In the next section, we shall explicitly define this problem.

6.1.2. Formulating the adversary's MILO problem.

We denote the set of all states $\sigma = (S, F)$ as $\mathcal{V}$. To decrease the state space of the scenario tree, certain states are discarded (see also the example described in Section 6.1.1). In particular,

- We omit states where $|S| = 1$, since they correspond to an initialization of the system in which one machine is idle, thus an additional task must be scheduled immediately. Hence, the states that follow the initial state are characterized by $|S| = 2$ (both machines are busy).

Table 4
The sets of states for the scheduler $\mathcal{D}$ (left) and the adversary $\mathcal{N}$ (right) when $m = 2$.

S-length	***F***-length	***S***-length	***F***-length
0	0	2	0
2	1	3	1
3	2	⋮	⋮
⋮	⋮	$n-1$	$n-3$
$n-2$	$n-3$	n	n

- Similarly, the last decision to be made is the one after which a single task remains to be scheduled. Such a decision is made at a state where the length of S is $n-2$ and the length of F is $n-3$. Afterwards, the adversary decides which tasks to complete and in which order. Hence, the states that follow states with $|S| = n-1$ and $|F| = n-3$ are the end states in which $|S| = |F| = n$.

Given this, we consider the state space $\mathcal{V}$ consisting of the sets of the scheduler states $\mathcal{D}$ and adversary states $\mathcal{N}$ (Table 4).

Let t_σ denote the optimal worst-case makespan corresponding to being in state $\sigma \in \mathcal{D} \cup \mathcal{N}$. We denote the set of all final states of the tree, for which $|S| = |F| = n$, by $\mathcal{L}$. For each state $\sigma \in \mathcal{D}$, the scheduler selects the task that minimizes the worst-case remaining makespan (corresponding to the inner minimum of the DP formulation in (12)). Thus,

$$t_\sigma = \min_{\delta \in \text{Children}(\sigma)} t_\delta, \ \sigma \in \mathcal{D}.$$

For each state $\sigma \in \mathcal{N} \setminus \mathcal{L}$, the adversary determines task completion times that maximize the worst-case remaining makespan (corresponding to the outer maximum within the DP formulation in (12)). Thus,

$$t_\sigma = \max_{\delta \in \text{Children}(\sigma)} t_\delta, \ \sigma \in \mathcal{N} \setminus \mathcal{L}.$$

For each end state $\sigma \in \mathcal{L}$, we define a separate decision variable of the adversary being a vector $d_\sigma \in \mathbb{R}^n$.

As illustrated in Section 6.1.1, for each final state σ, the set of inequalities that accommodates the task durations within the given scenario can be formulated in terms of the vector d_σ as follows:

$$E_\sigma d_\sigma \le 0, \ t_\sigma = e_\sigma^\top d_\sigma,$$

where, $E_\sigma \in \mathbb{R}^{(n-1)\times n}$ and $e_\sigma \in \mathbb{R}^n$, and for any $r \in [n-1]$ and $j \in [n]$,

$$(E_\sigma)_{r,j} = \begin{cases} 1, & j \in \{S_1, \ldots, S_{r+1}\},\ j \text{ and } F_r \text{ were scheduled to the same machine,} \\ -1, & j \in \{S_1, \ldots, S_{r+1}\},\ j \text{ and } F_r \text{ were not scheduled to the same machine,} \\ 0, & \text{otherwise,} \end{cases}$$

$$(e_\sigma)_j = \begin{cases} 1, & j \text{ and } F_n \text{ were scheduled to the same machine,} \\ 0, & \text{otherwise.} \end{cases}$$

An explicit algorithm for deriving E_σ and e_σ directly from σ is given in Appendix B.3. To achieve non-anticipativity of the adversary decision with respect to future scheduler decisions, the adversary decision vectors $d_\sigma, d_\delta \in \mathbb{R}^n$ for $\sigma, \delta \in \mathcal{L}$ have to be the same as long as sequences σ and δ are indistinguishable from each other in terms of scheduler decisions. Therefore, denoting $\mathcal{A}(\sigma, \delta) \in \mathcal{D}$ as the last common ancestor of both states σ and δ before a different scheduling decision is made, we can define

$$(d_\sigma)_i = (d_\delta)_i, \quad \forall i \in F, \quad (S, F) = \mathcal{A}(\sigma, \delta).$$

The combination of task duration inequalities, $E_\delta d_\delta \le 0$, and the non-anticipativity constraints can potentially lead to a situation

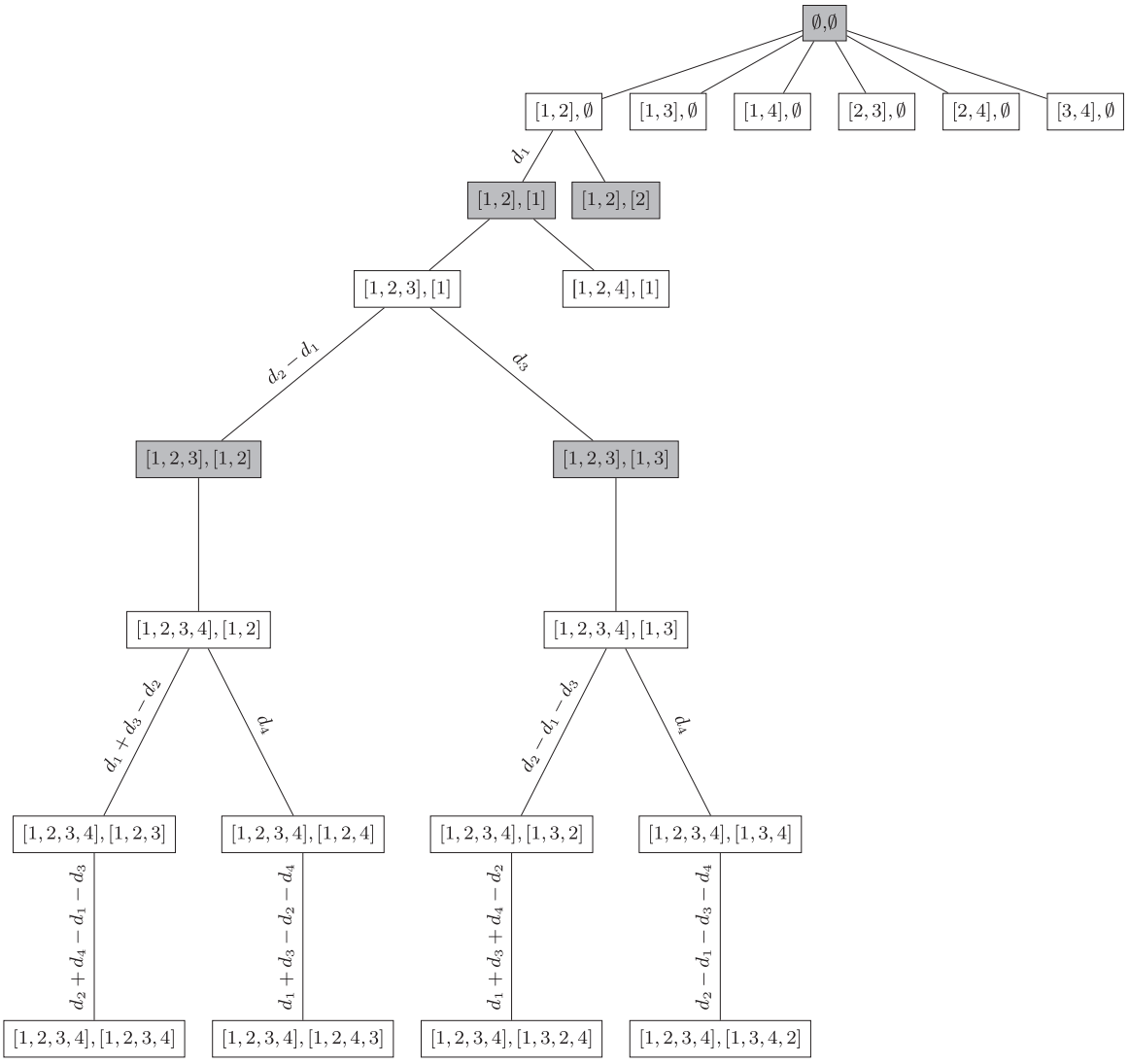

Fig. 2. A partial representation of a scenario tree for the case of $n = 4$ and $m = 2$. As the scheduling process progresses, one moves from the initial node $(\emptyset, \emptyset)$ downwards. On gray nodes the scheduler decides to move to one of the direct children. The other nodes are adversary nodes, in which the adversary controls the branching direction. The labels on the edges starting from the adversary nodes denote the time it takes to move from one node to another.

where a realization chosen for some $\sigma \in \mathcal{L}$, denoted by $\bar{d}_\sigma$, makes the feasible set for another $\delta \in \mathcal{L}$ empty:

$$\{d \in U : \ E_\delta d_\delta \leq 0, \ \ (d_\delta)_i = (\bar{d}_\sigma)_i, \quad \forall i \in F, \quad (S, F) = \mathcal{A}(\sigma, \delta)\} = \emptyset,$$

i.e., a specific realization of the first (few) tasks makes ending up in a given $\delta \in \mathcal{L}$ impossible. We will use big-M reformulations of $E_\sigma d_\sigma \leq 0$ to account for this possibility correctly.

We proceed to the problem formulation. If we denote by t_0 the optimal worst-case makespan corresponding to the root decision node $\sigma = (\emptyset, \emptyset)$, the adversary's problem (P) can be stated as:

$$\max_{d_\sigma, t_\sigma, z_\sigma} \ t_0 \tag{P}$$

$$\text{s.t. } t_\sigma = \min_{\delta \in \text{Children}(\sigma)} t_\delta \qquad \forall \sigma \in \mathcal{D} \tag{2}$$

$$t_\sigma = \max_{\delta \in \text{Children}(\sigma)} t_\delta \qquad \forall \sigma \in \mathcal{N} \setminus \mathcal{L} \tag{3}$$

$$(d_\sigma)_i = (d_\delta)_i \qquad \forall i \in F, (S, F) = \mathcal{A}(\sigma, \delta), \sigma, \delta \in \mathcal{L} \tag{4}$$

$$t_\sigma = e_\sigma^\top d_\sigma - z_\sigma M \qquad \forall \sigma \in \mathcal{L} \tag{5}$$

$$E_\sigma d_\sigma \leq z_\sigma M \qquad \forall \sigma \in \mathcal{L} \tag{6}$$

$$z_\sigma \in \{0, 1\}, \ d_\sigma \in U \qquad \forall \sigma \in \mathcal{L},$$

where M is a large positive number. The big-M constraints, (5) and (6), jointly with the binaries z_σ, enable one of two situations to occur for each of the corresponding paths d_σ for $\sigma \in \mathcal{L}$: (i) d_σ satisfies constraint (6) with $z_\sigma = 0$ (which is possible depending on the uncertainty set structure and the non-anticipativity constraint (4)) and has a corresponding finite makespan, (ii) if constraint (6) cannot be satisfied, then z_σ is set to 1 and the makespan corresponding to this path will be $-\infty$, thus it will never qualify as the worst-case makespan. We also note that since the adversary solves a maximization problem, it will never allow all the children leaves of a certain scheduler decision branch to take a value of $-\infty$, since that would imply that this will be the value of the entire problem.

Because the problem is a maximization one, constraints (2), involving minimum terms, can be reformulated as

$$t_\sigma \leq t_\delta, \quad \forall \delta \in \text{Children}(\sigma), \quad \sigma \in \mathcal{D},$$

while constraints (3), involving maximum terms, need to be implemented using auxiliary binary variables and a big-M constraint as follows:

$$t_\sigma \le t_\delta + M \cdot w_{\sigma,\delta}, \qquad \forall \delta \in \text{Children}(\sigma), \sigma \in \mathcal{N} \setminus \mathcal{L}$$

$$\sum_{\delta \in \text{Children}(\sigma)} w_{\sigma,\delta} \le |\text{Children}(\sigma)| - 1 \qquad \forall \sigma \in \mathcal{N} \setminus \mathcal{L}$$

$$w_{\sigma,\delta} \in \{0,1\}, \qquad \forall \delta \in \text{Children}(\sigma), \sigma \in \mathcal{N} \setminus \mathcal{L}.$$

In Appendix B.4, we present the MILO formulation for settings with more than two machines. Although exact, this formulation's complexity is exponential, as shown in Appendix C. For that reason, in the next section we develop an adaptive heuristic.

Remark 1. The MILO formulation (P) is also useful for computing the worst-case makespan of a specific SL policy. Specifically, for a given permutation of the tasks, π, we can eliminate some of the branches of the tree, which do not correspond to the policy. For example, for $n = 4$ and permutation $\pi = (1, 2, 3, 4)$ nodes in $\mathcal{L}$, which are children of a state σ in which $S = [1, 2, 4]$, will be eliminated. The optimal worst-case makespan for permutation π is obtained by eliminating the irrelevant leaves from the set $\mathcal{L}$ and solving the MILO for the reduced problem. Thus, one can find the optimal SL by solving the problem for every permutation π of the n tasks, and choosing the one with the lowest worst-case makespan.

6.2. Two-stage SA heuristic (2SSA)

As the exact MILO formulation of AR does not scale well with the problem size, we propose a heuristic for the two machine setting ($m = 2$) that takes into account only one adaption of the task allocations, as opposed to $n - 2$ such adaptations in the exact reformulation.

The idea of the heuristic is as follows: for every task pair $i < j$, we schedule the two tasks in parallel initially and, based on which task finishes first and its duration, apply the SA to the remaining tasks. The schedule of the remaining tasks adapts itself thus only to the durations of the two initial tasks, remaining fixed afterwards. This process is repeated for each pair of tasks ($(n-1)n/2$ times), and the pair with the lowest predicted worst-case makespan is chosen to be scheduled. Intuitively, the heuristic strives to place tasks whose durations are most informative about the remaining task durations as the initial tasks. The heuristic is then implemented in a rolling-horizon fashion, allowing to change decisions about tasks not yet scheduled based on newly available information.

For each pair $i < j$, in order to predict worst-case makespan, we look at the problem in two stages: before and after the first of the two initial tasks completes. Thus, in order to compute the worst-case makespan, we solve a two-stage robust optimization problem from the adversary point of view. To formulate this problem, we define $x \in \{0,1\}^n$ to be a vector of task allocations as in Section 4.3.1 where without loss of generality we have $x_i = 1$ and $x_j = 0$, *i.e.*, task i is scheduled on the first machine and task j on the second machine. The two-stage problem is then given by:

$$\max \left\{ \begin{array}{l} \sup\limits_{\tilde{d}_i:\ \tilde{d}_i \in U,\ \tilde{d}_i < \tilde{d}_j} \ \min\limits_{x \in \{0,1\}^n:\ x_i = 1,\ x_j = 0} \ \sup\limits_{d \in U:\ d_i = \tilde{d}_i,\ d_i < d_j} \max\{x^\top d, (e-x)^\top d\}, \\ \sup\limits_{\tilde{d}_j:\ \tilde{d}_i \in U,\ \tilde{d}_i \ge \tilde{d}_j} \ \min\limits_{x \in \{0,1\}^n:\ x_i = 1,\ x_j = 0} \ \sup\limits_{d \in U:\ d_j = \tilde{d}_j,\ d_i \ge d_j} \max\{x^\top d, (e-x)^\top d\} \end{array} \right\}. \tag{7}$$

In this formulation, the first-stage adversary decision, represented by the outer maximum and the first supremum, is which task, i or j, finishes first and its corresponding duration. Based on this information, in the next minimum stage, the scheduler decides on the SA schedule for the remaining tasks. In the second-stage, represented by the last supremum and max, the adversary decides on the exact durations of the remaining tasks to maximize the makespan.

For fixed $i < j$, this two-stage problem, can be solved using a discrete uncertainty modification of the Column-and-Constraint generation algorithm of Zeng & Zhao (2013) for two-stage robust optimization problems with continuous uncertainty. In Appendix E, we outline the exact implementation of 2SSA. In Section 7, we show that 2SSA admits reasonable running times for problems with $n \le 20$ with which we experimented.

In order to illustrate the 2SSA heuristic, we go back to the four task example presented in 6.1.1. Assume that (1,4) are scheduled first, then 2SSA separates the rest of the schedule as follows: scenarios 4 or 5 (1 finished before 4 and has duration lower than 2.5), task 2 will be scheduled after task 1 and task 3 after task 4, with a worst-case duration of 7.5. For scenario 1 (1 finishes before 4 and has duration 3) we will schedule task 3 after task 1 and task 2 after task 4, resulting in a makespan of 7.5. Finally for scenarios 2 and 3 (4 finishes before 1 with duration 4) task 3 will be scheduled after 4 and task 2 after task 1, resulting in a worst-case makespan of 7.5. Thus, we can conclude that initially scheduling tasks 1 and 4 is optimal for 2SSA, since it has a worst-case makespan of 7.5, equal to the optimal AR policy.

7. Numerical study

Our analytical work is accompanied by a numerical study that focuses on problem instances with two machines. We start with five-task instances, which are large enough to differentiate between alternative policies and elicit some insights, yet small enough to allow us to run the MILO formulation presented in the last section. We use these instances to demonstrate the value of adaptivity, and to compare the performance of SA, SL, AR, and 2SSA. We then proceed to investigate larger instances with 10–20 tasks, comparing the performance of SA to that of 2SSA in favor of validating our theoretical bounds in terms of identifying the settings in which accounting for adaptivity in the planning stage leads to a significant benefit.

7.1. The setup

We investigate the SA, SL, AR and 2SSA policies. In the coming experiments, SA is computed using a standard MILO formulation given in Appendix D. AR and SL are implemented using the full and pruned versions of the MILO problem of Section 6.1. Finally, 2SSA is computed using the algorithm outlined in Appendix E.

For a realistic comparison, we implement the policies for each scenario in a rolling-horizon fashion similarly to the way they are expected to be applied in reality. We note that while we implemented the policies without the re-optimizing via rolling-horizon we do not include the related results, since they were consistently inferior compared to the rolling-horizon implementations. A scenario represents a particular realization of task durations for the tasks. We conduct each experiment as follows: At the beginning of the planning horizon, when task durations are only known to belong to U, we select the optimal two first tasks to be scheduled, according to the considered policy. If multiple initial scheduling decisions give the same worst-case makespan, we choose between them according to lexicographic ordering (w.r.t. task indices). Then, we determine which of the two running tasks finishes first (based on the known duration for the considered scenario). Next, we update the uncertainty set to include the information about the duration of the completed task and for how long the other task has

been processing so far, and calculate the optimal next scheduling decisions by the considered policy. For AR, for example, this means solving the MILO using the current information about task duration and completion times in order to select the task to be scheduled for the idle machine. Once both machines are busy, we again determine which of the running tasks finishes first, and optimally select the next task to be scheduled for the first-to-be-idle machine. This sequence of events continues until all tasks have been scheduled and finished processing.

We find a lower bound on the obtainable makespan, by applying a PH policy for each scenario. Namely, we assume that the specific scenario is known in advance and determine the optimal task-to-machine allocations that minimize the makespan. The makespan obtained by the PH policy is a lower bound on the possible makespan achieved by any policy. Therefore, we use the PH makespan as a benchmark for the investigated policies.

To define the performance measures, we denote $p \in \{AR, SL, SA, 2SSA, PH\}$ as the selected policy, $k \in \{1, \ldots, N\}$ identifies a problem instance and $s \in \{1, \ldots, R\}$ denotes the scenario number. Accordingly, we denote $\tilde{T}_{k,p,\cdot}$ and $T_{k,p,s}$ for problem instance k, policy p, and scenario s, as the promised worst-case makespan at the beginning of the planning horizon and the realized makespan when the policy is applied via a rolling-horizon approach, respectively. Notice that the promised worst-case makespan $\tilde{T}_{k,p,\cdot}$ is independent of the scenario's realization and is only affected by the specification of the uncertainty set, since it is the makespan for the case in which the worst-case scenario is realized.

The mathematical formulas and names of the measures that we report are described in Table 5. The first measure is the percentage of instances where the policy's initial decision was not optimal for the AR policy. The next three measures specify the actual values (max and average) of the makespan, for comparison purposes. The final four measures give us an indication of how well the policies perform relative to the AR and PH solutions:

- **Promised to max PH.** The upper bound on the "price" that a risk-averse scheduler who commits (*e.g.*, in a contract) to the promised makespan is expected to pay with respect to a PH policy. The reasoning behind this measure is that before any of the tasks starts processing, the scheduler does not know which scenario will be realized, thus she compares the promised makespan to the maximal perfect hindsight makespan across scenarios.
- **Max makespan to max PH.** Following the previous measure, a ratio between the maximal realized makespan to the maximal perfect hindsight makespan gives an indication into the expected price that a risk averse scheduler who applies the rolling-horizon policies would contractually pay with respect to the PH policy.
- **Makespan to PH.** While RO does not optimize for non-worst-case scenarios, we approximate the average ratio for each scenario between the realized makespan by the applied policy and its perfect hindsight policy. This measure gives an indication into the expected price of the risk-averse scheduler for an "average" scenario when compared with the corresponding perfect hindsight makespan.
- **Max makespan to max AR.** To benchmark the best robust policy, which is AR with its alternatives – SA, SL and 2SSA – we replicate the "Max makespan to max PH" measure with AR replacing PH as the reference point.

We note that these experiments evaluate the performance of the policies under conditions of worst-case (denoted as the promised makespan) and non-worst-case scenarios in which we uniformly sample task durations from the respective scenario uncertainty sets. It is important to experiment with both conditions since the RO methodology is indifferent with respect to non-worst-case scenarios, thus an optimal robust policy may perform poorly when applied on such scenarios (see, Iancu & Trichakis, 2014).

We conduct two types of experiments, one includes small problem instances and the second larger problem instances. For the former experiment, we use discrete uncertainty sets, which enable to manually verify the optimal policy. For the latter experiment type, we use continuous budgeted uncertainty sets in order to demonstrate the connection to the theoretical results. Moreover, using this variety of uncertainty sets allows us to demonstrate the versatility of the suggested solution approach.

The code was run on a PowerEdge R740xd server with two Intel Xeon Cold 6254 3.1GHz processors, each with 18 cores, and a total RAM of 384GB.

7.2. Small problem instances

First, we test the policies over $N = 500$ problem instances, where each instance k has a discrete uncertainty set $U^k \subseteq \mathbb{R}^n$ ($n = 5$) consisting of $|U^k| = R = 15$ scenarios. Each problem instance $k \in \{1, \ldots, N\}$ is generated by drawing a vector of the nominal durations $d^{0,k}$ and their perturbations $\bar{d}^k$ out of a uniform distribution according to $d^{0,k} = (d_1^{0,k}, d_2^{0,k}, \ldots, d_n^{0,k}) \sim \text{Unif}([0.1, 2.0]^n)$ and $\bar{d}^k = (\bar{d}_1^k, \bar{d}_2^k, \ldots, \bar{d}_n^k) \sim \text{Unif}([0.1, 5.0]^n)$, respectively. Each of the $s \in \{1, \ldots, R\}$ scenarios of instance k is sampled as

$$\tilde{d}_j^{k,s} = d_j^{0,k} + u_j^{k,s} \bar{d}_j^k, \quad j = 1, \ldots, n,$$

with $u^{k,s} \in \mathbb{R}^n$ sampled uniformly from one of the following sets:

(I) An n-dimensional ball $\{u \in \mathbb{R}_+^n : \ \|u\|_2 \leq 1\}$ (type I).
(II) An n-dimensional box $\{u : \ \|u\|_\infty \leq 1\}$ (type II)

We rounded $\tilde{d}^{k,s}$ to multiples of 0.1 to allow for the situation that two tasks finish simultaneously. We refer to the resulting discrete uncertainty sets as uncertainty sets of type I and II based on the method in which we generate $u^{k,s}$. Because the results for both set types are very similar, we discuss here only the type I sets, and relegating the type II results in Appendix F.

In Table 6 we summarise the first 4 measures comparing the different methods and the average CPU times required to obtain the optimal initial decisions and run the entire RH sequence, per method. The significance of AR is underlined by the fact that its first-stage decisions were different from the other policies in a substantial portion of the problems. Indeed, we found that there is a high correlation between percent of different initial decisions compared to AR and the policies performances compared to AR. That is, 2SSA is expected to be the best performing among the remaining policies, followed closely by SL and SA being the last.

Our results indicate that the AR promised makespan was the shortest, with 2SSA nearly the same (longer by 0.4%), SL very close behind (longer by 0.7%) and SA trailing behind with a longer makespan by 9.7%. The upper bound on the "price" that a risk-averse scheduler, who commits to the promised makespan, is expected to pay upfront with respect to a PH policy (as indicated by the Promised to max PH measure) was very small for AR (0.1%), 2SSA (0.5%), SL (0.8%), and much higher for SA (3.6%).

As visible from the CPU times results, 2SSA provides a performance similar to SL, albeit at a significantly lower computational cost. As expected, SA is the fastest of all methods, however, providing very poor performance. Summarizing, 2SSA is the only method that provides good performance at a reasonable computational cost.

Fig. 3 presents the Cumulative Distribution Function (CDF) of the last four performance measures in Table 5, by means of empirical CDF across instances. Fig. 3(a) shows the inferior performance of SA compared to the other policies in providing good promised makespans. Among the remaining policies, AR is the best, followed

Table 5
Description and formulas of the performance measures.

Short name	Description	Calculation
Suboptimal initial decision	Percentage of instances with initial decisions different from AR	-
Promised makespan	Average promised makespan of policy p	$\frac{1}{N}\sum_{k=1}^{N}\tilde{T}_{k,p,\cdot}$
Max makespan	Average maximal makespan of policy p	$\frac{1}{N}\sum_{k=1}^{N}\max_{s\in[R]}T_{k,p,s}$
Makespan	Average makespan of policy p	$\frac{1}{N}\sum_{k=1}^{N}\frac{1}{R}\sum_{s=1}^{R}T_{k,p,s}$
Promised to max PH	Ratio between the promised of policy p to the maximal makespan under perfect hindsight	$\frac{\tilde{T}_{k,p,\cdot}}{\max_{s\in[R]}T_{k,PH,s}}-1$
Max makespan to max PH	Ratio between the maximal of policy p to the maximal makespan under perfect hindsight	$\frac{\max_{s\in[R]}T_{k,p,s}}{\max_{s\in[R]}T_{k,PH,s}}-1$
Makespan to PH	Ratio between the makespan of policy p to the makespan under perfect hindsight	$\frac{1}{R}\sum_{s=1}^{R}\frac{T_{k,p,s}}{T_{k,PH,s}}-1$
Max makespan to max AR	Ratio between the maximal makespan of policy p to the maximal makespan the AR policy	$\frac{\max_{s\in[R]}T_{k,p,s}}{\max_{s\in[R]}T_{k,AR,s}}-1$

Table 6
Performance measure results for type I uncertainty sets.

Short name	AR	2SSA	SL	SA	PH
Suboptimal initial	-	7%	11%	50%	-
Promised makespan	5.628(0.914)	5.653(0.913)	5.668(0.928)	6.174(1.034)	-
Max makespan	5.628(0.914)	5.641(0.913)	5.648(0.918)	5.824(0.962)	5.624(0.913)
Makespan	4.832(1.013)	4.821(1.015)	4.830(1.017)	4.874(1.042)	4.743(1.015)
CPU time initial decision (sec)	8.27	0.26	17.09	0.004	-
CPU time RH (sec)	19.26	1.93	40.53	0.05	-

Average makespan values are presented with their standard deviations in parentheses.

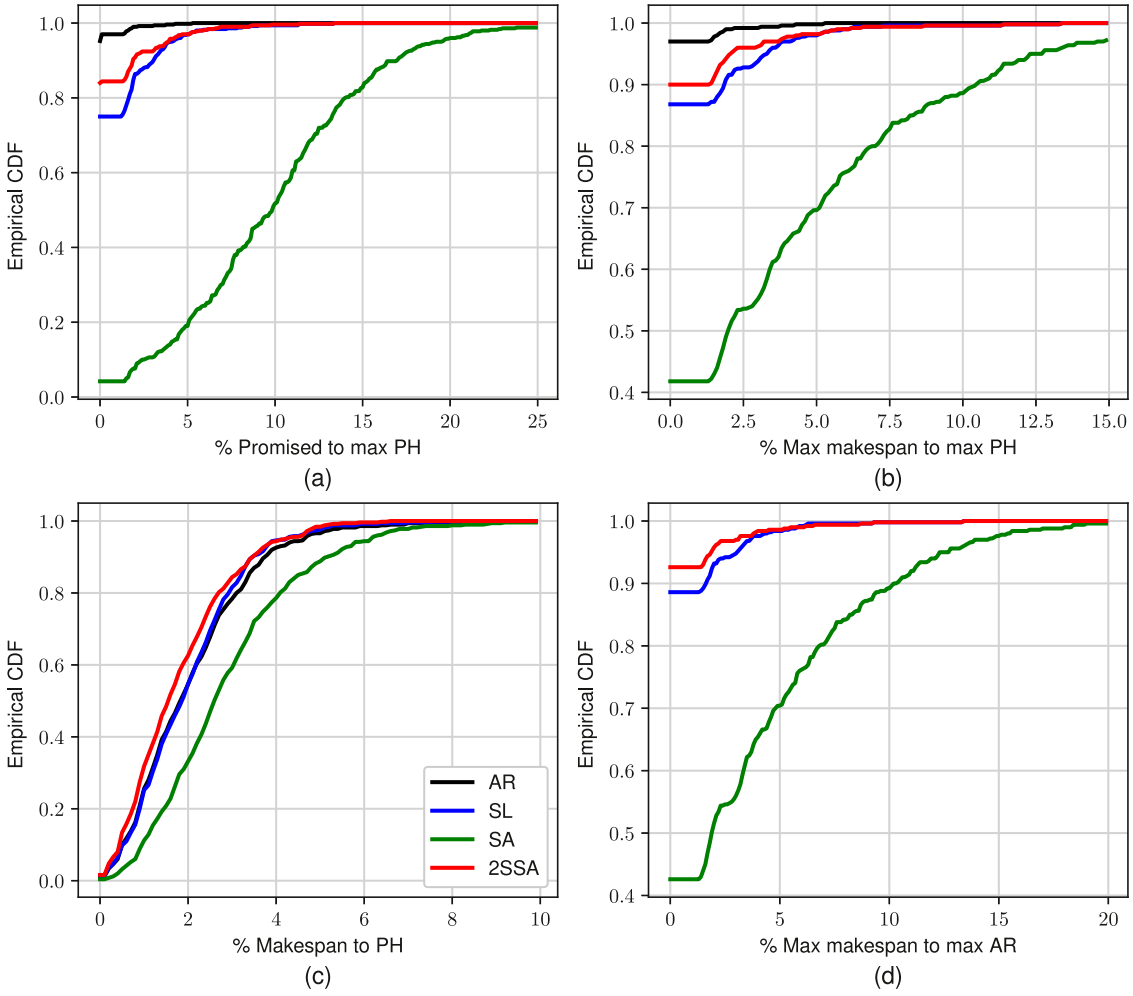

Fig. 3. Empirical CDF (over instances) of the last four performance measures of Table 5 for Type I uncertainty set.

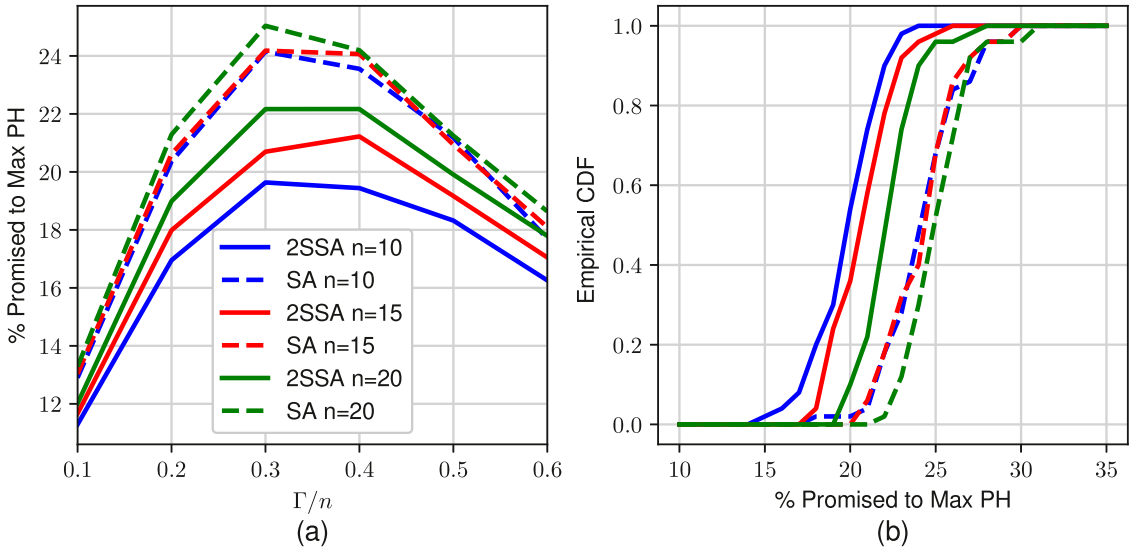

Fig. 4. Promised to Max PH ratio for $n = 10, 15, 20$ over $N = 50$ instances: (a) average for different values of Γ (b) empirical CDF for $\Gamma = 0.3n$.

by 2SSA and SL. Fig. 3(b) shows similar relationships between the different methods with respect to the ratio of the worst-realization of a given policy compared to the worst PH duration, with the SA being clearly the worst. When taking the less conservative measure of taking the average ratio of scenarios (Fig. 3(c)), the four policies are closer to each other, with 2SSA becoming surprisingly the best. In the end, in Fig. 3(d) we see the worst-case performance of the policies SL, SA and 2SSA compared to AR. While 2SSA and SL with comparable performance to that of the AR, the SA can go as much as 20% worse.

To recapitulate, AR is the best policy for the risk-averse scheduler but it is not scalable. Our heuristic, the 2SSA, and the SL achieve comparable performance with respect to AR, while 2SSA is more computationally efficient. SA demonstrates the worst performance. As expected, the gaps between all policies are smaller when applying the policies on average (non-worst-case) scenarios, yet SA is still inferior with respect to the other policies.

7.3. Large problem instances

In this set of experiments, we test the policies for $n = 10, 15, 20$ tasks. Because of the problem sizes we could only test the SA and 2SSA policies. For each value of n, and budget $\Gamma = 0.1n, 0.2n, 0.3n, 0.4n, 0.5n, 0.6n$, we generated $N = 50$ problem instances. For each instance, we used a budgeted uncertainty set $U^k \subseteq \mathbb{R}^n$ of the form

$$U^k = \left\{ d \in \mathbb{R}^n : d_i = d_i^{0,k} + u_i \bar{d}_i^k,\ i \in [n],\ u \in [0,1]^n, \sum_{i=1}^{n} u_i \le \Gamma \right\},$$

where the nominal durations $d^{0,k}$ and their perturbations $\bar{d}^k$ are randomly generated such that $d_i^{0,k} \sim \text{Unif}(0.5, 5.0)$ and $\bar{d}_i^k \sim \text{Unif}(0.5, 1.0) \cdot d_i^0$, respectively for each $i \in [n]$. For each realized scenario, we found the PH solution by assuming that the realized durations are known in advance.

We start by comparing the promised duration of SA and 2SSA to the worst-case duration of PH on a sample of $K = 50$ scenarios, which we present in Fig. 4. We observe the same phenomenon predicted by our bound in Section 4.3.1, *i.e.*, SA promised duration gets close to those of the PH when Γ/n gets close to 0 or 1, and is further away from PH when $\Gamma/n \in [0.3, 0.4]$ in Fig. 4(c). We also note that this maximal difference is around 22% implying a potential for improvement by methods which take into account adaptivity. Indeed, we see that the promised makespan of the 2SSA heuristic improves upon SA in this region, by $2\% - 4\%$. Moreover, as n increases both methods' have worse promised duration with respect to the PH, and the gap between them decreases. We explain the latter by the limited way in which 2SSA accounts for adaptivity in the planning stage. Since only one stage of adaptivity is taken into account, as n increases there are more stages for which we do not account for adaptivity and thus, the relative benefit decreases. We note that the increase in the promised to PH ratio for both methods as n grows larger, is in contrast to their actual performance when implemented in a RH fashion, as we discuss next. This gap between the promised and actual performance due to not taking into account enough of the future adaptivity may put a decision maker at a disadvantage when contractually committing to a specific makespan.

To observe the actual performance of the methods, as they are applied in a RH fashion, we generated $K = 50$ random scenarios and computed, for each, the makespan to PH ratio. Fig. 5 depicts the performance measures of RH-SA and RH-2SSA as a function of both n and Γ. Additionally, we present the empirical CDF of 'Max makespan to max PH' for $\Gamma = 0.3n$. We see that as n increases both the average and maximum gap between RH-SA and PH decreases, as predicted by the bound in Section 4.3.2. In Fig. 5(b) we observe that for the lower values of Γ/n, 2SSA reduces the suboptimality of SA w.r.t. the PH by roughly 50%. Importantly, Fig. 5 demonstrates that, even in the rolling-horizon implementation, 2SSA has favorable performance compared to the SA for all tested settings.

7.4. Computational times

We compared the computational times of the two policies, SA and 2SSA, over instances with $n = 5, 10, 15, 20$ tasks and under budgeted uncertainty. As the 2SSA can be easily parallelized over the $n(n-1)/2$ pairs of tasks to be scheduled first, we tested the running times with and without 2SSA parallelization. Fig. 6 summarizes the average run times of the entire rolling-horizon simulations for different Γ/n ratios. It indicates that, given parallelization, the performances of SA and 2SSA are comparable for the tested problem sizes. We note that the presented running times for both SA and 2SSA include all the re-optimization rounds over the entire rolling-horizon. Thus, for the case of n tasks, the time to adapt the scheduling decision each time a task completes is,

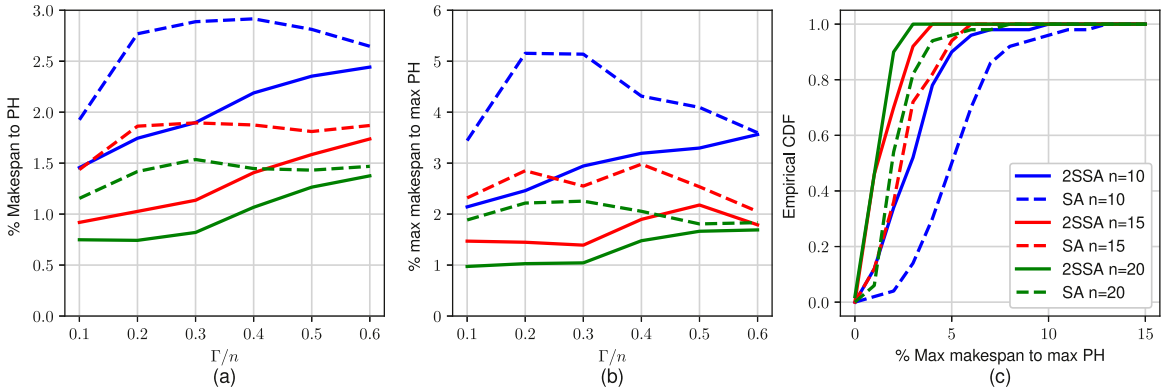

Fig. 5. Makespan to PH ratios for $n = 10, 15, 20$ over $N = 50$ instances: (a) average 'Makespan to PH' (b) average 'Max makespan to max PH' (c) CDF of 'Max makespan to max PH' for $\Gamma = 0.3n$.

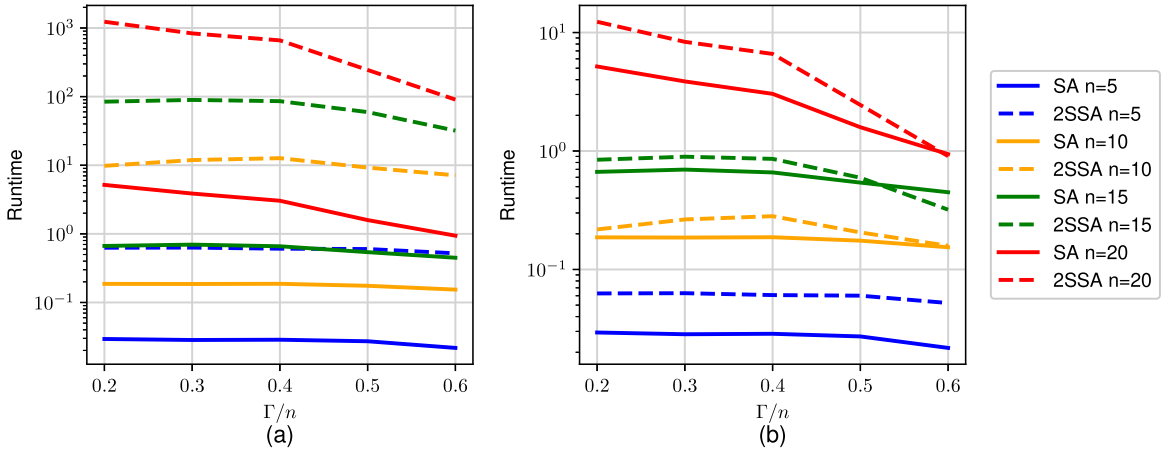

Fig. 6. Computational times for $n = 5, 10, 15, 20$ over the entire rolling-horizon: (a) average 'raw' computational times (b) average computational times with parallelization on up to 100 cores.

roughly, the presented time divided by the $n-2$ times that each policy is re-optimized. These times are expected to be reasonable for most real-world settings, in the sense that machine will be idle only a short time while the problem is resolved. Practically, even these idle times can be decreased in classical manufacturing processes; this is due to the fact that the machines can accurately predict residual processing times when a part is close to completing its processing, allowing to solve the next re-optimization ahead of time.

8. Conclusions

We study adaptive and static robust optimization policies for the PMS, which is an important problem in multiple domains such as production lines in which machines process a set of tasks, computer multiprocessors ("cloud processing") for processing jobs, shipyards and ports in which ships are loaded and unloaded, doctors who treat patients in a walk-in clinic or triage setting, and teachers who educate student groups.

The suggested methodology suits settings in which the probabilistic knowledge about task durations is limited or costly to attain. In such circumstances, it is rather easy to design a polyhedral or ellipsoidal uncertainty set that frames the involved uncertainty. Ben-Tal, El Ghaoui, & Nemirovski (2009) provide guidance and probabilistic guarantees in favor of designing uncertainty sets that balance the level of conservatism and the probability that a constraint is violated by a scenario.

We demonstrate that ARO models that consider, in the planning stage, the optimal future (wait-and-see) decisions are preferable over non-adaptive RO models since they lower the maximum possible makespan that the scheduler can promise. Our experiments point out that the average advantage of adaptive-based bids is estimated to be $9-10\%$ over its non-adaptive RO policy. To this end, we offer a heuristic adaptive policy, the 2SSA, that can be used to solve larger problems.

Our analyses indicate that a good adaptive policy may be important for mid-range Γ values rather than when uncertainty is very small (*i.e.*, $\Gamma \to 0$) or large (*i.e.*, $\Gamma \to n$).

Adaptive robust policies are not only superior over static policies in the planning stage – they are also preferable in the implementation stage. Specifically, policies that take the later-stage adaptivity of the decisions into account remain preferable even when the static policies are re-optimized every time new information becomes available (rolling-horizon). A hint into the reason for this is provided by the $42-59\%$ of the problem instances in which an adaptive policy yielded different first-stage decisions compared to a SA policy. That means that the adaptive policies not only offer better makespan guarantees, but also select decisions that lead to better realized duration. Our results shows clearly that the algorithms' performance deteriorate with the fraction of different first-stage decisions compared to fully adaptive AR. Accordingly, 2SSA (7% different first-stage decisions) was the best performing algorithm, followed by SL (11%), and lastly the static SA policy (50%).

To conclude, while robust SA policies are widely investigated and used in risk averse settings, they may achieve inferior performance in practice compared to adaptive alternatives. Since the performance gap between an optimal adaptive policy and a static one is quite significant, and alternative existing policies such as the SL are not computationally efficient, we suggested the new 2SSA heuristic. 2SSA can grant the scheduler competitive advantages both in the planning stage and in the implementation stage.

9. Future research

Following this research, we recommend pursuing the following research directions:

1. Extending the scope of scheduling problems beyond PMS. As a first step, we suggest two problems that consider precedence constraints between tasks: the job-shop problem, in which tasks require a unitary resource (machine) and the task includes sub-tasks with precedence constraints, and the resource constrained project scheduling problem, in which a task may require more than one type and/or more than one resource unit to be processed.
2. It would be interesting to investigate the types of problems and settings under which first-stage decisions are different for adaptive and non-adaptive scheduling policies.
3. Finally, it would be worthwhile to explore more efficient MILO formulations that are more amenable to approximation algorithms.

Acknowledgements

We are grateful to Michael Pinedo for consultations at the early stage of this work. The research was supported by the Israel Science Foundation grant No. 226/21 (IC). The second author's work was financed by The Dutch Research Council (NWO) Grant VI.Veni.191E.035.

Appendix A. Proofs

Proof of Proposition 1. For all $i \in [m]$, let $\bar{d}_i = \max_{d_i \in \mathcal{U}_i} d_i$. Let $J \in \mathcal{J}_{n.m}$ be some partition of the tasks to machines. Then, the makespan duration induced by this partition is given by

$$\max_{d\in\mathcal{U}} \max_{j\in[m]} \sum_{i\in J_j} d_i = \max_{j\in[m]} \sum_{i\in J_j} \bar{d}_i,$$

and the optimal SA policy $J^* = (J_1^*, \ldots, J_m^*)$ satisfies

$$\max_{j\in[m]} \sum_{i\in J_j^*} \bar{d}_i \le \max_{j\in[m]} \sum_{i\in J_j} \bar{d}_i, \ \forall J \in \mathcal{J}_{n.m}\ .$$

Now let $\bar{d} = (\bar{d}_1, \ldots, \bar{d}_n)$ and define $J^{\mathrm{AR}*} = \mathcal{M}(P^{\mathrm{AR}*}, \bar{d})$, where $P^{\mathrm{AR}*}$ is an optimal AR policy. Then,

$$\max_{j\in[m]} \sum_{i\in J_j^*} \bar{d}_i \le \max_{j\in[m]} \sum_{i\in J^{\mathrm{AR}*}} \bar{d}_i \le \max_{d\in\mathcal{U}, J=\mathcal{M}(P^{\mathrm{AR}*},d)} \max_{j\in[m]} \sum_{i\in J_j} d_i, \tag{8}$$

where the first inequality follows from the optimality of J^* with respect to $J^{\mathrm{AR}*}$, and the last inequality follows from adding the maximum over all $d \in \mathcal{U}$. Since, however, the worst-case makespan of optimal AR is always shorter than or equal to that of the optimal SA policy, both worst-case makespans are equal, and thus both inequalities in (8) are in fact equalities.

We now turn to prove the equality with respect to the worst-case makespan by the optimal SL. We first show that for any $\pi \in \Pi_n$, using the realization $\bar{d}$ results in the worst-case makespan. Assume, to the contrary, that for some $\pi \in \Pi_n$ a worst-case makespan is achieved by a realization $\tilde{d}$ that contains a component π_j such that $\tilde{d}_{\pi_j} < \bar{d}_{\pi_j}$. The starting times (and, therefore, ending times) of all tasks π_i for $i > j$ using $\tilde{d}$ would be earlier or the same as those with $\bar{d}$, leading to a shorter or identical makespan. Thus, realization $\bar{d}$ would always lead to the worst-case makespan. Now, let $\pi^\dagger \in \Pi_n$ be a permutation such that $\mathcal{M}(P^{\mathrm{SL},\pi^\dagger}, \bar{d}) \equiv J^*$ (such a permutation can always be constructed by the order of starting times), let π^* be the permutation associated with the optimal SL policy, and let $J^{\mathrm{SL}*} \equiv \mathcal{M}(P^{\mathrm{SL}*}, \bar{d}) \equiv \mathcal{M}(P^{\mathrm{SL},\pi^*}, \bar{d})$. Then,

$$\begin{aligned}\max_{d\in\mathcal{U}, J=\mathcal{M}(P^{\mathrm{SL},\pi^\dagger},d)} \max_{j\in[m]} \sum_{i\in J_j} d_i &= \max_{j\in[m]} \sum_{i\in J_j^*} \bar{d}_i \le \max_{j\in[m]} \sum_{i\in J^{\mathrm{SL}*}} \bar{d}_i \\ &= \max_{d\in\mathcal{U}, J=\mathcal{M}(P^{\mathrm{SL},\pi^*},d)} \max_{j\in[m]} \sum_{i\in J_j} d_i,\end{aligned}$$

where the first equality follows from the definition of $\pi^\dagger$, the inequality follows from the optimality of the static policy J^* (from the point of view of SA, $J^{\mathrm{SL}*}$ is suboptimal) and optimality of $\bar{d}$ for the worst-case for static allocation policies, and the second equality follows from the definition of $J^{\mathrm{SL}*}$ and again the fact that $\bar{d}$ is a worst-case for π^*. Combining this inequality with the optimality of π^*, we obtain that the worst-case makespans of the optimal SL and SA policies must be equal. □

Proof of Theorem 1. In order to lower bound the worst-case makespan of the PH policy, we can restrict the adversary to split the uncertainty equally between the tasks. For these scenarios the PH policy would use allocation $\tilde{x}$. Thus,

$$\begin{aligned}PH &= \max_{d\in U} \min_{x\in\{0,1\}^n} \max\{x^\top d, (e-x)^\top d\} \\ &\ge \max\left\{\tilde{x}^\top\left(d^0 + \frac{\Gamma}{n}\bar{d}\right), (e-\tilde{x})^\top\left(d^0 + \frac{\Gamma}{n}\bar{d}\right)\right\} \\ &= \max\left\{\sum_{i\in\mathcal{I}} d_i^0\left(1+\frac{\alpha\Gamma}{n}\right), \sum_{i\in\bar{\mathcal{I}}} d_i^0\left(1+\frac{\alpha\Gamma}{n}\right)\right\} \\ &= \left(1+\frac{\alpha\Gamma}{n}\right)\max\left\{\sum_{i\in\mathcal{I}} d_i^0, \sum_{i\in\bar{\mathcal{I}}} d_i^0\right\}\end{aligned}$$

Moreover, the SA worst-case is not worse than the one obtain by using $\tilde{x}$ as the allocation, and thus,

$$\begin{aligned}SA &= \min_{x\in\{0,1\}^n} \max_{d\in U} \max\{x^\top d, (e-x)^\top d\} \\ &\le \max_{d\in U} \max\{\tilde{x}^\top d, (e-\tilde{x})^\top d\} \\ &= \max_{d\in U} \max\left\{\sum_{i\in\mathcal{I}} d_i, \sum_{i\in\bar{\mathcal{I}}} d_i\right\} \\ &= \max\left\{\sum_{i\in\mathcal{I}} d_i^0 + \alpha \max_{\mathcal{S}\subseteq\mathcal{I}, |\mathcal{S}|=\min\{\lfloor\Gamma\rfloor, |\mathcal{I}|\}} \sum_{j\in\mathcal{I}\setminus\mathcal{S}} \left\{\sum_{k\in\mathcal{S}} d_k^0 + (\min\{\Gamma, |\mathcal{I}|\} - |\mathcal{S}|)d_j^0\right\},\right. \\ &\qquad \left.\sum_{i\in[n]\setminus\mathcal{I}} d_i^0 + \alpha \max_{\mathcal{S}\subseteq[n]\setminus\mathcal{I}, |\mathcal{S}|=\min\{\lfloor\Gamma\rfloor, n-|\mathcal{I}|\}} \sum_{j\in[n]\setminus\mathcal{I}\cup\mathcal{S}} \left\{\sum_{k\in\mathcal{S}} d_k^0 + (\min\{\Gamma, n-|\mathcal{I}|\} - |\mathcal{S}|)d_j^0\right\}\right\}.\end{aligned}$$

Using our defined notation, can rewrite the above inequality as

$$SA \le \max\left\{\sum_{i\in\mathcal{I}} d_i^0, \sum_{i\in\bar{\mathcal{I}}} d_i^0\right\} + \alpha \max_{\mathcal{S}\in\{\mathcal{I},\bar{\mathcal{I}}\}} \left\{\sum_{k=1}^{\min\{|\mathcal{S}|,\lfloor\Gamma\rfloor\}} d_{(k)}^{0,\mathcal{S}} + \Delta^{\mathcal{S}}(\Gamma) d_{(\min\{|\mathcal{S}|,\lfloor\Gamma\rfloor\}+1)}^{0,\mathcal{S}}\right\}.$$

Dividing the two bounds we obtain

$$\begin{aligned}\frac{SA}{PH} &\le \frac{1}{1+\frac{\alpha\Gamma}{n}} + \frac{\alpha \max_{\mathcal{S}\in\{\mathcal{I},\bar{\mathcal{I}}\}} \left\{\sum_{k=1}^{\min\{|\mathcal{S}|,\lfloor\Gamma\rfloor\}} d_{(k)}^{0,\mathcal{S}} + \Delta^{\mathcal{S}}(\Gamma) d_{(\min\{|\mathcal{S}|,\lfloor\Gamma\rfloor\}+1)}^{0,\mathcal{S}}\right\}}{(1+\frac{\alpha\Gamma}{n}) \max\left\{\sum_{i\in\mathcal{I}} d_i^0, \sum_{i\in[n]\setminus\mathcal{I}} d_i^0\right\}} \\ &\le \frac{n}{n+\Gamma\alpha} + \frac{\alpha n}{n+\alpha\Gamma} \max_{\mathcal{S}\in\{\mathcal{I},\bar{\mathcal{I}}\}} \frac{\sum_{k=1}^{\min\{|\mathcal{S}|,\lfloor\Gamma\rfloor\}} d_{(k)}^{0,\mathcal{S}} + \Delta^{\mathcal{S}}(\Gamma) d_{(\min\{|\mathcal{S}|,\lfloor\Gamma\rfloor\}+1)}^{0,\mathcal{S}}}{\sum_{i\in\mathcal{S}} d_i^0} \\ &= \frac{n}{n+\Gamma\alpha}\left(1+\alpha \max_{\mathcal{S}\in\{\mathcal{I},\bar{\mathcal{I}}\}} \frac{\sum_{k=1}^{\min\{|\mathcal{S}|,\lfloor\Gamma\rfloor\}} d_{(k)}^{0,\mathcal{S}} + \Delta^{\mathcal{S}}(\Gamma) d_{(\min\{|\mathcal{S}|,\lfloor\Gamma\rfloor\}+1)}^{0,\mathcal{S}}}{\sum_{i\in\mathcal{S}} d_i^0}\right). \end{aligned} \quad (9)$$

We now look at (A.9) for different values of Γ. Specifically, we look at the term inside the maximum. When $\Gamma = 0$, $\Delta^{\mathcal{S}}(\Gamma) = 0$, and thus it is equals zero and upper bound is equal to 1. When $\Gamma = n$ then again $\Delta^{\mathcal{S}}(\Gamma) = 0$, its numerator and denominator are equal, and so the upper bound is again equal to 1. □

Proof of Theorem 2. By construction we have that

$$PH \ge \max_{d\in U} \frac{\sum_{i=1}^n d_i}{2}.$$

Moreover, as explained above, for any RH methodology, we have the following upper bound

$$RH \le \max_{d\in U, j\in[n]} \frac{\sum_{i=1}^n d_i}{2} + \frac{d_j}{2}.$$

Using these bounds and Assumption 1, we have that

$$\begin{aligned}\frac{RH}{PH} &\le \frac{\max_{d\in U, j\in[n]} \left\{\sum_{i=1}^n \frac{d_i}{2} + \frac{d_j}{2}\right\}}{\max_{d\in U} \sum_{i=1}^n \frac{d_i}{2}} \\ &\le \frac{\max_{d\in U} \sum_{i=1}^n \frac{d_i}{2} + \frac{\bar{a}\min\{(1+\alpha),(1+\alpha\Gamma)\}}{2}}{\max_{d\in U} \sum_{i=1}^n \frac{d_i}{2}} \\ &\le 1 + \frac{\bar{a}\min\{(1+\alpha),(1+\alpha\Gamma)\}}{\underline{a}(n+\alpha\Gamma)}. \end{aligned}$$

□

Appendix B. Extensions for more than two machines

B1. The scheduler's DP

When $m > 2$, more than one task may still be in process when a given task completes. Thus, we extend the definition of a state to $S, F, D, I, \bar{D}$, where I is the set of indices of running tasks, and their respective processing duration thus far is represented by vector $\bar{D}$. Decisions can be made when there is an idle machine, *i.e.*, when one task completes ($|I| < m$,) and there are still tasks to schedule, ($|S| < n$). When $|S| = n$, all decisions have been already made.

Similar to $m = 2$ machines, at each decision point, the scheduler seeks the allocation policy that achieves the minimal worst-case remaining makespan. The recursive formulation is:

$$\begin{aligned}&T(S,F,D,I,\bar{D}) = \\ &\min_{k\notin S} \max\left\{\max_{\substack{d_k: d\in U_{S,F,D,I,\bar{D}},\\ d_k \le \min_{j\in I}(d_j-\bar{D}_j)}} d_k + T([S,k],[F,k],[D,d_k],I,(\bar{D}_j+d_k)_{j\in I}),\right. \\ &\left.\max_{\substack{(l,d_l):\\ d\in U_{S,F,D,I,\bar{D}}\\ l=\operatorname{argmin}_{j\in I}(d_j-\bar{D}_j)\\ d_k>d_l-\bar{D}_l}} d_l - \bar{D}_l + T([S,k],[F,l],[D,d_l],[I\setminus\{l\},k],[(\bar{D}_j)_{j\in I\setminus\{l\}},0]+d_l-\bar{D}_l)\right\}. \end{aligned} \quad (10)$$

The objective is to allocate the task k that minimizes the worst-case remaining makespan. For each k, there are two types of worst-case scenarios to consider. The first term in the first max of (10) relates to the possibility that under the worst-case scenario, k completes its processing before the completion of any of the other tasks already underway in I. In such a case, the next decision point is when task k completes and all tasks in I are still in processing (thus, the composition of I does not change). The second term describes the scenarios in which the next task completes before the newly scheduled task k. Thus, the time until the next decision point is $\min_{j\in I}(d_j - D_j)$, in which case the composition of set I changes to include k and to exclude the task that just finished processing.

The boundary case occurs when all tasks have been scheduled, yet some of them are still in processing; that is, $|S| = n$, $|F| < n$. In such a case, the remaining makespan amounts to the time until the completion of the last task among the tasks still being processed:

$$T(S,F,D,I,\bar{D}) = \max_{i\in I} \max_{d_i: d\in U_{S,F,D,I,\bar{D}}} d_i - \bar{D}_i.$$

B2. The adversary's DP

As in the two-machine formulation, the adversary makes her decisions immediately after a new task k has been scheduled in states $([S,k],F,D,I,\bar{D})$ where $k \notin S \equiv F \cup I$. The recursive formula for the worst-case remaining makespan T' is:

$$\begin{aligned}&T'([S,k],F,D,I,\bar{D}) = \max\left\{\max_{\substack{d_k: d\in U_{[S,k],F,D,I,\bar{D}},\\ d_k \le \min_{j\in I}(d_j-\bar{D}_j)}} d_k + \min_{l\notin S\cup\{k\}} T'([S,k,l],[F,k],[D,d_k],I,\bar{D}+d_k),0]),\right. \\ &\left.\max_{\substack{(i,d_i): d\in U_{[S,k],F,D,I,\bar{D}},\\ i\in\operatorname{argmin}_{j\in I}(d_j-\bar{D}_j),\\ d_k>d_i-\bar{D}_i}} d_i - \bar{D}_i + \min_{l\notin S\cup\{k\}} T'([S,k,l],[F,i],[D,d_i],[I\setminus\{i\},k],[(\bar{D}_j+d_i-\bar{D}_i)_{j\in I\setminus\{i\}},d_i-\bar{D}_i]\right\}. \end{aligned} \quad (11)$$

as long as $|S| \leqslant n-2$ (meaning that there are still more tasks to schedule). For the case $|S| = n-1$ and $|F| < n$, we have

$$T'([S,k],F,D,I,\bar{D}]) = \max\left\{\max_{i\in I} \max_{d_i: d\in U_{[S,k],F,D,I,\bar{D}}} d_i - \bar{D}_i, \max_{d_k: d\in U_{[S,k],F,D,I,\bar{D}}} d_k\right\}$$

which is the longest possible time until the last task completes.

B3. Computing E_σ and e_σ for two-machines MILO

Here, we present a formal algorithm that constructs the matrix E_σ and vector e_σ for the case of $m = 2$. The algorithm, shown in Algorithm 1 is based on iteratively constructing two binary vectors x and y for tasks assigned to machine 1 and 2, respectively.

B4. MILO formulation for m machines

We consider a setting with m machines and n tasks ($m < n$) and denote the set of all states $\sigma = (S,F)$ as $\mathcal{V}$. To decrease the state

Algorithm 1 Algorithm determining the E_σ and vector e_σ for $m = 2$ machines.

Initialize: $x = 0_{n\times 1}, y = 0_{n\times 1}, x_{S_1} = 1, y_{S_2} = 1$.
for $r = 1, \ldots, n-2$ **do**
  if $x_{F_r} = 1$ **then**
    $(E_\sigma)_{r,\cdot} = (x-y)^\top$
    Set $x_{S_{r+2}} = 1$
  else
    $(E_\sigma)_{f,\cdot} = (y-x)^\top$
    Set $y_{S_{r+2}} = 1$
  end if
end for
if $x_{F_{n-1}} = 1$ **then**
  $(E_\sigma)_{n-1,\cdot} = (x-y)^\top$
  $e_\sigma = y$
else
  $(E_\sigma)_{n-1,\cdot} = (y-x)^\top$
  $e_\sigma = x$
end if

Table 7
The sets of states for the scheduler $\mathcal{D}$ (left) and the adversary $\mathcal{N}$ (right).

S-length	***F***-length	***S***-length	***F***-length
0	0	m	0
m	1	$m+1$	1
$m+1$	2	$\vdots$	$\vdots$
$\vdots$	$\vdots$	$n-1$	$n-m-1$
$n-2$	$n-m-1$	n	n

space of the scenario tree, certain states are discarded (see also the example described in Section 6.1.1). In particular,

- we omit states where the length of S is smaller than m, since they correspond to an initialization of the system in which at least one machine is idle, thus additional tasks must be scheduled immediately. Hence, the states that follow the initial state are characterized by $|S| = m$ (all machines are busy).
- Similarly, the last decision to be made is the one after which a single task remains to be scheduled. Such a decision is made at a state where the length of S is $n-2$ and the length of F is $n-m-1$. Afterwards, the adversary decides which tasks to complete and in which order. Hence, the states that follow states with $|S| = n-1$ and $|F| = n-m-1$ are the end states in which $|S| = |F| = n$.

Given this, we consider the state space $\mathcal{V}$ consisting of the sets of the scheduler states $\mathcal{D}$ and adversary states $\mathcal{N}$ (Table 7).

With these formulations, we can define the MILO formulation in the same way as in Section 6.

Appendix C. Complexity of the MILO formulation (P)

In this appendix, we study the complexity of the MILO formulation (P) needed to solve the scheduling problem. The scheduler's first decision concerns which m tasks out of the n to schedule, followed by the adversary's decision regarding which of the m tasks to end first. Next, the scheduler has to choose one of the remaining tasks to schedule to the newly idle machine. This sequence repeats itself until the scheduler has no more tasks to schedule, in which case the adversary has to decide on the termination order of the m in-process tasks. Thus, the number of end states in the scenario tree of the mixed-integer formulation is given by

$$\begin{aligned}|\mathcal{L}| &= \binom{n}{m} m(n-m)m(n-m-1)\cdot\ldots\cdot m\cdot 2\cdot m! \\ &= \binom{n}{m} m^{n-m-1}(n-m)!m! \\ &= n!m^{n-m-1}.\end{aligned}$$

The number of scheduler nodes is:

$$\begin{aligned}|\mathcal{D}| &= 1 + \binom{n}{m}m + \binom{n}{m}m(n-m)m + \ldots \\ &\quad + \binom{n}{m}m(n-m)m(n-m-1)m\ldots 3\cdot m \\ &= 1 + \sum_{i=0}^{n-m-2}\binom{n}{m}\frac{(n-m)!}{(n-m-i)!}m^{i+1} \\ &= 1 + \frac{n!}{m!}\sum_{i=1}^{n-m-1}\frac{m^i}{(n-m-i+1)!}.\end{aligned}$$

The number of binary variables required for the nature nodes' maximum reformulation as MILO constraints is, therefore, $|\mathcal{D}| + |\mathcal{L}| - 1$. Thus, the MILO formulation requires an exponential number of binary variables.

Consequently, the MILO problem (P) is not scalable. In the next section we conduct a numerical study with small problems to compare the MILO formulation (*i.e.*, AR) to other alternatives (*e.g.*, SA, SL), which will enable us to evaluate the possible benefits of applying an adaptive or at least partially adaptive policy for which finding solutions is less computationally demanding.

Appendix D. MILO formulation for the SA policy

The following optimization problem is the MILO problem to be solved to obtain the optimal SA policy.

$$\begin{aligned}\min_{t,x\in\{0,1\}^n}\ & t \\ \text{s.t. } & t \geq x^\top d && \forall d\in U \\ & t \geq (\mathbf{e}-x)^\top d && \forall d\in U.\end{aligned}$$

If U is finite, the constraints can be enforced by enumerating each of them over the elements of U. If U is convex, they can be enforced using standard RO techniques of dualizing the inner maximization problem in $\sup_{d\in U} x^\top d$ and $\sup_{d\in U}(\mathbf{e}-x)^\top d$.

Appendix E. Column-and-constraint generation algorithm for the 2SSA

In this section, we will discuss how to solve the two-stage problem (7), which computes the worst-case makespan of 2SSA for the choice of i, j as the tasks initially scheduled. We focus on the case where $d_i < d_j$ (first term in the outer max in (20)). The second case, in which $d_i \geq d_j$ is solved similarly. We solve this problem via a column-and-constraint generating procedure, similar to Zeng & Zhao (2013), that is given in Algorithm 2. We note that of U is convex the condition $d_i < d_j$ can be replaced by $d_i \leq d_j$ (which we use for the rest of the section), while for discrete U, the original condition needs to be maintained.

The algorithm, consists of iterating between solving two problems. The master problem (12), corresponds to the first-stage adversarial problem and provides an upper bound $\bar{t}^k$ on the worst-case makespan, and the scheduler subproblem (13) is associated with the scheduler best response to the adversarial decision and provides a lower bound $\bar{v}^k$ on the worst-case makespan.

Algorithm 2 Adversarial CCG.

Initialization: Set $\mathcal{X} = \{x^1\}$for some $x^1 \in \{0,1\}^n$ satisfying $x^1_i = 1$, and $x^1_j = 0$.
For $k = 1, 2, \ldots,$:

(1) Solve the master problem, given by

$$\bar{t}^k = \max_{t, \tilde{d}_i, t^k, d^k \in U, z^k \in \{0,1\}} t \tag{12}$$
$$\text{s.t. } t \le t^k \qquad \forall k = 1 \ldots, |\mathcal{X}|$$
$$t^k \le x^{k\top} d^k + M(1 - z^k) \qquad \forall k = 1 \ldots, |\mathcal{X}|$$
$$t^k \le (\mathbf{e} - x^k)^\top d^k + M z^k \qquad \forall k = 1 \ldots, |\mathcal{X}|$$
$$d^k_i = \tilde{d}_i \qquad \forall k = 1 \ldots, |\mathcal{X}|$$
$$d^k_i \le d^k_j \qquad \forall k = 1 \ldots, |\mathcal{X}|.$$

Denote by $\bar{d}_i$ the optimal value of $\tilde{d}_i$.

(2) Solve the scheduler subproblem for the choice of $\bar{d}_i$:

$$\bar{v}^k = \min_{x \in \{0,1\}^n, v} v \tag{13}$$
$$\text{s.t. } v \ge d^\top x \qquad \forall d \in U : d_i = \tilde{d}_i, d_i \le d_j$$
$$v \ge (\mathbf{e} - x)^\top d \qquad \forall d \in U : d_i = \tilde{d}_i, d_i \le d_j$$
$$x_i = 1,\ x_j = 0.$$

Denote its optimal solution by x^{k+1}.

(3) If $\bar{v}^k < \bar{t}^k$ set $\mathcal{X} := \mathcal{X} \cup \{x^{k+1}\}$. Otherwise, finish and return worst-case makespan $\bar{t}^k$.

In the master problem (12), the adversary aims to find the first-stage decision $\tilde{d}_i$ that maximizes the worst-case makespan, for all schedules x^k in a specified set of static allocation $\mathcal{X}$. Specifically, for each $x^k \in \mathcal{X}$, a different second stage duration $d^k \in U$ vector can be chosen, as long as it satisfies $d_j \ge d_i$ and $d_i = \tilde{d}_i$. The auxilary indicator z^k, identifies which machine finished last, and thus determined the makespan. Thus, t^k is an upper bound on the worst-case makespan that can be obtained, since it only considers a subset of scheduler responses.

In the scheduler subproblem (13), the scheduler attempts to find a static schedule $x^{k+1} \in \{0,1\}^n$, $x^{k+1}_i = 1$, $x^{k+1}_j = 0$ which solves the robust problem of minimizing the worst-case makespan for any duration realization $d \in U$ satisfying $d_i \le d_j$ and $d_i = \tilde{d}_i$. Thus, the optimal value $\bar{v}^k$ provides a lower bound for the worst-case makespan. If $\bar{v}^k < \bar{t}^k$ it means that the schedule x^{k+1} provides a response to $\bar{d}_i$ which improves the makespan. Thus, x^{k+1} is added to $\mathcal{X}$ and the master problems needs to be resolved. If $\bar{v}^k = \bar{t}^k$ then the worst-case makespan is obtained. Importantly, subproblem (13) is a standard linear one-stage robust problem. Specifically, if U is a convex set the first two constraint in the problem can be transformed from semi-infinite constraints to their robust variant by dualization, and if U is a discrete set the problem can be solved through enumeration or a feasibility cut method.

Appendix F. Box uncertainty set sampled results

The results for this uncertainty set are presented in Table 8 and Fig. 7. For SA, SL, and 2SSA, the percentages of different first-stage decisions, compared to AR, are 32%, 4%, and 2% respectively.

The AR promised makespan was the shortest, with 2SSA being nearly the same, SL very close behind (longer by 0.17%), and SA trailing behind with a longer makespan by 5.3%. The upper bound on the "price" that a risk-averse scheduler, who commits to the promised makespan, is expected to pay upfront with respect to a PH policy (as indicated by the Promised to max PH measure) was very small for AR (0.0%), 2SSA (0.1%) SL (0.2%), and much higher for SA (5.8%).

Table 8
Performance measure results for type II uncertainty sets.

Short name	AR	2SSA	SL	SA	PH
Suboptimal initial	-	2%	4%	32%	-
Promised makespan	7.784(1.430)	7.788(1.428)	7.798(1.429)	8.217(1.470)	-
Max makespan	7.784(1.430)	7.787(1.429)	7.791(1.429)	7.913(1.425)	7.783(1.430)
Makespan	6.105(1.493)	6.097(1.496)	6.098(1.495)	6.139(1.513)	5.966(1.493)

Makespan values are presented with their standard deviations in parentheses.

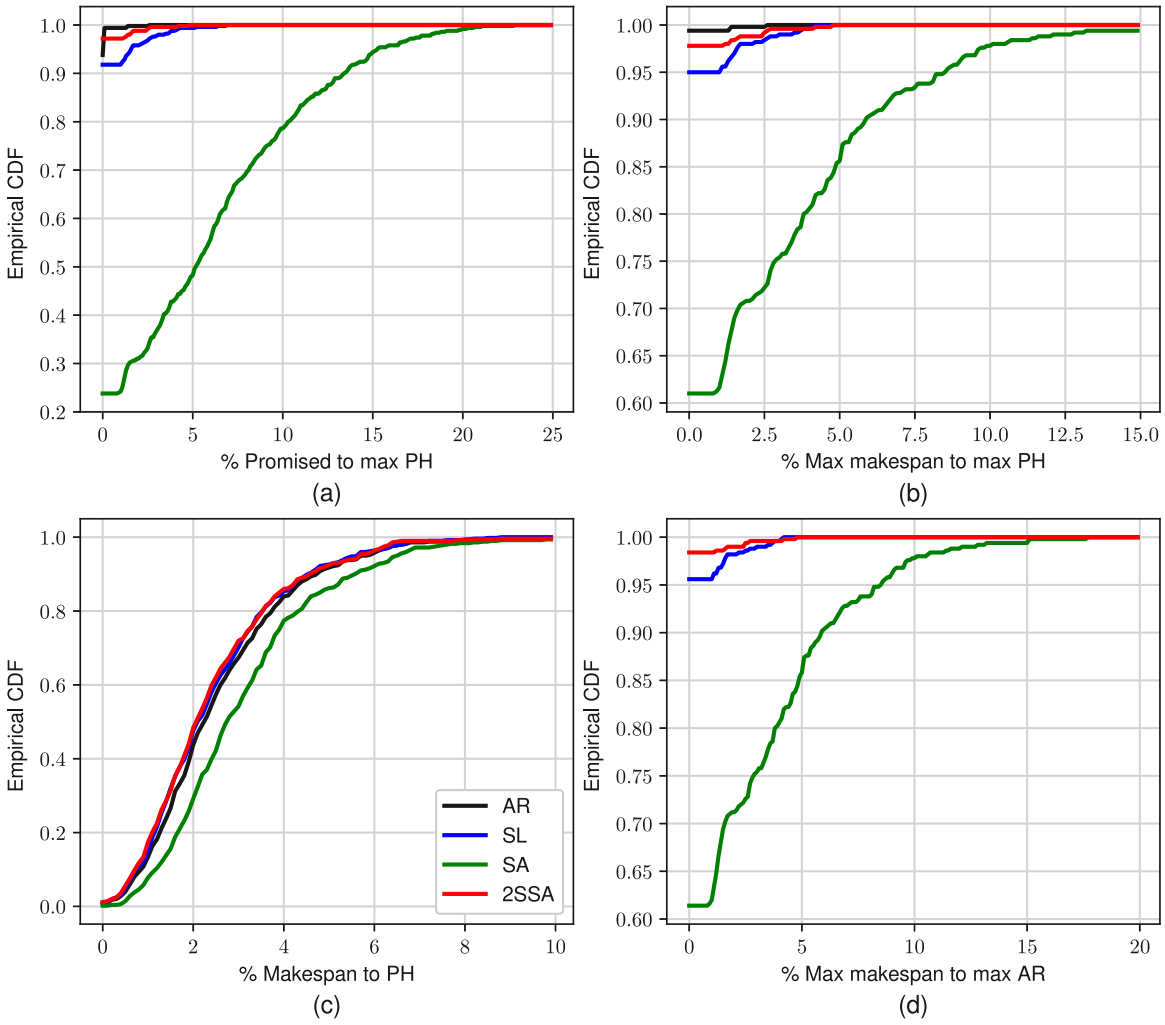

Fig. 7. Empirical cumulative distribution functions (over instances) of the last four performance measures of Table 5 for Type II uncertainty set.

References

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

Balin, S. (2011). Parallel machine scheduling with fuzzy processing times using a robust genetic algorithm and simulation. *Information Sciences, 181*(17), 3551–3569.

Balouka, N., & Cohen, I. (2019). A robust optimization approach for the multi-mode resource-constrained project scheduling problem. *European Journal of Operational Research*.

Beck, J. C., & Wilson, N. (2007). Proactive algorithms for job shop scheduling with probabilistic durations. *Journal of Artificial Intelligence Research, 28*, 183–232.

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

Ben-Tal, A., Goryashko, A., Guslitzer, E., & Nemirovski, A. (2004). Adjustable robust solutions of uncertain linear programs. *Mathematical Programming, 99*(2), 351–376.

Bertsimas, D., & Dunning, I. (2016). Multistage robust mixed-integer optimization with adaptive partitions. *Operations Research, 64*(4), 980–998.

Bertsimas, D., Iancu, D. A., & Parrilo, P. A. (2010). Optimality of affine policies in multistage robust optimization. *Mathematics of Operations Research, 35*(2), 363–394.

Bertsimas, D., Litvinov, E., Sun, X. A., Zhao, J., & Zheng, T. (2012). Adaptive robust optimization for the security constrained unit commitment problem. *IEEE Transactions on Power Systems, 28*(1), 52–63.

Bertsimas, D., & Sim, M. (2004). The price of robustness. *Operations Research, 52*(1), 35–53.

Bougeret, M., Jansen, K., Poss, M., & Rohwedder, L. (2021). Approximation results for makespan minimization with budgeted uncertainty. *Theory of Computing Systems, 65(6)*, 903–915.

Bougeret, M., Pessoa, A. A., & Poss, M. (2019). Robust scheduling with budgeted uncertainty. *Discrete Applied Mathematics, 261*, 93–107.

Cai, X., Wu, X., & Zhou, X. (2014). *Optimal stochastic scheduling*. Springer.

Cohen, I., Golany, B., & Shtub, A. (2007). The stochastic time–cost tradeoff problem: A robust optimization approach. *Networks: An International Journal, 49*(2), 175–188.

Conde, E. (2014). A MIP formulation for the minmax regret total completion time in scheduling with unrelated parallel machines. *Optimization Letters, 8*(4), 1577–1589.

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

Davari, M., & Demeulemeester, E. (2019). The proactive and reactive resource-constrained project scheduling problem. *Journal of Scheduling, 22*(2), 211–237.

Georghiou, A., Kuhn, D., & Wiesemann, W. (2019). The decision rule approach to optimization under uncertainty: Methodology and applications. *Computational Management Science, 16*(4), 545–576.

Gupta, J. N. D., & Ruiz-Torres, A. J. (2001). A LISTFIT heuristic for minimizing makespan on identical parallel machines. *Production Planning & Control, 12*(1), 28–36.

Hanasusanto, G. A., Kuhn, D., & Wiesemann, W. (2015). *K*-adaptability in two-stage robust binary programming. *Operations Research, 63*(4), 877–891.

Iancu, D. A., & Trichakis, N. (2014). Pareto efficiency in robust optimization. *Management Science, 60*(1), 130–147.

Lin, J., & Ng, T. S. (2011). Robust multi-market newsvendor models with interval demand data. *European Journal of Operational Research, 212*(2), 361–373.

Liu, M., Liu, X., Chu, F., Zheng, F., & Chu, C. (2019). Service-oriented robust parallel machine scheduling. *International Journal of Production Research, 57*(12), 3814–3830.

Möhring, R. H., Schulz, A. S., & Uetz, M. (1999). Approximation in stochastic scheduling: The power of LP-based priority policies. *Journal of the ACM (JACM), 46*(6), 924–942.

Pinedo, M. (2002). *Scheduling: Theory, algorithms and systems*. Prentice Hall.

Postek, K., Den Hertog, D., Kind, J., & Pustjens, C. (2019). Adjustable robust strategies for flood protection. *Omega, 82*, 142–154.
Postek, K., & Hertog, D. d. (2016). Multistage adjustable robust mixed-integer optimization via iterative splitting of the uncertainty set. *INFORMS Journal on Computing, 28*(3), 553–574.
Ranjbar, M., Davari, M., & Leus, R. (2012). Two branch-and-bound algorithms for the robust parallel machine scheduling problem. *Computers & Operations Research, 39*(7), 1652–1660.
de Ruiter, F., Brekelmans, R., & den Hertog, D. (2016). The impact of the existence of multiple adjustable robust solutions. *Mathematical Programming, 160*(1–2), 531–545.
Tighazoui, A., Sauvey, C., & Sauer, N. (2021). Predictive-reactive strategy for identical parallel machine rescheduling. *Computers & Operations Research*, 105372.
Wang, S., & Cui, W. (2020). Approximation algorithms for the min-max regret identical parallel machine scheduling problem with outsourcing and uncertain processing time. *International Journal of Production Research*, 1–14.
Weber, R. R. (1982). Scheduling jobs with stochastic processing requirements on parallel machines to minimize makespan or flowtime. *Journal of Applied Probability*, 167–182.
Xu, X., Cui, W., Lin, J., & Qian, Y. (2013). Robust makespan minimisation in identical parallel machine scheduling problem with interval data. *International Journal of Production Research, 51*(12), 3532–3548.
Xu, X., Lin, J., & Cui, W. (2014). Hedge against total flow time uncertainty of the uniform parallel machine scheduling problem with interval data. *International Journal of Production Research, 52*(19), 5611–5625.
Yanıkoğlu, I., Gorissen, B. L., & den Hertog, D. (2019). A survey of adjustable robust optimization. *European Journal of Operational Research, 277*(3), 799–813.
Yeh, W., Lai, P., Lee, W., & Chuang, M. (2014). Parallel-machine scheduling to minimize makespan with fuzzy processing times and learning effects. *Information Sciences, 269*, 142–158.
Zeng, B., & Zhao, L. (2013). Solving two-stage robust optimization problems using a column-and-constraint generation method. *Operations Research Letters, 41*(5), 457–461.