Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Control Engineering

Doctoral Thesis

Scheduling with uncertain processing times given by empirical distributions

Antonín Novák

August 2022

Scheduling with uncertain processing times given by empirical distributions

by

Antonín Novák

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Control Engineering

Doctoral Thesis

Supervisor: prof. Dr. Ing. Zdeněk Hanzálek
Co-supervisor: doc. Ing. Přemysl Šůcha, Ph.D.
Ph.D. programme: Electrical Engineering and Information Technology
Branch of study: Control Engineering and Robotics
Submission date: August 2022

Antonín Novák: Scheduling with uncertain processing times given by empirical distributions, Ph.D. Thesis, Czech Technical University in Prague, Faculty of Electrical Engineering, Department of Control Engineering, August 2022, Prague.

Acknowledgments

I would like to express my gratitude to my supervisor Zdenek Hanzalek, who inspired me to pursue the field of mathematical optimization and optimization algorithms. I believe that I have learned a lot from him during the years, both professionally and personally. For that, I respect him greatly both as a researcher and as a person. Next, I thank Premysl Sucha, with whom I have worked the most. I always enjoyed our discussions, and I want to thank him for his will to share with me the joy when something worked out and the disappointment when something did not.

During this journey, I was lucky enough to be surrounded by amazing friends and colleagues: István Módos, Marek Vlk, Jan Dvořák, Libor Bukata, Aasem Ahmad, Roman Václavík, Anna Minaeva, Michal Sojka, Theodor Krocan, Ondřej Benedikt, Jaroslav Klapálek, Oxana Kovbasjukova, Joel Matějka, Pavla Svítilová, Vilém Heinz, Matěj Novotný and many others. I will never forget the good times we had together solving problems, traveling to conferences, teaching courses, and attending all different kinds of social activities. Moreover, I thank our climbing crew for the time we spent together and for helping me to find the passion for the sport.

Last but not least, I thank my parents, Antonín and Jana, and my brother Tomáš for their support during my whole life. Finally, I am grateful to my wife Bára, who always has been supportive and never doubted about the path I have chosen. My successes are also yours.

Finally, I would like to acknowledge all the agencies and projects that funded my research. Namely, the work was supported by Czech Technical University under SGS16/233/OHK3/3T/13 and SGS19/175/OHK3/3T/13, by project CAK 3: Centre for Applied Cybernetics, funded by the Technology Agency of the Czech Republic under TE01020197, SALTT—Scheduling Algorithms for Time-Triggered Systems, U.S. Office of Naval Research Global under N62909-15-1-N094, Factory of Future, funded by EU and the Ministry of Industry and Trade of the Czech Republic, Connected Motor Starter, funded by EU and the Ministry of Industry and Trade of the Czech Republic, Arrowhead tools, under by ECSEL Ministry of Education of the Czech Republic under 826452, by H2020 THERMAC project, funded by European Commission under 832011, National Competence Center—Cybernetics and Artificial Intelligence, funded by the Technology Agency of the Czech Republic under TN01000024 and by the Grant Agency of the Czech Republic under the Project GACR 22-31670S.

Antonín Novák
Prague, August 2022

Declaration

This doctoral thesis is submitted in partial fulfillment of the requirements for the degree of doctor (Ph.D.). The work submitted in this thesis is the result of my own investigation, except where otherwise stated. I declare that I worked out this thesis independently and I quoted all used sources of information in accord with Methodical instructions about ethical principles for writing academic thesis. Moreover, I declare that it has not already been accepted for any degree and is also not being concurrently submitted for any other degree.

Antonín Novák
Prague, August 2022

Abstract

The processing time of a job is arguably one of the most important parameters of scheduling problems, but often it is also the one that is very difficult to obtain—either it is not known, costly to measure, or it does not have a deterministic nature. Classical deterministic scheduling methods that replace the missing information with, e.g., single-point estimates may produce solutions that are fragile and susceptible to unexpected realizations of the parameters. Therefore, the optimization approaches that can deal with imprecise knowledge of the problem parameters must be used in cases where protection against unfavorable realizations is vital. The important essence in the design of such optimization algorithms is the choice of the uncertainty model and the approach to mitigate its effects. Is the estimation of, e.g., the mean and variance enough? Can we assume that, at most, a certain number of parameters will deviate from their estimates? Do we need to have a risk-averse objective, and what is the price for that? These are essential considerations since too simplistic models can lead to conservative and inefficient solutions or may fail to protect from uncertainty at all. Luckily, the advancements in collecting large datasets for many aspects of industrial processes allow us to apply powerful data-driven approaches to scheduling under uncertainty.

In this work, we study different means of integrating distributional knowledge of processing times into scheduling problems. The part of the proposed methods in this thesis takes inspiration from the machine learning field, as the techniques used there are scalable and deal with uncertain parameters also. As an example, we have used in-sample data to estimate the parameters of an ambiguity set to obtain a distributionally robust solution via norm regularization techniques which greatly reduces the run time of the algorithm. Another way of improving the scalability of the scheduling algorithms lies in the choice of a suitable model of uncertainty. For example, we have proposed a way to model jobs with uncertain processing times with an abstraction called F-shape, which discretizes the cumulative distribution function of processing times. The resulting problem formulation is essentially a packing problem that can be solved efficiently in practice with the advantage that the probabilistic guarantees of the resulting schedules can be verified within the framework of Bayesian networks. Finally, we study a setting where the distribution of processing times can be approximated by a normal distribution. The full distributional knowledge is utilized in a β-robust formulation that maximizes the probability that all jobs are completed before a given common due date. For the researched scheduling problems with uncertain processing times utilizing distributional knowledge, we have obtained their complexity characterizations, and we have developed scalable algorithms that provide high-quality solutions for instances with hundreds of jobs.

Keywords: processing time uncertainty, robust scheduling, mixed-criticality scheduling, distributionally robust optimization

Abstrakt

Doba zpracování úlohy je jedním z nejdůležitějších parametrů rozvrhovacích problémů, ovšem je také jedním z těch, které je velmi obtížné získat—obvykle není znám, je drahý na měření nebo nemá deterministickou povahu. Klasické metody deterministického rozvrhování, které nahrazují chybějící informaci například bodovými odhady, mohou produkovat výsledy které jsou nestabilní a obecně náchylné na nečekané realizace hodnot parametrů. Kvůli tomu je potřeba v prostředí s nepřesnou znalostí parametrů použít optimalizační přístupy, které jsou schopny vypořádat se s neurčitostí parametrů. Důžitá část v návrhu takových algoritmů je volba modelu neurčitosti a přístupu k potlačení jeho důsledků. Je například odhad střední hodnoty a rozptylu dostatečný? Můžeme předpokládat že pouze jistý počet parametrů se při realizaci odkloní od své odhadované hodnoty? Potřebujeme optimalizovat kriteriální funkci zohlednující riziko, a pokud ano, jaká je za to cena? Toto jsou zásadní otázky, protože příliš jednoduché modely vedou ke konzervativním a neefektivním řešením nebo naopak dokonce mohou zcela selhat při ochraně proti neurčitosti. Díky pokrokům technologií, umožnující sběr velikých souborů dat různých aspektů průmyslových procesů, můžeme zmíněné výzvy adresovat daty-řízenými přístupy k rozvrhování za neurčitosti.

V této práci studujeme různé způsoby včlenění distribuční znalosti časů zpracování do rozvrhovacích problémů. Metody navržené v této práci jsou z části inspirovány technikami ze strojového učení, kde známe škálující algoritmy a přičemž také čelíme neurčinosti parametrů. Například jsme použili navzorkovaná data k odhadu parametrů množiny nejednoznačnosti ke konstrukci distribučně-robustního řešení metodami regularizace normami, což přineslo velké urychlení běhů algoritmů. Další cesta ve vylepšování výkonu rozvrhovacích algoritmů se nachází ve vhodné volbě modelu neurčitosti. Zde jsme navrhli metodu jak modelovat úlohy s neurčitou dobou zpracování pomocí abstrakce takzvaných F-tvarých úloh která diskretizuje kumulativní distribuční funkci časů zpracování. Výsledná formulace problému je v podstatě balící problém, který je velmi dobře řešitelný v praxi s výhodou, že pravděpodobností garance výsledných rozvrhů jsou verifikovatelné pomocí Bayesovských sítí. Na závěr studujeme situaci kdy distribuční funkce dob zpracování lze nahradit normálním rozdělením. Úplná znalost distribuční funkce je využita v β-robustní formulaci kdy se maximalizuje pravděpodobnost, že všechny úlohy jsou dokončeny před společným termínem. Zmíněné problémy jsme charakterizovali z hlediska výpočetní složitosti a navrhli pro ně škálující algoritmy které poskytují vysoce kvalitní řešení pro instance se stovkami úloh.

Klíčová slova: neurčitost doby zpracování, robustní rozvrhování, smíšeně-kritické rozvrhování, distribučně robustní optimalizace

Goals and Objectives

This thesis focuses on scheduling problems and algorithms with uncertain processing times utilizing the full distributional knowledge. The main goals of the work are:

1. Study the related scheduling literature in robust, stochastic, and distributionally robust optimization fields. Identify new challenges, promising approaches, and possible improvements to existing modeling methods, problem formulations, and algorithms.

2. Develop formal descriptions of the deduced problems. Formulate the uncertainty models, constraints, and objective functions considering the distributional knowledge.

3. Propose heuristic and exact algorithms that compute robust schedules with respect to the processing time uncertainty.

4. Benchmark the developed algorithms by numerical experiments. Discuss the obtained results from the perspective of quality and time complexity. Demonstrate the scalability of the proposed approaches and discuss the robustness of their solutions.

Contents

Chapter 1

Introduction

The processing time of a job is arguably one of the most important parameters of scheduling problems, but often it is also the one that is very difficult to obtain. Either it is not known, is costly to measure, or it does not have a deterministic nature. Scheduling problems where the uncertainty of parameters cannot be disregarded belong to the robust optimization field. The goal is to construct schedules that account for uncertainty in advance. This approach belongs to the so-called *proactive* methods that absorb the effects of uncertainty by the design of the schedule. This is different from *reactive* approaches that aim to repair the schedule during the online execution. When we design algorithms that deal with uncertainty, we face the following two related questions. Namely, *how do we describe such uncertainty* and *what does it mean to account for it* in advance? The goal of the thesis is to develop scalable optimization algorithms for scheduling problems with uncertain processing times given by empirical distributions.

In our view, and in the view of others [134, 69], the fields of stochastic and distributionally robust optimization are related to the machine learning discipline. In statistical machine learning, one uses a collected empirical data sample (i.e., in-sample) to estimate the parameters of a model such that it would perform well on out-of-sample data. This is essentially also a form of decision-making under uncertainty. Of course, it appears that training a machine learning model (typically a continuous problem) is an easier job than solving a scheduling problem, which is discrete by its nature. However, many machine learning models have a discrete structure as well, e.g., construction of decision trees or models with $\|\cdot\|_0$ (sparsity) regularization [118]. Despite that, it is apparent that machine learning algorithms are nowadays successful, massively scalable, and utilize the knowledge of the distribution of the data. In our view, scheduling algorithms for stochastic problems are still to catch up. On the other hand, machine learning algorithms might benefit [100] from a more systematic treatment of robustness and incomplete information that is nowadays well-established in stochastic optimization. Thus, it seems to us that looking at related techniques and approaches might cross-fertilize both fields.

The first group of methods proposed in this work is inspired by techniques used in machine learning. For example, in Chapter 2, we study the effect of different objective regularization techniques to obtain an efficiently solvable approximation of the original distributional robust optimization model for the scheduling problem with the total flow time minimization. As another example, in Chapter 4, we utilize the framework of Bayesian networks to compute the execution probability of jobs with uncertain processing time in a setting where we allow each job to be replicated, i.e., scheduled more than once.

The other group of methods in this work is based on the idea that often, a suitable problem formulation is already a part of its successful solution. For example, in Chapter 3, we proposed how to discretize a cumulative distribution function of processing times into a finite number of values depending on the criticality of

the job. This transformation reduces the problem with uncertain processing times to a deterministic scheduling problem with alternatives. Finally, the formulation resembles an interesting packing problem that is efficiently solvable in practice. A similar idea is investigated in Chapter 5, where we deal with the problem where the processing time can be approximated by a normal distribution. Again, the resulting formulation resembles a kind of packing problem. Adapting solution concepts for the Knapsack problem with stochastic items and the study of the two-machine case of the problem allowed us to develop scalable scheduling algorithms that can solve instances with hundreds of jobs.

1.1 Related work

The state-of-the-art optimization approaches differ in their strategies for tackling uncertainty. In the following section, we discuss solution concepts and techniques related to our work. We consider this as a high-level overview of the area; therefore, the list of works should not be understated as complete by any means. Instead, we aim to introduce arguably the most influential works and ideas. A more detailed discussion of related work for each problem is presented in its respective chapter. Nevertheless, arguably the most commonly used paradigms are known as robust optimization (RO), stochastic programming (SP), and distributionally robust optimization (DRO).

Robust optimization RO assumes that uncertain parameters belong to a given uncertainty set, and the goal is to optimize for the possible worst-case realization or to ensure that constraints hold for all realizations of parameters [166]. It is apparent that such formalism may result in overly conservative and pessimistic solutions, thus being costly. To counteract this, notable development in this area by Ben-Tal and Nemirovski [24, 26, 25], Bertsimas, and many others led to the development of an uncertainty set centered around an in-sample estimate of parameters with controllable uncertainty budget [31]. These controllable parameters allow different trade-offs between the robustness and quality of solutions. In recent years, we have seen the development of methods for the design of uncertainty sets that go well beyond budgeted sets toward more data-driven approaches [30]. Uncertainty sets are often designed to have a polyhedral or conic representation, thus, leading to models efficiently solvable in practice. For a more detailed treatment of the history, theory, and applications of RO, we refer the reader to many excellent books and surveys, e.g., [33, 22, 29, 167].

Stochastic programming Another popular theoretical framework is SP which typically optimizes the expected value of the objective function with respect to a known probabilistic distribution over parameters values. The additional information about parameter distribution may allow obtaining solutions that are less conservative than RO solutions. The SP formulations of the problem can also be extended to incorporate so-called *stages*. The stages can be used to model the decision-making process. Depending on how uncertainties are unfolded during the execution, the decision-maker can react to them with different decisions. Depending on how

the problem is formulated, this model leads to the notion of single, two-stage, or multistage optimization [140]. For further interesting connections between SP and sequential decision making, we refer the reader to [132, 131].

Depending on the uncertainty model and the form of the objective function, solving the exact formulation of an SP formulation is almost always intractable [18]. However, when the distribution is supported on a finite set (i.e., scenarios) that is not very large, the formulation can be solved by translating it into a finite-sized mathematical program whose structure can be exploited by decompositions or by cutting plane methods [149]. Alternatively, the problem needs to be solved approximately with methods such as Sample Average Approximation (SAA) [154]. There, one collects a finite set of samples and creates a surrogate formulation of the problem, where, e.g., the expectation of an objective function is replaced with the sample mean of the objective with the individual realizations. Such a method can be used to approximate solutions to problems with uncertain parameters in terms of the expected utility while having probabilistic approximation guarantees for its out-of-sample performance [156, 105, 18].

Despite SP being an attractive and natural tool for decision-making under uncertainty, it is well-known that SP formulations have several limitations. The first comes from the fact that the expectation is only a measure of centrality; thus, it does not reflect, e.g., the solution variance. Therefore, even though the performance fluctuations cancel each other out in the expectation, it does not protect against sudden disturbances [137] that may not be desired, e.g., in safety-critical systems. Naturally, this was well understood in the past, and several methods were developed to counteract this. Perhaps one of the most intuitive solutions is to include standard deviation of the objective as a safety margin [111, 156] or to replace expectation with different quantities, such as *conditional value-at-risk* (CVaR) [138, 45], or the introduction of *chance constraints* [47].

The other problematic scenario occurs when the out-of-sample distribution differs from the in-sample one. These are the results of the distribution shift or simply the cases of insufficient knowledge of the target distribution. In such scenarios, the decision-making done by an SP model may lead to suboptimal results.

Interestingly, these raised issues and their solutions are, in fact, closely connected and have motivated the development of *minimax stochastic optimization* [148], which connects RO and SP. Nowadays, it is commonly referred to as distributionally robust optimization (DRO).

Distributionally robust optimization DRO refers to problems where one minimizes the worst-case expected objective over a set of possible distributions, i.e., the so-called *ambiguity set*. An alternative view on distributionally robust formulations is offered via the *coherent risk measures* [137]. In contrast to the modeling of the ambiguity set in the inner expectation problem, one can examine the properties of the risk measure (objective function) of the formulation. If it satisfies a certain set of natural properties, then one can show that the optimal solution to such a problem will be distributionally robust in a certain sense [134].

DRO solution concept has been around due to Scarf [148] already in 1958, by Dupačová [62, 63] in 1966, and later revisited by many more people. Recently, a new wave of interest in DRO was sparked, namely by advancements of mathematical

programming solvers and by novel formulations of expressive ambiguity sets that can incorporate a large amount of information. A DRO problem is typically reformulated to a computationally tractable deterministic mathematical programming problem, whose complexity depends on the used formulation of the ambiguity set and the structure of the objective function. The ambiguity sets in the literature are often categorized into two main groups: (i) moment-based and (ii) discrepancy-based ambiguity sets [134].

The moment-based ambiguity sets aim to constrain moments of the distributions (e.g., the expected value and the variance); the ambiguity set then contains all distributions with the specific values of the moments. Despite its seeming simplicity, for certain objective functions, it already leads to intractable formulations. For example, in the case of a single-machine scheduling problem with the total tardiness, just the inner worst-case expectation problem alone leads to an exponentially-sized semidefinite program (SDP), even when just the exected value and covariance are constrained [158]. Besides computability, another consideration is the expressive power of the moment-based ambiguity sets. More specifically, what kind of families of distributions can it model, but more importantly, what are the properties of the extremal distributions that are attained in the inner worst-case expectation problem? A known shortcoming of the moment-based ambiguity sets is that the worst-case realization of distributions might have unrealistic form [180]. For example, it is known that under mild assumptions on the objective function, its worst-case expectation is attained at the distribution having support in at most $m+1$ points if m moments are constrained [156, 23]. Therefore, it is possible that a decision-maker might optimize against worst-case distribution that even contradicts the empirical sample that was collected for the design of the ambiguity set in the first place. To the increase expressivness of moment-based ambiguity sets, one can use *generalized moment functions* [153] or construct *nested confidence sets* [181] to incorporate more data-driven information into the robust formulation.

The discrepancy-based ambiguity sets use an estimate of a nominal distribution and a discrepancy measure to model the set of all distributions with the prescribed discrepancy radius centered around the nominal distribution [134]. Perhaps the most notable discrepancy measures are ϕ-divergence [27, 17], Wasserstein distance [70, 68, 100], and likelihood measure [180]. Their advantage over moment-based ambiguity sets lies in more realistic modeling of the ambiguous distribution, thus leading to less conservative solutions.

Despite the growing popularity of discrepancy-based ambiguity sets in the continuous optimization problems, their adaptation in the scheduling domain remains relatively limited, see, e.g., [49, 179, 153, 127] with a few notable exceptions, such as [157]. Hence, there is a fairly obvious yet understandable gap between the development of theoretical tools and their application in the scheduling domain.

1.2 Contributions

The methods proposed in this work have contributed to the scheduling literature dealing with processing time uncertainty with the following. In the area of robust scheduling, we have proposed to treat the discretization of the cumulative distri-

bution function of the processing time as an *F-shaped* job. This abstraction is inspired by the so-called Mixed-Criticality model [175, 40] that assigns different processing times for different criticality levels of the execution. We have studied problem properties and algorithms for problems connected to makespan minimization [6, 7], periodic scheduling [123] and for the environments with release times and deadlines [5]. Furthermore, we have introduced the *job replication* for reliability problems, where the goal is to increase the execution probability of jobs in the schedule, which extends the work of [151]. We have used the framework of Bayesian networks to analyze the complexity of the problem and to reduce the computation of the execution probability to the inference in the resulting networks. Our proposed model was recognized [40, 43, 185], adopted, and extended by [88] who applied it to message scheduling in industrial 5G NR networks and verified it on a real-life hardware testbed.

In the area of stochastic scheduling, we have worked on a β-robust scheduling problem on parallel machines with normally distributed processing times of jobs, originally introduced by [135]. In our work [168, 8], we have designed an efficient exact method for the problem with two machines which was utilized as an intensification step in our heuristic for large problem instances. We have coined a question about whether it is possible to obtain approximation guarantees for some of the natural list scheduling rules. This question was picked up by [169], who ruled out the existence of a polynomial-time algorithm with a constant approximation ratio under $\mathsf{P} \neq \mathsf{NP}$ assumption. To solve the problem, we have applied integer programming techniques which improved the scalability of the algorithm for instances with a larger number of machines. Additionally, as part of our industrial collaboration, we have utilized our experience with data-driven stochastic problems to design an SAA algorithm to optimize laboratory workflow in a medical laboratory of our industrial partner.

In the area of distributionally robust optimization, we have asked whether, with the given evaluation protocol of out-of-sample performance, is it necessary to solve the original problem optimally, or can a comparable performance be achieved by solving a related problem, perhaps at a much-reduced computation cost? In our study [122] we have analyzed the computational complexity of distributionally robust total flow time minimization problem on parallel machines originally proposed by [44]. Inspired by the regularization techniques used in statistical machine learning, we have reformulated the problem using vector norm regularizations. We have characterized the complexity of the problem based on the used ℓ_p norm and proposed new algorithms that also can handle problems with dependent jobs. Our main finding is that the formulation with the ℓ_1 norm achieves nearly identical price-robustness trade-offs as the original ℓ_2 formulation at a much lower computational cost.

1.3 Outline

The rest of the thesis is structured as follows. In Chapter 2, we deal with a distributionally robust scheduling problem concerning the total flow time minimization. We investigate the use of different vector norms as regularizations of solution variance

in the context of solution price, robustness, and computational complexity. In Chapter 3, we introduce the F-shape job model, which acts as a discretization of the cumulative distribution function of processing times. We demonstrate its use for the single machine problem with makespan minimization in a Mixed-Criticality environment. In Chapter 4, we deal with, in some sense, oposite problem. The goal is to maximize the execution probability of F-shaped jobs, which is done by displacing them along the scheduling horizon. We introduce job replication as a mechanism for increasing the execution probability, and we study how it affects the computational complexity of the evaluation of the schedule's objective. In Chapter 5, we solve the problem of the maximization of the probability that all jobs with normally distributed processing times are completed before a common due date. Again like in the previous cases, the problem formulation and its solution utilizes the knowledge of the whole distribution function of processing times. Finally, Chapter 6 concludes the work, reviews the fulfillment of the thesis, and outlines the future work.

Chapter

Distributionally robust scheduling algorithms for total flow time minimization on parallel machines using norm regularizations

2

Antonín Novák, Andrzej Gnatowski, and Premysl Sucha. "Distributionally robust scheduling algorithms for total flow time minimization on parallel machines using norm regularizations". In: *European Journal of Operational Research* 302.2 (2022), pp. 438–455. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2022.01.002

2.1 Introduction

Real-life processes often involve uncertainty, i.e., values of some parameters of the system are not known beforehand. Provided a sufficient quantity of empirical data, it is usually possible to build models describing how the uncertain parameters relate to each other and what values they can attain. Then, the description of the uncertainty can be utilized during the optimization, following various paradigms, for instance, Robust Optimization (RO, optimizing for the worst-case realization), Stochastic Programming (SP, optimizing the expected value), or DRO (distributionally robust optimization, optimizing for the worst-case expectation) that we address in this paper.

An example of such a process is unit testing, a crucial part of modern software development [163]. Each time the unit tests are executed, they cover slightly different parts of the source code, resulting in random pass/failure ratios and the run times. Since the test batches are performed repeatedly, one can build empirical distributions of the aforementioned parameters. Moreover, the tests are usually not independent—a failure of one test might lead to an automatic failure of an entire batch. Thus, it is beneficial to model the uncertainty, e.g., with multivariate distributions. When the number of tests is large, they are scheduled and performed in parallel, using a cluster of servers. When the computing nodes are identical, the problem can be modeled as parallel identical machines scheduling with uncertain job durations. The total flow time is usually chosen as the objective function [102], as minimizing average user waiting time ensures the tests can be executed frequently.

2.1.1 Problem statement

In this work, we focus on a distributionally robust scheduling introduced in [44]. The problem considers n jobs $\mathcal{J} = \{1, 2, \ldots, n\}$ that need to be scheduled on m identical machines $\mathcal{M} = \{1, 2, \ldots, m\}$. Each job is available at time 0 and can be processed on any machine, while preemption is not allowed. Job $j \in \mathcal{J}$ is characterized by uncertain processing time $\tilde{p}_j \in \mathbb{R}_+$. The processing times can be expressed as random vector $\tilde{\boldsymbol{p}} \in \mathbb{R}_+^n$ subject to an ambiguous probability distribution $P \in \mathcal{D}$, $\mathcal{D} \subseteq \mathcal{P}_0(\mathbb{R}_+^n)$, where $\mathcal{P}_0(\mathbb{R}_+^n)$ is the set of all probability distributions on $\mathbb{R}_+^n$. Finally, the objective is to minimize the worst-case expected total flow time f, i.e., a sum of completion times of jobs ($\sum C_j$ or TFT).

A solution to this problem is a schedule that assigns the jobs to the machines and sequences the assigned jobs on each of them. In [44], it was shown that the representation of the solution can completely disregard the assignment of a job to a specific machine. This property leads to a concise representation of the solution by vector $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n) \in \mathbb{Z}_+^n$ of n positive integers. In this representation, $\pi_j = l$ if and only if job j is scheduled as the l-th job from the end of the schedule (e.g., $l = 1$ means the last position) on *some* machine. The number of available machines is reflected by the property that any feasible assignment $\boldsymbol{\pi}$ contains at most m elements of the same value. In addition, any optimal solution is of a specific structure. That is, having an instance with n jobs and m machines, an optimal solution $\boldsymbol{\pi}^\star$ to the studied problem has exactly m elements with the value of 1, exactly m elements with the value of 2, and so on up to $\lfloor n/m \rfloor$. Finally, when n is not divisible by m, we have additional $n - \lfloor n/m \rfloor \cdot m$ positions, with the value of $\lfloor n/m \rfloor + 1$. Actually, since the assignment of a job to the specific machine is disregarded by $\boldsymbol{\pi}$, then whenever $m > 1$, a single $\boldsymbol{\pi}$ defines more solutions identical up to the permutations of machines with the identical objective values (see Example 1 bellow). Nevertheless, since the solutions are identical from the objective function point of view, we treat $\boldsymbol{\pi}$ as a single solution.

Considering the structure of solutions explained above, we have that the set of (potentially) optimal assignments $\boldsymbol{\pi}$ is given as

$$\boldsymbol{\Pi} = \left\{ \boldsymbol{\pi} \in \{1, \ldots, \lfloor n/m \rfloor + 1\}^n \,\middle|\, \begin{array}{l} c^{\boldsymbol{\pi}}(\lfloor n/m \rfloor + 1) = n - \lfloor n/m \rfloor \cdot m, \\ \forall l \in \{1, \ldots, \lfloor n/m \rfloor\} : \quad c^{\boldsymbol{\pi}}(l) = m \end{array} \right\}, \tag{2.1.1}$$

where $c^{\boldsymbol{\pi}}(l) = |\{j \in \mathcal{J} : \pi_j = l\}|$ is the number of jobs assigned to l-th position from the end. The positions of jobs are indexed in the reversed order, as it leads to a simplified representation of the objective function. Subsequently, the objective function f of the problem can be written as

$$f \equiv f(\boldsymbol{\pi}, \tilde{\boldsymbol{p}}) = \boldsymbol{\pi}^\intercal \tilde{\boldsymbol{p}}. \tag{2.1.2}$$

When the probability distribution P of the processing times $\tilde{\boldsymbol{p}} \sim P$ is known exactly, then one can utilize Stochastic Programming (SP) solution, which minimizes the expectation of the objective function:

$$\text{SP-PTFT} \equiv \min_{\boldsymbol{\pi} \in \boldsymbol{\Pi}} \mathbb{E}_P\left[\boldsymbol{\pi}^\intercal \tilde{\boldsymbol{p}}\right] = \min_{\boldsymbol{\pi} \in \boldsymbol{\Pi}} \boldsymbol{\pi}^\intercal \mathbb{E}_P\left[\tilde{\boldsymbol{p}}\right]. \tag{2.1.3}$$

Example 1. Let us have a problem instance with $m = 2$ machines and $n = 5$ jobs with uncertain processing times $\tilde{\boldsymbol{p}} = (\tilde{p}_1, \ldots, \tilde{p}_5)$. Suppose that we have a feasible solution $\boldsymbol{\pi} = (1, 1, 2, 3, 2)$ to the problem (2.1.3). The solution $\boldsymbol{\pi}$ represents $2^3 = 8$ different job orders. One of these orders is illustrated in Figure 2.1 (a), where job 4 is scheduled as the first job on the first machine, followed by job 3 and job 1 on the same machine. The remaining jobs are allocated to the second machine, where job 5 is followed by job 2. Using the linearity of the expected value, the objective for order (a) can be rewritten as

$$\mathbb{E}_P\big[\boldsymbol{\pi}^\intercal \tilde{\boldsymbol{p}}\big] = 3 \cdot \mathbb{E}_P\big[\tilde{p}_4\big] + 2 \cdot \mathbb{E}_P\big[\tilde{p}_3\big] + 1 \cdot \mathbb{E}_P\big[\tilde{p}_1\big] + 2 \cdot \mathbb{E}_P\big[\tilde{p}_5\big] + 1 \cdot \mathbb{E}_P\big[\tilde{p}_2\big].$$

Note that the order in Figure 2.1 (b) leads to the same result, as the multipliers of $\tilde{\boldsymbol{p}}$ are identical.

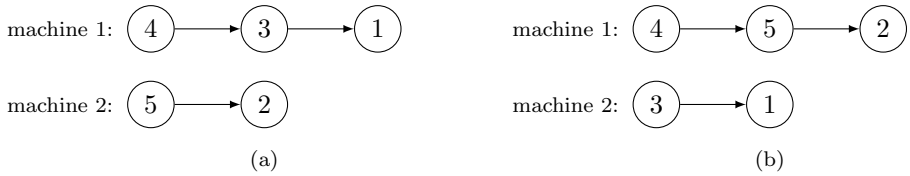


Figure 2.1: Two (out of eight) different job orders on two machines represented by $\boldsymbol{\pi} = (1, 1, 2, 3, 2)$.

2.1.2 Distributionally robust solution

Although tt can be seen that the SP solution is optimal in the sense of expected (or long term) performance, it does not hedge against variances in solution quality, thus, it may not be suitable for the risk-averse decision maker. Moreover, the probability distribution P is often not precisely known, therefore, it is advantageous to protect ourselves from sudden disturbances in solution quality caused by the changes in the distribution parameters. This is the reason why it is often useful to assume a broader concept, i.e., a set of probability distributions called an ambiguity set $\mathcal{D}$. Such a set can be in practice build with historical data, model assumptions, or problem constraints, using specific rules. Therefore, the problem is seen as *distributionally robust optimization* (DRO), which aims to find a solution that yields the best expected value of the objective function f for the worst-case distribution in $\mathcal{D}$. Thus, the probability distribution acts as a decision variable and the goal is to find solution $\boldsymbol{\pi}$ that minimizes its expected objective value with respect to the worst-case realization of the probability distribution.

Using the introduced notation, the distributionally robust problem (denoted as *DR-PTFT*) becomes

$$\text{DR-PTFT} \equiv \min_{\boldsymbol{\pi} \in \boldsymbol{\Pi}} \max_{P \in \mathcal{D}} \mathbb{E}_P\big[\boldsymbol{\pi}^\intercal \tilde{\boldsymbol{p}}\big]. \tag{2.1.4}$$

There are numerous ways to define an ambiguity set. The ambiguity set used in paper [44], as well as in this paper, constrains the first two moments of the

distribution, and it is defined as

$$\mathcal{D} = \left\{ P \;\middle|\; \begin{array}{l} \mathbb{P}_P\big[\tilde{\boldsymbol{p}} \in \mathbb{R}^n_+\big] = 1 \\ \big(\mathbb{E}_P[\tilde{\boldsymbol{p}}] - \hat{\boldsymbol{\mu}}\big)^\intercal \hat{\boldsymbol{\Sigma}}^{-1} \big(\mathbb{E}_P[\tilde{\boldsymbol{p}}] - \hat{\boldsymbol{\mu}}\big) \leq \gamma_1^2 \\ \mathbb{E}_P\Big[(\tilde{\boldsymbol{p}} - \hat{\boldsymbol{\mu}})(\tilde{\boldsymbol{p}} - \hat{\boldsymbol{\mu}})^\intercal\Big] \preceq \gamma_2^2 \hat{\boldsymbol{\Sigma}} \end{array} \right\}, \tag{2.1.5}$$

where $\hat{\boldsymbol{\mu}} \geq \mathbf{0}$ is an estimate of the mean vector, and $\hat{\boldsymbol{\Sigma}} \succeq \mathbf{0}$ is an estimate of the covariance matrix. The parameters γ_1 and γ_2 ($\gamma_1 \geq 0$, $\gamma_2 \geq 1$) define confidence in the estimates. The set can be interpreted such that the mean vector $\mathbb{E}_P[\tilde{\boldsymbol{p}}]$ is restricted in an ellipsoid of size γ_1; centered at its estimate $\hat{\boldsymbol{\mu}}$. The covariance of the distribution P, in turn, lies in a positive semidefinite cone defined by $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\Sigma}}$. The reason why it is convenient to model the ambiguity set with the two first central moments follows from the difficulty of estimating higher-order moments in case of a lack of data. Indeed, the statistical estimators of covariance matrices (e.g., sample covariance) have higher variance than the estimators of the expected value (e.g., sample mean). Thus, the higher moment one wants to estimate, the more data for its reliable estimation is needed. Furthermore, in the case of multivariate distributions (joint distributions), there are several different methods how to measure and interpret skewness (i.e., the standardized third moment), even for distribution from skewed-normal family [10]. Thus, different measures might be suitable for different applications, which makes the estimation of higher-order moments rather complicated for multivariate distributions.

Using the above notation, the problem studied in this paper can be denoted with three-field notation as $\mathrm{P}|\mathbb{P}[\tilde{\boldsymbol{p}}] \in \mathcal{P}^{DY}|\sum C_j$, where $\mathcal{P}^{DY}$ stands for Delange and Ye's ambiguity set [44, 134]. To keep the notation short, we refer to the studied problem as to DR-PTFT. Note that one of the advantageous properties of ambiguity set $\mathcal{D}$ is that when $\gamma_1 = 0$, then the worst-case expectation problem of the DRO formulation reduces exactly to the ordinary expectation of the objective function which matches SP formulation (2.1.3). Thus, the DRO formulation (2.1.4) contains an SP solution (2.1.3) as a special case.

2.1.3 Contributions and paper organization

In this paper, we revisit DR-PTFT problem from the perspective of the design and analysis of the algorithms. We demonstrate that its solution for large problem instances is computationally intractable in practice, especially with dependent jobs. Thus, we aim to design algorithms with a (pseudo) polynomial complexity, and simultaneously, to provide solutions with the same or almost the same desired properties as the optimal solutions to formulation (2.1.4). We achieve this aim by expressing the variance of a solution with a robust term, and by considering its different forms. Namely, the main contributions of this paper are:

1. we reformulate DR-PTFT as a minimization of a linear function plus a robust term in the sense of ℓ_2 norm (see Section 2.3.2);

2. we investigate the effect of the form of the robust term on the computational complexity and, as a special case of our theorem, we improve the best-known

upper bound of [44] on the complexity for the problem with independent jobs (see Section 2.3.3);

3. we extend our methods to the case when processing times of jobs are dependent and we show that the source of the hardness of the problem arises from the presence of large negative correlations, not from the simple fact that jobs are dependent (see Section 2.3.4 and Section 2.3.5);

4. we show that the robust term in the sense of ℓ_1 norm allows to solve the problem in polynomial time, and we provide the explicit definition of the corresponding ambiguity set (see Section 2.3.5);

5. we relate the proposed methods to multi-objective optimization setting in terms of expected quality and the solution variance; we significantly improve a method for uniform sampling of the solution Pareto set (see Section 2.4);

6. the experimental results show that our polynomial approximations have nearly identical performance to the formerly known second-order cone integer programming formulation from [44] while being much faster (see Section 2.5).

The rest of the paper is organized as follows. In Section 2.2, we survey the related work. In Section 2.3, we study the computational complexity of the problem in terms of ℓ_p norm with independent jobs. Then, we focus on a particular case of ℓ_1 norm, for which we propose a polynomial-time algorithm with the extension for the case of dependent jobs. In Section 2.4, we point out the relation between the form of the objective function of the problem and the multi-objective optimization in terms of solution quality and its robustness and discuss some practical concerns for solving the problem. Finally, in Section 2.5, we perform numerical experiments with our algorithms, and we provide a comparison to the state-of-the-art methods. Section 2.6 concludes the work.

2.1.3.0.1 Notation Generally we use calligraphic letters ($\mathcal{A}$) to denote sets, for vectors and matrices we use bold ($\boldsymbol{a}$, $\boldsymbol{A}$), tilde ($\tilde{a}$) for random variables, and for the estimates the hat ($\hat{a}$). By $\mathbf{0}$ and $\mathbf{1}$ we denote, respectively, vectors of zeros and ones of appropriate sizes. The diagonal matrix with vector $\boldsymbol{\lambda}$ on its diagonal is denoted as $\mathrm{diag}(\boldsymbol{\lambda})$. The set of all probability distributions on $\mathbb{R}^n$ is written as $\mathcal{P}_0(\mathbb{R}^n)$. Element-wise comparison of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ is defined as $\boldsymbol{a} \circ \boldsymbol{b} \iff \forall i : \ a_i \circ b_i$, where $\circ \in \{>, \geq, <, \leq, =\}$. We define ℓ_p norm of a vector $\boldsymbol{x} \in \mathbb{R}^n$ as $\|\boldsymbol{x}\|_p = \left(\sum_{i=1}^n |x_i|^p\right)^{1/p}$. Furthermore, we denote the set of all symmetric real positive semidefinite matrices of size $n \times n$ as $\mathbb{S}_+^n$, the set of non-negative reals as $\mathbb{R}_+$, the set of non-negative integers as $\mathbb{Z}_+$, and positive integers as $\mathbb{N}$.

2.2 Related work

Distributionally robust optimization (DRO) was introduced by Scarf [148] back in 1958. The aim of DRO is to minimize the worst-case expectation with respect to the uncertainty of the underlying distribution of the parameters, i.e., the so-called ambiguity set. The new wave of interest in DRO was sparked namely by recent

advancements of mathematical programming solvers and tractable formulations of ambiguity sets [30]. A DRO problem is typically reformulated to a deterministic mathematical programming problem, whose complexity depends on the used formulation of the ambiguity set. Often, such reformulation is more computationally attractive than stochastic and robust optimization counterparts.

2.2.1 DRO in the scheduling literature

The majority of the existing works dealing with scheduling problems and DRO have applied ambiguity sets defined by estimates of the first two moments. Wang *et al.* [179] solve the assignment of surgery blocks to operating rooms, which leads to the objective function containing a non-linear term $\sum_i \max\{0, \boldsymbol{d}_i^\intercal \boldsymbol{x} - T\}$ ($\boldsymbol{d}_i$ is a random vector of surgery durations, $\boldsymbol{x}$ are decision variables, and constant T is the regular operating room opening hours). Processing times of surgeries $\boldsymbol{d}_i$ are subject to a probability distribution contained in the ambiguity set defining bounds on mean values and mean absolute deviations. The proposed reformulation of the DRO problem formulation leads to a mixed-integer linear program (MILP) of exponential size in the number of operating rooms. The approach is able to solve problems with about 15 surgery blocks within an hour.

A DRO variant of a single machine total tardiness problem with uncertain processing times was addressed in [121]. The authors used an ambiguity set enforcing equality of the first two moments. The exact reformulation has high complexity, with the inner problem being an exponential-sized SDP (semidefinite programming) problem. Therefore, they solved a surrogate SOCP (second-order cone programming) problem instead by a custom branch-and-bound algorithm. They have been able to solve instances with 30 jobs within 40 seconds. Shang *et al.* [153] use the generalized moment functions with a piece-wise linear form, given by so-called truncation points, to define the ambiguity set. They can be used to constrain the first-order deviation projected along the selected direction. The authors also show a problem reformulation leading to a MILP. Furthermore, they propose a data-driven procedure based on principal component analysis to construct an ambiguity set from the historical data, and they apply the framework to a process scheduling problem.

A problem with bimodal distributions was studied in [158] in the context of outpatient colonoscopy scheduling. Colonoscopy duration is uncertain, and it is conditioned by the bowel preparation quality, which is uncertain as well. Moreover, uncertainty in the time when the patient will show up for the procedure is considered as well. The goal is to sequence patients such that the worst-case expectation of the weighted sum of patient and provider waiting with the overtimes is minimized. The authors use an ambiguity set that enforces support (i.e., lower and upper bounds) and the mean value for all uncertain parameters. The problem is translated into a MILP and solved by CPLEX solver.

In this paper, we build on the work of [44]. They have proposed a distributionally robust variant of the parallel identical machine scheduling problem with the minimization of the worst-case expected total flow time. The processing times of jobs are subject to uncertainty belonging to the ambiguity set constraining the first two moments. The problem is reduced to an integer SOCP, and the case

with independent jobs is solved by an exact algorithm that explores all solutions satisfying necessary optimality conditions. The proposed approach for independent jobs was able to solve instances with 100 jobs and 5 machines within several seconds whereas the integer SOCP formulation does not scale well. In our work, we address this problem from the perspective of surrogate problems and their complexity. That is, we classify related problems with respect to their complexity and we show when it is possible to obtain identical quality and robustness of solutions at a much lower computational cost. What is more, we extend the proposed methodology for the case of dependent jobs which displays excellent scaling capabilities.

2.2.2 Ambiguity set expressivity and robustness evaluation

A known shortcoming of the moment-based ambiguity sets is that the worst-case distribution might have an unrealistic form [180]. For example, it is known that under mild assumptions on the objective function, its worst-case expectation is attained at the distribution having support in at most $m + 1$ points, if m moments are constrained [156, 23]. If such distribution leads to overly conservative solutions for the target application, then it is better to use, e.g., likelihood or phi-divergence ambiguity sets [17].

The above points raise a question of whether it is appropriate to solve problems with certain ambiguity sets optimally when neither the protection against the (unrealistic) worst-case distribution is not required nor is in the interest of the decision maker. Indeed, the majority of DRO applications in scheduling do not evaluate solutions with respect to the worst-case distribution which was chosen by their DRO algorithm. Instead, the authors assume various selected distributions or they choose different evaluation protocols that are suited to the target application [179, 153, 49]. Nevertheless, the way a DRO algorithm is tested may result in a situation where, sometimes, even a heuristic solution can achieve a better performance than the exact approach under some sensible evaluation protocol, e.g., as in [179]. Therefore, this paper tries to answer a question, that is, given an evaluation protocol of out-of-sample performance, is it necessary to solve the original problem optimally, or can a comparable performance be achieved by solving a related problem, perhaps at a much-reduced computation cost?

2.2.3 Total flow time scheduling and other problems

From the perspective of the scheduling problem solved in this work, several works on total flow time scheduling are closely related. For example, in [99], the authors study a deterministic parallel identical machines scheduling problem with weighted completion time. With their enhanced arc-flow MILP formulation, they have been able to solve instances having up to 400 jobs, whereas former approaches were limited to around 100 jobs. A robust approach for parallel machines total flow time problem is studied in [2]. The authors treat the problem with normally distributed processing times as a β-robust optimization problem, where the objective is to maximize the probability that the total flow time does not exceed the given level. They developed a branch-and-bound algorithm that was able to solve instances of up to 45 jobs and 5 machines. Another total flow time scheduling problem

with sequence-dependent setups is addressed by [107]. The uncertain processing times and setups are represented by interval data. The goal is to minimize the worst-case absolute deviation of the total flow time from the optimal scenario. They have formulated the problem as a resource-constrained shortest path and devised a simulated annealing algorithm to solve it. They were able to solve instances with 200 jobs in about 20 seconds.

From the perspective of the used approaches, our methods share similarities with linear optimization problems containing absolute values of variables. Problems such as *absolute value equations* (AVE) [110] or *linear complementarity problem* (LCP) [50] are known to be $\mathcal{NP}$-hard even over real variables. We further consider problems related to ℓ_p norm minimization [72, 182], which are typically studied in the context of robust estimation and fitting. However, the current results are not directly applicable to our problem as we optimize over a set of constraints having a form of a totally unimodular matrix rather than an unconstrained case.

2.3 Solving methods for ℓ_p norm formulations

In Section 2.3.1, we outline the second-order cone program (SOCP) formulation of problem (2.1.4) from [44]. In Section 2.3.2, we express the objective function as a sum of a linear term and a robust term in the sense of some ℓ_p norm. Section 2.3.3 investigates the properties and complexity of the formulation with respect to the used norm for independent jobs. Finally, Section 2.3.4 focuses on the reformulation with dependent jobs, while Section 2.3.5 on ℓ_1 norm specifically. Our analysis shows that the problem with dependent jobs is hard only when the large negative correlations are present. Furthermore, we give a polynomial algorithm for a tractable subclass of the problem, and we investigate its robustness, given by the corresponding ambiguity set.

2.3.1 Deterministic reformulation of the stochastic problem

In [44], it was shown that when parameters $\gamma_1 \geq 0$ and $\gamma_2 \geq 1$, defining the ambiguity set (2.1.5), satisfy $\gamma_2 \geq \gamma_1$, then the stochastic problem in the form of (2.1.4) is equivalent to the following deterministic integer second-order cone program

$$\min_{\boldsymbol{\pi}\in\boldsymbol{\Pi}} \boldsymbol{\pi}^\intercal \hat{\boldsymbol{\mu}} + \gamma_1 \cdot \sqrt{\boldsymbol{\pi}^\intercal \hat{\boldsymbol{\Sigma}} \boldsymbol{\pi}}. \tag{2.3.1}$$

Interestingly, the resulting formulation does not depend on the particular value of γ_2, as long as $\gamma_2 \geq \gamma_1$. For more details, we refer the reader to [44]. Furthermore, note that when $\gamma_1 = 0$, then (2.3.1) matches SP solution (2.1.3).

In [44], authors have dealt with the special case of the problem with independent random variables, i.e., $\hat{\boldsymbol{\Sigma}} = \operatorname{diag}(\hat{\sigma}_1^2, \hat{\sigma}_2^2, \ldots, \hat{\sigma}_n^2)$ where $\hat{\sigma}_j$, $j \in \mathcal{J}$ is a standard deviation of $\tilde{p}_j$. We note that when $\hat{\boldsymbol{\Sigma}} = \mathbf{0}$ (i.e., when the processing times are deterministic), then the formulation (2.3.1) reduces to the classical, deterministic parallel identical machines total flow time problem ($P||\sum C_j$), with processing times given by $\hat{\boldsymbol{\mu}}$. The deterministic problem is solvable in $\mathcal{O}(n \log n)$ time by sorting the jobs according to $\hat{\boldsymbol{\mu}}$ values in non-increasing order. The optimal $\boldsymbol{\pi}$ is obtained by setting $\pi_j = 1$ to the first m sorted jobs, $\pi_j = 2$ to the next m jobs

until all the elements in $\boldsymbol{\pi}$ are assigned (see, e.g., [39, p. 133–134] for a similar algorithm applicable to a more general $Q||\sum C_j$ problem).

Finally, note that since $\hat{\boldsymbol{\mu}}$ (i.e., sample mean) is unbiased estimator of $\mathbb{E}[\tilde{\boldsymbol{p}}]$, we can build a direct connection between the Stochastic Programming formulation SP-PTFT and the considered DRO formulation. That is, the solution of (2.3.1) with $\gamma_1 = 0$ is equivalent to SP-PTFT formulation.

2.3.2 Robustness as a norm of solution variance

In this section, we express formulation (2.3.1) using a vector norm of the solution variance, which provides new insights to the problem. Let us assume that the estimate of covariance matrix $\hat{\boldsymbol{\Sigma}}$ is a positive semidefinite (PSD) matrix. Indeed, this is without a loss of generality as the covariance matrix of any distribution is a PSD matrix (if it exists). From the practical standpoint, the covariance matrix is typically estimated from data using the sample covariance, which provably always results in a PSD matrix. Thus, $\hat{\boldsymbol{\Sigma}}$ admits factorization into $\hat{\boldsymbol{\Sigma}} = \boldsymbol{V}\boldsymbol{D}\boldsymbol{V}^{-1}$, where $\boldsymbol{V}$ is an orthogonal matrix and $\boldsymbol{D}$ is a diagonal matrix with eigenvalues $\lambda_j \geq 0$ of $\hat{\boldsymbol{\Sigma}}$. Let us define a square root of $\hat{\boldsymbol{\Sigma}}$ as

$$\hat{\boldsymbol{\Sigma}}^{1/2} \equiv \boldsymbol{V}\boldsymbol{D}^{1/2}\boldsymbol{V}^{-1}, \tag{2.3.2}$$

where $\boldsymbol{D}^{1/2}$ is a diagonal matrix computed as element-wise square root of $\boldsymbol{D}$.

Lemma 1. *Problem* (2.3.1) *can be equivalently expressed as*

$$\textit{DR-PTFT}(\ell_2) \equiv \min_{\boldsymbol{\pi}\in\boldsymbol{\Pi}} \boldsymbol{\pi}^\intercal\hat{\boldsymbol{\mu}} + \gamma_1 \cdot \left\|\hat{\boldsymbol{\Sigma}}^{1/2}\boldsymbol{\pi}\right\|_2, \tag{2.3.3}$$

where $\|\cdot\|_2$ *is* ℓ_2 *(euclidean) norm.*

Proof. Observe that $\|\boldsymbol{x}\|_2 = \sqrt{\boldsymbol{x}^\intercal\boldsymbol{x}}$ and $\hat{\boldsymbol{\Sigma}}^{1/2}$ is a PSD matrix and is symmetric. Then, since $\boldsymbol{V}^{-1} = \boldsymbol{V}^\intercal$, it follows that

$$\left\|\hat{\boldsymbol{\Sigma}}^{1/2}\boldsymbol{\pi}\right\|_2 = \sqrt{\left(\hat{\boldsymbol{\Sigma}}^{1/2}\boldsymbol{\pi}\right)^\intercal\hat{\boldsymbol{\Sigma}}^{1/2}\boldsymbol{\pi}} = \sqrt{\boldsymbol{\pi}^\intercal\boldsymbol{V}\boldsymbol{D}^{1/2}\boldsymbol{V}^{-1}\boldsymbol{V}\boldsymbol{D}^{1/2}\boldsymbol{V}^{-1}\boldsymbol{\pi}} = \sqrt{\boldsymbol{\pi}^\intercal\hat{\boldsymbol{\Sigma}}\boldsymbol{\pi}}. \tag{2.3.4}$$

By substituting (2.3.4) into (2.3.3), we obtain (2.3.1). □

While the reformulation (2.3.3) itself does not bring anything novel, it provides an interesting insight into the connections of (2.1.4) with the related problems. Namely, we see certain similarities with robust regression methods used in machine learning (ML). There, typically one does not have a precise knowledge of the underlying distribution of the data. Instead, one has access to a finite sample set that can be used to estimate the ambiguity. The training of a prediction model is treated as an optimization problem, minimizing a function defined as a mean error on input samples plus a complexity measure of the model, which typically refers to the number of degrees of freedom used in the learned model. The end goal is to find a model that achieves a small error on unseen data. The resulting model (by analogy, here—a solution to the scheduling problem) is chosen such that it does not overfit the training data (sampled processing times) by having some

level of generalization to unseen data (robustness with respect to the processing times uncertainty). Viewing the researched problem from the perspective of ML analogy, $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\Sigma}}^{1/2}$ are derived from the training set, $\boldsymbol{\pi}$ represents the model, $\hat{\boldsymbol{\mu}}^\intercal\boldsymbol{\pi}$ is its error on input samples (performance), while an estimate of model complexity (variance) is expressed as $\|\hat{\boldsymbol{\Sigma}}^{1/2}\boldsymbol{\pi}\|_2$ and acts as a regularization term. The level of protection against overfitting is typically controlled by a weight term for the regularization term, in our case, corresponding to γ_1. There are several common methods in ML how to penalize the model complexity, e.g.,: *ridge regression*, *support vector machines (SVM)* (squared ℓ_2 norm), *lasso regression* (ℓ_1 norm), or *smoothing regularization* ($\|\boldsymbol{D}\boldsymbol{\pi}\|_p$ for some suitably chosen matrix $\boldsymbol{D}$) [118]. Frequently, an ℓ_p norm of the model parameters is used. Different choices of penalty terms lead to different models and training (optimization) algorithms [69]. Similar connections between DRO and regularization approaches were observed by other authors as well, see, e.g., [134].

Therefore, the above reformulation stimulates several interesting questions. Namely, we ask whether the ℓ_2 norm used in DR-PTFT(ℓ_2) formulation (2.3.3) is essential to preserving the quality of solutions, or maybe rather, can it be replaced with a different penalty (e.g., ℓ_1 norm)? What are then the performance guarantees of such a model and how does the change of regularization affects the complexity of the problem? In the following sections, we provide answers to these questions.

2.3.3 Complexity of ℓ_p formulation with independent jobs

In the paper [44], the computational complexity problem of (2.3.1) was not studied. Regarding the complexity of their algorithm for independent jobs (called NPSA) was shown that terminates within the finite number of iterations, but no specific time bound was given. In this section, we provide a complexity characterization for the problem formulation with independent jobs in sense of any ℓ_p norm:

$$\text{DR-PTFT}(\ell_p) \equiv \min_{\boldsymbol{\pi}\in\boldsymbol{\Pi}} \hat{\boldsymbol{\mu}}^\intercal\boldsymbol{\pi} + \gamma_1 \cdot \left\|\hat{\boldsymbol{\Sigma}}^{1/2}\boldsymbol{\pi}\right\|_p. \tag{2.3.5}$$

We show that the particular case $p = 2$ of the following proposition reduces to problem (2.3.1) with independent jobs which, in turn, establish a new complexity result and provides a new algorithm for problem (2.3.1) with independent jobs studied in [44].

In this subsection, we study the case when the processing times of jobs are independent, i.e., $\hat{\boldsymbol{\Sigma}} = \text{diag}(\hat{\sigma}_1^2, \hat{\sigma}_2^2, \ldots, \hat{\sigma}_n^2)$. Provided that the processing times are non-negative, without a loss of generality we assume that all elements of $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\Sigma}}$ are non-negative integers. Finally, the following proposition provides a characterization of the computational complexity and the solution algorithm for DR-PTFT(ℓ_p) with independent jobs.

Proposition 1. *For any fixed integer $p \geq 1$ and a diagonal covariance matrix $\hat{\boldsymbol{\Sigma}} = \text{diag}(\hat{\sigma}_1^2, \ldots, \hat{\sigma}_n^2)$, problem DR-PTFT($\ell_p$) admits a pseudopolynomial algorithm in $\max_{j\in\mathcal{J}} \{\hat{\mu}_j \cdot \lambda, (\hat{\sigma}_j \cdot \lambda)^p\}$, where $\lambda = \min\left\{\hat{\boldsymbol{\mu}}^\intercal\mathbf{1}, \mathbf{1}^\intercal\hat{\boldsymbol{\Sigma}}\mathbf{1}\right\}$.*

Proof. We prove the statement by reducing DR-PTFT(ℓ_p) problem to a non-linear perfect matching problem in a bipartite graph [28] with a suitably defined non-linear

objective function. Such problem is given as the maximization of d-dimensional convex function $q(z_1, \ldots, z_d) : \mathbb{R}_+^d \mapsto \mathbb{R}$ over a set of all perfect matchings $\boldsymbol{g} \in \{0,1\}^{n^2}$ in complete bipartite graph $K_{n,n}$. The arguments of the function q are given as mere linear combinations of the characteristic vector $\boldsymbol{g}$ of the matching and weights $z_i = \boldsymbol{w}_i^\intercal \boldsymbol{g}$, $\boldsymbol{w}_i \in \mathbb{Z}_+^{n^2}$ where $i \in \{1, \ldots, d\}$. In other words, each matching $\boldsymbol{g}$ is scored by d different non-negative integer weights, which are aggregated into a single convex scoring function to be maximized. When the number of arguments d of the function to be maximized is fixed to a constant, then such problem can be solved in a pseudopolynomial time in the maximal weight, as shown in [28]. The reduction given below preserves the pseudopolynomial time complexity with respect to the sum of means and variances, and, thus, the algorithm scheme of [28] is applicable to solve the problem.

Let us define complete bipartite graph $K_{n,n} = (\mathcal{J}, \mathcal{L}, \mathcal{E})$, where $\mathcal{J}$ is a set of all jobs and $\mathcal{L}$ is a multiset of all eligible positions (i.e., including their multiplicity, see the definition of $\boldsymbol{\Pi}$ in (2.1.1)) for any job given as

$$\mathcal{L} = \{ \underbrace{1, \ldots, 1}_{m \text{ elements}}, \underbrace{2, \ldots, 2}_{m \text{ elements}}, \ldots, \underbrace{\lfloor n/m \rfloor, \ldots, \lfloor n/m \rfloor}_{m \text{ elements}}, \underbrace{\lfloor n/m \rfloor + 1, \ldots, \lfloor n/m \rfloor + 1}_{(n - \lfloor n/m \rfloor \cdot m) \text{ elements}} \}.$$

The first part of the graph represents jobs, and the other part represents all possible job positions, including their multiplicity given by the number of machines. We associate each edge $(j,l) \in \mathcal{E}$, $j \in \mathcal{J}, l \in \mathcal{L}$ with two weights $(\hat{\mu}_j \cdot l, |\hat{\sigma}_j|^p \cdot l^p)$ and we collect all first and second weights into vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{Z}_+^{n^2}$.

Next, let us denote a perfect matching in $K_{n,n}$ as $M \subseteq \mathcal{E}$. Such matching M can be represented by the characteristic vector $\boldsymbol{g} \in \{0,1\}^{n^2}$ with i-th entry being one if and only if the corresponding edge is contained in M, and zero otherwise. Finally, let us define function $q : \mathbb{R}_+^2 \mapsto \mathbb{R}$ given as

$$q(z_1, z_2) = -z_1 - \gamma_1 \cdot {z_2}^{1/p}.$$

Note that, as $\gamma_1 \geq 0$, we have that q is convex on $\mathbb{R}_+^2$ for any $p \in \mathbb{N}$. Then, for any perfect matching $\boldsymbol{g}$, we have that

$$\begin{aligned} q(\boldsymbol{a}^\intercal \boldsymbol{g}, \boldsymbol{b}^\intercal \boldsymbol{g}) = -\boldsymbol{a}^\intercal \boldsymbol{g} - \gamma_1 \cdot (\boldsymbol{b}^\intercal \boldsymbol{g})^{1/p} &= -\boldsymbol{\pi}^\intercal \hat{\boldsymbol{\mu}} - \gamma_1 \cdot \left(\sum_{j=1}^{n} |\hat{\sigma}_j|^p \cdot \pi_j^p \right)^{1/p} \\ &= -\boldsymbol{\pi}^\intercal \hat{\boldsymbol{\mu}} - \gamma_1 \cdot \left\| \hat{\boldsymbol{\Sigma}}^{1/2} \boldsymbol{\pi} \right\|_p, \end{aligned} \tag{2.3.6}$$

where $\pi_j = l$ if and only if edge $(j,l) \in M$. The first equality follows from the definition of q while the second one follows from the definition of $\boldsymbol{a}$ and $\boldsymbol{b}$ and the fact that $\boldsymbol{g}$ is a perfect matching in a bipartite graph. Thus, for each $j \in \mathcal{J}$, exactly one edge to some $l \in \mathcal{L}$ is selected. Finally, any perfect matching $\boldsymbol{g}^\star$ maximizing $q(\boldsymbol{a}^\intercal \boldsymbol{g}^\star, \boldsymbol{b}^\intercal \boldsymbol{g}^\star)$ corresponds to minimizing (2.3.3) which solves DR-PTFT(ℓ_p) problem.

Example 2 (cont.)**.** We continue with the example introduced in Section 2.1.1. Let $\hat{\boldsymbol{\mu}} = (5, 3, 3, 1, 2)$ be a vector of estimated means and let $\hat{\boldsymbol{\Sigma}} = \operatorname{diag}(1, 2, 1, 4, 3)$ be a diagonal covariance matrix (i.e., the jobs processing times are independent).

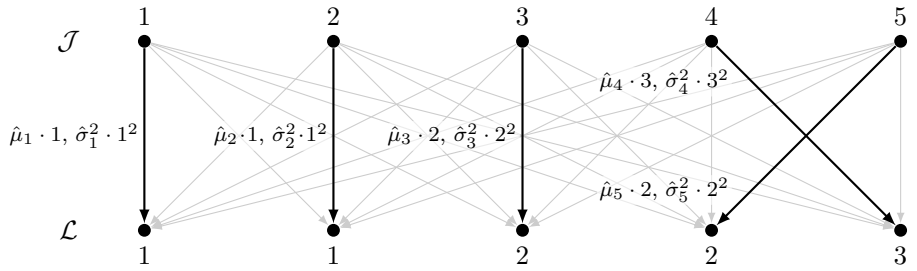


Figure 2.2: Graph $K_{n,n}$ for instance with $n = 5$, $m = 2$, and optimal solution $\boldsymbol{\pi}^\star = (1, 1, 2, 3, 2)$.

Finally, let $\gamma_1 = 1$ and $p = 2$. The graph $K_{n,n}$ for this problem instance is depicted in Figure 2.2. The edges in bold correspond to the optimal solution $\boldsymbol{\pi}^\star$.

The remaining question is, how to find such $\boldsymbol{g}^\star$ in the required time. We see that our function q and weights $\boldsymbol{a}$ and $\boldsymbol{b}$ satisfy assumptions of Theorem 1.2 of [28] for the case with $d = 2$. Their algorithm runs in a polynomial time in n and $\max_i\{\boldsymbol{a}_i, \boldsymbol{b}_i\}$ provided that q is polynomially computable. In our setting, function q can be evaluated in a polynomial time in n and the constants involved can be upper bounded as $\max_i\{\boldsymbol{a}_i, \boldsymbol{b}_i\} \leq \max_j\{\hat{\mu}_j \cdot n, (\hat{\sigma}_j \cdot n)^p\}$ with $n \leq \min\{\hat{\boldsymbol{\mu}}^\intercal \mathbf{1}, \mathbf{1}^\intercal \hat{\boldsymbol{\Sigma}} \mathbf{1}\}$ since the parameters are non-negative integers. Thus, applying the algorithm described in [28] with the above-mentioned setting concludes the proof. □

Remark 1. *We note that when the robust term is p-th power of an ℓ_p norm (e.g., ℓ_2 norm squared), then the problem* (2.3.5) *with independent jobs becomes an ordinary min-cost perfect bipartite matching problem with a linear objective function. It can be seen from the equation* (2.3.6)*, when the term $\boldsymbol{b}^\intercal \boldsymbol{g}$ is raised to the p-th power, then the function q becomes separable. Therefore, only a single coefficient $c_{j,l} = \hat{\mu}_j \cdot l + \gamma_1 \cdot |\hat{\sigma}_j|^p \cdot l^p$ for an edge (j, l) suffices to resemble the problem with p-th power of an ℓ_p norm. Such problem can be solved as the ordinary min-cost perfect bipartite matching problem in polynomial time by, e.g.,* HUNGARIAN *algorithm in $\mathcal{O}(n^3)$ [89].*

The difficulty of extending the above proposition to the case of dependent jobs lies in the necessity of having the number of arguments of function q fixed to some constant d. The reason is that an underlying step of the algorithm from [28], which solves the non-linear bipartite matching problem constructs a d-dimensional integer lattice to examine, thus having an exponential complexity in d. When $\hat{\boldsymbol{\Sigma}}$ is a full covariance matrix, then we would need to have $d = 1 + n$ to exploit the same approach as described above. A possible way to go around it would be to compress the information contained in $\hat{\boldsymbol{\Sigma}}$ to a smaller matrix while preserving norms of vectors transformed by the corresponding linear mappings, which is the idea we will explore in the following section.

2.3.4 Approximate solution for ℓ_p norm with dependent jobs

The purpose of this section is to analyze to which extent the ideas developed in the previous section can be applied to the problem with dependent jobs. In the context

of ℓ_1 norm, we show that the difficulty of the problem with dependent jobs lies in the presence of large negative correlations between jobs, not just in the plain fact that jobs are dependent.

To overcome the exponential grow of complexity in n, the trick is to compress the information about the norm of $\hat{\boldsymbol{\Sigma}}^{1/2}\boldsymbol{\pi}$ vector using a different vector $\hat{\boldsymbol{\Sigma}}_k^{1/2}\boldsymbol{\pi}$ of a fixed length k independent from n. However, the same algorithm as in the case with independent jobs cannot be applied. The difficulty is that for dependent jobs, the objective function used in non-linear bipartite perfect matching now loses its convexity. Thus, we will employ a weaker version of the algorithm for non-linear bipartite perfect matching which maximizes an arbitrary function $q : \mathbb{R}^{k+1} \mapsto \mathbb{R}$ over a set of perfect matchings in the complete bipartite graph. In [28], such algorithm is given which runs also in a pseudopolynomial time with the caveat that it is a randomized algorithm — i.e., an algorithm with access to the random bit generator which for any input returns an optimal solution with the probability of at least $1/2$.

The rationale behind the approach with a compressed covariance matrix is that performing computations over a matrix that is similar in some sense to the original one should yield similar results as performing the computation over the original matrix, but with a significantly reduced computational cost. Indeed, this scheme is frequently exploited in numerical linear algebra, e.g., to approximate solutions to problems such as multiplication of large matrices, matrix decompositions, approximate regression problems, and finds many other applications [164].

The general idea of the reduction is similar to the one used in Proposition 1 except for some minor differences, which we describe below. Then, we formulate the task of finding an approximation (compression) of the original matrix and present some solutions to this problem.

The underlying bipartite graph has the same structure as in Proposition 1. We associate each edge (j, l) with $k + 1$ values:

$$(\hat{\mu}_j \cdot l, s_{1,j} \cdot l + v, \ldots, s_{k,j} \cdot l + v)\,,$$

where $s_{i,j}$ is (i, j)-th element of $\hat{\boldsymbol{\Sigma}}_k^{1/2}$ matrix and $v = n \cdot \max_{i,j} |s_{i,j}|$. We collect all $k + 1$ weights along all edges into vectors $\boldsymbol{w}_0, \ldots, \boldsymbol{w}_k \in \mathbb{R}^{n^2}$. Next, we denote the characteristic vector of matching $M \subseteq \mathcal{E}$ in $K_{n,n} = (\mathcal{J}, \mathcal{L}, \mathcal{E})$ as $\boldsymbol{g} \in \{0, 1\}^{n^2}$ with i-th entry being one if and only if the corresponding edge is contained in matching M. Next, we define function $q : \mathbb{R}^{k+1} \mapsto \mathbb{R}$,

$$q(z_0, z_1, \ldots, z_k) = -z_0 - \gamma_1 \cdot \|(z_1 - nv, \ldots, z_k - nv)\|_p \tag{2.3.7}$$

to be maximized over a set of all perfect matchings $\boldsymbol{g}$ with $z_i = \boldsymbol{w}_i^\intercal \boldsymbol{g}$.

In contrast to the case when $\hat{\boldsymbol{\Sigma}}^{1/2}$ was a diagonal matrix, some entries of $\hat{\boldsymbol{\Sigma}}_k^{1/2}$ might be negative, but the underlying algorithm for non-linear bipartite matching [28] requires weights which are non-negative integers. This is why we add a positive constant v to the last k values of each edge and subtract them back in (2.3.7). Furthermore, if some $s_{i,j}$ is not an integer, then we need to multiply all weights by a sufficiently large constant. Then, we can use randomized version of the algorithm for non-linear bipartite perfect matching, which performs maximization of an arbitrary function $q(\boldsymbol{w}_0^\intercal \boldsymbol{g}, \ldots, \boldsymbol{w}_k^\intercal \boldsymbol{g})$ given by a polynomial-time comparison

oracle. Such a method follows from Theorem 1.3 of [28] with $d = k + 1$ running in a pseudopolynomial time, hence, avoiding an exponential complexity in n.

The question that remains is how to find a suitable approximation of $\hat{\mathbf{\Sigma}}^{1/2}$. Given parameter $k \in \mathbb{N}$, the goal is to find matrix $\hat{\mathbf{\Sigma}}_k^{1/2} \in \mathbb{R}^{k\times n}$ which does not yield to a large error for vectors $\boldsymbol{\pi} \in \mathbf{\Pi}$ in the sense of some ℓ_p norm. That is, one wishes to find

$$\min_{\hat{\mathbf{\Sigma}}_k^{1/2}\in\mathbb{R}^{k\times n}} \max_{\boldsymbol{\pi}\in\mathbf{\Pi}} \left| \|\hat{\mathbf{\Sigma}}^{1/2}\boldsymbol{\pi}\|_p - \|\hat{\mathbf{\Sigma}}_k^{1/2}\boldsymbol{\pi}\|_p \right|. \tag{2.3.8}$$

We call $\hat{\mathbf{\Sigma}}_k^{1/2} \in \mathbb{R}^{k\times n}$ matrix as *rank-k* approximation of $\hat{\mathbf{\Sigma}}^{1/2}$ and its *distortion* is defined as the maximum absolute difference of the norms of the two vectors over $\mathbf{\Pi}$ in the sense of ℓ_p.

An obvious question to ask is how to look for good approximations of $\hat{\mathbf{\Sigma}}^{1/2}$ with small ranks and how large distortions are incurred. The answer depends on the used norm. This problem is, in fact, very closely related to the *subspace embedding problem* [178], which has many applications, namely in numerical linear algebra. Since the solution of (2.3.8) in its generality goes well beyond the scope of this paper, we rather briefly describe some particular results related to our application. We provide below some examples of good approximations for some matrices under ℓ_1 norm. The following lemma addresses the case when jobs are positively correlated.

Lemma 2. *For any $\hat{\mathbf{\Sigma}}^{1/2} \in \mathbb{R}_+^{n\times n}$, there exists a rank-1 approximation $\hat{\mathbf{\Sigma}}_1^{1/2}$ with zero distortion in sense of ℓ_1 norm.*

Proof. Set $\hat{\mathbf{\Sigma}}_1^{1/2} = \mathbf{1}^\intercal\hat{\mathbf{\Sigma}}^{1/2}$, i.e., a matrix with column sums. Then, for $k = 1$, we have

$$\|\hat{\mathbf{\Sigma}}_k^{1/2}\boldsymbol{\pi}\|_1 = \|\mathbf{1}^\intercal\hat{\mathbf{\Sigma}}^{1/2}\boldsymbol{\pi}\|_1 = \left|\sum_{i=1}^n\sum_{j=1}^n \hat{\mathbf{\Sigma}}_{ij}^{1/2}\pi_j\right| = \sum_{i=1}^n\left|\sum_{j=1}^n \hat{\mathbf{\Sigma}}_{ij}^{1/2}\pi_j\right| = \|\hat{\mathbf{\Sigma}}^{1/2}\boldsymbol{\pi}\|_1,$$

where the third equality follows from the fact that $\hat{\mathbf{\Sigma}}^{1/2} \in \mathbb{R}_+^{n\times n}$. □

The above lemma also suggests how to obtain good approximations for covariance matrices with a small number of negative entries:

Corollary. *For any $\hat{\mathbf{\Sigma}}^{1/2}$ with at most k rows with a negative entry, there exists rank-$(k+1)$ approximation $\hat{\mathbf{\Sigma}}_{k+1}^{1/2}$ with zero distortion in sense of ℓ_1 norm.*

The construction is straightforward — keep all k rows with a negative element and, for the rest, apply Lemma 2, yielding a rank-$(k+1)$ approximation. The above approximations suggest that the complexity of the problem with correlated jobs in the sense of ℓ_1 norm is closely connected to the presence of negative correlations in the covariance matrix. In fact, based on the above construction it can shown that the distortion for a rank-1 approximation is proportional to n, $\mathsf{nne}(\hat{\mathbf{\Sigma}}^{1/2})$ and $\max_j \hat{\mathbf{\Sigma}}_{jj}^{1/2}$, where $\mathsf{nne}(\hat{\mathbf{\Sigma}}^{1/2})$ is the number of negative elements of $\hat{\mathbf{\Sigma}}^{1/2}$. Obviously, one can again trade-off the rank of the approximation for its precision by keeping some of the rows intact, yielding distortion dependant on $\mathsf{nne}(\cdot)$ of the remaining

matrix. For other related results on ℓ_1 subspace embedding, we refer the reader to, e.g., [84, 46, 164].

In the following section, we turn our attention to the formulation utilizing ℓ_1 norm from the perspective of robustness and its computational complexity.

2.3.5 ℓ_1 norm formulation for dependent jobs

In this section, we will focus specifically on the problem with ℓ_1 norm and dependent jobs. As we show below, this particular case leads to favorable computational complexity as well as both theoretical guarantees of the robustness and its empirical performance. These favorable properties make this case the most practical one.

Let us consider the following problem

$$\text{DR-PTFT}(\ell_1) \equiv \min_{\boldsymbol{\pi}\in\boldsymbol{\Pi}} \hat{\boldsymbol{\mu}}^{\intercal}\boldsymbol{\pi} + \gamma_1 \cdot \left\|\hat{\boldsymbol{\Sigma}}^{1/2}\boldsymbol{\pi}\right\|_1, \tag{2.3.9}$$

where the robust term is expressed in the sense of ℓ_1 norm. As it will be shown below, the benefit of such formulation is that problem (2.3.9) can be solved in strongly polynomial time when $\hat{\boldsymbol{\Sigma}}$ fulfills so-called copositivy condition, which is related to the relative magnitude of negative elements in $\hat{\boldsymbol{\Sigma}}^{1/2}$ matrix. At the same time, as it is shown later in Section 2.5.3, the quality of solutions to problem (2.3.9) is comparable to the solutions of a more complex ℓ_2 formulation.

Speaking about general PSD covariance matrices, the complexity of problem (2.3.9) arises from the presence of the absolute value inside ℓ_1 norm. It is known that equations with absolute values are hard to solve even over the domain of real numbers [110], suggesting that solving (2.3.9) in its generality might be hard as well. Therefore, we will focus on the cases where each element of the vector $\hat{\boldsymbol{\Sigma}}^{1/2}\boldsymbol{\pi}$ is non-negative, which is more general than simply requiring $\hat{\boldsymbol{\Sigma}}^{1/2} \in \mathbb{R}^{n\times n}_+$. We show that such cases of the problem can be solved in polynomial time. For that, we introduce a subclass of matrices which acts as a generalization of strictly positive covariances:

Definition (Copositivity with respect to $\boldsymbol{\Pi}$)**.** *Let us define a set of matrices*

$$\mathcal{C}^{\boldsymbol{\Pi}}_+ = \left\{\boldsymbol{A} \in \mathbb{S}^n_+ \,\middle|\, \forall \boldsymbol{\pi} \in \boldsymbol{\Pi} \subset \mathbb{Z}^n_+ : \boldsymbol{A}\boldsymbol{\pi} \geq \boldsymbol{0}\right\},$$

which is the set of PSD matrices that maps $\boldsymbol{\Pi}$ *into* $\mathbb{R}^n_+$.

Intuitively, the set of matrices $\mathcal{C}^{\boldsymbol{\Pi}}_+$ relates to the notion of diagonally-dominant matrices. Obviously, it follows that any covariance matrix of independent jobs (or strictly positively correlated) is contained in $\mathcal{C}^{\boldsymbol{\Pi}}_+$. Next, when $\hat{\boldsymbol{\Sigma}}^{1/2}$ matrix has diagonal elements that are about a factor $\mathcal{O}((n/m)^2)$ larger than the absolute value of the largest negative off-diagonal element, then it is likely to be contained in $\mathcal{C}^{\boldsymbol{\Pi}}_+$. Such covariance matrices appear, e.g., in distributions of so-called *weakly correlated random variables* [109]. As an example, we list some particular matrices below.

Example 3. Consider the following example covariance matrices:

$$\boldsymbol{A}_1 = \begin{bmatrix} 3 & 1 & 0 & 2 \\ 1 & 3 & 0 & 1 \\ 0 & 0 & 2 & 1 \\ 2 & 1 & 1 & 4 \end{bmatrix}, \quad \boldsymbol{A}_2 = \begin{bmatrix} 3 & -1 & 0 & 2 \\ -1 & 3 & 1 & 1 \\ 0 & 1 & 2 & -1 \\ 2 & 1 & -1 & 4 \end{bmatrix}, \quad \boldsymbol{A}_3 = \begin{bmatrix} 4 & -2 & 0 & 2 \\ -2 & 3 & 0 & 1 \\ 0 & 0 & 2 & 1 \\ 2 & 1 & 1 & 4 \end{bmatrix}.$$

All matrices are PSD. Next, it can be verified, e.g., by enumeration of all $\boldsymbol{\pi} \in \boldsymbol{\Pi}$, that $\boldsymbol{A}_1, \boldsymbol{A}_2 \in \mathcal{C}_+^{\boldsymbol{\Pi}}$ but $\boldsymbol{A}_3 \notin \mathcal{C}_+^{\boldsymbol{\Pi}}$ for $\boldsymbol{\Pi}$ corresponding to set of assignments for a single machine. As it was discussed in Section 2.3.4, Lemma 2 would suggest approximating $\boldsymbol{A}_1$ with a rank-1 matrix, $\boldsymbol{A}_2$ with a rank-4, and $\boldsymbol{A}_3$ with a rank-3 matrix. Thus, the notions of rank-k approximation and $\mathcal{C}_+^{\boldsymbol{\Pi}}$ are generally incomparable. Finally, note that when $\boldsymbol{\Pi}'$ corresponds to the set of assignments for two machines, then $\boldsymbol{A}_3 \in \mathcal{C}_+^{\boldsymbol{\Pi}'}$.

Another useful property of $\mathcal{C}_+^{\boldsymbol{\Pi}}$ is that it forms a convex cone, meaning that whether $\boldsymbol{A}, \boldsymbol{B} \in \mathcal{C}_+^{\boldsymbol{\Pi}}$, then $\alpha\boldsymbol{A} + \beta\boldsymbol{B} \in \mathcal{C}_+^{\boldsymbol{\Pi}}$ for any $\alpha, \beta \geq 0$. This property will be utilized later. First, we ask the question of whether it is possible to test the membership in $\mathcal{C}_+^{\boldsymbol{\Pi}}$ for a matrix $\boldsymbol{A}$ efficiently. Since $\boldsymbol{\Pi}$ is a finite set for any n and m (although a large one), one could enumerate all its elements and test the inequality for each element of $\boldsymbol{\Pi}$. However, a more efficient way exists. The test whether a given matrix $\boldsymbol{A} \in \mathbb{S}_+^n$ is contained in $\mathcal{C}_+^{\boldsymbol{\Pi}}$ can be performed in $\mathcal{O}(n^2 \log n)$ time. The idea is the following. If the copositivity condition holds, there must not exist a pair of $\boldsymbol{\pi} \in \boldsymbol{\Pi}$ and $i \in \mathcal{J}$, such that $\boldsymbol{e}_i^\intercal \boldsymbol{A}\boldsymbol{\pi} < 0$, where $\boldsymbol{e}_i = (0, \ldots, 0, 1, 0, \ldots, 0)$ is an i-th basis vector. Essentially, it selects i-th row of matrix $\boldsymbol{A}$ and multiplies it with a $\boldsymbol{\pi}$, which as to be a non-negative number. If one wants to know if this holds for the given i for any $\boldsymbol{\pi} \in \boldsymbol{\Pi}$, then it is enough to examine the worst-case $\boldsymbol{\pi}$ vector. That is, the test checks for each $i \in \mathcal{J}$, whether $\min_{\boldsymbol{\pi} \in \boldsymbol{\Pi}} \boldsymbol{e}_i^\intercal \boldsymbol{A}\boldsymbol{\pi} \geq 0$. The minimum can be evaluated by sorting $\boldsymbol{e}_i^\intercal \boldsymbol{A}$ and $\boldsymbol{\pi}$; — assigning the lowest values of $\boldsymbol{e}_i^\intercal \boldsymbol{A}$ with the highest of $\boldsymbol{\pi}$. This step takes $\mathcal{O}(n \log n)$ time, and thus the overall complexity is $\mathcal{O}(n^2 \log n)$.

In the rest of this section, we will analyze the properties of problem DR-PTFT(ℓ_1) with copositive covariance matrices. We will show that solutions of the problem (2.3.9) have similar robust properties as in ℓ_2 case. When $\hat{\boldsymbol{\Sigma}}^{1/2} \in \mathcal{C}_+^{\boldsymbol{\Pi}}$, the solution of (2.3.9) corresponds exactly to a distributionally robust solution over a specific ambiguity set.

Proposition 2. *Assuming $\hat{\boldsymbol{\Sigma}}^{1/2} \in \mathcal{C}_+^{\boldsymbol{\Pi}}$, the problem DR-PTFT($\ell_1$) is a distributionally robust formulation for $P|| \sum C_j$ with an ambiguity set given by*

$$\mathcal{D}_{\ell_1} = \left\{ P \in \mathcal{P}_0(\mathbb{R}^n) \;\middle|\; \begin{array}{l} \mathbb{P}_P[\tilde{\boldsymbol{p}} \geq \boldsymbol{0}] = 1 \\ \mathbb{E}_P[\tilde{\boldsymbol{p}}] \leq \hat{\boldsymbol{\mu}} + \gamma_1 \cdot \hat{\boldsymbol{\Sigma}}^{1/2}\boldsymbol{1} \end{array} \right\}.$$

Proof. We will start with a high-level sketch of the proof. We substitute the definition of an ambiguity set $\mathcal{D}_{\ell_1}$ into the stochastic formulation of the DRO problem considered (2.1.4). We focus on the inner expectation problem of finding the worst-case probability distribution. First, we reformulate the inner problem using the definition of the expected value. Then, we derive the dual problem and observe that strong duality holds (both problems have the same optimal solutions). Finally, we transform the dual problem and substitute it back into the outer problem (finding optimal $\boldsymbol{\pi}$), obtaining the DR-PTFT(ℓ_1) problem.

Starting with the definition of the problem in (2.1.4)

$$\min_{\boldsymbol{\pi} \in \boldsymbol{\Pi}} \max_{P \in \mathcal{D}} \mathbb{E}_P\big[f(\boldsymbol{\pi}, \tilde{\boldsymbol{p}})\big] = \min_{\boldsymbol{\pi} \in \boldsymbol{\Pi}} \max_{P \in \mathcal{D}} \boldsymbol{\pi}^\intercal \mathbb{E}_P\big[\tilde{\boldsymbol{p}}\big]. \tag{2.3.10}$$

Let us focus on the inner maximum, i.e., finding the worst-case probability distribution P from the ambiguity set $\mathcal{D}$. By letting $\mathcal{D} \equiv \mathcal{D}_{\ell_1}$, the maximum can be calculated from the definition of the expected value

$$\max_{P \in \mathcal{D}_{\ell_1}} \int_{\mathbb{R}^n_+} \mathrm{f}_P(\boldsymbol{p}) \boldsymbol{\pi}^\intercal \boldsymbol{p} \, d\boldsymbol{p} \tag{2.3.11}$$

$$\text{s.t.} \int_{\mathbb{R}^n_+} \mathrm{f}_P(\boldsymbol{p}) \, d\boldsymbol{p} = 1 \tag{2.3.12}$$

$$\int_{\mathbb{R}^n_+} \mathrm{f}_P(\boldsymbol{p}) \boldsymbol{p} \, d\boldsymbol{p} \leq \hat{\boldsymbol{\mu}} + \gamma_1 \cdot \hat{\boldsymbol{\Sigma}}^{1/2} \mathbf{1} = \boldsymbol{h}, \tag{2.3.13}$$

where f_P is a probability density function of probability distribution P and $\boldsymbol{p}$ is a value in the support (i.e., the set of possible realizations) of $\tilde{\boldsymbol{p}}$, i.e., $\mathbb{R}^n_+$. Now, we will derive a dual problem for (2.3.11)–(2.3.13). Essentially, we put constraints into the objective multiplied with newly introduced multipliers. The Lagrangian of the problem is given by the equation

$$L(P, \alpha, \boldsymbol{\beta}) = \int_{\mathbb{R}^n_+} \mathrm{f}_P(\boldsymbol{p}) \Big(\boldsymbol{\pi}^\intercal \boldsymbol{p} - \alpha - \boldsymbol{\beta}^\intercal \boldsymbol{p} \Big) \, d\boldsymbol{p} + \alpha + \boldsymbol{\beta}^\intercal \boldsymbol{h}, \tag{2.3.14}$$

where $\alpha \in \mathbb{R}$ and $\boldsymbol{\beta} \in \mathbb{R}^n_+$ are the introduced Lagrange multipliers. The dual Lagrangian function is obtained with taking maximum over the original variables. Thus, the dual for the Lagrangian above is

$$g(\alpha, \boldsymbol{\beta}) = \max_{P \in \mathcal{D}_{\ell_1}} L(P, \alpha, \boldsymbol{\beta}) \tag{2.3.15}$$

$$= \max_{P \in \mathcal{D}_{\ell_1}} \left(\int_{\mathbb{R}^n_+} \mathrm{f}_P(\boldsymbol{p}) \Big(\boldsymbol{\pi}^\intercal \boldsymbol{p} - \alpha - \boldsymbol{\beta}^\intercal \boldsymbol{p} \Big) \, d\boldsymbol{p} + \alpha + \boldsymbol{\beta}^\intercal \boldsymbol{h} \right) \tag{2.3.16}$$

$$= \alpha + \boldsymbol{\beta}^\intercal \boldsymbol{h} + \max_{P \in \mathcal{D}_{\ell_1}} \int_{\mathbb{R}^n_+} \mathrm{f}_P(\boldsymbol{p}) \Big(\boldsymbol{\pi}^\intercal \boldsymbol{p} - \alpha - \boldsymbol{\beta}^\intercal \boldsymbol{p} \Big) \, d\boldsymbol{p}. \tag{2.3.17}$$

If there exists $\boldsymbol{p} \in \mathbb{R}^n_+$, such that $\boldsymbol{\pi}^\intercal \boldsymbol{p} - \alpha - \boldsymbol{\beta}^\intercal \boldsymbol{p} \geq 0$, then g is unbounded:

$$g(\alpha, \boldsymbol{\beta}) = \begin{cases} \alpha + \boldsymbol{\beta}^\intercal \boldsymbol{h} & \text{if } \boldsymbol{\pi}^\intercal \boldsymbol{p} - \alpha - \boldsymbol{\beta}^\intercal \boldsymbol{p} \leq 0, \\ +\infty & \text{otherwise.} \end{cases} \tag{2.3.18}$$

Finally, disregarding the unbounded case, the dual problem for (2.3.11) is

$$\min_{\alpha, \boldsymbol{\beta}} \alpha + \boldsymbol{\beta}^\intercal \boldsymbol{h} \tag{2.3.19}$$

$$\text{s.t. } \boldsymbol{\beta} \geq \mathbf{0} \tag{2.3.20}$$

$$\alpha + \boldsymbol{\beta}^\intercal \boldsymbol{p} \geq \boldsymbol{\pi}^\intercal \boldsymbol{p} \quad \forall \boldsymbol{p} \in \mathbb{R}^n_+. \tag{2.3.21}$$

The problem satisfies conic duality [155], thus strong duality holds. Therefore, an optimal solution to the dual is also optimal w.r.t. (2.3.11). By transforming the problem further, we obtain

$$\min_{\alpha, \boldsymbol{\beta}} \alpha + \boldsymbol{\beta}^\intercal \boldsymbol{h} \tag{2.3.22}$$

$$\text{s.t. } \boldsymbol{\beta} \geq \mathbf{0} \tag{2.3.23}$$

$$(\boldsymbol{\pi}^\intercal - \boldsymbol{\beta}^\intercal) \boldsymbol{p} \leq \alpha \quad \forall \boldsymbol{p} \in \mathbb{R}^n_+. \tag{2.3.24}$$

Next, the value of the left hand side expression in (2.3.24) must be investigated. With respect to variable α, there are two possible cases to consider:

1. $\exists j \in \mathcal{J} : \beta_j < \pi_j$. Then, because for $p_j \to +\infty$, left hand side of (2.3.24) goes to infinity, and also $\alpha \to +\infty$. Therefore, in this case, (2.3.22)–(2.3.24) becomes infeasible.

2. $\boldsymbol{\pi} \leq \boldsymbol{\beta}$. Then $\alpha = 0$ and then (2.3.22)–(2.3.24) can be written as

$$\min_{\boldsymbol{\beta}} \boldsymbol{\beta}^\intercal \boldsymbol{h} \tag{2.3.25}$$

$$\text{s.t. } \boldsymbol{\beta} \geq \mathbf{0} \tag{2.3.26}$$

$$\boldsymbol{\pi} \leq \boldsymbol{\beta}, \tag{2.3.27}$$

   and thus $\boldsymbol{\beta} = \boldsymbol{\pi}$.

This investigation shows that the optimal value for the multipliers are: $\alpha = 0$ and $\boldsymbol{\beta} = \boldsymbol{\pi}$, and the solution to the dual problem is $0 + \boldsymbol{\pi}^\intercal \boldsymbol{h} = \boldsymbol{\pi}^\intercal(\hat{\boldsymbol{\mu}} + \gamma_1 \cdot \hat{\boldsymbol{\Sigma}}^{1/2}\mathbf{1})$. Substituting the result into the outer minimization problem from (2.3.10), we have

$$\min_{\boldsymbol{\pi} \in \boldsymbol{\Pi}} \boldsymbol{\pi}^\intercal(\hat{\boldsymbol{\mu}} + \gamma_1 \cdot \hat{\boldsymbol{\Sigma}}^{1/2}\mathbf{1}) = \boldsymbol{\pi}^\intercal \hat{\boldsymbol{\mu}} + \gamma_1 \cdot \mathbf{1}^\intercal \hat{\boldsymbol{\Sigma}}^{1/2}\boldsymbol{\pi} = \boldsymbol{\pi}^\intercal \hat{\boldsymbol{\mu}} + \gamma_1 \cdot \left\|\hat{\boldsymbol{\Sigma}}^{1/2}\boldsymbol{\pi}\right\|_1, \tag{2.3.28}$$

where the last equality follows from the fact that $\hat{\boldsymbol{\Sigma}}^{1/2} \in \mathcal{C}_+^{\boldsymbol{\Pi}}$.

□

Ambiguity set $\mathcal{D}_{\ell_1}$ from Proposition 2 is relatively simple as it imposes the upper limit on the first moment of the random variables only. Utilizing ambiguity sets based only on the first moment is not uncommon, see for example, [158]. The main reason why they are being used is to improve the computational tractability of the resulting problem. On the other hand, our ambiguity set does not disregard the second moment. As it was explained in Section 2.3.2, the second moment is reflected in the same way as ML algorithms treat model complexity via regularization. This was shown in Proposition 2, which explains the link between DR-PTFT(ℓ_1) and ambiguity set $\mathcal{D}_{\ell_1}$ where parameter γ_1 is used to control the robustness of the solution. The principal advantage of ambiguity set $\mathcal{D}_{\ell_1}$ is its favorable computational properties, as reflected by the following complexity characterization:

Proposition 3. *Problem DR-PTFT(ℓ_1) is solvable in $\mathcal{O}(n \log n)$ time when $\hat{\boldsymbol{\Sigma}}^{1/2}$ is a diagonal matrix; and in $\mathcal{O}(n^2)$ when $\hat{\boldsymbol{\Sigma}}^{1/2} \in \mathcal{C}_+^{\boldsymbol{\Pi}}$.*

Proof. Using the fact that $\operatorname{diag}(\hat{\boldsymbol{\mu}}) \in \mathcal{C}_+^{\boldsymbol{\Pi}}$, $\hat{\boldsymbol{\Sigma}}^{1/2} \in \mathcal{C}_+^{\boldsymbol{\Pi}}$, $\gamma_1 \geq 0$ and $\mathcal{C}_+^{\boldsymbol{\Pi}}$ is a convex cone, problem DR-PTFT(ℓ_1) can be reformulated as

$$\begin{aligned}\min_{\boldsymbol{\pi} \in \boldsymbol{\Pi}} \hat{\boldsymbol{\mu}}^\intercal \boldsymbol{\pi} + \gamma_1 \cdot \|\hat{\boldsymbol{\Sigma}}^{1/2}\boldsymbol{\pi}\|_1 &= \min_{\boldsymbol{\pi} \in \boldsymbol{\Pi}} \left\| \left(\operatorname{diag}(\hat{\boldsymbol{\mu}}) + \gamma_1 \cdot \hat{\boldsymbol{\Sigma}}^{1/2}\right) \boldsymbol{\pi} \right\|_1 \\ &= \min_{\boldsymbol{\pi} \in \boldsymbol{\Pi}} \mathbf{1}^\intercal \left(\operatorname{diag}(\hat{\boldsymbol{\mu}}) + \gamma_1 \cdot \hat{\boldsymbol{\Sigma}}^{1/2}\right) \boldsymbol{\pi}. \end{aligned} \tag{2.3.29}$$

Next, let us denote $\boldsymbol{h} = \mathbf{1}^\intercal \left(\operatorname{diag}(\hat{\boldsymbol{\mu}}) + \gamma_1 \cdot \hat{\boldsymbol{\Sigma}}^{1/2}\right) \in \mathbb{R}^n$. We can see that the problem (2.3.29) is tantamount to deterministic $P||\sum C_j$ with job durations given

by $\boldsymbol{h}$. There are known, efficient polynomial exact algorithms for the problem, based on sorting (for more details, refer to, e.g., [39, p. 133–134]). For the convenience of the reader, we present the full procedure in Algorithm 2.1. Vector $\boldsymbol{h}$ can be computed in $\mathcal{O}(n^2)$ (line 1), and as a result, the overall complexity is $\mathcal{O}(n^2 + n \log n) = \mathcal{O}(n^2)$. When $\hat{\boldsymbol{\Sigma}}^{1/2}$ is diagonal, the complexity is just $\mathcal{O}(n \log n)$. Note that when $\hat{\boldsymbol{\Sigma}}^{1/2}$ is not part of the input, then it needs to be computed from covariance matrix $\hat{\boldsymbol{\Sigma}}$ first, which can be done in $\mathcal{O}(n^3)$ time. □

The algorithm given by Proposition 3 is formulated in Algorithm 2.1. At line 2, the jobs are sorted in non-increasing order, by the weight defined in $\boldsymbol{h}$, which takes $\mathcal{O}(n \log n)$ time. Then, in the loop at lines 3–5, they are sequentially inserted into the solution. Each time a job is assigned to a machine with the least jobs assigned so far. In $\boldsymbol{\pi}$ representation, we only store the number of jobs preceding the job J on the machine, so the exact machine number is not computed. Each operation in the loop takes $\mathcal{O}(1)$ time, thus the entire loop takes $\mathcal{O}(n)$ time. In conclusion, line 1 determines the overall time complexity, depending on the form and values of the covariance matrix. We refer to this algorithm as *Sort Optimizer with Ravishing Technique* (i.e., SORT(ℓ_1)).

Algorithm 2.1: Sort Optimizer with Ravishing Technique (SORT(ℓ_1))

input : estimates of the parameters: $\hat{\boldsymbol{\mu}}$, $\hat{\boldsymbol{\Sigma}}^{1/2}$, trade-off parameter γ_1.
output : optimal solution $\boldsymbol{\pi}$.

1 $\boldsymbol{h} \leftarrow \mathbf{1}^\intercal \left(\operatorname{diag}(\hat{\boldsymbol{\mu}}) + \gamma_1 \cdot \hat{\boldsymbol{\Sigma}}^{1/2} \right)$
2 jobs ← sort jobs in non-increasing order, job $j \in \mathcal{J}$ has weight h_j
3 **for** $j = 1, 2, \ldots, n$ **do**
4 $\quad \mathsf{J} \leftarrow \mathsf{jobs}(j)$ `// take the next job`
5 $\quad \pi_{\mathsf{J}} \leftarrow \lceil \frac{j}{m} \rceil$
6 **return** $\boldsymbol{\pi}$

Obviously, when $\hat{\boldsymbol{\Sigma}}^{1/2} \notin \mathcal{C}_+^{\boldsymbol{\Pi}}$, then Proposition 3 does not apply, and it is an open question whether a polynomial algorithm for this case exists as well. In any case, DR-PTFT(ℓ_1) can be still expressed as a mixed-integer linear program (2.3.29) and solved by a general-purpose solver. As it will be shown in experiments in Section 2.5, even this method is very efficient, while allowing to tackle any covariance matrix.

2.4 Multi-objective optimization perspective

The purpose of this section is to point out a relation between the form of the objective function of the problem and the multi-objective optimization. In contrast to the previous sections, the aim here is to discuss some practical concerns related to the solution of the problem (2.3.5). One of them the obvious questions that the decision maker faces is, how to set the value of γ_1 parameter. Although there are some methods of how to set γ_1, e.g., to which extent we want to cover the target distribution [55], these will not provide the price to be paid for such a solution beforehand. Thus, we will argue that obtaining a single solution for some γ_1 is not likely to be very useful in practice. Instead, we will provide a method for uniform

sampling of solutions in the Pareto front, which exhibits different optimal trade-offs between the mean and variance.

Finally, we reveal that the objective function of problem (2.3.3) directly optimizes the metrics used to assess its out-of-sample performance. We identify this as a striking difference from many other scheduling problems and their evaluation in DRO scheduling literature, where this correspondence is often not present.

2.4.1 Relation to multi-objective optimization

First, let us introduce the two solution quality metrics of a robust solution used in [44] — *robust price* (RP) and *robust benefit* (RB). They are defined with respect to some testing probability distribution $P \in \mathcal{P}_0(\mathbb{R}^n)$ of processing times $\tilde{\boldsymbol{p}} \sim P$:

$$\mathrm{RP}\big(\boldsymbol{\pi}^R\big) = \Big(\mathbb{E}_P\big[f(\boldsymbol{\pi}^R, \tilde{\boldsymbol{p}})\big] - \mathbb{E}_P\big[f(\boldsymbol{\pi}^D, \tilde{\boldsymbol{p}})\big]\Big) / \mathbb{E}_P\big[f(\boldsymbol{\pi}^R, \tilde{\boldsymbol{p}})\big], \tag{2.4.1}$$

$$\mathrm{RB}\big(\boldsymbol{\pi}^R\big) = \Big(\mathrm{Var}_P\big[f(\boldsymbol{\pi}^D, \tilde{\boldsymbol{p}})\big]^{1/2} - \mathrm{Var}_P\big[f(\boldsymbol{\pi}^R, \tilde{\boldsymbol{p}})\big]^{1/2}\Big) / \mathrm{Var}_P\big[f(\boldsymbol{\pi}^R, \tilde{\boldsymbol{p}})\big]^{1/2}. \tag{2.4.2}$$

The solution of the robust formulation (2.3.5) is denoted as $\boldsymbol{\pi}^R$, while $\boldsymbol{\pi}^D$ is an optimal solution of the problem with deterministic processing times, i.e., (2.3.5) with $\gamma_1 = 0$, with the processing times set as their true (or estimated) means. Thus, $\mathrm{RP}(\boldsymbol{\pi}^R)$ measures the relative difference between the expected quality of the robust and deterministic solutions $\boldsymbol{\pi}^R$ and $\boldsymbol{\pi}^D$, respectively. Similarly, $\mathrm{RB}(\boldsymbol{\pi}^R)$ is the relative difference of standard deviations of solutions under the distribution P. Note that the testing distribution P may or may not be known; nevertheless, we assume that one has access to a finite sample set from P.

Example 4. Assume that for an instance with 4 jobs and a single machine, the parameters of the jobs are estimated as $\hat{\boldsymbol{\mu}} = (1.96, 1.39, 1.39, 1.39)$ and $\hat{\boldsymbol{\Sigma}} = \mathrm{diag}\,(0, 0, 0.072, 0.209)$. For this instance, consider two solutions: the deterministic (or SP, they are equivalent) one $\boldsymbol{\pi}^D = (1, 2, 3, 4)$ and the robust one $\boldsymbol{\pi}^R = (4, 3, 2, 1)$ (not necessarily optimal). As an example to demonstrate the above-defined quantities, let us pick a testing distribution $P = \mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})$ to display the empirical densities of the objective values of the aforementioned solutions (shown in Figure 2.3). Under this test distribution, $\boldsymbol{\pi}^D$ is an optimal solution for SP formulation. However, we see that for a price measured in terms of $\mathrm{RP}(\boldsymbol{\pi}^R)$, robust solution $\boldsymbol{\pi}^R$ achieves a smaller variance and also a smaller worst-case objective value, thus being a more stable solution for a risk-aware decision maker.

Obviously, one would like to have RP the smallest possible and RB the largest. Different robust solutions $\boldsymbol{\pi}^R$ can perform differently under these two, generally conflicting criteria. Thus, such a problem can be formulated from the perspective of the multi-objective optimization with the two criteria $g(\boldsymbol{\pi}) \in \mathbb{R}^2$:

$$\boldsymbol{\pi}^R = \arg\min_{\boldsymbol{\pi} \in \boldsymbol{\Pi}} g(\boldsymbol{\pi}) = \arg\min_{\boldsymbol{\pi} \in \boldsymbol{\Pi}} \big(\mathrm{RP}(\boldsymbol{\pi}), -\mathrm{RB}(\boldsymbol{\pi})\big). \tag{2.4.3}$$

What will be shown next is that formulation (2.3.3), in fact, solves this multi-objective optimization problem with the *scalarization approach* [66]. To show that,

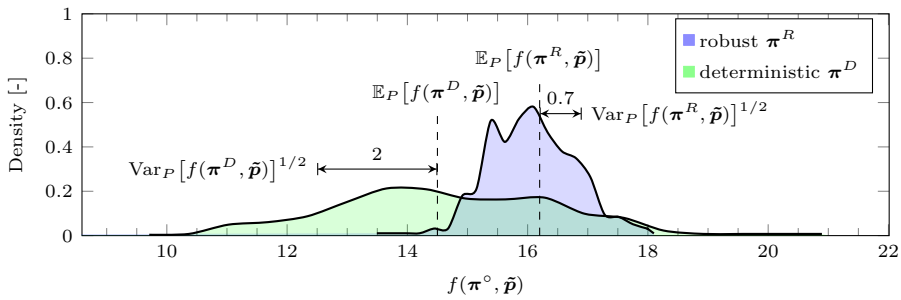


Figure 2.3: Illustration of robust price and robust benefit for different solutions $\boldsymbol{\pi}^\circ$, $\circ \in \{R, D\}$.

let us analyze the two criteria separately. For the RP, we have that

$$\begin{aligned}\boldsymbol{\pi}^\star_{RP} = \arg\min_{\boldsymbol{\pi}^R \in \boldsymbol{\Pi}} \mathrm{RP}(\boldsymbol{\pi}^R) &= \arg\min_{\boldsymbol{\pi}^R \in \boldsymbol{\Pi}} \left\{ 1 - \frac{\mathbb{E}_P\big[f(\boldsymbol{\pi}^D, \tilde{\boldsymbol{p}})\big]}{\mathbb{E}_P\big[f(\boldsymbol{\pi}^R, \tilde{\boldsymbol{p}})\big]} \right\} \\ &= \arg\min_{\boldsymbol{\pi}^R \in \boldsymbol{\Pi}} \mathbb{E}_P\big[f(\boldsymbol{\pi}^R, \tilde{\boldsymbol{p}})\big] \\ &= \arg\min_{\boldsymbol{\pi}^R \in \boldsymbol{\Pi}} \boldsymbol{\mu}^\intercal \boldsymbol{\pi}^R \approx \arg\min_{\boldsymbol{\pi}^R \in \boldsymbol{\Pi}} \hat{\boldsymbol{\mu}}^\intercal \boldsymbol{\pi}^R,\end{aligned}$$

where $\boldsymbol{\mu}$ is (potentially unknown) mean value of $\tilde{\boldsymbol{p}}$ and $\hat{\boldsymbol{\mu}}$ is the sample mean obtained from P. The second equality follows from the fact that $\mathbb{E}_P[f(\boldsymbol{\pi}^D, \tilde{\boldsymbol{p}})]$ is a constant as long as P is fixed. Analogously for RB, we obtain

$$\begin{aligned}\boldsymbol{\pi}^\star_{RB} = \arg\max_{\boldsymbol{\pi}^R \in \boldsymbol{\Pi}} \mathrm{RB}(\boldsymbol{\pi}^R) &= \arg\max_{\boldsymbol{\pi}^R \in \boldsymbol{\Pi}} \left\{ \frac{\mathrm{Var}_P[f(\boldsymbol{\pi}^D, \tilde{\boldsymbol{p}})]^{1/2}}{\mathrm{Var}_P[f(\boldsymbol{\pi}^R, \tilde{\boldsymbol{p}})]^{1/2}} - 1 \right\} \\ &= \arg\min_{\boldsymbol{\pi}^R \in \boldsymbol{\Pi}} \mathrm{Var}_P[f(\boldsymbol{\pi}^R, \tilde{\boldsymbol{p}})]^{1/2} \\ &= \arg\min_{\boldsymbol{\pi}^R \in \boldsymbol{\Pi}} \|\boldsymbol{\Sigma}^{1/2} \boldsymbol{\pi}^R\|_2 \approx \arg\min_{\boldsymbol{\pi}^R \in \boldsymbol{\Pi}} \|\hat{\boldsymbol{\Sigma}}^{1/2} \boldsymbol{\pi}^R\|_2,\end{aligned}$$

where $\boldsymbol{\Sigma}$ is (potentially unknown) covariance and $\hat{\boldsymbol{\Sigma}}$ is the sample covariance of $\tilde{\boldsymbol{p}}$. Similarly, term $\mathrm{Var}_P[f(\boldsymbol{\pi}^D, \tilde{\boldsymbol{p}})]^{1/2}$ acts as a constant. Thus, it can be seen that $\boldsymbol{\pi}^\star_{RP} \in \boldsymbol{\Pi}$ minimizing $\hat{\boldsymbol{\mu}}^\intercal \boldsymbol{\pi}$ also minimizes $\mathrm{RP}(\cdot)$. Similarly, $\boldsymbol{\pi}^\star_{RB} \in \boldsymbol{\Pi}$ minimizing $\|\hat{\boldsymbol{\Sigma}}^{1/2} \boldsymbol{\pi}\|_2$ minimizes $-\mathrm{RB}(\cdot)$. Thus, it implies that (2.3.3) can be viewed as a scalarization method for the multi-objective optimization applied to (2.4.3). What else can be concluded is that formulation (2.4.3) suggests that a solution in the sense of ℓ_1 norm obtained from (2.3.9) is also likely to work well as a solution for multi-objective problem since $\|\boldsymbol{x}\|_1 \geq \|\boldsymbol{x}\|_2$ for any $\boldsymbol{x} \in \mathbb{R}^n$. Thus, its robust term acts as an upper bound on $\mathrm{Var}_P\left[f(\tilde{\boldsymbol{p}}, \boldsymbol{\pi})\right]^{1/2}$, which in turn leads to optimization of RB as well. Note that when one measures the out-of-sample performance in terms of RB and RP, then the optimization criterion matches the performance metric. This is not necessarily always the case with DRO scheduling problems described in the literature since they often just draw several samples from a mix of distribution (whose does not necessarily correspond to the worst-case distribution). After that, some quantities of the objective function are reported, such as the expected value,

maximum value or different quantiles. This leads us again back to the discussion in Section 2.2.2, where we have outlined some challenges when evaluating a DRO solution. Thus, we believe that this aspect of our problem worths pointing out.

In the following section, we describe an improved RP/RB trade-off parametrization, which provides a more uniform sampling of the Pareto front.

2.4.2 Pareto front sampling

As it was shown in the above section, parameter γ_1 in (2.3.1) can be used by the decision maker to control the trade-off between RP and RB of the resulting solution. However, an obvious disadvantage of such parametrization is that the actual value of γ_1 needed to achieve certain RB depends on the numerical scale of sample covariance matrix $\hat{\boldsymbol{\Sigma}}$. Concerning the practical use of such parametrization, it is difficult to guess desired values for γ_1. For a decision maker, an ideal parametrization would allow to choose any point on the RP/RB trade-off curve and obtain the desired balance between robustness and average performance of the system without having to re-run the solving procedure multiple times.

Unfortunately, a computationally efficient exact approach to this task might be hard to find. Instead, we propose a simple—yet useful—heuristic. The idea is to normalize the effect of parameter $\gamma_1 \geq 0$ with respect to the sample mean $\hat{\boldsymbol{\mu}}$ and covariance $\hat{\boldsymbol{\Sigma}}$. We introduce a single parameter $r \in [0, 1]$, that controls the emphasis between RP and RB. The problem with r becomes

$$\text{DR-PTFT}(\ell_p, r) \equiv \min_{\boldsymbol{\pi} \in \boldsymbol{\Pi}} \frac{1-r}{0.5n \cdot \hat{\boldsymbol{\mu}}^\intercal \mathbf{1}} \cdot \hat{\boldsymbol{\mu}}^\intercal \boldsymbol{\pi} + \frac{r}{\left\| 0.5n \cdot \hat{\boldsymbol{\Sigma}}^{1/2} \mathbf{1} \right\|_p^a} \cdot \left\| \hat{\boldsymbol{\Sigma}}^{1/2} \boldsymbol{\pi} \right\|_p^a, \quad (2.4.4)$$

where $a \in \mathbb{N}$ is used when the ℓ_p norm is raised to the a-th power (e.g., ℓ_2 norm squared, see Remark 1). The denominators play the role of normalization constants, by the values the both terms might likely attain. Technically, these values also depend on the number of machines m; however, we have observed that for small values of m it does not affect it heavily. Equation (2.4.4) is designed such that for $\boldsymbol{\pi} = \frac{n}{2} \cdot \mathbf{1}$ (note that such $\boldsymbol{\pi}$ is infeasible, yet it may represent an "averaged" solution for a small m), both terms are normalized and the overall value is invariant in respect to $r \in [0, 1]$, i.e.,

$$\frac{1-r}{0.5n \cdot \hat{\boldsymbol{\mu}}^\intercal \mathbf{1}} \cdot \hat{\boldsymbol{\mu}}^\intercal \cdot 0.5n \cdot \mathbf{1} + \frac{r}{\left\| 0.5n \cdot \hat{\boldsymbol{\Sigma}}^{1/2} \mathbf{1} \right\|_p^a} \cdot \left\| \hat{\boldsymbol{\Sigma}}^{1/2} \cdot 0.5n \cdot \mathbf{1} \right\|_p^a = 1 \quad \forall r \in [0, 1]\,.$$

Note that DR-PTFT$(\ell_p, r = 0)$ is equivalent to DR-PTFT$(\ell_p, \gamma_1 = 0)$, thus the solution of (2.4.4) with $r = 0$ resembles a Stochastic Programming solution as well.

To assess the benefits of the new parametrization, we have performed the following experiment. We generated 100 random instances, for each combination of $n \in \{10, 15, 20, 30, 50, 100, 150\}$ and $m \in \{3, 4, 5\}$. Each instance was solved both with ℓ_1 and ℓ_2 norms and with parametrizations using γ_1 and r. For each solution, RP and RB were calculated according to (2.4.1)–(2.4.2) and averaged over all instances. See the results in Figure 2.4. On the left-hand side, we can see RP/RB trade-off curves with the default parametrization using 10 values of $\gamma_1 \in \{0, 1, 2, \ldots, 9\}$, applied to both problems with ℓ_1 and ℓ_2 norm. We can see that

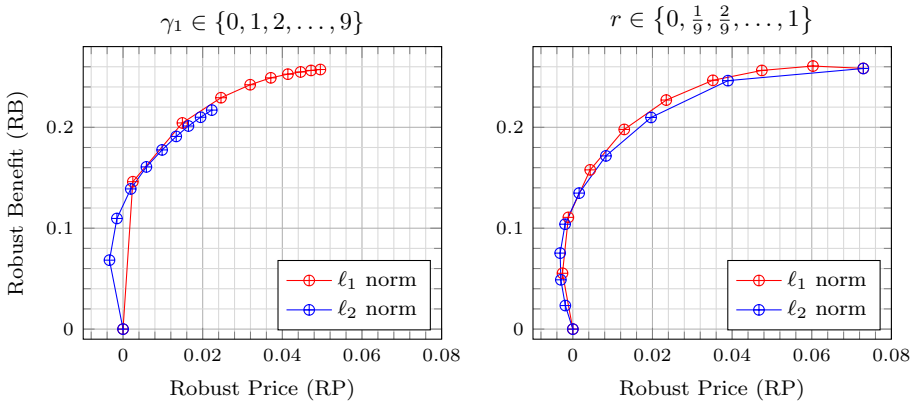


Figure 2.4: Distribution of solutions on RP/RB trade-off curves for different parametrizations.

in the case of ℓ_1 and ℓ_2 norm, the points on the Pareto front tend to get dense with the increasing value of γ_1 (i.e., the greater γ_1, the greater RB). On the right-hand side, we see the results for parametrization using $r \in \{\frac{0}{9}, \frac{1}{9}, \frac{2}{9}, \ldots, \frac{9}{9}\}$ (i.e., also 10 different values). We can observe that for ℓ_1 norm, this parametrization leads to solutions distributed evenly along the Pareto front. Similarly, for ℓ_2 norm, the parametrization using r also leads to a more even distribution of the solutions, than in the case of γ_1. Hence, in subsequent experiments, the parametrization with r will be utilized instead of γ_1.

2.5 Numerical experiments

In this section, we perform the experimental evaluation of the proposed methods. Specifically, for problem DR-PTFT(ℓ_1) we benchmark the sort-based method from Proposition 3 (denoted as SORT(ℓ_1)), and the MILP model given by Equation (2.3.28) (i.e., MILP(ℓ_1)). For problem DR-PTFT(ℓ_2), we evaluate the SOCP given by Equation (2.3.1) (i.e., SOCP(ℓ_2)), and the NOC-points Search Algorithm (denoted as NPSA(ℓ_2) introduced by Chang *et al.* [44]. Algorithm NPSA(ℓ_2) inspects a finite number of so-called NOC-points (*necessary optimality condition* points) that are suspected of being extreme. For more details, we refer the reader to [44].

In the case of problem DR-PTFT(ℓ_2^2), we assume the min-cost bipartite perfect matching with Hungarian algorithm [89] mentioned in Remark 1 (i.e., HUNGARIAN(ℓ_2^2)). We study the quality, robustness of the solutions, and computation times of the algorithms, for problems with respect to different norms, both for independent and dependent jobs.

2.5.1 Experimental setup

For experimental evaluation, we have used a workstation equipped with AMD Ryzen Threadripper 3990X @2.90GHz (during computations boosted to about 4.00 GHz), 64 GB RAM, running Windows 10 Pro. Due to the operating system shortcomings

(limited support for more than 64 cores per CPU), Simultaneous Multi-Threading (SMT) was disabled. All algorithms have been implemented in Python 3. As a solver, we have used Gurobi 9.0 with default parameter configuration. The source codes and test instances can be found at our Github page. When measuring time, a single run of an algorithm utilizes a single CPU core. Otherwise, all 64 physical cores are available for Gurobi solver (in case of solving MILP(ℓ_1) and SOCP(ℓ_2)); however, it rarely utilizes more than 12 and never more than 32.

2.5.2 Evaluation protocol

To test the robustness and quality of solutions, we adopted the evaluation protocol proposed in [44]. The protocol focuses on independent jobs and compares robust and deterministic solutions from the perspective of average quality and stability. It proceeds as follows:

1. Generate 1,000 random $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$, representing the true moments of distributions. The mean duration for each job μ_j is generated as $\mu_j \sim \mathcal{U}(10, 60)$ and its standard deviation is distributed $\sigma_j \sim \mathcal{U}(0.1\mu_j, 0.9\mu_j)$.

2. For each $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$, the protocol generates 10,000 random samples (i.e., one sample is a single realization of $\tilde{\boldsymbol{p}}$) from the mix of distributions: Gamma, uniform, normal and Laplace, with the given fixed mean and variance. From each distribution, 2,500 samples are taken. Then, the samples are shuffled randomly to simplify the next step.

3. From each set of 10,000 random samples, the protocol creates several sets of subsamples by selecting the given samples. When $n \leq 20$ it creates 100 subsample sets and when $n > 20$ it is only 20 of them. The size of the subsample set is determined by a sample rate (referred to as *S-rate* in [44]). If it is not specified otherwise, we assume 10 subsamples in each subsample set. Each subsample set is used to define the ambiguity set by estimates $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\sigma}}$ of the given true moments $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$. Solution of (2.3.5) and its deterministic counterpart result in $\boldsymbol{\pi}^R$ and $\boldsymbol{\pi}^D$ respectively.

4. For each $\boldsymbol{\pi}^\circ$, $\circ \in \{R, D\}$, their expectations $\mathbb{E}_P[f(\boldsymbol{\pi}^\circ, \tilde{\boldsymbol{p}})]$ and standard deviations $\mathrm{Var}_P[f(\boldsymbol{\pi}^\circ, \tilde{\boldsymbol{p}})]^{1/2}$ are estimated from $500,000$ samples from the mix of distributions with the given fixed true moments $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$. The samples are generated in the same way as in the third step.

5. Estimate RP and RB for each distribution given by its true moments $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ via sample mean along the subsamples.

Note that since [44] dealt with independent jobs only, it did not provide a protocol for dependent jobs' durations. Hence, in this paper, we propose the following modification for dependent jobs. True expected values of job durations were taken from the uniform distribution $\mathcal{U}(10, 50)$, but we replace the generation of the true $\boldsymbol{\sigma}$, with a full true covariance matrix $\boldsymbol{\Sigma}$. In order to cover a wide range of distributions, we consider covariance matrices to be samples from the Wishart distribution. Wishart distribution is a distribution over symmetric positive definite matrices, possibly with some negative off-diagonal elements. It appears as a distribution

over sample covariance matrices produced by samples from a multivariate normal distribution. In our experiments, the distribution of covariances is given as $\hat{\boldsymbol{\Sigma}} \sim W_n(\nu, \lambda \cdot \boldsymbol{I}) + \operatorname{diag} \boldsymbol{d}$ where $\boldsymbol{d} \sim \mathcal{N}(\boldsymbol{\mu'}, 1)$. Hence, they are samples from Wishart distribution $W_n(\,\cdot\,,\,\cdot\,)$ with ν degrees of freedom and a scale matrix given as $\lambda \cdot \boldsymbol{I}$ of $n \times n$ real symmetric PSD matrices. Furthermore, we add a normally distributed vector $\boldsymbol{d}$ to the diagonal to control the likelihood that $\hat{\boldsymbol{\Sigma}}$ is copositive with respect to $\boldsymbol{\Pi}$. We have chosen $\boldsymbol{\mu'} = 5 \cdot \mathbf{1}$, $\lambda = 1$ and $\nu = n + 40$. After a true covariance matrix $\boldsymbol{\Sigma}$ is generated, an algorithm has then access to a limited number of samples from a mix of multivariate normal and multivariate uniform distributions with covariance $\boldsymbol{\Sigma}$. The reason behind using just these two, and not all four types as in the independent case, is that for many distributions, there is no single, well-established multivariate extension. It follows from the fact, that the covariance has a good meaning as a measure of variability for symmetric, elliptically contoured distributions. That is, it is not straightforward to define a natural extension of these distributions, and—as a result—multiple possible generalizations exist, often emerge from different applications (see, e.g., [94]). Therefore, in our experiments addressing dependent jobs' durations, we have limited ourselves to multivariate uniform and multivariate Gaussian distributions. The details regarding generating covariance matrices are covered in Sections 2.5.4 and 2.5.5.

In the following sections, we utilize RP and RB obtained via the above evaluation protocol as the primary performance metrics based on the out-of-sample evaluation.

2.5.3 Independent jobs: quality, stability and performance

In this section, we compare the quality of solutions obtained by assuming different objective functions, for the case of independent jobs. In addition, we present computational times of different algorithms. All the variants of the problem are solved exactly with respect to their objective functions, i.e., ℓ_1 norm is solved with our SORT(ℓ_1) method, ℓ_2 norm with SOCP(ℓ_2) and ℓ_2^2 norm with HUNGARIAN(ℓ_2^2). In the evaluation of the computational times, we also present the state-of-the-art method NPSA(ℓ_2) proposed in [44].

Effect of the used ℓ_p norm The first experiment studies the trade-off between RP and RB when different norms are used. The instances are generated and evaluated according to the protocol described in Section 2.5.2. The results for instances with $n \in \{10, 15, 20, 30, 50, 100, 150\}$ and $m \in \{3, 4, 5\}$ are displayed in Figure 2.5 altogether, as we have observed that the general shape of trade-off curves do not depend heavily on the particular values of m and n. A more detailed comparison of the differences between the individual curves with particular values of m and n for ℓ_1 norm is displayed in Figure 2.6.

The coordinates of each point on the curve in Figure 2.5 correspond to RP and RB of the solution averaged over $100 \times 7 \times 3 = 2100$ instances. Each point is obtained with a different value of r parameter, i.e., the position of a point on the curve is completely parameterized by r. We have taken 100 values of r distributed uniformly on $[0, 1]$ for each method. In Figure 2.5, it can be seen that all the methods achieve comparable trade-off curves in terms of ℓ_1, ℓ_2 and ℓ_2^2 (although technically speaking ℓ_2^2 is not a norm), yet each norm gets more advantage in

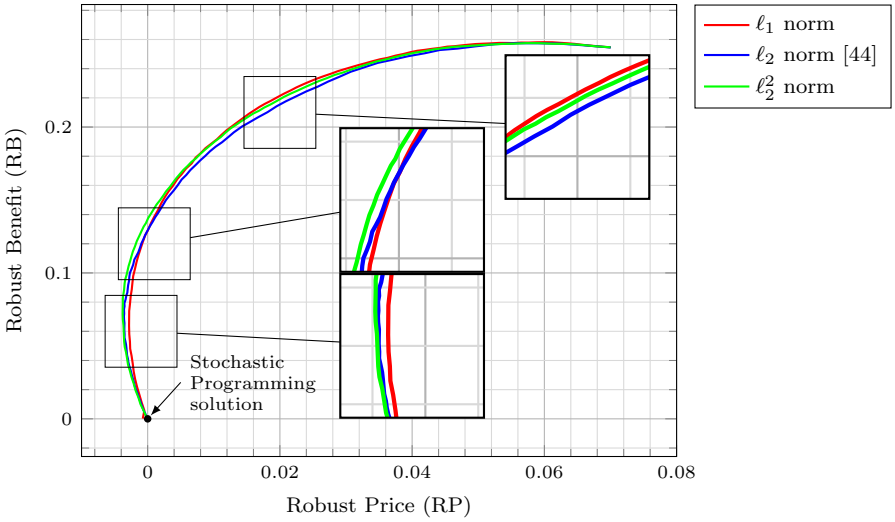


Figure 2.5: Trade off between RP and RB for ℓ_1, ℓ_2 and ℓ_2^2 formulations with independent jobs.

different parts of the curve. Thus, they are incomparable but essentially identical. What is particularly interesting, is that all methods allow obtaining solutions with a positive RB while having a negative RP. Thus, in this setting, a free lunch is possible and one can get a more stable and cheaper solution than the deterministic one. It is likely due to the fact that the deterministic solution completely lacks the information about variances, which robust solutions can take advantage of. An interesting question to ask is, how the choice of the particular norm affects the decision maker. As we have seen in Figure 2.5, ℓ_1 norm achieves similar trade-offs between RP and RB as ℓ_2 norm. On one hand, the advantage of ℓ_2 norm may be seen by the fact that it exactly resembles the standard deviation of the solution objective, thus the weight of the norm term can be directly interpreted. On the other hand, ℓ_1 norm does not offer this, but its advantage for the decision maker lies in its practical computational tractability. In other words, with ℓ_1 norm it is affordable to compute the whole RP/RB curve and pick any solution that suits the requirements of the decision maker.

Computational times In this experiment, we provide a measuring of time needed to solve the problem depending on the used norm and algorithm. In Figure 2.7 and Table 2.1, we display comparison of computational times SOCP(ℓ_2), SORT(ℓ_1), MILP(ℓ_1), and NPSA(ℓ_2). The x axis in Figure 2.7 represents instances with different values of n and m. Axis y depicts the computational time in seconds. Note the logarithmic scale of the y axis. Each data point is given by an average of over 100 instances. One can see that SOCP(ℓ_2) is computationally the most expensive, while MILP(ℓ_1) and SORT(ℓ_1) are far less demanding. Indeed, MILP(ℓ_1) is 10 times faster than SOCP(ℓ_2) while SORT(ℓ_1) is even more than three orders of magnitude faster than SOCP(ℓ_2) for the largest instances. In addition, all the methods solving DR-PTFT(ℓ_1) have consistent running times — the error bars (± 1 sigma) are virtually

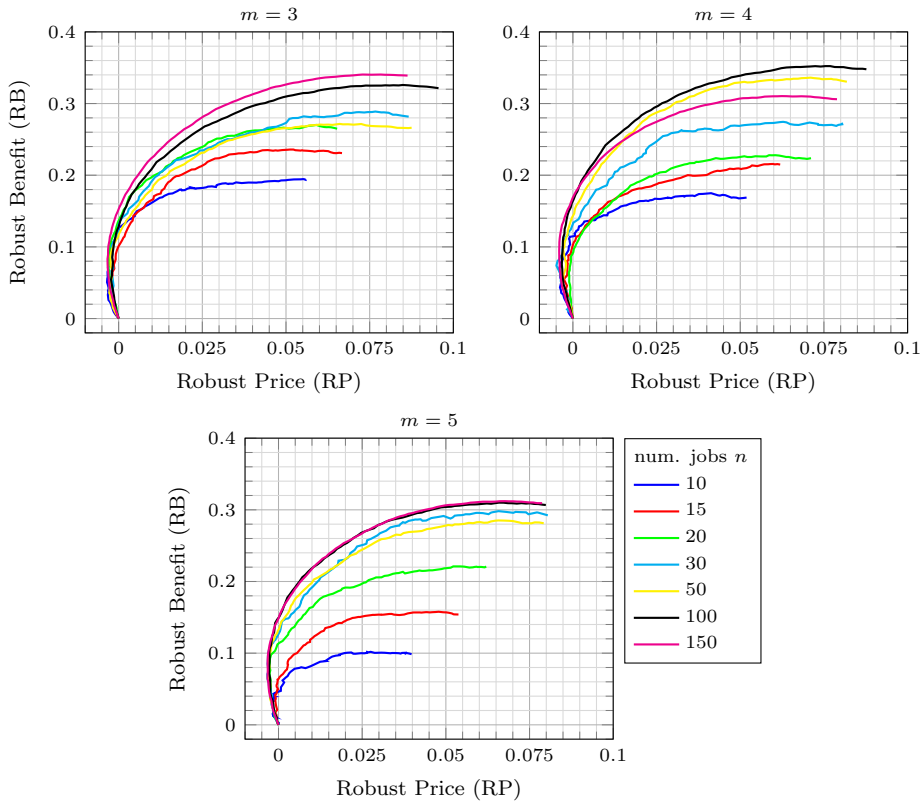


Figure 2.6: Relation between RP/RB curve and instance parameters for ℓ_1 formulation.

non-existent. While it is not surprising for SORT(ℓ_1), in the case of MILP(ℓ_1) this phenomenon is explained by the observation that the solver has solved every problem instance in the root node, yielding to consistent times. For more detailed results see Table 2.1. There, we can see that for a larger number of machines m, the problem becomes simpler. For the largest instances with $n = 150$ jobs, SOCP(ℓ_2) has lower average number of visited nodes than for $n = 100$. This is due to occasional timeouts that have occurred on the largest instances, especially for larger values of r which put more emphasis on the robust term. Nevertheless, the average optimality gaps reported by the solver were in these cases very small.

Next, we have compared our algorithms to the state-of-the-art method NPSA(ℓ_2) proposed in [44]. It is an exact algorithm solving the same problem as SOCP(ℓ_2), but with independent jobs only. For the sake of comparison, we have scaled their runtimes by the relative single-core performance of their and our CPU (approximately 1.5 times). It can be observed that even though NPSA(ℓ_2) outperforms SOCP(ℓ_2), it is still much slower than SORT(ℓ_1). The error bars are not displayed, as they were not reported in their paper.

To summarize, the results show that (i) one can obtain comparable RP/RB trade-offs for the problem with ℓ_1 norm as for ℓ_2 norm, and (ii) the computational time for the problem with ℓ_1 is much shorter than with ℓ_2 norm. Moreover, ℓ_1

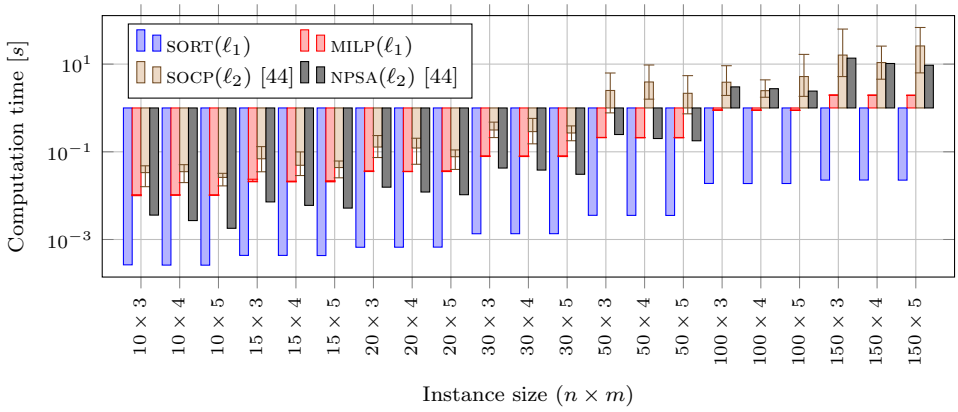


Figure 2.7: Comparison of averaged computational times for different methods and instance sizes for problem with independent jobs and $\gamma_1 = 4$.

formulation enjoys polynomial-time exact algorithm, while for ℓ_2 there is no such guarantee on the time required for calculations. The likely non-existence of this polynomial time bound is reflected in a higher spread of computation times observed for SOCP(ℓ_2), with some instances taking over 1000 times longer to be solved than the average. Although it may not be that dramatic for instances benchmarked in Figure 2.7, this difference even increases with the size of the instance, turning the solution to instances with more than hundreds of jobs nearly intractable with SOCP(ℓ_2). Such large instances occur, e.g., when scheduling unit test batches (as described in the introduction), with tens of thousands of jobs possible.

2.5.4 Dependent jobs: quality, stability and performance

In this section, we study the computational properties of the problem with dependent jobs. Specifically, we investigate: (i) what are the benefits of using information about the correlations between jobs to quality/stability of the schedule, and (ii) what amount of data is needed to reliably estimate covariance matrix such that it brings a meaningful benefit over just a diagonal covariance. Furthermore, we report computational times needed to solve such a problem, depending on the properties of the covariance matrix and the used norm.

Effect of the used ℓ_p norm In the first experiment, we compare RP/RB trade-off curves obtainable with ℓ_1 and ℓ_2 norms. The experiment assumes perfect information about the covariance, i.e., algorithms have access to the true covariance matrix. Since the solution of a single instance with $n = 10$ jobs with full covariance takes more than one hour to compute with SOCP(ℓ_2) model, we have limited ourselves to instances only with $n = 10$ and $m = 3$ in this experiment. As the solution of ℓ_1 formulation is much faster, we took 100 values of r uniformly distributed on $[0, 1]$ while for ℓ_2 formulation, we have used 25 values of r. The results can be seen in the left part of Figure 2.8.

Similarly, as for the independent case, the trade-off curves are very similar to each other, but here the solution with ℓ_2 achieves slightly better RB values. We

		SORT(ℓ_1)	NPSA(ℓ_2) [44]	MILP(ℓ_1)		SOCP(ℓ_2) [44]		
n	m	time [s]	time [s]	time [s]	nodes [$-$]	time [s]	nodes [$-$]	gap [%]
10	3	0.0003	0.0036	0.010	0.0	0.03	2.76	0.001
	4	0.0003	0.0027	0.010	0.0	0.04	2.36	0.000
	5	0.0003	0.0018	0.010	0.0	0.03	1.94	0.000
15	3	0.0004	0.0072	0.021	0.0	0.07	5.64	0.001
	4	0.0004	0.0060	0.021	0.0	0.05	5.93	0.001
	5	0.0004	0.0052	0.021	0.0	0.04	4.97	0.001
20	3	0.0007	0.0156	0.036	0.0	0.13	21.62	0.002
	4	0.0007	0.0121	0.036	0.0	0.12	11.34	0.002
	5	0.0007	0.0105	0.036	0.0	0.08	9.88	0.001
30	3	0.0014	0.0426	0.079	0.0	0.31	399.74	0.003
	4	0.0014	0.0383	0.079	0.0	0.29	139.16	0.003
	5	0.0014	0.0307	0.079	0.0	0.27	47.28	0.002
50	3	0.0036	0.2477	0.212	0.0	2.50	2377.50	0.003
	4	0.0035	0.2006	0.210	0.0	3.91	1885.92	0.007
	5	0.0035	0.1784	0.210	0.0	2.16	449.87	0.004
100	3	0.0190	3.0238	0.892	0.0	3.86	36029.88	0.015
	4	0.0189	2.7509	0.888	0.0	2.48	24225.20	0.010
	5	0.0189	2.4252	0.885	0.0	5.21	20566.85	0.012
150	3	0.0226	13.6820	1.964	0.0	16.07	12858.49	0.033
	4	0.0227	10.3278	1.953	0.0	10.84	13555.14	0.020
	5	0.0226	9.4144	1.942	0.0	25.95	15514.36	0.024

Table 2.1: Detailed comparison of performance indicators for independent jobs.

believe that the reason for it is related to the fact that PSD matrices are closely connected to ℓ_2 norm, which also takes a unique role among the different ℓ_p norms. That is, ℓ_2 is the only norm among ℓ_p norms which is induced by a scalar product, and by Riesz representation theorem, its dual norm is also ℓ_2. Actually, every scalar product on $\mathbb{R}^n$ corresponds to exactly one positive-definite $n \times n$ matrix. We believe the ℓ_2 norm is, therefore, more intuitive to work with when working with PSD matrices, and hence, leads to slightly better results. These differences did not play a significant influence for the case of independent jobs, but it seems to have a bigger impact for the dependent case. However, what will be shown in a subsequent experiment, when one does not have the perfect knowledge of covariance (i.e., it has to be estimated from a finite sample set), then the differences between ℓ_1 and ℓ_2 norms become negligible again. Thus, even with dependent jobs, both norms allow obtaining solutions of comparable quality/stability, especially considering the fact that the solution with ℓ_1 norm is much faster. Again, we see this as a benefit for the decision maker which can afford to compute the whole RP/RB trade-off curve and choose the desired balance between these two.

Full and diagonal covariances with perfect information The second experiment assesses the effect of using the full covariance matrix, assuming the perfect knowledge of it. Hence, in this setting, the algorithm has access to the true covari-

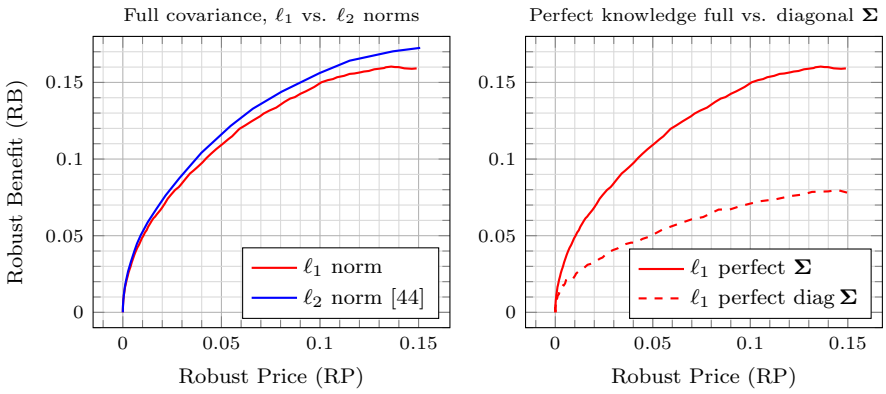


Figure 2.8: Trade-off between RP and RB with ℓ_1 and ℓ_2 formulations for dependent jobs with perfect covariance knowledge and its diagonal part.

ance matrix and we measure which values of RP/RB are achievable compared to the case when one uses just its diagonal part. The results are displayed in the right part of Figure 2.8. There, we have generated 100 instances with $n = 10$ jobs and $m = 3$ machines. Each curve is obtained with 100 values of r parameter controlling the trade-off between RP and RB. The solid curve is obtained using full covariance with the perfect information; hence, it represents an upper bound on the RP/RB curve. The dashed curve represents the performance of a solution when just the diagonal part of the covariance matrix is used, i.e., when (potential) dependency between jobs is ignored (but being tested against distributions with non-diagonal covariance). It can be seen that using the information from the full covariance matrix allows obtaining about two times larger RB for the same RP than using just its diagonal part.

Full covariance with imperfect information In practice, one might not have access to the true covariance, but rather the covariance needs to be estimated from empirical data. Hence, in the following experiment, we study how many data samples are needed to obtain an estimate of the covariance matrix, which actually produces an additional benefit over the ignorance of mutual correlations.

Instance	MILP-DIAG(ℓ_1)		MILP(ℓ_1)		SOCP(ℓ_2) [44]	
$n \times m$	time [s]	std [s]	time [s]	std [s]	time [s]	std [s]
8×3	6.44×10^{-3}	1.7×10^{-5}	7.29×10^{-3}	1.8×10^{-4}	0.14	0.08
9×3	7.83×10^{-3}	1.1×10^{-5}	8.66×10^{-3}	8.1×10^{-5}	0.30	0.22
10×3	9.41×10^{-3}	5.6×10^{-5}	1.04×10^{-2}	1.1×10^{-4}	3.09	3.88
11×3	1.10×10^{-2}	2.4×10^{-5}	1.22×10^{-2}	1.3×10^{-4}	100.13	787.20
12×3	1.29×10^{-2}	3.9×10^{-5}	1.46×10^{-2}	9.9×10^{-4}	488.95	1087.30

Table 2.2: Computational times of different solving methods and instance sizes for dependent jobs.

The setup is the following. We generate 100 instances with $n = 10$ and $m = 3$.

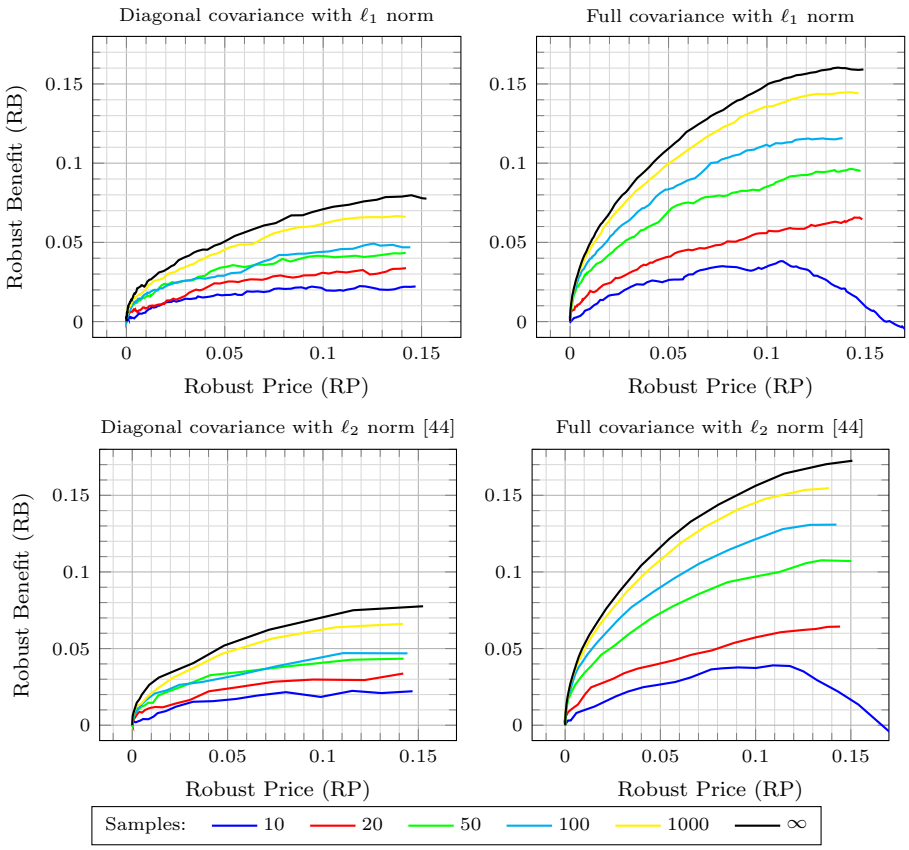


Figure 2.9: Effect of S-rate to RP/RB curve with dependent jobs.

Then, we follow the evaluation protocol described in Section 2.5.2, with the exception that S-rate is now a parameter whose effect is investigated. The tested values of S-rate are $0.001, 0.002, 0.005, 0.01$ and 0.1, which corresponds to $10, 20, 50, 100$ and 1000 samples used for the estimation of mean $\hat{\boldsymbol{\mu}}$ and covariance $\hat{\boldsymbol{\Sigma}}$.

The results are shown in Figure 2.9, where the results on the left side ignore the mutual correlations and the results on the right assume the full covariance.

Curves for ℓ_1 norm were obtained using 100 values of r parameter wheres the curves for ℓ_2 with 25 values (since its solution is much more computationally demanding). The top curve, denoted as ∞ samples, corresponds to results when the algorithm has the perfect knowledge of the covariance. When the number of samples is decreasing, the achievable RB for a fixed RP is decreasing as well. However, it can be seen that when the number of samples drops below a certain level, then we may obtain a solution that performs even worse than the solution ignoring mutual correlations, i.e., assuming independence. This effect can be observed in Figure 2.9 for the full covariance estimated with just 10 samples, where for some RP values the solution has worse RB than when just the diagonal is estimated. Again, the trade-off curves are comparable for both ℓ_1 and ℓ_2 norms. Moreover, the computational results displayed in Table 2.2 show that both the solution in

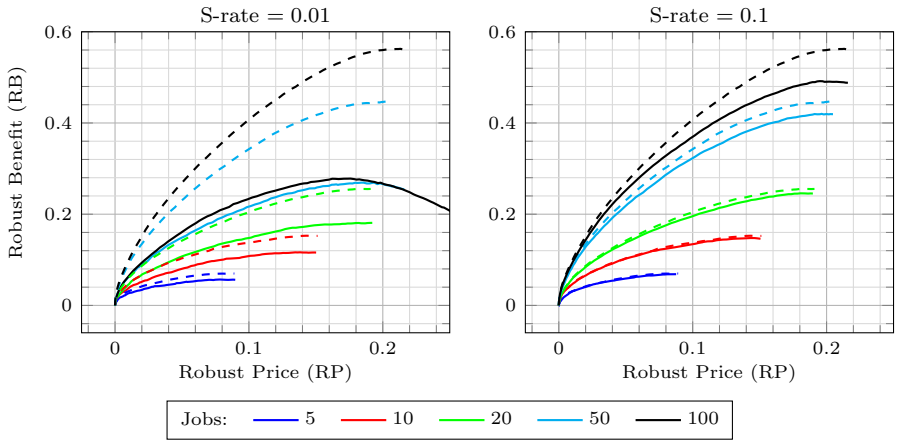


Figure 2.10: Effect of the number of dependent jobs n to RP/RB curve under ℓ_1 norm with variable S-rate.

Instance	SORT(ℓ_1)		MILP(ℓ_1)		SOCP(ℓ_2) [44]	
$n \times m$	time [s]	std [s]	time [s]	std [s]	time [s]	std [s]
8×3	2.21×10^{-4}	1.03×10^{-6}	6.0×10^{-3}	7.02×10^{-5}	0.13	0.08
9×3	2.45×10^{-4}	1.04×10^{-6}	7.3×10^{-3}	3.18×10^{-5}	0.29	0.40
10×3	2.72×10^{-4}	1.54×10^{-6}	9.0×10^{-3}	5.44×10^{-5}	1.29	1.78
11×3	3.07×10^{-4}	1.97×10^{-6}	1.1×10^{-2}	2.53×10^{-5}	7.07	13.79
12×3	3.39×10^{-4}	1.97×10^{-6}	1.2×10^{-2}	3.44×10^{-5}	25.88	59.59

Table 2.3: Computational times of different solving methods and instance sizes for dependent jobs with copositive covariance matrices.

terms of ℓ_1 norm with full covariance (i.e., MILP(ℓ_1)) and just its diagonal part (denoted as MILP-DIAG(ℓ_1)) are much faster than SOCP(ℓ_2). As a result, the above experiment suggests that when enough data is available, then it is advantageous for the decision maker to solve the problem with full covariance rather than assuming the independence between jobs as it provides more protection against solution variance.

Scaling with respect to the sample quantity Evidently, the amount of data that is needed to achieve the required RB depends on the size of the problem instance, i.e., the number of jobs n. Therefore, we have performed an experiment, where we fixed S-rate to 0.01 and 0.1, and we change the number of jobs $n \in \{5, 10, 20, 50, 100\}$ while we keep the number of machines $m = 3$. The results are displayed in Figure 2.10, where the RP/RB curves are obtained by MILP(ℓ_1) with a varying number of jobs n, but with a fixed S-rate (0.01 on the left side, 0.1 on the right side). For each n, two curves are reported — the dashed curve is the one with perfect information, whereas the solid curve corresponds to the case when the limited number of samples (given by S-rate) is available. Therefore, the absolute values of RB are not that important (as it changes with the number of jobs n), but the difference between the two curves matters. We can see that, e.g., for $n = 50$ jobs, the

S-rate equal to 0.1 (i.e., 1000 samples) is essentially enough to obtain a theoretically optimal trade-off curve (see the right plot in Figure 2.10). On the other hand, with S-rate equal to 0.01 (i.e., 100 samples), a similar level of discrepancy between the two curves is achieved for just $n = 10$ jobs (see the left plot in Figure 2.10). These values for S-rate correspond to the number of free parameters of a PSD matrix that need to be estimated, which is roughly quadratic in n.

2.5.5 Copositive covariance matrices

In this section, we focus on copositive covariance matrices, which is a special class of covariance matrices allowing us to solve the problem in polynomial time, as shown in Section 2.3.5. There are several natural questions connected with this class of covariance matrices. For example, do they bring any additional benefits in terms of RP/RB curve in comparison to using just their diagonal part? How often these matrices appear among the ones generated by the evaluation protocol, and is the solution of the problem in terms of ℓ_1 norm still comparable to ℓ_2 norm?

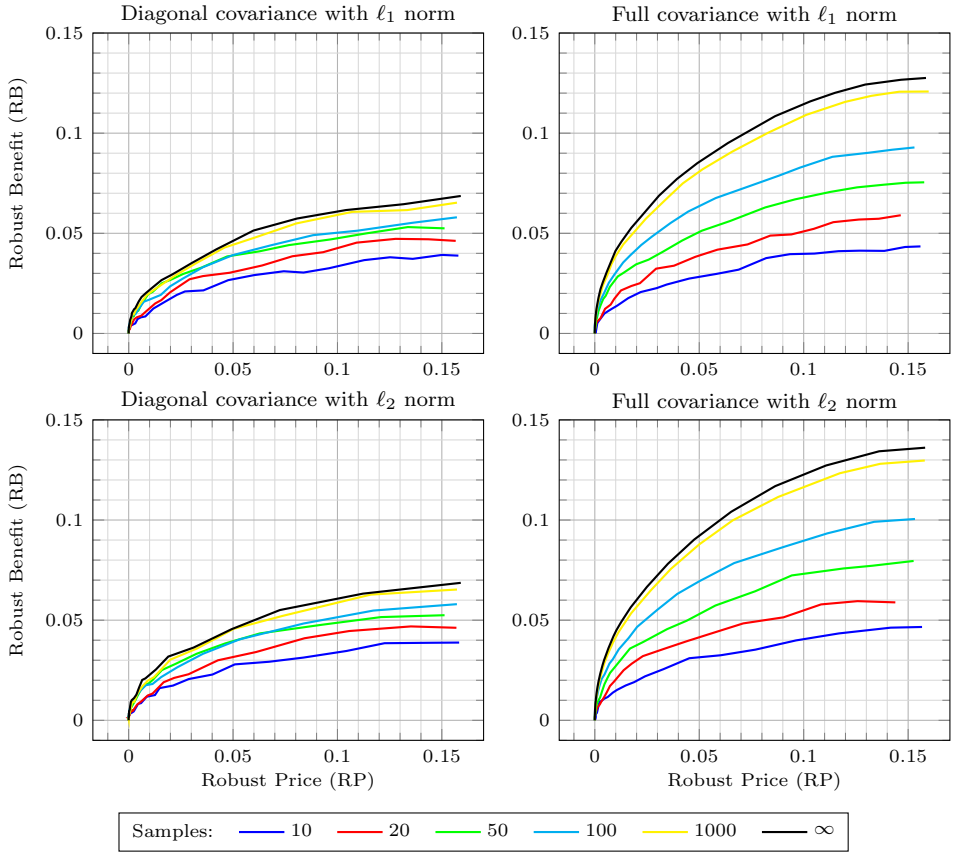


Figure 2.11: Effect of S-rate to RP/RB curve with copositive covariance matrices for instances with $n = 10$ and $m = 3$.

The setup is similar to the experiments in Section 2.5.4. Due to the limited performance of SOCP(ℓ_2) with dependent jobs, we have restricted the comparison to the instances with $m = 3$ machines and the maximum of $n = 12$ jobs. The true covariances are drawn from a distribution over PSD matrices described in Section 2.5.4, with the same parameters, i.e., $\nu = n+40$, $\lambda = 1$ and $\boldsymbol{\mu'} = 5 \cdot \mathbf{1}$. With the given parameters, we have observed that the sampled matrices are copositive with respect to the used values of m and n in about 30% of cases and these were used in the following experiments.

Effect of the used ℓ_p norm First, we discuss achievable trade-off curves. The results are displayed in Figure 2.11. There, we can see several essential differences from the results for general covariance matrices presented in Figure 2.9. First, we see that differences between ℓ_1 and ℓ_2 are much smaller for copositive covariance matrices in comparison to general ones (displayed in the left plot of Figure 2.8). Next, it can be seen that the achievable RB for a fixed RP is about 0.02 (for $n = 10$ jobs) smaller than in the case of general covariance matrices. Both these observations are explained by the fact that the copositive matrices tend to be more diagonally dominant, which is in line with the results observed in experiments with independent jobs. Furthermore, the results also show that using a full covariance matrix brings significantly more robustness than using just a diagonal part of the matrix. Thus, dealing with copositive covariance matrices is indeed meaningful.

Computational times The last experiment measures the computational times of different methods. Interestingly, when we compare computational times of SOCP(ℓ_2) method for general covariance matrices in Table 2.2 and the computational times for copositive covariance matrices in Table 2.3, we see that the runtimes are significantly smaller for the later ones. This again points to a relation between copositive and diagonally dominant matrices, for which SOCP(ℓ_2) scales better than in the general case. On the other hand, for MILP(ℓ_1) method, the computational times are comparable regardless of the type of covariance matrix. Finally, it can be seen that SORT(ℓ_1) method is superior, being about 100 times faster than MILP(ℓ_1), which indicates a good scaling to even larger instances (given by polynomial computational complexity).

2.6 Conclusion

In this paper, we study a distributionally robust scheduling problem with the total flow time criterion. The distribution of uncertain processing times is subject to ambiguity belonging to a set of distributions with constrained first two central moments. A prior work [44] has established that such a problem can be translated into a second-order conic programming problem. We have noticed that this optimization problem can be viewed as a minimization of a linear function plus a regularization term expressed in terms of ℓ_2 norm. A natural question immediately arises — is the use of a particular norm essential, or can it be replaced with some other ℓ_p norm, perhaps with more favorable computational properties while providing a similar level of robustness?

We answer this question affirmatively. We have provided a characterization of complexity for the problem with independent jobs in the sense of any ℓ_p norm. As a special case of our theorem, we have improved the upper bound on the complexity for the case of ℓ_2 formulation proposed in [44]. For the ℓ_1 norm, we obtained even stronger results, leading to a polynomial-time algorithm. For the case of dependent jobs, we identified a class of covariance matrices admitting an efficient, polynomial-time solution algorithm, when ℓ_1 regularization term is used. Interestingly, carefully conducted experiments have shown that solutions with ℓ_1 regularization term provide almost identical trade-offs between the quality and robustness to the more complex ℓ_2 regularization. This result comes as a surprise, considering that the best-known solution for dependent jobs in the sense of ℓ_2 regularization is able to solve only problems with 10 jobs and 3 machines within an hour, wheres our algorithm for ℓ_1 can successfully solve instances with hundreds of jobs, and for a class of generalized positive covariance matrices, is of polynomial time complexity.

The results also demonstrate the importance of utilizing the information about potential correlations between jobs, even when one does not have the perfect knowledge of covariance. This realistic case also shows that it is not crucial to use the formulation with ℓ_2 norm, but it can be replaced with ℓ_1 norm with essentially identical quality and stability — at a much reduced computational cost. This stimulates to study further the relation between tractable solutions for (conservative) ambiguity sets and approximate solutions for more expressive (but intractable) DRO formulations in environments with limited data availability. It is subject to a further study to which extent are the ideas developed in this paper applicable for more complex objective functions such as, e.g., total tardiness. Furthermore, the complexity of ℓ_1 formulation for general covariance matrices remains as an open question as well structural differences between the solutions obtained with different regularization norms.

Chapter

Scheduling with uncertain processing times in mixed-criticality systems

3

Antonín Novák, Premysl Sucha, and Zdenek Hanzalek. "Scheduling with uncertain processing times in mixed-criticality systems". In: *European Journal of Operational Research* 279.3 (2019), pp. 687–703. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2019.05.038

3.1 Introduction

This paper addresses scheduling in *mixed-criticality* systems where tasks have different degrees of importance (*criticalities*) and share a common resource. The key requirement of these systems is to isolate tasks such that a lower-criticality task does not influence any higher-criticality task. When the processing time of tasks is uncertain, the unexpected prolongation of a task may affect the execution of another task with higher criticality, which is extremely dangerous for safety-critical systems. A naive solution assuming *the worst-case* processing times leads to inefficient utilization of the resource. This is problematic, especially for embedded systems having limited computational and hardware resources.

To overcome the processing time uncertainty, we utilize the so-called *F-shaped tasks*, where each task has an integer *criticality* and a set of alternative processing times. The schedules with F-shaped tasks are proactive and contain exponentially many alternative schedules, with the alternative being selected based on the realized processing time of a task that occurs during the runtime execution of the schedule. The structure of the schedule guarantees that in any of these alternatives, all highly critical tasks are performed, rejecting low-criticality tasks only if a more critical one is prolonged. At the same time, the resource is efficiently utilized since when critical tasks are not prolonged, low-criticality tasks may use the resource. Therefore, the proactive schedules with F-shaped tasks achieve a trade-off between the required safety margins and an efficient resource usage. An important advantage of this approach is that despite such flexibility, the schedules only take polynomial-sized space. In addition, even though the corresponding optimization problem is $\mathcal{NP}$-hard, our exact algorithms are computationally efficient in practice.

In the following text, we formally define a single resource scheduling problem with non-preemptive F-shaped tasks to minimize the maximum completion time. The relation between real-world applications and this scheduling problem is provided in Section 3.2.

3.1.1 Problem statement

We assume a set of F-shaped tasks $I_{\mathcal{MC}} = \{T_1, \ldots, T_n\}$ to be scheduled on a single resource. We define an F-shaped task (or F-shape for short) and its criticality as follows:

Definition (F-shaped task)**.** *The F-shaped task T_i is a pair $(\mathcal{X}_i, \boldsymbol{P}_i)$ where $\mathcal{X}_i \in \{1, \ldots, \mathcal{L}\}$ is the task criticality and $\boldsymbol{P}_i \in \mathbb{N}^{\mathcal{X}_i}$, $\boldsymbol{P}_i = \left(p_i^{(1)}, p_i^{(2)}, \ldots, p_i^{(\mathcal{X}_i)}\right)$ is a vector of processing times such that $p_i^{(1)} < p_i^{(2)} < \ldots < p_i^{(\mathcal{X}_i)}$.*

Furthermore, we refer to $p_i^{(\ell)}$ as the processing time of T_i at level ℓ. Let us denote $\mathcal{L}$ as the highest criticality in $I_{\mathcal{MC}}$, i.e., $\mathcal{L} = \max_{T_k \in I_{\mathcal{MC}}} \mathcal{X}_k$. Having a set $I_{\mathcal{MC}}$ of F-shaped tasks, we define a feasible schedule of $I_{\mathcal{MC}}$ as follows:

Definition (Feasible Schedule)**.** *By the schedule for a set of F-shaped tasks $I_{\mathcal{MC}} = \{T_1, T_2, \ldots, T_n\}$, we refer to the assignment of start times $\boldsymbol{s} = (s_1, s_2, \ldots, s_n) \in \mathbb{N}_0^n$. We say that schedule $(s_1, s_2, \ldots, s_n)$ for $I_{\mathcal{MC}}$ is feasible if and only if $\forall i, j \in \{1, \ldots, n\}, i \neq j$:*

$$\left(s_i + p_i^{(\min\{\mathcal{X}_i, \mathcal{X}_j\})} \leq s_j\right) \vee \left(s_j + p_j^{(\min\{\mathcal{X}_i, \mathcal{X}_j\})} \leq s_i\right). \tag{3.1.1}$$

The sufficient and necessary conditions for the feasibility of a schedule with F-shaped tasks state that tasks are non-preemptive and do not overlap on any criticality level. For example, in Figure 3.1a where T_5 follows T_4, F-shaped task T_5 cannot start earlier than that at $s_4 + p_4^{(2)}$, since $\min\{\mathcal{X}_4, \mathcal{X}_5\} = 2$, which is the highest common criticality level of T_4 and T_5.

Given the schedule $\boldsymbol{s}$, we say that the completion time of a task is given by its start time in $\boldsymbol{s}$ plus the processing time at the highest criticality level:

Definition (Makespan of a Schedule)**.** *Given a feasible schedule $\boldsymbol{s} = (s_1, s_2, \ldots, s_n)$, the completion time of task T_j is given as $C_j = s_j + p_j^{(\mathcal{X}_j)}$. The maximal makespan of the schedule $\boldsymbol{s}$ is the latest completion time, i.e., $C_{max} = \max_j C_j = \max_j \left\{s_j + p_j^{(\mathcal{X}_j)}\right\}$.*

Further in the text, we will use the term makespan instead of the maximal makespan for simplicity. The problem we deal with in this paper is to find a feasible schedule for the given set of F-shaped tasks with criticality at most $\mathcal{L}$, which has the minimal makespan:

Definition (MC–$\mathcal{L}$ Problem Statement)**.** *Given the set $I_{\mathcal{MC}}$ of F-shaped tasks with maximum criticality $\mathcal{L}$, find a feasible schedule minimizing the makespan, i.e.,*

$$\begin{aligned} &\min_{\boldsymbol{s}} C_{max} \\ \textit{subject to} \quad & \\ &\textit{feasibility conditions } (3.1.1) \\ &\boldsymbol{s} \in \mathbb{N}_0^n. \end{aligned}$$

In the three-field Graham-Blazewicz scheduling notation [75], the problem is denoted as $1|mc = \mathcal{L}|C_{\max}$, where 1 denotes the scheduling on a single resource, $mc = \mathcal{L}$ stands for the mixed-criticality aspect of tasks of maximal criticality $\mathcal{L}$, and $C_{\max}$ stands for the minimization of the maximum completion time. This problem is known to be $\mathcal{NP}$-hard in the strong sense even for the special case $mc = 2$ (two criticality levels), as shown by the reduction from 3-Partition problem in [81].

3.1.2 Related work

The study of mixed-criticality systems originates from real-time scheduling community due to its practical applications. In the seminal paper [175], Vestal proposed a model of mixed-criticality that understands each task as a set of different processing times for discrete levels of assurance. This understanding of mixed-criticality was later adopted by many others in the following works, e.g., Baruah [12, 15], Burns [40, 41] and Davis [54]. This line of research mostly deals with response time analysis of different scheduling policies considering preemptive tasks in so-called event-triggered systems [93].

Often cited disadvantage of complex event-triggered systems is their inability to be certified for safety-critical applications [87, 3]. Therefore, researches have turned their attention toward static scheduling in mixed-criticality systems [93, 170, 20] that solves the problem with certification and predictability. The problem with preemptive tasks with two criticality levels was studied in [20]. They proposed a heuristic algorithm that constructs a static schedule for multiple resources while considering precedence constraints. Hanzalek *et al.* [81] were the first to state the mixed-criticality as a static non-preemptive scheduling problem, for which they proposed the relative-order MIP model to solve the problem with release times r_i and deadlines $\tilde{d}_i$, i.e., $1|r_i, \tilde{d}_i, mc = \mathcal{L}|C_{\max}$ and they proved that minimizing the makespan is strongly $\mathcal{NP}$-hard for two criticality levels.

The follow-up works aimed to study different problems arising from the scheduling of F-shaped tasks. The idea of approximating cumulative distribution functions with F-shapes has appeared in [6]. Dürr *et al.* [64] studied the case, where each task is given by a single number $p_i \in \mathbb{N}$ defining p_i criticality levels with unit processing time prolongation; hence they appear as equilateral triangles in Gantt charts. They refer to this special case of the scheduling with F-shaped tasks as the triangle scheduling problem. Their main results are the proof that the makespan minimization with triangular tasks is at least weakly $\mathcal{NP}$-hard and a quasipolynomial-time approximation scheme for the problem. Seddik [151] noted that makespan minimization with F-shaped tasks decreases the probability of tasks execution. Hence, instead of making compact schedules, they proposed a non-regular criterion that maximizes the execution probability of the tasks — spreading them as much as possible under deadline constraints. They presented the proof that finding optimal start times remains $\mathcal{NP}$-hard under the fixed permutation and they proposed (i) dynamic programming for the case of two criticality levels and (ii) MIP model for the general problem.

Makespan minimization with tasks up to two criticality levels (i.e., MC-2) is closely related to classical parallel machine scheduling problems [129] such as uniformly related machines with makespan minimization (i.e., $Q||C_{\max}$ [97]). The

machines represent critical tasks while the speeds of the machines are set proportionally to the difference of processing times $p_i^{(2)} - p_i^{(1)}$ at their both levels. However, the makespan minimization in parallel uniform machines environments leads to suboptimal solutions for MC-2 since makespan minimization disregards makespans on machines with smaller load than $C_{\max}$.

A closer problem is the scheduling on identical parallel machines with the total tardiness criterion, i.e., $P||\sum T_j$ [159]. The total tardiness criterion minimizes the total sum of processing processing times of jobs that exceed their due date, which relates to makespan minimization criterion in MC-2. This relation is further discussed at the end of Section 3.3.2. However, for the general problem MC-2, the transformation cannot be used.

Moreover, the problem with positive time lags $1|l_{ij} > 0|C_{\max}$ with chain precedence [117] can be used to solve MC-2. Even thought it is possible to reduce to more complex problems to obtain a solution, in practice, it is computationally inefficient method, as the structure of the original problem that can be exploited is not exposed to the algorithm.

Another related problem is the bin packing [71], which considers an unlimited number of bins (optionally of different sizes) and a set of items to pack. The goal is to pack the items using the minimum number of bins while their capacity is not exceeded. Further connections can be seen also with 1D cutting stock problem [57], where one cuts items of different size from material rolls of the given length such that the residual waste is minimized. The main difference from those two problems is that the size of the bin (material roll) cannot be exceeded (contrary to MC-2).

Solving problems with more criticality levels brings yet another level of complexity, yielding looser relation to the above mentioned problems. The makespan scheduling with more criticality levels can be then related to more general packing problems, such as polyominoes [73].

Taking a broader perspective, the problem in this study is related to stochastic optimization [144] due to the uncertainty of processing times [79]. Moreover, it contains aspects of task disruption [133] and rejection [152] due to flexible execution of schedules, and robust scheduling [29] due to the robustness with respect to processing time prolongation. To the best of our knowledge, none of these approaches alone can be applied to our problem, as we need a combination of uncertainty, robustness and task rejection at once. The problem of static non-preemptive mixed-criticality scheduling has been addressed by [81, 6] only; however they lack computationally efficient exact solution method, which is presented in this paper.

3.1.3 Contribution and paper outline

This paper focuses on the fundamental properties of F-shaped tasks that arise from scheduling problems in mixed-critical environments. We study the problem of the makespan minimization with F-shaped tasks (i.e., $1|mc = 2|C_{\max}$ and $1|mc = 3|C_{\max}$) and develop fast exact algorithms for solving the problems. The main contributions of this paper are as follows:

– an approximation algorithm for the problem with two criticality levels (see Section 3.3.1),

– an exact efficient block MIP model that optimizes over non-isomorphic permutations (see Section 3.3.2),

– a branch-and-price algorithm with a pseudopolynomially solvable pricing problem (see Section 3.3.3),

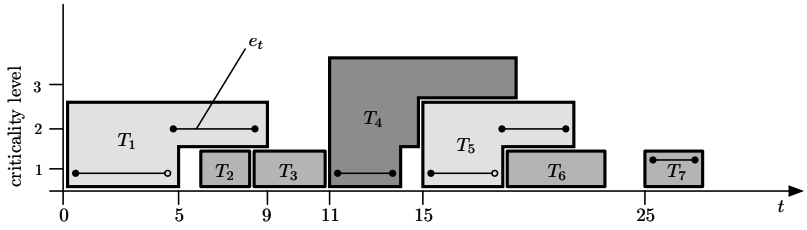
– a structural result on optimal permutations and a generalization of the branch-and-price for more criticality levels (see Section 3.4.1 and Section 3.4.2), and

– the experimental evaluation of the proposed algorithms (see Section 3.5).

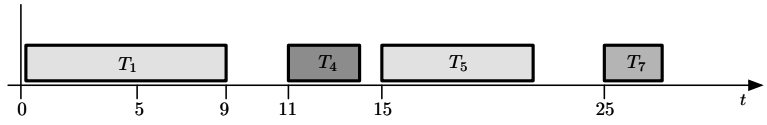
The rest of this paper is organized as follows. In Section 3.2, we describe our model for processing time uncertainty, explain the online execution of the schedule, and show real-life applications of the model. In Section 3.3, we derive a factor-two approximation algorithm for the problem with two criticality levels, unveil the structure of optimal schedules, and propose an efficient MIP formulation that optimizes over non-isomorphic permutations. In Section 3.4, we generalize the method for more criticality levels. The numerical experiments are described in Section 3.5, where we demonstrate the efficiency of our algorithms and bounds distinguishing easy instances from the difficult ones. The conclusions are drawn in Section 3.6.

3.2 Uncertainty and execution model

In this section, we explain how uncertain processing time of a task, given by a probability distribution, can be modeled by an F-shaped task. Next, we will show how F-shapes form static schedules, that encapsulate different alternative runtime scenarios. Finally, we describe some real-life applications suitable for the proposed model.



(a) Schedule with F-shaped tasks and the execution scenario e_t.

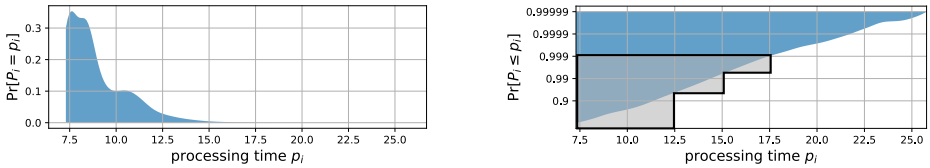


(b) Executed alternative.

Figure 3.1: Schedule with F-shaped tasks and the executed alternative.

3.2.1 Approximation of a distribution function

The processing time uncertainty may be expressed by a *probability density function* (PDF). Figure 3.2a shows a real-life PDF of computational times of an algorithm used in autonomous driving. This algorithm, described in [113] consists of matrix multiplications, fast Fourier transform, inverse transform, and a binary search.



(a) PDF of computational times. (b) F-shape as an approximation of CDF.

Figure 3.2: Approximation of computational times represented as an F-shaped task.

Real-life distributions of computational times often have a long tail, i.e., the actual computational times can be significantly larger than the expected value, but with a decreasing probability. Therefore, it is convenient to display *cumulative distribution functions* (CDFs) with a logarithmic scale on the y-axis (see Figure 3.2b). Each task has criticality prescribed by the application requirements (e.g., the pedestrian tracking has higher criticality than adaptive light shaping). Processing time $p_i^{(\ell)}$ at each criticality level ℓ is given by the CDF and the corresponding probability threshold.

The choice of probability thresholds is dependent on the target application and its safety requirements. For example, often the automotive safety integrity levels (ASIL) standard [87] states that the system must guarantee that all high-criticality activities will be successfully completed with the probability of at least 0.999, medium-criticality with at least 0.99, and low-criticality with 0.9. Then, it can be analytically computed by examining the worst-case coverage, that the choice of thresholds 0.999 for the high level, $\sqrt{0.99} \approx 0.995$ for the medium level and $\sqrt{\frac{0.9}{\sqrt{0.99}}} \approx 0.952$ for the low critical level guarantees that in any feasible schedule, all tasks will be successfully completed at least with the required probability. See Figure 3.2b for the resulting F-shape.

3.2.2 Runtime execution scenarios

A schedule with F-shaped tasks is the same as any other static schedule, i.e., it is a static assignment of tasks to start times. However, it can be executed under different scenarios that emerge from the processing time uncertainty. Hence, we distinguish two concepts — a schedule and an *execution scenario*. The schedule is a static assignment of F-shapes to start times, and is computed from the given set of F-shaped tasks; thus, it is known before the runtime execution. On the other hand, the execution scenario is a function of the schedule and the observed processing time prolongations; therefore, it is not known in advance. The criticality of an F-shaped task plays a role in the runtime execution — a more critical task is allowed to consume the resource time of a less critical task to compensate for its prolongation if needed. This can happen in cases where a more critical F-shape

covers a less critical one (e.g., T_5 covers T_6 in Figure 3.1a).

An example of a static schedule of F-shaped tasks can be seen in Figure 3.1a. Since the exact processing time of tasks is not known in advance, the schedule needs to account for the observed processing time prolongations, i.e., provide an alternative for each possible scenario. The realized scenario is described in terms of the execution level e_t of the static schedule at each time instance t. Denoting $\mathcal{L}$ as the maximum criticality among all tasks, the execution level $e_t : t \to \{0, 1, \ldots, \mathcal{L}\}$ is a piecewise constant function. In Figure 3.1a, one of the possible execution scenarios is depicted by the black line. Its value corresponds to the current system criticality level with value 0 used in cases where the resource is idle.

Example We will describe the execution policy through a specific example depicted in Figure 3.1a. In this case, the execution has begun at time $t = 0$ at the first level, i.e., $e_0 = 1$. The task T_1 is executed until time $t = 5$. Here, it is observed that T_1 is not finished by that time. Therefore, its processing time is prolonged; i.e., the realized processing time is greater than 5. The execution level is raised to the second level, i.e., $e_5 = 2$, and the execution of T_1 continues. At time $t = 9$, T_1 is completed. However, tasks T_2 and T_3 are rejected during this scenario since the more critical task T_1 is prolonged and the execution of T_2 and T_3 would collide with it. Hence, if a prolongation occurs, it is compensated by rejecting some of the low-criticality tasks.

When a task is completed, the execution *matches-up* [19] with the base level (i.e., $e_t = 0$). In our example, $e_9 = 0$ denotes that the resource was available at time $t = 9$ during the considered scenario. The next task executed is T_4, since at its start time $s_4 = 11$, the resource was available; i.e., $e_{11} = 0$ and its execution starts at the first level. This sequence of observed events and reactions of the execution policy results in the executed alternative depicted in Figure 3.1b.

3.2.3 Real-world applications

As it was described in the previous subsection, static schedules with F-shaped tasks contain exponentially many alternatives, and it might be the case that for a schedule, there are scenarios that reject some or even all low-criticality tasks in the schedule. Nevertheless, it is still reasonable to schedule all tasks and not to exclude them from scheduling in advance because this behavior has support in the applications.

First, most of real-life embedded systems perform a periodic workload [40, 170, 175], i.e., the same tasks (given in advance) are repeated over time (e.g., periodical measurement of oil temperature). In these cases, the rejected task might be executed again in few milliseconds in the next period (see, e.g., [64] for application to retransmission of communication messages in safety-critical embedded systems). In non-periodic environments, such as production scheduling or scheduling of surgeries in an operating theater [151], the low-criticality tasks rejected in the current scheduling horizon are transferred to the following one where they will be scheduled again. Secondly, the rejection of a task occurs rarely, and it is reasonable to assume that in practical applications, we talk about exceptions.

Furthermore, many of today's real-time applications, such as advanced driver

assistance systems, demand both high computing power and safety guarantees. A real-life example of such systems is NVIDIA DRIVE™ PX2, which contains a powerful graphics processing unit that runs deep neural networks for computer vision that secure autonomous driving capabilities. A common property of such algorithms is that their computational time is not deterministic since it frequently depends on the content of the input image. For example, the computational load in the problem of visual object tracking increases with the number of objects in the camera image. Furthermore, the additional uncertainty comes from low-level mechanisms such as the shared access to the main memory, the processor caches and interconnects.

3.3 Problem with two criticality levels

In this section, we deal with the problem restricted to two criticality levels, i.e., MC-2. This problem models an environment that distinguishes between critical and non-critical activities. The critical activities are those that cannot be rejected under any circumstance, whereas non-critical are the ones that can be if a critical one is prolonged. Concerning practical applications, the number of criticality levels $\mathcal{L}$ might be relatively low, i.e., usually $\mathcal{L} \ll n$, where n is the number of tasks in $I_{\mathcal{MC}}$. Indeed, without loss of generality, we can assume that $\mathcal{L}$ is bounded above by n, as proposed by Lemma 3:

Lemma 3. *For any instance $I_{\mathcal{MC}}$ of the problem MC-$\mathcal{L}$, there exists an instance $I'_{\mathcal{MC}}$ of the problem MC-$\mathcal{L}'$, $\mathcal{L}' \leq \mathcal{L}$, such that $\mathcal{L}' \leq n$ and that any feasible schedule for $I'_{\mathcal{MC}}$ is a feasible schedule of $I_{\mathcal{MC}}$ with the same makespan.*

Proof. Suppose we have a feasible schedule $\boldsymbol{s}$ for $I_{\mathcal{MC}}$. If there is no task $T_i \in I_{\mathcal{MC}}$ with criticality $\ell = \mathcal{X}_i$, then there is no T_j such that $\ell = \min\{\mathcal{X}_j, \mathcal{X}_i\}$. Therefore, removing level ℓ from all tasks $T_i \in I_{\mathcal{MC}}$, $\mathcal{X}_i > \ell$ while keeping the start times $\boldsymbol{s}$ fixed will not violate the feasibility conditions in Definition . Moreover, the makespan $C_{\max} = \max_k \left\{ s_k + p_k^{(\mathcal{X}_k)} \right\}$ is preserved since $\mathcal{X}_k \neq \ell$.

By removing the level ℓ, we effectively reduce the maximum criticality in the instance $I_{\mathcal{MC}}$, since $\ell \leq \max_k \mathcal{X}_k = \mathcal{L}$. Therefore, we obtain an instance $I'_{\mathcal{MC}}$ of the problem MC-$\mathcal{L}'$ such that $\mathcal{L}' < \mathcal{L}$. This transformation can be chained until there is such unused level ℓ. Furthermore, since $\mathcal{L}' \leq |\{\mathcal{X}_i \,|\, \forall T_i \in I'_{\mathcal{MC}}\}| = |\{\mathcal{X}_i \,|\, \forall T_i \in I_{\mathcal{MC}}\}| \leq n$, the claim follows. □

The corollary of Lemma 3 is that if $I_{\mathcal{MC}}$ is an instance of the problem MC-$\mathcal{L}$, then without loss of generality, $\forall \ell \in \{1, \ldots, \mathcal{L}\} \quad \exists T_i \in I_{\mathcal{MC}} : \mathcal{X}_i = \ell$, i.e., we can assume that for each criticality level $\ell \in \{1, \ldots, \mathcal{L}\}$, a task with the same criticality exists. The schedules are defined in terms of start times of tasks. However, it is easy to see that the search for schedules can be reduced to an optimization problem over a set of permutations of tasks:

Definition (Left-shifted Schedule)**.** *Let $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ be a permutation of a set of tasks $I_{\mathcal{MC}}$. Then, the left-shifted schedule of permutation π is a schedule $\boldsymbol{s}$, where the task $\pi(1)$ starts at time 0 and all other tasks start at their earliest*

start times such that they do not overlap on any level with any preceding task in the order given by π, i.e.,

$$s_{\pi(1)} = 0$$
$$s_{\pi(i)} = \max_{j<i} \left\{ s_{\pi(j)} + p_{\pi(j)}^{(\min\{\mathcal{X}_{\pi(i)}, \mathcal{X}_{\pi(j)}\})} \right\} \quad \forall i \in \{2, \ldots, n\}$$

We say that a schedule $\boldsymbol{s}$ of a permutation π is *dominant* for π, if it has the minimum makespan among the set of all possible schedules of the permutation π.

Lemma 4. *For any instance of MC-$\mathcal{L}$, the left-shifted schedule is dominant for any permutation π.*

Proof. By contradiction. Suppose we have a left-shifted schedule $\boldsymbol{s}$ of a permutation π and a feasible schedule $\boldsymbol{s}'$ of the same permutation π that is not left-shifted, such that $C_{\max}(\boldsymbol{s}') < C_{\max}(\boldsymbol{s})$. Since $\boldsymbol{s}'$ is not left-shifted, then either $s'_{\pi(1)} > 0$ or $s'_{\pi(i)} > \max_{j<i} \left\{ s'_{\pi(j)} + p_{\pi(j)}^{(\min\{\mathcal{X}_{\pi(i)}, \mathcal{X}_{\pi(j)}\})} \right\}$ for some $i \in \{2, \ldots, n\}$. Therefore, it holds that $s'_j > s_j$ for some task $T_j \in I_{\mathcal{MC}}$. However, since $C_{\max}(\boldsymbol{s}) = \max_k \left\{ s_k + p_k^{(\mathcal{X}_k)} \right\}$ is a non-decreasing function of start times, then it follows that $C_{\max}(\boldsymbol{s}') \geq C_{\max}(\boldsymbol{s})$, which leads to the contradiction. □

That is, given the permutation of tasks, the optimal makespan is achieved by shifting all tasks to the left while maintaining feasibility, i.e., overlapping conditions from Definition . Moreover, it can be shown that for the case of ℓ criticality levels, the makespan of such schedule will always be at most ℓ-times larger than the optimal one.

Proposition 4. *Any algorithm for the problem MC-$\mathcal{L}$ producing the left-shifted schedule is $\mathcal{L}$-approximation algorithm.*

Proof. Let us denote the makespan of an optimal solution of $I_{\mathcal{MC}}$ instance as $\mathrm{OPT}(I_{\mathcal{MC}})$ and the makespan of any left-shifted solution as $\mathrm{LS}(I_{\mathcal{MC}})$. Since $\max_{\ell \leq \mathcal{L}} \left\{ \sum_{T_j \in I_{\mathcal{MC}} : \mathcal{X}_j \leq \ell} p_j^{(\ell)} \right\}$ is a lower bound on $\mathrm{OPT}(I_{\mathcal{MC}})$, we can write

$$\max_{\ell \leq \mathcal{L}} \left\{ \sum_{T_j \in I_{\mathcal{MC}} : \mathcal{X}_j \leq \ell} p_j^{(\ell)} \right\} \leq \mathrm{OPT}(I_{\mathcal{MC}}) \leq \mathrm{LS}(I_{\mathcal{MC}}) \leq \sum_{\ell=1}^{\mathcal{L}} \sum_{T_j \in I_{\mathcal{MC}} : \mathcal{X}_j = \ell} p_j^{(\ell)} \leq$$
$$\leq \mathcal{L} \cdot \max_{\ell \leq \mathcal{L}} \left\{ \sum_{T_j \in I_{\mathcal{MC}} : \mathcal{X}_j \leq \ell} p_j^{(\ell)} \right\} \leq \mathcal{L} \cdot \mathrm{OPT}(I_{\mathcal{MC}}),$$

where the third inequality follows from the fact that the longest possible left-shifted schedule has tasks sorted in a non-decreasing order of criticalities. □

In the next section, we will propose an approximation algorithm for problem MC-2, that achieve on average better results than its the worst-case guarantees.

3.3.1 Approximation algorithm

We propose the following approximation algorithm. The main concept of the algorithm is to build the schedule using basic units, which we call *blocks*. Let us partition the input instance $I_{\mathcal{MC}}$ into two disjoint subsets L and H, $I_{\mathcal{MC}} = L \cup H$, $L \cap H = \emptyset$. Let L be the set of tasks with low criticality $L = \{T_j \,|\, \forall T_j \in I_{\mathcal{MC}} : \mathcal{X}_j = 1\}$, $|L| = n_L$ and H be the set of tasks with high criticality $H = \{T_i \,|\, \forall T_i \in I_{\mathcal{MC}} : \mathcal{X}_i = 2\}$, $|H| = n_H$. Note that by Lemma 3, we can assume that $L, H \neq \emptyset$. The algorithm constructively partitions tasks into the so-called *coverage sets*.

Definition (Coverage set)**.** *Let $T_i \in I_{\mathcal{MC}}$ be an F-shaped task. Then,*

$$cov(T_i) \subseteq \{T_j \,|\, \forall T_j \in I_{\mathcal{MC}} : \mathcal{X}_j = \mathcal{X}_i - 1\}$$

is a subset of tasks with criticality $\mathcal{X}_i - 1$.

The coverage set $\mathrm{cov}(T_i)$ can be viewed as a set of less critical tasks, which immediately follows T_i in a schedule. If $T_j \in \mathrm{cov}(T_i)$, then T_j is *covered by* T_i. The tasks $\{T_i\} \cup \mathrm{cov}(T_i)$ form a *block* (see Figure 3.3a with three different blocks). The algorithm constructs blocks that are used later to derive the whole schedule. In each iteration, the algorithm takes an unassigned task $T_j \in L$ with the longest processing time $p_j^{(1)}$ and a task $T_i \in H$, which currently has the largest available gap, defined as $W_i = p_i^{(2)} - p_i^{(1)} - \sum_{T_k \in \mathrm{cov}(T_i)} p_k^{(1)}$. Note that the gap W_i can be even negative if the sum of processing times of tasks in $\mathrm{cov}(T_i)$ is larger than $p_i^{(2)}$. After $T_j \in L$ and $T_i \in H$ are selected, T_j is assigned to the coverage set of T_i, i.e., $T_j \in \mathrm{cov}(T_i)$. When the task T_j is assigned, the gap W_i is decreased by the processing time $p_j^{(1)}$. This procedure is repeated until all tasks in L are assigned. In fact, the algorithm works similarly as the LPT (longest processing time first) rule for $Q||C_{\max}$ problem [97].

task	$\mathcal{X}_i$	$\boldsymbol{P}_i$
T_1	2	$(3, 9)$
T_2	2	$(4, 8)$
T_3	2	$(2, 9)$
T_4	1	$(8, -)$
T_5	1	$(4, -)$
T_6	1	$(3, -)$
T_7	1	$(3, -)$

(a) Input instance

iteration	
#1	$\mathrm{cov}(T_3) \leftarrow \{T_4\}$, $W_3 \leftarrow 7 - 8 = -1$
#2	$\mathrm{cov}(T_1) \leftarrow \{T_5\}$, $W_1 \leftarrow 6 - 4 = 2$
#3	$\mathrm{cov}(T_2) \leftarrow \{T_6\}$, $W_2 \leftarrow 4 - 3 = 1$
#4	$\mathrm{cov}(T_1) \leftarrow \{T_5, T_7\}$, $W_1 \leftarrow 2 - 3 = -1$
permutation	$\pi = (T_1, T_5, T_7, T_2, T_6, T_3, T_4)$
schedule	$\boldsymbol{s} = (0, 10, 18, 20, 3, 14, 7)$

(b) Iterations of the algorithm and the solution

Table 3.1: Illustrative example of (APX-MC-2) algorithm.

The output of the algorithm is a permutation π of all tasks in $I_{\mathcal{MC}}$. The permutation is formed by all tasks in H sorted in a non-decreasing order of W_i, each of them interleaved by assigned tasks $T_j \in \mathrm{cov}(T_i)$. The resulting schedule is given by the left-shifted schedule of the permutation π. Table 3.1 shows an illustrative example of how the algorithm proceeds. The pseudocode can be seen in the (APX-MC-2) algorithm.

Algorithm 3.1: 2-Approximation algorithm for MC-2 (APX-MC-2).

input : an instance $I_{\mathcal{MC}}$ of MC-2
output : a vector of start times $s_1, \dots s_n$

1 let $p_1^{(1)} \geq p_2^{(1)} \geq \dots \geq p_j^{(1)} \geq \dots \geq p_{n_L}^{(1)}$
2 $W_i \leftarrow p_i^{(2)} - p_i^{(1)} \quad \forall T_i \in H$
3 **for** $j = 1$ **to** n_L **do**
4 $\quad k \leftarrow \arg\max_i W_i$
5 $\quad W_k \leftarrow W_k - p_j^{(1)}$
6 $\quad \mathrm{cov}(T_k) \leftarrow \mathrm{cov}(T_k) \cup \{T_j\}$
7 $\pi \leftarrow ()$
8 **for** $i = 1$ **to** n_H **do**
9 $\quad \pi \leftarrow (\pi, T_i)$
10 $\quad$ **for** $T_j \in cov(T_i)$ **do**
11 $\quad\quad \pi \leftarrow (\pi, T_j)$
12 **return** LEFT-SHIFTED(π)

The algorithm runs in $\mathcal{O}(n_L(\log n_H + \log n_L) + n_H)$. The dominant operations are sorting (line 1) and preservation of the max-heap of W_i's (line 5). The (APX-MC-2) algorithm ensures that the makespan of any produced schedule is at most twice worse than the optimal one, which can be seen directly from Proposition 4. The difficulty of improving the upper bound on the approximation factor is introduced by the presence of "long" tasks in L together with uneven length of differences $p_i^{(2)} - p_i^{(1)}$ of tasks in $T_i \in H$. However, for some specific classes of instances we can obtain tighter factor: (i) when all tasks in H have the same constant difference $\Delta > 0$ between the second and the first level, i.e., $\forall T_i \in H : p_i^{(2)} - p_i^{(1)} = \Delta$, then the method of [67] gives us a PTAS (*polynomial-time approximation scheme*), (ii) when $\max_{T_j \in L} p_j^{(1)} \leq \min_{T_i \in H} p_i^{(2)} - p_i^{(1)}$, then (APX-MC-2) has factor at most 3/2, as will be shown below. We note that the case (ii) is the most practical one, since such instances arise from problems where the original processing time distributions have long tails.

For the problem with two criticality levels, i.e., MC-2, where it holds that the longest task in L is not longer than the difference between the second and the first level of any task in H, i.e.,

$$\max_{T_j \in L} p_j^{(1)} \leq \min_{T_i \in H} \left(p_i^{(2)} - p_i^{(1)} \right), \tag{SLT}$$

we can obtain factor 3/2 for (APX-MC-2) algorithm. Such instances arise from the practical problems where the original distribution functions describing processing time uncertainty have long tails, which is the realistic case. Note that such condition does not rule out solutions where a task in L overlaps the second criticality level of some task in H.

Proposition 5. *(APX-MC-2) is a* 3/2*-approximation algorithm for the problem MC-2 satisfying condition* (SLT).

Proof. Let $lb = \sum_{Tr_i \in H} p_i^{(1)} + \max\left\{\sum_{T_i \in H}\left(p_i^{(2)} - p_i^{(1)}\right), \sum_{T_j \in L} p_j^{(1)}\right\}$ be a lower bound on the optimal makespan. Furthermore, let us denote the makespan of the schedule produced by (APX-MC-2) for instance $I_{\mathcal{MC}}$ as $\mathrm{APX}(I_{\mathcal{MC}})$. Without loss of generality, we may assume that $n_L > n_H$; otherwise, the algorithm returns an optimal schedule due to assumption (SLT). We say that a task $T_j \in L$ is *assigned* if inserting it into coverage set $\mathrm{cov}(T_i)$ of the chosen $T_i \in H$ does not decrease the available gap below zero (i.e., $W_i < 0$). Otherwise, we say T_j *overlaps*. We denote by $\overline{L}$ the set of all tasks that overlap and the processing time of the longest overlapping task by $\overline{p} = \max_{T_j \in \overline{L}} p_j^{(1)}$.

First, we note that (APX-MC-2) *assigns* at least n_H largest tasks in L. Indeed, let $T_j \in L$ be the first task of L that overlaps at the k-th step of the algorithm. Suppose that $k \leq n_H$. Since T_j overlaps a task with the largest available gap $T_{i^\star} = \arg\max_{T_i \in H} W_i$, then T_j overlaps any other task $T_i \in H$ during k-th iteration. However, since $k \leq n_H$, then there is either (i) a task $T_{i'} \in H$ with the currently available gap $W_{i'} = p_{i'}^{(2)} - p_{i'}^{(1)}$, i.e., with no assigned tasks so far or (ii) every task $T_i \in H$ has exactly one task in its coverage set. In the case (i), by the assumption (SLT) we have that $W_{i'} \geq p_j^{(1)}$, which contradicts the choice of $T_{i^\star}$. In the case (ii), $T_{i^\star}$ violates assumption (SLT) since $p_{i^\star}^{(2)} - p_{i^\star}^{(1)} < p_j^{(1)}$. Therefore, $k > n_H$. We proceed by splitting the proof into two cases.

Case 1: $\sum_{T_i \in H}\left(p_i^{(2)} - p_i^{(1)}\right) \leq \sum_{T_j \in L} p_j^{(1)}$**.** We start by bounding the cardinality of $\overline{L}$. To do so, suppose that $|\overline{L}| \geq n_H$. Since all tasks in $\overline{L}$ have the property that their assignment into any task $T_i \in H$ would lead to $W_i < 0$, we could put at least one task in $\overline{L}$ to the coverage set of every task in H. Hence, it would hold that $\forall T_i \in H : W_i < 0$, which implies an optimal solution. Therefore, suppose that $|\overline{L}| \leq n_H - 1$, i.e., we have at most $n_H - 1$ overlapping tasks. Then we can write

$$\begin{aligned}
\frac{\mathrm{APX}(I_{\mathcal{MC}})}{\mathrm{OPT}(I_{\mathcal{MC}})} &\leq \frac{\sum_{T_i \in H} p_i^{(2)} + \sum_{T_j \in \overline{L}} p_j^{(1)}}{lb} = \frac{\sum_{T_i \in H} p_i^{(2)} + \sum_{T_j \in \overline{L}} p_j^{(1)}}{\sum_{T_i \in H} p_i^{(1)} + \sum_{T_j \in L} p_j^{(1)}} \\
&\leq 1 + \frac{\sum_{T_j \in \overline{L}} p_j^{(1)}}{\sum_{T_j \in L} p_j^{(1)}} \leq 1 + \frac{\sum_{T_j \in \overline{L}} p_j^{(1)}}{n_H \cdot \overline{p} + \sum_{T_j \in \overline{L}} p_j^{(1)}} \\
&\leq 1 + \frac{\sum_{T_j \in \overline{L}} p_j^{(1)}}{2 \cdot \sum_{T_j \in \overline{L}} p_j^{(1)}} \leq \frac{3}{2},
\end{aligned}$$

where the first inequality follows from the fact that the expression in the numerator is an upper bound on the $\mathrm{APX}(I_{\mathcal{MC}})$, the equality from the definition of lb, the second inequality follows from Case 1 assumption, the third from the fact that at least n_H tasks in L of length at least $\overline{p}$ are assigned and the fourth inequality from the fact that $n_H \cdot \overline{p} \geq (n_H - 1) \cdot \overline{p} \geq \sum_{T_j \in \overline{L}} p_j^{(1)}$.

Case 2: $\sum_{T_i \in H}\left(p_i^{(2)} - p_i^{(1)}\right) > \sum_{T_j \in L} p_j^{(1)}$**.** Similarly, as in Case 1, we may assume that at most $n_H - 1$ tasks from L are overlapping, i.e., $|\overline{L}| \leq n_H - 1$. If

this would not be the case, we would have at least n_H tasks from L with the property that assigning any of them to arbitrary $T_i \in H$ would lead to $W_i < 0$, which contradicts Case 2 assumption. Then, similarly as in Case

$$\frac{\mathrm{APX}(I_{\mathcal{MC}})}{\mathrm{OPT}(I_{\mathcal{MC}})} \leq \frac{\sum_{T_i \in H} p_i^{(2)} + \sum_{T_j \in \overline{L}} p_j^{(1)}}{\sum_{T_i \in H} p_i^{(2)}} \leq 1 + \frac{\sum_{T_j \in \overline{L}} p_j^{(1)}}{\sum_{T_i \in H} p_i^{(2)}} \leq 1 + \frac{\sum_{T_j \in \overline{L}} p_j^{(1)}}{\sum_{T_j \in L} p_j^{(1)}} \leq \frac{3}{2}.$$

where the third inequality follows from Case 2 assumption and the fourth inequality from the same arguments as in Case 1. □

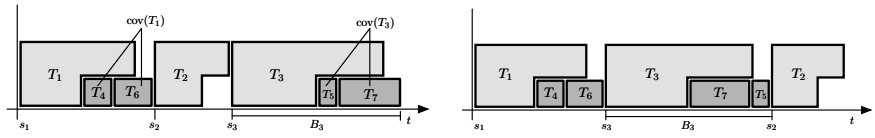
Note that (APX-MC-2) works well in practice even for the general problem, hence we use it as the initial heuristic for branch-and-price algorithm proposed in Section 3.3.3. In the next section, we derive the block MIP formulation that utilizes the structure of optimal permutations.

3.3.2 Block MIP formulation

The proposed MIP model is based on a similar concept as the approximation algorithm described in the previous subsection. The model exploits three symmetries in the problem. The first symmetry comes from the fact that tasks in L with the same processing time are indistinguishable. Therefore, the constraint to schedule all tasks can be given by the requirement to schedule a given number of tasks with a specific processing time, instead of scheduling unique tasks' occurrences. The second symmetry occurs in the ordering of tasks in $\mathrm{cov}(T_i)$. The last symmetry comes from the ordering of sets of tasks that are covered since the $C_{\max}$ criterion is invariant with respect to the ordering of blocks. This property becomes apparent from Figure 3.3, and is proven below.

Let $P = \left\{ p_j^{(1)} \,\middle|\, \forall T_j \in L \right\}$ be the set of unique processing times of tasks in L (i.e., it *is not* a superset) and let $n_p = \left| \left\{ T_j \,|\, \forall T_j \in L : p_j^{(1)} = p \right\} \right|$, i.e., the number of tasks in L with processing time equal to $p \in P$.

The decision variable $x_{i,p}$ states the number of tasks in L with processing time equal to $p \in P$ that are covered by $T_i \in H$, i.e., $x_{i,p} = |\{T_j \,|\, \forall T_j \in \mathrm{cov}(T_i) : p_j^{(1)} = p\}|$. The continuous variable B_i corresponds to the length of $\{T_i\} \cup \mathrm{cov}(T_i)$ block, e.g., see B_3 in Figure 3.3a. The first symmetry is broken by constraint (3.3.3), while the second symmetry is broken by constraint (3.3.2). Finally, the third symmetry is broken by the objective function.



(a) Left-shifted schedule of the canonical permutation.

(b) Equivalent schedule with the same coverage sets represented by a non-canonical permutation.

Figure 3.3: Schedule with two criticality levels.

$$\min \sum_{T_i \in H} B_i \tag{MIP-MC-2}$$

subject to

$$B_i \geq p_i^{(2)} \quad \forall T_i \in H \tag{3.3.1}$$

$$B_i \geq p_i^{(1)} + \sum_{p \in P} p \cdot x_{i,p} \quad \forall T_i \in H \tag{3.3.2}$$

$$\sum_{T_i \in H} x_{i,p} = n_p \quad \forall p \in P \tag{3.3.3}$$

where

$$B_i \geq 0 \quad \forall T_i \in H \tag{3.3.4}$$

$$x_{i,p} \in \mathbb{Z}_0^+ \quad \forall (T_i, p) \in H \times P \tag{3.3.5}$$

The model contains $\Theta(n_H|P|) \subseteq \mathcal{O}(n_H n_L)$ integer variables $x_{i,p}$, which define for each $T_i \in H$, how many tasks in L with the given processing time follow immediately after T_i in a permutation. The final schedule $\boldsymbol{s}$ is given by the left-shifted permutation of tasks $T_i \in H$ interleaved by $T_j \in \mathrm{cov}(T_i)$. In fact, for each solution of MIP formulation (MIP-MC-2), there are $n_H!$ different but equivalent solutions. Figure 3.3b shows one particular solution equivalent to the one in Figure 3.3a. Hence, to obtain a representative solution for this equivalence class, we define *canonical permutation*, which we use to reconstruct the schedule $\boldsymbol{s}$ from a solution of (MIP-MC-2).

The permutation π is the canonical permutation if $\forall T_i, T_j \in H : i < j \implies \pi(i) < \pi(j)$ and $\forall T_i \in H, \forall T_k, T_l \in \mathrm{cov}(T_i) : k < l \implies \pi(i) < \pi(k) < \pi(l)$. Therefore, in a canonical left-shifted schedule, $T_1 \in H$ is scheduled at time $s_1 = 0$. The start time of task $T_q \in H$, $q > 1$ is given by the following recurrent formula:

$$s_q = s_{q-1} + \max \left\{ p_{q-1}^{(2)},\, p_{q-1}^{(1)} + \sum_{T_k \in \mathrm{cov}(T_{q-1})} p_k^{(1)} \right\} \quad \forall q : 1 < q \leq n_H. \tag{3.3.6}$$

The start times of the tasks $T_k \in \mathrm{cov}(T_q)$, $\forall T_q \in H$ are given as

$$s_k = s_q + p_q^{(1)} + \sum_{T_{k'} \in \mathrm{cov}(T_q) : k' < k} p_{k'}^{(1)} \quad \forall T_k \in \mathrm{cov}(T_q). \tag{3.3.7}$$

The makespan $C_{\max}$ of the schedule $\boldsymbol{s} = (s_1, s_2, \ldots, s_n)$ is then

$$C_{\max} = s_{n_H} + \max \left\{ p_{n_H}^{(2)},\, p_{n_H}^{(1)} + \sum_{T_k \in \mathrm{cov}(T_{n_H})} p_k^{(1)} \right\}. \tag{3.3.8}$$

Now, we show that MIP formulation (MIP-MC-2) is correct.

Proposition 6. *Given the optimal solution of* (MIP-MC-2)*, the schedule* $\boldsymbol{s}$ *is feasible and optimal.*

Proof. First, we will show that such schedule $\boldsymbol{s}$ is feasible, and later, that it is optimal. To ensure feasibility, for all tasks in $I_{\mathcal{MC}}$, the conditions specified in Definition need to be satisfied. For all $T_i, T_j \in H$, $i < j$, the start times are set such that

$$s_j \geq \ldots \geq s_{i+1} = s_i + \max \left\{ p_i^{(2)},\, p_i^{(1)} + \sum_{T_k \in \mathrm{cov}(T_i)} p_k^{(1)} \right\} \geq s_i + p_i^{(2)},$$

where the equality follows from (3.3.6). Since $\mathcal{X}_i = \mathcal{X}_j = 2$, the maximal common criticality level of T_i and T_j is 2; thus, $s_j \geq s_i + p_i^{(\min\{\mathcal{X}_i, \mathcal{X}_j\})}$ follows. For all $T_i, T_j \in L$, it holds that $T_i \in \mathrm{cov}(T_k)$, $T_j \in \mathrm{cov}(T_q)$ for some $T_k, T_q \in H$. If $T_k \neq T_q$, then without loss of generality, let us assume that $k < q$, and thus, $s_k \leq s_q$. Therefore in this case, it follows from the definition of the schedule $\boldsymbol{s}$ that $s_k + p_k^{(1)} \leq s_i + p_i^{(1)} \leq s_q \leq s_j$. If $T_k = T_q$ and $i < j$, then $s_i + p_i^{(1)} \leq s_j$ by (3.3.7). For all $T_i \in H$, $T_j \in L$, there are essentially two cases. If $T_j \in \mathrm{cov}(T_i)$, then immediately $s_j \geq s_i + p_i^{(1)}$. If $T_j \notin \mathrm{cov}(T_i)$, then there exists some T_k such that $T_j \in \mathrm{cov}(T_k)$. For $k > i$, we have $s_i + p_i^{(2)} \leq s_k \leq s_j$, and for $k < i$, we have $s_k \leq s_j + p_j^{(1)} \leq s_i$.

Now, we show that $\boldsymbol{s}$ has the optimal makespan. Applying recursively (3.3.6) to makespan (3.3.8) leads to

$$C_{\max} = \sum_{q=1}^{n_H} \max \left\{ p_q^{(2)},\, p_q^{(1)} + \sum_{T_k \in \mathrm{cov}(T_q)} p_k^{(1)} \right\}.$$

Since the objective of (MIP-MC-2) is a sum of B_is and each B_i is by constraints (3.3.1) and (3.3.2) equal to the maximum of terms in the above expression, (MIP-MC-2) minimizes $C_{\max}$. □

The formulation (MIP-MC-2) provides additional insights into MC-2 problem. Its structure is related to the scheduling problem of parallel machines with the total tardiness criterion $P||\sum T_j$ [159]. It is possible to polynomially reduce a special case of MC-2 when all tasks in $T_i \in H$ have the same constant difference $p_i^{(2)} - p_i^{(1)} = \Delta$ between the second and the first level to problem $P|d_j = \Delta|\sum T_j$. The transformation generates n_H machines and n_L tasks with a common due date Δ. Then, it can be shown that for every optimal solution of such instance of $P|d_j = \Delta|\sum T_j$ holds that (i) the completion time of the last task on each machine is greater than or equal to Δ or (ii) all start times are smaller than Δ. Under the considered transformation, in case (i) the solution produced by $P|d_j = \Delta|\sum T_j$ is optimal for MC-2 since its $C_{\max}$ matches a lower bound $\sum_{T_k \in I_{\mathcal{MC}}} p_k^{(1)}$. In case (ii), the total tardiness of this instance of $P|d_j = \Delta|\sum T_j$ is equal to the sum of processing times that exceed the common due date Δ, since at most n_H tasks have non-zero tardiness in an optimal solution. We note that the reduction of the general case of MC-2 to $Q||\sum T_j$, where the speeds of machines are set proportionally to the differences $p_i^{(2)} - p_i^{(1)}$ in order to capture the fact that tasks in H are unequal is not exact since in case (ii) the contribution of each machine to the total tardiness is skewed by the machine speed.

3.3.3 Branch-and-price decomposition

In this section, we propose a branch-and-price decomposition algorithm [11] to solve the problem. In general, the problem is decomposed into several *pricing problems* and a single *master problem* that couples them. We view tasks in H as individual subproblems that resolve the question which tasks in L should be covered by which task $T_i \in H$. These subproblems are coupled by the criterion that minimizes the sum of amounts by which the second levels $p_i^{(2)}$ of tasks in H are exceeded. See, for example, the schedule in Figure 3.3b. Here, the second level of task T_3 is exceeded by the amount of $p_5^{(1)}$. This is an equivalent way of expressing $C_{\max}$ criterion. To find out how to improve the current solution, we solve a pricing problem, which suggests new coverage sets $\mathrm{cov}(T_i)$ that can improve the objective with the current solution of the master problem.

The master problem contains cover constraints requiring that all tasks in L are scheduled. Individual pricing problems communicate with the master problem through shadow prices of cover constraints. Shadow prices express the need to schedule the particular tasks in L. The problem (BNP-MC-2) represents the master problem, which is a *linear programming* (LP) problem with an exponential number of variables (i.e., all possible coverage sets). Such problems can be solved efficiently through *column generation* (CG) [59], which utilizes the fact that only a polynomial-sized subset of variables has a non-zero value in an optimal solution of the problem. Each variable is associated with a column of coefficients in the constraint matrix and the objective coefficient. CG starts with a few columns and progressively puts new variables into the model. New columns are generated by a dedicated algorithm that takes the current dual LP solution of (BNP-MC-2) and produces a new column that can improve the objective value, or the algorithm proves that the current solution of LP is optimal. Using CG, we can prove the optimality of the full model (with an exponential number of variables) even without enumerating all variables. Therefore, by solving the model, only a small subset of all variables is typically generated.

3.3.3.1 Master problem

The master problem resolves the question, how to split a set of tasks L into coverage sets such that $L = \bigcup_{T_i \in H} \mathrm{cov}(T_i)$ while the makespan is minimal. It uses an indicator variable $x_i^{(s)}$, stating whether the particular *configuration* $s \in S_i$ is covered by $T_i \in H$. A configuration $s \in S_i$ encodes the number of tasks with the given processing time occurring in $\mathrm{cov}(T_i)$ into the vector $\boldsymbol{a}_i^{(s)}$. Hence, the entry $a_{i,p}^{(s)}$ denotes the number of tasks in L with processing time equal to p, which is covered by T_i in configuration s. S_i is the set of all configurations available for task T_i. See an example in Figure 3.3a. Here, T_4, T_5, T_6, and T_7 have different processing times. Hence, the schedule displays the following three configurations: $\boldsymbol{a}_1^{(s_1)} = (1,0,1,0)^\top$, $s_1 \in S_1$ $\boldsymbol{a}_2^{(s_2)} = (0,0,0,0)^\top$, $s_2 \in S_2$, and $\boldsymbol{a}_3^{(s_3)} = (0,1,0,1)^\top$, $s_3 \in S_3$.

The master problem can be stated with the following LP:

$$\min_{\boldsymbol{x}} \sum_{T_i \in H} \sum_{s \in S_i} O_i^{(s)} x_i^{(s)} \qquad \text{(BNP-MC-2)}$$

subject to

$$\sum_{T_i \in H} \sum_{s \in S_i} a_{i,p}^{(s)} x_i^{(s)} \geq n_p \quad \forall p \in P \tag{3.3.9}$$

$$\sum_{s \in S_i} x_i^{(s)} \leq 1 \quad \forall T_i \in H \tag{3.3.10}$$

where

$$x_i^{(s)} \geq 0 \quad \forall s \in S_i, \forall T_i \in H \tag{3.3.11}$$

The objective coefficient is given as $O_i^{(s)} = \max\left\{p_i^{(1)} + \sum_{p \in P} p \cdot a_{i,p}^{(s)} - p_i^{(2)}, 0\right\}$, $\forall T_i \in H, \forall s \in S_i$, where $a_i^{(s)} \in \mathbb{Z}_0^{|P|}$. The constraint (3.3.9) ensures that each task in L is scheduled, while the constraint (3.3.10) states that each task $T_i \in H$ covers at most one configuration $s \in S_i$. In the beginning, the master problem is solved with restricted configuration sets S_i containing only the minimal number of configurations, ensuring the feasibility of the model (BNP-MC-2) and with an empty configuration $s_0 \in S_i, \forall T_i \in H$. The empty configuration s_0 denotes the empty covering set, i.e., $\text{cov}(T_i) = \emptyset$. During the solution of (BNP-MC-2), more configurations are being added. To efficiently determinate which configuration to add at each step, we need to consider the dual form of the LP problem, which is stated as follows:

$$\max_{\boldsymbol{y},\boldsymbol{\gamma}} \sum_{p \in P} n_p y_p + \sum_{T_i \in H} \gamma_i \tag{BNP-DMC2}$$

subject to

$$\sum_{p \in P} a_{i,p}^{(s)} y_p + \gamma_i \leq O_i^{(s)} \quad \forall T_i \in H, \forall s \in S_i \tag{3.3.12}$$

where

$$y_p \geq 0 \quad \forall p \in P \tag{3.3.13}$$

$$\gamma_i \leq 0 \quad \forall T_i \in H \tag{3.3.14}$$

The values of dual variables $\boldsymbol{y}$ and $\boldsymbol{\gamma}$ are used to decide which configuration to generate to improve the current solution of (BNP-MC-2). This is achieved using the pricing problem, which generates a constraint of type (3.3.12) that is violated by the current values of $\boldsymbol{y}$ and $\boldsymbol{\gamma}$. In the next section, we will derive the pricing problem.

3.3.3.2 Pricing problem

The pricing problem determines whether there exists a constraint that violates the current dual solution or whether the primary solution is optimal and no such constraint can be found. Due to LP duality, each constraint (3.3.12) corresponds to the $x_i^{(s)}$ variable in the primary model (BNP-MC-2), and hence, to the whole column. To determine which column can enter the basis, one needs to find a violated constraint in the dual form (BNP-DMC2). Therefore, at each iteration of the branch-and-price algorithm, we ask whether there exists a configuration $s \in S_i$

(a column $\boldsymbol{a}_i^{(s)}$ and objective coefficient $O_i^{(s)}$) that violates one of the constraints (3.3.12) with the current dual solution $\hat{\boldsymbol{y}}$, $\hat{\boldsymbol{\gamma}}$ of the master problem. This involves deciding whether the following expression

$$0 > \max\{p_i^{(1)} + \sum_{p\in P} p \cdot a_{i,p}^{(s)} - p_i^{(2)}, 0\} - \hat{\gamma}_i - \sum_{p\in P} a_{i,p}^{(s)} \hat{y}_p = \mu_i \tag{3.3.15}$$

holds for the given fixed values of $\hat{\gamma}_i$ and $\hat{\boldsymbol{y}}$. If one is interested in a column with the lowest reduced cost μ_i, it is equivalent to the problem

$$\min_{\boldsymbol{a}} \max\{p_i^{(1)} + \sum_{p\in P} p \cdot a_{i,p}^{(s)} - p_i^{(2)}, 0\} - \sum_{p\in P} a_{i,p}^{(s)} \hat{y}_p \tag{BNP-MC-2-PP}$$

$$a_{i,p}^{(s)} \in \mathbb{Z}_0^+ \quad \forall p \in P \tag{3.3.16}$$

Writing it down as an MIP leads to

$$\max_{\boldsymbol{a},z} \quad \sum_{p\in P} a_{i,p}^{(s)} \hat{y}_p - z \tag{3.3.17}$$

subject to

$$\sum_{p\in P} p \cdot a_{i,p}^{(s)} \leq p_i^{(2)} - p_i^{(1)} + z \tag{3.3.18}$$

where

$$z \geq 0 \tag{3.3.19}$$

$$a_{i,p}^{(s)} \in \mathbb{Z}_0 \quad \forall p \in P \tag{3.3.20}$$

which can be seen as a variant of *Knapsack Problem* [112] with items whose values are given by the current shadow prices of assignment constraints (3.3.9) and weights are given by processing times of tasks that need to be fitted into the knapsack of size given by the size of the gap of $p_i^{(2)} - p_i^{(1)}$. However, the difference is that there is a possibility to enlarge the size of the knapsack by some amount while incurring the identical loss in the objective function. The structure of the pricing problem shows a connection to 1D cutting stock problem [57], where the pricing problem is the classical Knapsack Problem, since the length of any material roll in the cutting stock cannot be exceeded.

An example of the pricing problem with $n_L = 5$ tasks for the particular $T_i \in H$ is displayed in Table 3.2 and the corresponding optimal solution in Figure 3.4. In this solution, T_1, T_3, and T_5 are selected to form configuration $s \in S_i$ with $\boldsymbol{a}_i^{(s)} = (1, 0, 1, 0, 1)^\top$ and $O_i^{(s)} = 1$.

Therefore, for $z = 0$, the pricing problem is an ordinary Knapsack Problem. Since the processing times are integers, the variable z will also be always an integer in an optimal solution. Having a pseudopolynomial upper bound on z, we can solve different knapsack problems for all possible values of z separately. However, the pricing problem can be solved even faster. Next, we will show that the pricing problem is solvable in a pseudopolynomial time, and propose a dynamic programming algorithm to solve it.

j	1	2	3	4	5
$p_j^{(1)}$	2	10	3	7	5
$\hat{y}_j$	6.0	0.5	5.5	1.0	4.5

Table 3.2: Example instance of the pricing problem for $T_i \in H$, $\boldsymbol{P}_i = (4, 13)$.

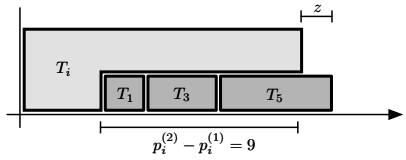


Figure 3.4: Optimal solution to the pricing problem instance from Table 3.2.

The constraints (3.3.9) in a master problem enforce the required number of tasks in L with the given processing time to be scheduled. For convenience, let us work in the pricing problem with the specific occurrences of tasks in L instead. Hence, for each specific task size $p \in P$, we choose to work with n_p number of tasks with values $\hat{y}_j = \hat{y}_p$. In this way, the pricing problem respects the available number of tasks in L with the given processing time.

Proposition 7. *The pricing problem can be solved in a pseudopolynomial time in the maximal length of a task.*

Proof. For any fixed $z \in \mathbb{N}_0$, the pricing problem corresponding to task $T_i \in H$ with $W = p_i^{(2)} - p_i^{(1)}$ becomes the knapsack problem with maximum capacity $W+z$, which can be solved in $\mathcal{O}(n_L(W + z))$ by dynamic programming. Since in any solution of the pricing problem, we pack items with total size of at most $K = \sum_{T_j \in L} p_j^{(1)}$, we can set an upper bound on z as $K \leq n_L \max_{T_j \in L} p_j^{(1)}$. Therefore, the pricing problem can be solved as K independent knapsack problems while picking the best solution among them in total $\mathcal{O}(n_L K(W + K))$ time. □

However, we can do better. The pricing problem can be solved by the following dynamic programming recurrence relation. Let $U(k, j)$ be an optimal solution to the pricing problem with capacity k and tasks $\{T_1, \ldots, T_j\} \subseteq L$. Let $W = p_i^{(2)} - p_i^{(1)}$. For any $k, j \leq 0$, we set $U(k, j) = 0$. Then, the recurrent relation is given for $k \leq W$ as follows:

$$U(k,j) \leftarrow \begin{cases} \max\{U(k, j-1),\ \hat{y}_j + U(k - p_j^{(1)}, j-1)\} & \text{if } p_j^{(1)} \leq k \\ U(k, j-1) & \text{otherwise} \end{cases} \tag{3.3.21}$$

and for $k > W$, as

$$U(k,j) \leftarrow \begin{cases} \max\{U(k, j-1),\ \hat{y}_j + U(k - p_j^{(1)}, j-1) - (k - W)\} & \text{if } k - p_j^{(1)} \leq W \\ \max\{U(k, j-1),\ \hat{y}_j + U(k - p_j^{(1)}, j-1) - p_j^{(1)}\} & \text{if } k - p_j^{(1)} > W \end{cases} \tag{3.3.22}$$

The optimal solution of the pricing problem is then given as $\hat{k} = \arg\max_{k \in [W+K]} U(k, n_L)$ with the objective value $U(\hat{k}, n_L)$. If $-U(\hat{k}, n_L) - \gamma_i < 0$, then the set of tasks in the solution corresponds to the new column that can enter the basis (i.e., it has the so-called *negative reduced cost*). The new column $\boldsymbol{a}_i^{(s)}$ has the objective coefficient $O_i^{(s)} = \max\{\hat{k} - W, 0\}$ and its entries are given by the number of tasks with the given size contained in the solution of $U(\hat{k}, n_L)$.

The worst-case total running time of the algorithm is $\mathcal{O}(n_L(W+K))$. However, in some cases, the pricing problem can be further simplified by fixing the set of tasks that are necessarily included in an optimal solution.

Lemma 5. *Every task $T_j \in L$ with $\hat{y}_j / p_j^{(1)} \geq 1$ is included in an optimal solution of the pricing problem.*

Lemma 5 is due to the influence of $z \in \mathbb{R}_0^+$ variable in MIP (3.3.17) to its criterion. If for a task, $T_j \in L$ holds $\hat{y}_j \geq p_j^{(1)}$, then taking it into the solution cannot hurt the objective, since an improvement $\hat{y}_j - p_j^{(1)} \geq 0$ is achieved by enlarging z by the amount of $p_j^{(1)}$. In the example in Table 3.2, this rule suggests us to include tasks T_1 and T_3. Lemma 5 is used in the algorithm for solving the pricing problem in the following way. The set $Q \subseteq L$ of tasks satisfying $\forall T_j \in Q : \hat{y}_j / p_j^{(1)} \geq 1$ is taken out of the pricing problem instance and the capacity W is decreased by $\sum_{T_j \in Q} p_j^{(1)}$. Then, the pricing problem is solved only for the remaining tasks.

3.3.3.3 Initial solution and branching

The branch-and-price algorithm starts with an initial set of columns that leads to a feasible solution of the model (BNP-MC-2). In our case, the initial solution comes from the (APX-MC-2) approximation algorithm, where set $\mathrm{cov}(T_i)$ forms the corresponding column $\boldsymbol{a}_i^{(s)}$. After the master problem (BNP-MC-2) is solved with the given set of columns, a subproblem corresponding to some task $T_i \in H$ is selected. In our case, we solve the subproblems in the non-increasing order of $\hat{\gamma}_i$ until there are no more columns with a negative reduced cost.

The optimal solution to the master problem (BNP-MC-2) can be fractional in general. Therefore, to ensure an integer solution, a branching is employed inside the branch-and-price algorithm. Hence, every master problem acts as a node in the branch-and-bound tree. The tree is searched in the depth-first fashion. We introduce a branching strategy on the original variables, i.e., based on $x_{i,p}$ variables from (MIP-MC-2). It branches on the decision of how many tasks in L with processing time p are present in $\mathrm{cov}(T_i)$. Therefore, given a fractional value of the corresponding original variable $x_{i,p}^*$ obtained from the solution of the master problem, two branches with constraints $\lfloor x_{i,p}^* \rfloor \leq x_{i,p}$ and $\lceil x_{i,p}^* \rceil \geq x_{i,p}$ are created. In the first case, the constraint is reflected in the pricing problem by reducing the capacity W by $p \cdot \lfloor x_{i,p}^* \rfloor$ and taking those tasks into the solution. In the latter case, the constraint is enforced by setting shadow prices $\hat{y}_j$ to $-\infty$ for tasks $T_j \in L' \subseteq \left\{ T_j \,\middle|\, \forall T_j \in L : p_j^{(1)} = p \right\}$, $|L'| = n_p - \lceil x_{i,p}^* \rceil$. Note that in the $\lceil x_{i,p}^* \rceil \geq x_{i,p}$ branch, Lemma 5 may suggest to take some tasks that are forbidden in this branch. In this case, Lemma 5 does not apply. The choice of the variable to branch on is performed by selecting the corresponding original variable with the most fractional value, i.e., the one maximizing $\left| \lfloor x_{i,p}^* + 0.5 \rfloor - x_{i,p}^* \right|$ function.

3.4 Problem with three criticality levels

In this section, we generalize the results developed in Section 3.3 for working with more criticality levels. We show that optimal schedules for problems with an arbitrary number of criticality levels can be represented by trees. Based on this finding, we give a computationally efficient scheduling algorithm for the problem with three criticality levels.

3.4.1 Tree schedule structure

For simplicity, let us assume the problem with three criticality levels and its solution depicted in Figure 3.5a. Note that the makespan of the solution is given by the sum of lengths of blocks D_1 and D_2 formed by tasks with criticality level of three. This is due to the analogous reason as in the case with two criticality levels described in Section 3.3.2 since any permutation of blocks achieves the same makespan.

The length of the block D_1 is given by the maximum between $p_1^{(3)}$ and the sum of lengths of blocks B_1, B_3, and B_4 formed by tasks with the criticality of two. Applying the above reasoning recursively, an arbitrary order of blocks B_1, B_3, and B_4 achieves the same total length. To define the block B_1, let us introduce the so-called *restricted task*:

Definition. *Let $T_i \in I_{\mathcal{MC}}$, $\mathcal{X}_i > 1$ be an F-shaped task. Then, T_i' is called the restriction of T_i and is given as*

$$\mathcal{X}_i' = \mathcal{X}_i - 1, \quad \boldsymbol{P}_i' = \left(p_i^{(1)}, \ldots, p_i^{(\mathcal{X}_i - 1)}\right),$$

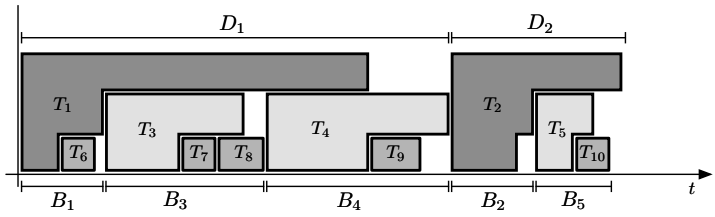
i.e., it is an F-shape that remains after removing the highest criticality level $\mathcal{X}_i$ from T_i.

In Figure 3.5a, the permutation defining the order of tasks in this complete solution is given by a nested system of sets $\{\mathrm{cov}(T_1), \mathrm{cov}(T_2)\}$, $\mathrm{cov}(T_1) = \{T_1', T_3, T_4\}$, $\mathrm{cov}(T_1') = \{T_6\}$ and $\mathrm{cov}(T_2) = \{T_2', T_5\}$, $\mathrm{cov}(T_3) = \{T_7, T_8\}$, $\mathrm{cov}(T_4) = \{T_9\}$, $\mathrm{cov}(T_2') = \emptyset$, $\mathrm{cov}(T_5) = \{T_{10}\}$.

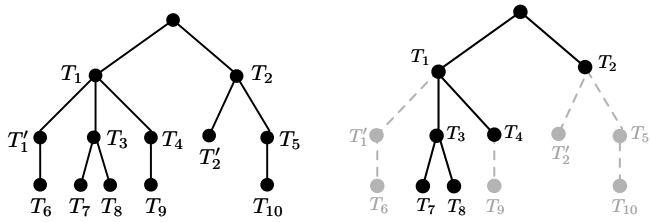
Such a system of sets can be conveniently represented by a tree describing coverage relations. Therefore, we establish the relation between the schedules and trees:

Lemma 6. *An optimal schedule of the problem MC-$\mathcal{L}$ is representable by a tree.*

For the problem with $\mathcal{L}$ criticality levels, the solution is given by a rooted tree with $\mathcal{L} + 1$ layers, where the root (0-th level) is a dummy vertex and vertices in ℓ-th layer, $\ell \geq 1$, are given by all tasks $T_i \in I_{\mathcal{MC}}$ with criticality $\mathcal{X}_i = \mathcal{L} - \ell + 1$. Furthermore, the immediate successors of a vertex T_i are tasks in $\mathrm{cov}(T_i)$ (including its restriction T_i'). Examples of a tree and the corresponding solution are depicted in Figure 3.5b and Figure 3.5a, respectively. Note that swapping subtrees rooted at T_3 and T_4 in Figure 3.5b leads to an isomorphic graph. This transformation can be viewed as permuting B_3 and B_4 blocks inside the schedule in Figure 3.5a, which leads to different but an equivalent schedule. Therefore, isomorphic trees



(a) Example of a schedule with three criticality levels.



(b) Tree representation.

(c) Critical subtree.

Figure 3.5: Schedule with three criticality levels and representation of its permutation as a tree.

represent equivalent schedules; hence, we optimize over non-isomorphic ones to mitigate symmetries.

The actual schedule corresponding to a tree is obtained by traversing the tree in the *preorder* fashion; every time a vertex of the tree corresponding to a non-restricted task is visited, the corresponding task is scheduled at the earliest possible start time. The makespan of the schedule is given by the so-called *critical subtree*, which is a subgraph of the tree of the solution.

Definition (Critical Subtree)**.** *Given a tree of solution K, a critical subtree $C \subseteq K$ is a minimal subgraph of K that achieves the same makespan as K.*

Figure 3.5c shows an example that highlights a critical subtree of the schedule in Figure 3.5a. Basically, this is the minimal set of tasks that causes the achieved makespan of the solution. Furthermore, we show that optimal trees consist of optimal subtrees, as stated by the following proposition:

Proposition 8. *There is an optimal tree of the solution of the problem MC-$\mathcal{L}$, such that every subtree rooted at a vertex corresponding to task $T_i \in I_{\mathcal{MC}}$, $\mathcal{X}_i > 1$ is an optimal tree of all its child vertices with respect to the problem MC-($\mathcal{X}_i - 1$).*

Proof. By contradiction. Let us denote the subtree rooted under T_i as tree(T_i). Suppose a unique optimal solution represented by a tree K that contains a subtree tree(T_i) that is not an optimal tree. For such a solution, there are two cases. Either for every critical subtree $C \subseteq K$, there exists a task $T_j \in \text{tree}(T_i) \cap C$ or not. If yes, then by rearranging tree(T_i) into an optimal one would decrease the makespan of tree K, which is by the assumption optimal. In the other case, by rearranging tree(T_i) into the optimal one would not increase its makespan, and thus, no task contained in tree(T_i) would enter a critical subtree C. Therefore, the makespan of C, and thus, K would not increase. □

Proposition 8 states that for any problem instance, there is an optimal solution with this property. However, in general, the optimal solution tree cannot be constructed in the bottom-up fashion, i.e., constructing optimal subtrees of tasks (and their restrictions) with criticality one and two and those joining with tasks of criticality three and so on. In fact, it can be shown that this procedure would yield suboptimal solutions. Hence, one has to first reason about which tasks fall into which subtree, and given that such subtree is an optimal tree. However, Proposition 8 still provides a useful insight into the structure of optimal solutions. We employ it in the branch-and-price decomposition algorithm for the problem with three criticality levels in the following section. The concept of the decomposition is similar to the one proposed in Section 3.3.3 – to form blocks of tasks of the highest criticality by exploring possible options of how to cover the remaining tasks by them. As a consequence of Proposition 8, given the set of tasks to be covered by another task, we know that they need to be scheduled there optimally according to the $C_{\max}$ criterion of the problem with one criticality level less. The master problem is used to efficiently explore the options of which task should be covered by which tasks, while the pricing problem, given the coverages, schedules them optimally.

3.4.2 Branch-and-price decomposition for MC-3

3.4.2.1 Master problem

For clarity, let us denote the set of all tasks with criticality three as $D = \{T_k \,|\, \forall T_k \in I_{\mathcal{MC}} : \mathcal{X}_k = 3\}$, while the meaning of sets H, L, and P remains the same as in Section 3.3.2. The general idea here is similar to that in Section 3.3.3 for two criticality levels. Therefore, the master problem assigns tasks in $H \cup L$ to coverage sets associated with tasks in D. This can be stated as follows:

$$\min_{\boldsymbol{x}} \sum_{T_k \in D} \sum_{s \in S_k} O_k^{(s)} x_k^{(s)} \tag{BNP-MC-3}$$

subject to

$$\sum_{T_k \in D} \sum_{s \in S_k} a_{k,p}^{(s)} x_k^{(s)} \geq n_p \quad \forall p \in P \tag{3.4.1}$$

$$\sum_{T_k \in D} \sum_{s \in S_k} b_{k,i}^{(s)} x_k^{(s)} \geq 1 \quad \forall T_i \in H \tag{3.4.2}$$

$$\sum_{s \in S_k} x_k^{(s)} \leq 1 \quad \forall T_k \in D \tag{3.4.3}$$

where

$$x_k^{(s)} \geq 0 \quad \forall s \in S_k, \forall T_k \in D \tag{3.4.4}$$

The column coefficient is given as $O_k^{(s)} = \max\left\{\sum_{T_i \in \mathrm{cov}(T_k)} B_i - p_k^{(3)},\, 0\right\}$, where the term $\mathrm{cov}(T_k)$ is a function of configuration s. The constant B_i denotes the length of the block given by $T_i \in \mathrm{cov}(T_k)$, defined in the same way as in (MIP-MC-2). The variable $x_k^{(s)}$ states whether $T_k \in D$ covers the set of trees $s \in S_k$, where s is given by two vectors $\boldsymbol{a}_k^{(s)}$ and $\boldsymbol{b}_k^{(s)}$. The coefficient $a_{k,p}^{(s)}$ states how many tasks in L with

processing time equal to p are rooted under the subtree of $T_k \in D$. The vector $\boldsymbol{b}_k^{(s)}$ is the characteristic vector (i.e., a vector with binary entries denoting the presence of an element) of tasks in H rooted under the subtree of $T_k \in D$.

The constraints (3.4.1) and (3.4.2) ensure that all tasks in $H \cup L$ are scheduled, while the constraint (3.4.3) states that at most one configuration is selected per task $T_k \in D$. The problem of how to generate a new configuration s that can improve the current solution and the computation of the column coefficient is solved by the pricing problem.

3.4.2.2 Pricing problem

Since now the pricing problem embeds the MC-2 problem, which is strongly $\mathcal{NP}$-hard, there is no pseudopolynomial algorithm solving the problem unless $\mathcal{P} = \mathcal{NP}$. Hence, we formulate it as an MIP model. The complete description of the pricing problem corresponding to a task $T_k \in D$ can be stated as follows:

$$\max \sum_{T_i \in H} \hat{y}_i x_i + \sum_{p \in P} \hat{y}_p \sum_{T_i \in H \cup \{T'_k\}} q_{i,p} - z \qquad \text{(BNP-MC-3-PP)}$$

subject to

$$\sum_{T_i \in H \cup \{T'_k\}} B_i - p_i^{(2)}(1 - x_i) \leq p_k^{(3)} + z \qquad (3.4.5)$$

$$B_i \geq p_i^{(2)} \quad \forall T_i \in H \cup \{T'_k\} \qquad (3.4.6)$$

$$B_i \geq p_i^{(1)} + \sum_{p \in P} p \cdot q_{i,p} \quad \forall T_i \in H \cup \{T'_k\} \qquad (3.4.7)$$

$$\sum_{T_i \in H \cup \{T'_k\}} q_{i,p} \leq n_p \quad \forall p \in P \qquad (3.4.8)$$

$$\sum_{p \in P} q_{i,p} \leq n_L x_i \quad \forall T_i \in H \cup \{T'_k\} \qquad (3.4.9)$$

$$x_k = 1 \qquad (3.4.10)$$

where

$$z \geq 0 \qquad (3.4.11)$$

$$B_i \geq 0 \quad T_i \in H \cup \{T'_k\} \qquad (3.4.12)$$

$$x_i \in \{0, 1\} \quad \forall T_i \in H \cup \{T'_k\} \qquad (3.4.13)$$

$$q_{i,p} \in \mathbb{Z}_0^+ \quad \forall T_i \in H \cup \{T'_k\}, \forall p \in P \qquad (3.4.14)$$

The coefficients $\hat{y}_p$ are shadow prices for constraints (3.4.1) and coefficients $\hat{y}_i$ correspond to shadow prices for constraints (3.4.2). The model assigns the given number of tasks in L with processing time equal to p using $q_{i,p}$ variable to the selected tasks from H that are selected using x_i variables. Moreover, for the given subproblem corresponding to the task $T_k \in D$, we work inside the model with its restriction T'_k, which is always included in every solution by constraint (3.4.10). Finally, if the optimal objective value is greater than $-\hat{\gamma}_k$, which is the shadow price for the constraint (3.4.3) associated with the current subproblem T_k, then a column

that can improve the current solution of the master problem exists. The column coefficient $O_k^{(s)}$ is then given as the value of z variable in an optimal solution.

The advantage of (BNP-MC-3-PP) MIP model is that it does not contain a big-M constant. Furthermore, the sufficient condition for selecting a task in L into an optimal solution suggested by Lemma 5 also applies here. Moreover, a similar statement about tasks in H is also valid; if $\hat{y}_i/p_i^{(2)} \geq 1$ for any $T_i \in H$, then T_i can be taken into an optimal solution too.

3.4.2.3 Initial solution and branching

As an initial solution, we use a greedy algorithm that works in two steps. First, a new instance $I'_{\mathcal{MC}}$ of the MC-2 problem is created by taking $I'_{\mathcal{MC}} = L \cup H \cup D'$, where $D' = \{T'_k \,|\, \forall T_k \in D\}$, i.e., the set of restrictions of tasks in D. A solution to this problem instance defines coverages corresponding to two bottom layers of the solution tree (Figure 3.5b). The coverages in the top level of the tree are determined by the solution of yet another MC-2 problem instance following from the solution of $I'_{\mathcal{MC}}$, consisting of tasks $T_{k^\star}$, $\mathcal{X}_{k^\star} = 2$ with processing times $p_{k^\star}^{(1)} = \max\left\{p_k^{(2)}, p_k^{(1)} + \sum_{T_j \in \mathrm{cov}(T'_k)} p_j^{(1)}\right\}$ and $p_{k^\star}^{(2)} = \max\left\{p_k^{(3)}, p_{k^\star}^{(1)}\right\}$ for all $T_k \in D$. Tasks with criticality one are given by the original tasks in H, with their coverage sets obtained from the solution of $I'_{\mathcal{MC}}$; e.g., T_3 and $\mathrm{cov}(T_3)$ from Figure 3.5a are treated as a single task with processing time $p_3^{(1)} = B_3$.

The branching is realized for each $T_k \in D$ both on the number of assigned tasks in L with the given $p \in P$ in the same way as in Section 3.3.3.3. For tasks in H, 0/1 branching is performed. We use the most fractional value strategy for selecting the variable to branch on. The conditions imposed by the branching are taken into the account by putting equivalent conditions into pricing problem (BNP-MC-3-PP).

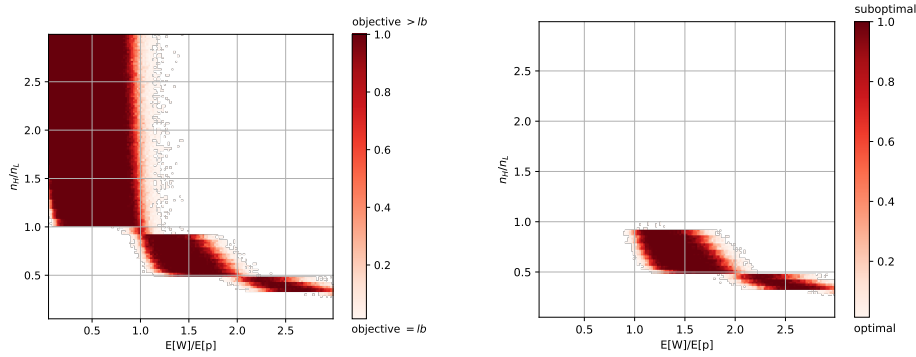
3.5 Computational experiments

In this section, we provide experimental results obtained using the above-described methods. The testing environment consists of a computer with Intel Xeon E5-2620 v2 @ 2.10 GHz equipped with 64 GB RAM running Gentoo Linux. The algorithms are implemented in Python 3.5 and Java 8. As external solvers, Gurobi Optimizer 7.0.2 and IBM CPLEX 12.7.1 are used.

3.5.1 Results of the approximation algorithm

First, we estimate the phase transition [162] of the problem MC-2. This is a set of threshold values on numerical parameters of instances of the problem that separates *easy* instances from the *hard* ones. From our computational experience, we have determined two main parameters that influence the difficulty of an instance the most. The first parameter is the ratio between the number of tasks of different criticalities, written as n_H/n_L. The second parameter is the ratio between the mean value of the gap $W_i = p_i^{(2)} - p_i^{(1)}$ of tasks in $T_i \in H$ and that of processing time $p_j^{(1)}$ of tasks in $T_j \in L$. We denote this ratio of processing times as $\mathbb{E}[W]/\mathbb{E}[p]$,

where $\mathbb{E}$ states for the expected value. The choice of these parameters naturally arises from the way the makespan of a solution is given.



(a) Fraction of solutions not matching a lower bound.

(b) Fraction of suboptimal solutions.

Figure 3.6: Results of (APX-MC-2) approximation algorithm in the instance space of MC-2.

We say that an instance is *easy* if the objective of the solution provided by the (APX-MC-2) approximation algorithm equals to a lower bound. Recall that a lower bound on the makespan in problem MC-2 is given as

$$lb = \max\left\{\sum_{T_i \in H} p_i^{(2)}, \sum_{T_k \in I_{\mathcal{MC}}} p_k^{(1)}\right\}.$$

Having an instance with relatively low (or high) ratios n_H/n_L and $\mathbb{E}[W]/\mathbb{E}[p]$ makes (APX-MC-2) approximation algorithm likely to result into a solution whose makespan matches the lower bound lb, and thus, solving the instance optimally. Therefore, to assess where the hard instances are located in the space of instances, we evaluate the solutions produced by the (APX-MC-2) approximation algorithm using the grid search on a large set of parameter values. Data in Figure 3.6 are obtained for the problem with $n = 50$ tasks, with each data point averaged over 75 independent samples. Figure 3.6a shows the fraction of instances where the makespan of solutions does not match the lower bound lb. Therefore, blank areas are filled with instances for which the (APX-MC-2) approximation algorithm produces solutions with the objective matching the lower bound lb.

In general, even when the solution objective value is not equal to a lower bound, the solution still might be optimal. Therefore, we compare the results obtained by the (APX-MC-2) approximation algorithm with those obtained by the optimal ones. In Figure 3.6b, the ratio of sub-optimally solved instances by (APX-MC-2) is shown. Here, even though solutions of instances with $\mathbb{E}[W]/\mathbb{E}[p] \leq 1$ do not match a lower bound, they are mostly solved optimally. Furthermore, it empirically shows all instances where $n_H \geq n_L$ are solved optimally by the (APX-MC-2) approximation algorithm.

We observe that the position of points in Figure 3.6 is invariant to the different values of n. The cluster of points in Figure 3.6b displays where the difficult instances of the MC-2 problem are located in the instance space.

3.5.2 Computational time for MC-2 problem

In this section, we evaluate algorithms proposed in Section 3.3.2 and 3.3.3. We have used three different sets of instances; each set consists of multiple batches that differ in the total number of tasks n. Each of these batches contains 40 instances. Table 3.3 summarizes the results for instances that are generated from the distribution corresponding to the cluster of points depicted in Figure 3.6b, which correspond to difficult instances. We denote this dataset as *MC-2-LOP*. Table 3.4 shows the results for instances that are located at the same position in the instance plane but have more than three times larger standard deviation of processing times of tasks in L, thus resulting in a larger set P. We denote this dataset as *MC-2-HIP*. In practical problems related to message scheduling [64], tasks usually have length given as a power of two [65]. This follows from the implementation aspects of real-life computer systems (i.e., lengths of packets). Thus, we also generate a set of instances where processing times of tasks and their prolongations are given as a 2^k, $k \in \mathbb{N}_0$, denoted as *MC-2-2K*. We perform experiments with range $k \in [0, 7]$, and display the results in Table 3.5.

In all tables, the column *gap* is the mean optimality gap proven by the solver within the time limit $t_{max} = 300$ s, and is given as $100 \cdot \frac{ub-lb}{ub}$, where ub is the objective value of the best solution found, while lb is the best proven lower bound. The column t denotes the mean computational time required to prove the optimality of an integer solution (measured in seconds) for the instances computed within the time limit. Columns *gap* and t report two values separated by the slash symbol according to whether multithreading with 12 CPU cores for a single run (MT) is allowed or just a single thread (ST) is used. In case of (BNP-MC-2) algorithm, only the ST performance is reported, owing to its implementation. For all methods, the lower bound computed in the root node is reported as a single value as it does not depend on the computing power available.

The dash symbol denotes that for no instance in the batch, the optimality of an integer solution is proven within the time limit (although a feasible solution is found for each instance in any experiment). Finally, the column denoted as *gen* states the mean number of columns generated during the whole run of (BNP-MC-2) algorithm (measured in *kilocolumns*, i.e., thousands of columns) across all visited nodes. We compare our methods with the currently best-known exact method [81]. The results of their MIP model are given in the column entitled *Relative-Order MIP*.

	(MIP-MC-2)		(BNP-MC-2)			Relative-Order MIP [81]	
n tasks	gap [%] MT/ST	t [s] MT/ST	gap [%] ST	gen [kcols]	t [s] ST	gap [%] MT/ST	t [s] MT/ST
10	0.00 (±0.00) / 0.00 (±0.00)	< 0.1 / < 0.1	0.00 (±0.00)	< 0.1	0.2 (±0.0)	0.00 (±0.00) / 0.00 (±0.00)	< 0.1 / 0.2 (±0.2)
15	0.00 (±0.00) / 0.00 (±0.00)	< 0.1 / < 0.1	0.00 (±0.00)	< 0.1	0.2 (±0.0)	0.00 (±0.00) / 0.00 (±0.00)	0.9 (±1.7) / 7.5 (±28.7)
20	0.00 (±0.00) / 0.00 (±0.00)	< 0.1 / < 0.1	0.00 (±0.00)	< 0.1	0.2 (±0.1)	14.46 (±8.61) / 20.83 (±11.51)	38.6 (±43.0) / 198.1 (±63.0)
40	0.00 (±0.00) / 0.00 (±0.00)	< 0.1 / 0.1 (±0.1)	0.00 (±0.00)	0.1 (±0.0)	0.6 (±0.5)	69.66 (±5.34) / 73.87 (±3.36)	—
50	0.00 (±0.00) / 0.00 (±0.00)	0.1 (±0.4) / 0.1 (±0.5)	0.00 (±0.00)	0.1 (±0.0)	0.6 (±0.4)	79.54 (±3.09) / 80.29 (±1.80)	—
100	0.00 (±0.00) / 0.00 (±0.00)	0.1 (±0.1) / 0.2 (±0.1)	0.00 (±0.00)	0.2 (±0.0)	1.8 (±0.7)	93.09 (±0.75) / 93.59 (±0.93)	—
200	0.21 (±0.00) / 0.21 (±0.00)	0.3 (±0.2) / 0.5 (±0.5)	0.00 (±0.00)	0.4 (±0.1)	13.0 (±5.0)	97.79 (±0.32) / 98.13 (±0.12)	—
400	0.00 (±0.00) / 0.00 (±0.00)	0.5 (±0.2) / 0.8 (±0.3)	0.00 (±0.00)	0.8 (±0.1)	167.4 (±64.3)	99.10 (±0.04) / 99.10 (±0.04)	—
800	0.00 (±0.00) / 0.00 (±0.00)	1.7 (±1.2) / 2.6 (±0.8)	2.90 (±0.60)	1.1 (±0.0)	—	99.54 (±0.02) / 99.54 (±0.02)	—
1000	0.00 (±0.00) / 0.00 (±0.00)	1.9 (±0.9) / 4.7 (±6.5)	2.68 (±0.55)	1.2 (±0.0)	—	99.63 (±0.01) / 99.63 (±0.01)	—

Table 3.3: Computational results for MC-2 problem on *MC-2-LOP* dataset.

We can see that the smaller the size of P is, the faster the instances are solved. This pattern is also spotted in results for both *MC-2-LOP* and *MC-2-2K* datasets, where the instances of the same size n are solved by (MIP-MC-2) about 10 times

	(MIP-MC-2)		(BNP-MC-2)			Relative-Order MIP [81]	
n tasks	gap [%] MT/ST	t [s] MT/ST	gap [%] ST	gen [kcols]	t [s] ST	gap [%] MT/ST	t [s] MT/ST
10	0.00 (±0.00) / 0.00 (±0.00)	< 0.1 / < 0.1	0.00 (±0.00)	< 0.1	0.2 (±0.0)	0.00 (±0.00) / 0.00 (±0.00)	0.1 (±0.1) / 0.4 (±0.4)
15	0.00 (±0.00) / 0.00 (±0.00)	< 0.1 / 0.1 (±0.2)	0.00 (±0.00)	< 0.1	0.2 (±0.0)	23.38 (±7.49) / 28.44 (±10.31)	16.5 (±41.4) / 18.6 (±47.6)
20	0.00 (±0.00) / 0.00 (±0.00)	< 0.1 / 0.1 (±0.1)	0.00 (±0.00)	< 0.1	0.2 (±0.0)	29.84 (±17.31) / 26.45 (±20.64)	46.4 (±23.0) / 220.1 (±42.6)
40	0.00 (±0.00) / 0.00 (±0.00)	0.3 (±0.9) / 0.5 (±1.2)	0.00 (±0.00)	0.1 (±0.0)	0.3 (±0.1)	65.08 (±7.83) / 68.92 (±5.14)	—
50	0.00 (±0.00) / 0.00 (±0.00)	0.6 (±1.1) / 1.3 (±2.2)	0.00 (±0.00)	0.1 (±0.2)	0.8 (±1.0)	74.89 (±4.58) / 76.27 (±3.81)	—
100	0.08 (±0.00) / 0.08 (±0.00)	2.0 (±3.9) / 6.4 (±12.5)	0.00 (±0.00)	0.5 (±0.9)	24.4 (±84.4)	90.75 (±1.51) / 91.39 (±1.31)	—
200	0.04 (±0.01) / 0.04 (±0.01)	15.2 (±41.2) / 16.2 (±34.7)	0.26 (±0.36)	0.8 (±0.8)	51.4 (±70.5)	96.60 (±0.77) / 97.24 (±0.21)	—
400	0.04 (±0.05) / 0.02 (±0.01)	13.4 (±20.1) / 11.8 (±14.6)	2.10 (±0.98)	1.0 (±0.3)	139.5 (±95.9)	98.79 (±0.11) / 98.79 (±0.11)	—
800	0.01 (±0.01) / 0.01 (±0.01)	30.7 (±43.0) / 31.3 (±35.4)	2.05 (±0.96)	1.0 (±0.1)	116.5 (±26.1)	99.37 (±0.04) / 99.37 (±0.04)	—
1000	0.05 (±0.07) / 0.03 (±0.03)	35.8 (±41.2) / 49.2 (±59.1)	1.90 (±0.96)	1.1 (±0.0)	—	99.49 (±0.04) / 99.49 (±0.04)	—

Table 3.4: Computational results for MC-2 problem on *MC-2-HIP* dataset.

	(MIP-MC-2)		(BNP-MC-2)			Relative-Order MIP [81]	
n tasks	gap [%] MT/ST	t [s] MT/ST	gap [%] ST	gen [kcols]	t [s] ST	gap [%] MT/ST	t [s] MT/ST
10	0.00 (±0.00) / 0.00 (±0.00)	< 0.1 / < 0.1	0.00 (±0.00)	< 0.1	0.2 (±0.0)	0.00 (±0.00) / 0.00 (±0.00)	0.3 (±0.4) / 1.0 (±1.4)
15	0.00 (±0.00) / 0.00 (±0.00)	< 0.1 / < 0.1	0.00 (±0.00)	< 0.1	0.2 (±0.0)	12.44 (±5.88) / 12.20 (±6.83)	22.7 (±56.1) / 15.5 (±28.3)
20	0.00 (±0.00) / 0.00 (±0.00)	< 0.1 / < 0.1	0.00 (±0.00)	< 0.1	0.2 (±0.1)	23.06 (±8.44) / 22.49 (±11.01)	65.1 (±58.1) / 199.2 (±66.5)
40	0.00 (±0.00) / 0.00 (±0.00)	< 0.1 / < 0.1	0.00 (±0.00)	< 0.1	0.3 (±0.1)	46.40 (±11.87) / 48.95 (±10.70)	—
50	0.00 (±0.00) / 0.00 (±0.00)	< 0.1 / < 0.1	0.00 (±0.00)	0.1 (±0.0)	0.4 (±0.1)	56.00 (±7.38) / 59.24 (±7.27)	—
100	0.00 (±0.00) / 0.00 (±0.00)	< 0.1 / < 0.1	0.00 (±0.00)	0.1 (±0.0)	0.7 (±0.2)	83.23 (±3.15) / 84.23 (±3.02)	—
200	0.00 (±0.00) / 0.00 (±0.00)	0.1 (±0.0) / 0.1 (±0.1)	0.00 (±0.00)	0.2 (±0.0)	2.2 (±0.6)	93.77 (±0.54) / 95.93 (±0.56)	—
400	0.00 (±0.00) / 0.00 (±0.00)	0.2 (±0.1) / 0.3 (±0.2)	0.00 (±0.00)	0.4 (±0.0)	12.4 (±2.1)	98.01 (±0.13) / 98.01 (±0.13)	—
800	0.00 (±0.00) / 0.00 (±0.00)	0.7 (±0.3) / 0.9 (±1.0)	0.00 (±0.00)	0.7 (±0.0)	107.2 (±14.8)	99.02 (±0.05) / 99.02 (±0.05)	—
1000	0.00 (±0.00) / 0.00 (±0.00)	0.9 (±0.8) / 1.3 (±1.0)	0.00 (±0.00)	0.9 (±0.0)	204.2 (±31.4)	99.20 (±0.02) / 99.20 (±0.02)	—

Table 3.5: Computational results for MC-2 problem on *MC-2-2K* dataset.

faster in comparison to the results for the *MC-2-HIP* dataset. Interestingly, e.g., for $n = 400$ tasks, even though the total number of instances unsolved to the optimality is larger in the ST mode than in the MT mode, the average gap for the former is smaller.

The relative-order MIP proposed in [81] particularly struggles with the *MC-2-2K* dataset, solving all instances only with $n = 10$ tasks, despite having a relatively small P. In comparison to [81], relative-order MIP can solve the instances with up to $n \approx 15$ tasks, whereas our proposed methods scale up to $n = 1000$ tasks. Moreover, relative-order MIP [81] does not gain any significant advantage for instances where $|P| < n_L$. Furthermore, lower bounds in the root node obtained by the relative-order method of [81] are weaker than those in cases of (MIP-MC-2) and (BNP-MC-2).

The median of the total number of columns generated by (BNP-MC-2) needed to prove the optimality of an integer solution for an instance is depicted in Figure 3.7. The figure shows that the number of generated columns needed to prove optimality is roughly linear in the number of tasks. The smallest number of columns generated is required in *MC-2-2K* dataset. The second smallest number of generated columns is observed in *MC-2-LOP* dataset, producing, on average, approximately twice as many columns. The most difficult dataset to solve is found to be *MC-2-HIP* in terms of both the number of columns generated and computational time. One can notice a spike in Figure 3.7 for the batch $n = 100$, where one instance took more than 7 kilocolumns to solve. We see that the cardinality of P influences the mean and variance of the number of columns generated.

For a better assessment, where the hotspots of our implementation of (BNP-MC-2) are, we measure the time spent in solving the master problem and pricing problem separately. We find out that over 95% of the total computational time is spent on the pricing problem. Hence, the algorithm can be accelerated if an efficient vectorized implementation of the algorithm for the pricing problem is used.

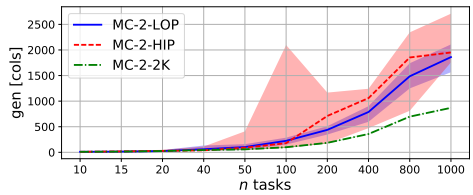


Figure 3.7: Median of the total number of columns generated for instances of MC-2.

3.5.3 Computational time for the MC-3 problem

We work with two datasets for MC-3 problem, denoted as *MC-3-HIP* and *MC-3-2K*. Each of them contains batches of instances with a different number of tasks n. For each size n, the batch has 40 instances. In both datasets, the ratio between the number of tasks with different criticalities is $n_D/n_H = n_H/n_L \approx 0.75$ for each instance. The datasets differ in the distribution of processing times. For *MC-3-HIP* dataset, the processing times are given such that for every two consecutive criticality levels, the ratios of the means of their prolongation is approximately 1.5 (i.e., the region of hardness displayed in Figure 3.6b). In dataset *MC-3-2K*, the processing times and their prolongations on each level are given as $2^k, k \in [0, 7]$ to mimic the problems of practical interests inspired by packet scheduling, similarly to *MC-2-2K*.

	(BNP-MC-3)			Relative-Order MIP [81]	
n tasks	gap [%] MT	gen [kcols]	t [s] MT	gap [%] MT	t [s] MT
10	0.00 (±0.00)	< 0.1	0.2 (±0.0)	0.00 (±0.00)	< 0.1
15	0.00 (±0.00)	< 0.1	0.3 (±0.1)	0.00 (±0.00)	0.6 (±0.2)
20	0.00 (±0.00)	< 0.1	0.3 (±0.2)	0.00 (±0.00)	8.1 (±22.7)
40	0.00 (±0.00)	0.1 (±0.0)	1.8 (±1.4)	44.40 (±8.98)	—
50	0.06 (±0.00)	0.2 (±0.7)	3.4 (±2.7)	61.11 (±9.22)	—
100	0.02 (±0.00)	0.2 (±0.3)	18.9 (±15.3)	88.39 (±1.20)	—
200	0.17 (±0.22)	0.3 (±0.3)	110.4 (±77.6)	96.20 (±0.66)	—
400	0.63 (±0.53)	0.8 (±0.7)	273.0 (±185.2)	98.72 (±0.07)	—
800	0.81 (±0.55)	1.0 (±0.1)	—	99.34 (±0.03)	—
1000	0.82 (±0.52)	1.1 (±0.1)	—	99.47 (±0.02)	—

Table 3.6: Computational results for MC-3 problem on *MC-3-HIP* dataset.

	(BNP-MC-3)			Relative-Order MIP [81]	
n tasks	gap [%] MT	gen [kcols]	t [s] MT	gap [%] MT	t [s] MT
10	0.00 (±0.00)	< 0.1	0.2 (±0.1)	0.00 (±0.00)	0.4 (±1.3)
15	0.00 (±0.00)	< 0.1	0.6 (±1.0)	8.44 (±0.00)	9.7 (±21.8)
20	0.50 (±0.00)	0.1 (±0.8)	0.5 (±1.0)	19.89 (±9.26)	50.1 (±70.1)
40	5.20 (±5.12)	0.2 (±0.6)	3.3 (±8.9)	38.99 (±11.16)	—
50	9.36 (±0.00)	0.4 (±2.2)	4.6 (±16.1)	50.87 (±9.95)	—
100	0.00 (±0.00)	< 0.1	0.3 (±0.1)	81.79 (±2.92)	—
200	0.00 (±0.00)	0.1 (±0.0)	0.9 (±0.5)	92.68 (±0.71)	—
400	0.00 (±0.00)	0.1 (±0.0)	4.3 (±1.7)	98.11 (±0.12)	—
800	3.33 (±0.75)	0.3 (±0.4)	29.1 (±21.6)	99.03 (±0.05)	—
1000	3.60 (±0.52)	0.6 (±0.5)	81.4 (±12.5)	99.23 (±0.04)	—

Table 3.7: Computational results for MC-3 problem on *MC-3-2K* dataset.

In dataset *MC-3-HIP*, the algorithm (BNP-MC-3) can optimally solve nearly all instances up to the size $n = 100$ within the time limit. It runs out of time only in a single case for sizes $n = 50$ and $n = 100$. However, for the instances where the optimality is not proven, the optimality gap, on average, is only 0.42%.

Furthermore, the number of generated columns is about the same as that observed in the dataset *MC-2-HIP*, showing that the scalability is also preserved for problems with more criticality levels. Tables 3.6 and 3.7 show that scaling capabilities of [81] approach are the same regardless of the number of criticality levels, thus being able to solve instances about $n \approx 15 - 20$ tasks. Similar to that in Section 3.5.2, (BNP-MC-3) can solve instances with almost twice the number of tasks than those in [81].

3.5.4 Discussion

For problem MC-2, both methods (MIP-MC-2) and (BNP-MC-2) proposed in this paper outperform Relative-Order MIP [81]. Under the used testing settings, the (MIP-MC-2) average computation time is faster than (BNP-MC-2) computation time, except for a few cases in batch $n = 100$ of the *MC-2-HIP* dataset and $n = 200$ in the *MC-2-LOP* dataset, where (BNP-MC-2) has closed all instances. It would be possible to further improve the performance of (BNP-MC-2), e.g., by solving pricing problems in parallel since they are independent. However, this is beyond the scope of this paper.

For some use-cases, using model (MIP-MC-2) might pose two disadvantages. One of them is that its performance depends on a commercial solver, which is not an affordable option for some applications. On the other hand, (BNP-MC-2) needs only an LP solver for solving the master problem, which is the task where non-commercial solvers perform better than those in MIP. Moreover, only a small part of the computational time is spent on the master problem. Most of the time is spent on the pricing problem, which is solved by a dynamic programming algorithm without any third-party software package.

Moreover, (BNP-MC-2) gains advantages in some special cases of the problem due to its pseudopolynomially solvable pricing problem. For example, if the values of processing times are restricted, then the pricing problem becomes solvable in polynomial time. Another disadvantage of (MIP-MC-2) is its memory complexity. It uses $\mathcal{O}(n_L n_H)$ variables, and therefore, $\Omega(n_L^2 n_H)$ memory space (i.e., the size of the constraint matrix), whereas (BNP-MC-2) is observed in Figure 3.7 to use $\mathcal{O}(n_L)$ variables, which can be further reduced (e.g., by removing old columns from the simplex tableau). Finally, (BNP-MC-2) is further generalized for more criticality levels.

For MC-3 problem, (BNP-MC-3) outperforms Relative-Order MIP [81]. The dataset *MC-3-2K* turns out to be easier than *MC-3-HIP*, in terms of the number of columns generated, computational time, and the number of instances solved. However, (BNP-MC-3) struggles to prove the optimality for a single instance in a batch with $n = 50$, causing a noticeable spike in the number of columns generated. On the other hand, it solves all instances in batches $n \in \{100, 200, 400\}$. Note that (BNP-MC-3) achieves smaller computational times than (BNP-MC-2) for the solved instances in batch $n = 1000$ on average. This is because while pricing problems in the case of (BNP-MC-3) are being solved as an MIP, (BNP-MC-2) implementation uses a dynamical programming algorithm that has pseudopolynomial time complexity in the total sum of all processing times of tasks. Hence, even though the pricing problem in (BNP-MC-2) is in some sense easier to solve (solvable in pseudopolynomial time)

than the pricing problem in (BNP-MC-3) (strongly $\mathcal{NP}$-hard), the average case computational time for the former tends to be lower with the tested lengths of processing of tasks.

It would be possible to improve (BNP-MC-2) and (BNP-MC-3) algorithms with other techniques, such as generating more columns at once, Lagrangian relaxation, primal heuristics, stabilization, and suppression of the tailing-off effect [108]. Hence, they still provide room for a performance improvement, but these are beyond the scope of this paper.

3.6 Conclusion

In this paper, we have studied the problem of scheduling F-shaped tasks to minimize the makespan of the schedule. This problem has applications such as those in real-life mixed-criticality systems, where high-criticality activities coexist with less-criticality ones on a shared resource. The processing time for such activities is uncertain. To overcome the uncertainty, an F-shape modeling the activity contains a set of alternative processing times. The schedules contain exponentially many alternative schedules, where the performed alternative is selected based on the observed execution scenario. The schedule remains static and its behavior is predictable. However, the synthesis of such flexible schedules is computationally expensive; hence, we proposed efficient exact algorithms to solve the problem.

We showed that optimal schedules are equivalent to trees consisting of optimal subtrees, and established the relation between problems with ℓ and $\ell + 1$ criticality levels. We suggested an approximation algorithm, a block MIP model, and a branch-and-price decomposition algorithm with a pseudopolynomially solvable pricing problem for a problem with two criticality levels, for which we proposed a dynamic programming algorithm. Furthermore, we generalized the proposed decomposition to obtain the exact algorithm for a problem with three criticality levels. The experimental results showed an excellent scaling ability of our approach on hard problem instances. We found that it takes only a few hundreds of generated columns, on average, in our decomposition algorithms to solve instances with up to 1000 tasks to the optimality.

A possible extension of the proposed model and algorithms might consider including a penalty for each covered task in the schedule or optimization of a bi-criteria objective function that would find a trade-off between the schedule length and the number of covered tasks in such a schedule. Both extensions can be developed as a generalization of the proposed methods, as they decide which tasks shall be covered by others during the solution.

Chapter

4 Computing the execution probability of jobs with replication in mixed-criticality schedules

Antonín Novák and Zdenek Hanzalek. "Computing the execution probability of jobs with replication in mixed-criticality schedules". In: *Annals of Operations Research* (2022). DOI: 10.1007/s10479-021-04445-x

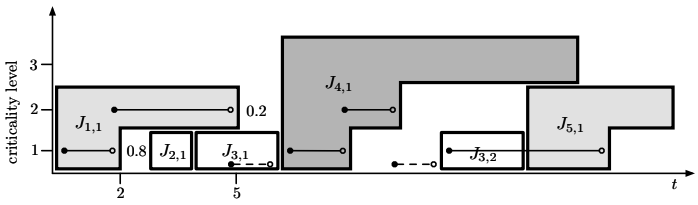
4.1 Introduction

This paper addresses the problem of computing the execution probability of jobs in mixed-criticality schedules. Mixed-criticality systems [175, 40, 41, 7] share resources among jobs with a given degree of importance, i.e., a criticality represented by a positive integer number. Although this paradigm reduces the costs of the system, it introduces new challenges, such as undesired interactions (causing delays or deadline misses) between jobs with uncertain processing times.

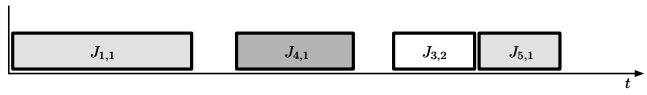
The traditional systems typically consider the allocation of critical jobs to a dependable resource (e.g., a dedicated communication line for critical messages) to achieve isolation of critical jobs, and thus, higher reliability of the system; however, the efficiency of such systems becomes an issue. Hence, the key challenge of mixed-criticality systems is to isolate jobs such that low-critical jobs (e.g., $J_{2,1}$, $J_{3,1}$ and $J_{3,2}$ in Figure 4.1a) do not influence any high-critical job (e.g., $J_{4,1}$ in Figure 4.1a) without the use of additional resources. Typically, the isolation of critical jobs with uncertain duration is achieved with the two aspects: (i) static scheduling of jobs [170, 20, 7] and (ii) online rejection of low-critical jobs to compensate for processing delays of more critical ones [40, 151, 13]. Although the static scheduling increases the predictability of the system (i.e., its behavior is given by a static schedule which can be analyzed offline), flawless execution of critical jobs may require occasionally to reject less critical jobs during online execution (e.g., $J_{2,1}$ and $J_{3,1}$ in Figure 4.1b). Thus, careful scheduling of jobs should be used [20, 151] to mitigate the degradation of the execution probability of non-critical jobs without affecting the requirements of critical jobs. To optimize the execution probability of jobs in a schedule, a scheduling algorithm needs to assess the quality (i.e., the objective function) of the current schedule in order to drive the search towards a good solution [80]. Hence, the computation of the objective function of a schedule is the central component of any algorithm that produces high-quality schedules.

In this paper, we introduce the concept of *job replication* to mixed-criticality schedules. The job replication is a mechanism that utilizes unused time slots in

a static schedule for additional jobs' execution attempts if the previous attempts have failed. This elegant mechanism increases the execution probability of jobs and does not require additional system resources. In fact, job replication acts as a natural generalization of the mixed-criticality model used, e.g., by Vestal [175] or Seddik *et al.* [151], but it introduces additional complexity to the computation of the objective function of a schedule. Currently known methods (such as [151]) that are used for the computation of the execution probability in mixed-criticality schedules do no longer work when the replication is introduced. Therefore, in this paper, we study the complexity of computing the execution probability of jobs in mixed-criticality schedules with replication (i.e., an objective function of a schedule related to reliability of the system), and we propose an algorithm to compute it that utilizes the theoretical framework of Bayesian networks.



(a) Mixed-criticality schedule with replication with five jobs where J_3 has two replicas.



(b) An execution scenario.

Figure 4.1: Mixed-criticality schedule with three criticality levels with one of the possible execution scenarios.

4.1.1 Contribution and outline

In this paper, we study the job replication, which is a mechanism for increasing the execution probability of jobs in mixed-criticality schedules. Specifically, the main contributions are:

- We introduce the concept of replication to mixed-criticality schedules as a mechanism for increasing the execution probability of jobs.
- We show that the general problem of computing execution probability for a job in a mixed-criticality schedule with replication is $\#\mathcal{P}$-hard, in contrast to the known polynomial-time algorithm for the case without replication. The proof shows that the problem remains hard even if one of the numbers of criticality levels or the maximum number of replicas is equal to a certain constant while the other is bounded by a polynomial in the number of jobs.
- We solve the problem by a reduction to the probabilistic inference in a suitably defined Bayesian network.

– Finally, we show that the cases of reasonable interests can be solved in polynomial time in the number of jobs, which enables practical usage of the job replication.

The rest of the paper is structured as follows. In the subsequent section, we first describe the functionality and an application example of mixed-criticality systems with replicated jobs. Then, we demonstrate the effect of job replication on the execution probability, and we rigorously define the problem statement and survey the related work in this area.

In Section 4.2, we study the complexity of the general problem. Section 4.3 deals with the actual algorithm for the computation of execution probability. In Section 4.3.1, we show the reduction to the probabilistic inference in Bayesian networks, and in Section 4.3.2, it is shown that a practical case of the problem admits an efficient computation algorithm. Finally, conclusions are drawn in Section 4.4.

4.1.2 Mixed-criticality systems with job replication

In this section, we describe an application example to illustrate the main concepts of mixed-criticality systems with job replication. Then, we show how the execution probability can be computed in the case without replication and how the job replication increases it.

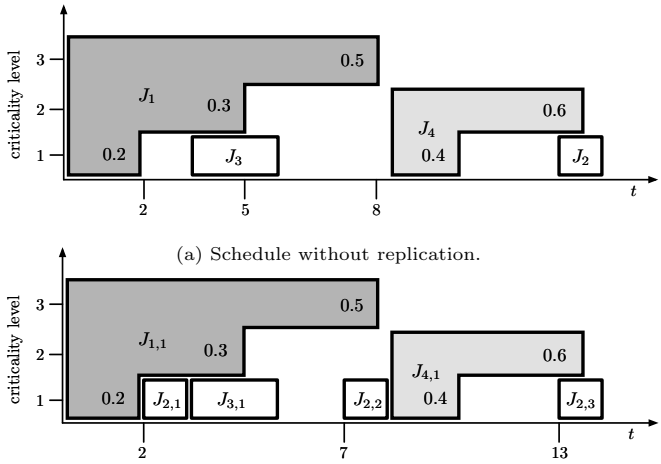
Application example Consider a message scheduling problem on a shared communication bus in modern cars. Safety-related standards such as ASIL (Automotive Safety Integrity Levels) [21] introduce the existence of messages with several levels of criticality:

– messages of high criticality (criticality 3) are used for safety-related functionalities (their failure may result in death or severe injury to people), such as steering;

– messages of medium criticality (criticality 2) are used for mission-related functionalities (their failure may prevent activity from being successfully completed), such as parking assist;

– messages of low criticality (criticality 1) are typically used for infotainment functionalities, such as automotive navigation system.

The messages are transmitted via the bus at the moments defined by the static time-triggered schedule [93] which improves determinism and predictability. The goal is to compute objective function reflecting statistical properties of a given static schedule that accounts for disruption of the communication according to the message criticality. In real-life environments, the execution of jobs is affected by various sources of uncertainty, causing, e.g., transmission delays. In the above example, the criticality expresses the commitment to the transmission when the original transmission is prolonged. Therefore, several transmission attempts are awarded to messages with a high criticality, whereas for low-criticality messages, it might be just a single one.



(a) Schedule without replication.

(b) Schedule with three replicas of job J_2.

Figure 4.2: Effect of job replication to the execution probability.

Vestal's [175] widely adopted model of mixed-criticality considers jobs with the criticality given by an integer number and set of different processing times associated with criticality levels. Let us demonstrate the main concepts with an example before providing a formal definition in Section 4.1.3. Figure 4.1a shows an example of a mixed-criticality static schedule with six job replicas. Each job has a given integer criticality as it is seen on the vertical axis. For example, job $J_{4,1}$ has criticality of three, job $J_{1,1}$ has criticality of two while $J_{2,1}$ has criticality of one. Notice that $J_{3,1}$ and $J_{3,2}$ are two replicas of the same job, hence, they have the same parameters but different start times. Job $J_{1,1}$ has the considered processing time 2 time units with probability 0.8, and 5 time units with probability 0.2. The considered values of processing times are derived from the (empirical) cumulative distribution function with respect to the selected probability thresholds [175, 7].

Mixed-criticality schedules contain several alternative execution scenarios, with the one being selected based upon the realized processing times of jobs that occur during the runtime execution. To compensate for unexpected prolongations of critical jobs observed at the runtime, some of the less critical ones might not be executed under the specific realization of processing times. This can be seen in Figure 4.1b, where jobs $J_{2,1}$ and $J_{3,1}$ are rejected if realized processing time of $J_{1,1}$ is equal to 5, happening with probability 0.2. However, the second replica $J_{3,2}$ was executed later on. Finally, we note that, e.g., $J_{1,1}$ is never rejected since it does not share its execution time with any other job with higher criticality.

The formal definition of the policy that guides the execution of the schedule will be given in Section 4.1.3. Next, let us illustrate how the execution probability can be computed and optimized in the case without replication and we show how does the job replication improve it.

Execution probability and job replication The first method for the optimization of the execution probability in mixed-criticality schedules was proposed by [151]. Given a schedule with jobs subject to deadlines, the start times of mixed-

criticality jobs are shifted in the schedule, such that low-critical jobs do not share execution time with high-critical ones as long as the deadlines are not violated. We illustrate the shifting of start time with the following example. Consider the schedule in Figure 4.2a. There, job J_3 has execution probability equal to 0.2 since with probability $0.3 + 0.5 = 0.8$ job J_1 will take longer than 2 time units to process which would reject J_3. If we would shift the start time of J_3 to time 5, then its execution probability would be increased to $0.2 + 0.3 = 0.5$. For more details, we refer the reader to [151].

The disadvantage of this approach, i.e., the shifting the start times, is that the resulting schedule might be too sparse, which results in low utilization of the resource (e.g., CPU in an embedded system). This is an undesired property when the resource is statically scheduled and no jobs are arriving dynamically (since it leads to more empty CPU cycles). Compare the utilization of schedules in Figure 4.2a (i.e., a lower) and Figure 4.2b (i.e., a higher).

The main idea in this paper is to utilize unused time slots in the schedule by replicating some of the scheduled jobs which increases their execution probability. First, let us introduce the formal definition of mixed-criticality jobs used thorough the paper. We use a standard mixed-criticality model [175, 151]:

Definition (Mixed-criticality job)**.** *Let* $J_i = (\boldsymbol{\pi}_i, \boldsymbol{\mu}_i, \mathcal{X}_i)$ *be a mixed-criticality job, where* $\mathcal{X}_i \in \mathbb{N}$ *is a positive integer criticality,* $\boldsymbol{\pi}_i = \left(\pi_i^{(1)}, \ldots, \pi_i^{(\mathcal{X}_i)}\right) \in \mathbb{N}^{\mathcal{X}_i}$ *is a vector of processing times of job* J_i *such that* $\pi_i^{(1)} < \ldots < \pi_i^{(\mathcal{X}_i)}$ *and* $\boldsymbol{\mu}_i = \left(\mu_i^{(1)}, \ldots, \mu_i^{(\mathcal{X}_i)}\right) \in [0,1]^{\mathcal{X}_i}$ *is a conditional probability distribution over* $\boldsymbol{\pi}_i$ *given that* J_i *is executed.*

See the example in Figure 4.2a. There, we can see, e.g., job J_1 has $\boldsymbol{\pi}_1 = (2, 5, 8)$, $\boldsymbol{\mu}_1 = (0.2, 0.3, 0.5)$ and $\mathcal{X}_1 = 3$. Let us denote q-th replica of job J_i as $J_{i,q}$, i.e., all replicas of the same job J_i have the same parameters $\mathcal{X}_i$, $\boldsymbol{\pi}_i$, and $\boldsymbol{\mu}_i$, but different start times.

Next, we demonstrate how the replication leads to an increased execution probability over the schedules without replication. For example, consider Figure 4.2b, where job replica $J_{2,2}$ can be executed at time 7 if $J_{2,1}$ is not executed at time 2. When the execution at time 7 fails as well, $J_{2,3}$ can be still executed at time 13. More generally, the replication of jobs allows us to achieve higher execution probabilities, since the replication may be seen as a form of relaxation, where we relax on the scheduling constraint "*each job is scheduled exactly once*" to "*each job is scheduled at least once*".

In this paper, we deal with the problem of computing execution probabilities of jobs with replication for the given fixed schedule. It is assumed that the schedule is a solution to the scheduling problem denoted in three-field scheduling notation [75] as $1|mc = \mathcal{L}, rep = \mathcal{R}| \sum P_i$. The first field stands for a single resource, $mc = \mathcal{L}$ stands for mixed-criticality jobs with an unspecified number of criticality levels, [81] and $rep = \mathcal{R}$ for the replication considering an unspecified number of replicas per job. The last field $\sum P_i$ indicates that the objective function is the sum of execution probabilities [151]. Note that without loss of generality, we can restrict ourselves to schedules with a single resource only. Indeed, as the schedule is fixed, then each resource can be treated separately.

In the following section, we define the problem statement of computing objective function $\sum P_i$ for the given schedule of problem $1|mc = \mathcal{L}, rep = \mathcal{R}|\sum P_i$, and potentially other constraints. Therefore, we assume that the start times of the jobs are provided, and the goal is to compute the execution probabilities of jobs.

4.1.3 Problem statement

First, we define an instance of the problem given by a set of mixed-criticality jobs and their schedule.

Definition (Mixed-criticality instance)**.** *Let $I_{\mathcal{MC}} = \{J_1, \ldots, J_n\}$ be a set of independent mixed-critical jobs. Let us denote the maximal criticality as $\mathcal{L} = \max_i \mathcal{X}_i$. Let $\mathcal{R} > 1$ be the maximal number of replicas of any job in a feasible schedule for $I_{\mathcal{MC}}$.*

Further in the text, we will refer to the original job J_i in the instance $I_{\mathcal{MC}}$ as a "*job*", whereas their individual occurrences $J_{i,q}$ in the schedule as "*(job) replicas*". Therefore, each job has at least one replica in the schedule, and each replica corresponds to exactly one job. We assume that we are provided with a feasible schedule containing up to $\mathcal{R}$ replicas per job with criticality up to $\mathcal{L}$:

Definition (Schedule with replication)**.** *Let us denote the schedule for a set of jobs $I_{\mathcal{MC}}$ as $\boldsymbol{s} = (s_{1,1}, \ldots, s_{1,n_1}, s_{2,1}, \ldots, s_{2,n_2}, \ldots, s_{n,1}, \ldots, s_{n,n_n})$, where $n_k \leq \mathcal{R}$ is the number of replicas of the job J_k in schedule $\boldsymbol{s}$. Let $s_{i,q} \in \mathbb{N}_0$ be the start time of replica $J_{i,q}$. Furthermore, we say that schedule $\boldsymbol{s}$ is feasible if and only if $\forall J_i, J_j \in I_{\mathcal{MC}}, \forall q \leq n_i, r \leq n_j$:*

$$\left(s_{i,q} + \pi_i^{(\min\{\mathcal{X}_i, \mathcal{X}_j\})} \leq s_{j,r}\right) \vee \left(s_{j,r} + \pi_j^{(\min\{\mathcal{X}_i, \mathcal{X}_j\})} \leq s_{i,q}\right),$$

i.e., replicas do not overlap on any criticality level.

Furthermore, we will assume without loss of generality that the start times of the first replicas are ordered $s_{1,1} \leq s_{2,1} \leq \ldots \leq s_{n,1}$ and replicas of each job $J_i \in I_{\mathcal{MC}}$ are ordered as $s_{i,1} \leq s_{i,2} \leq \ldots \leq s_{i,n_i}$.

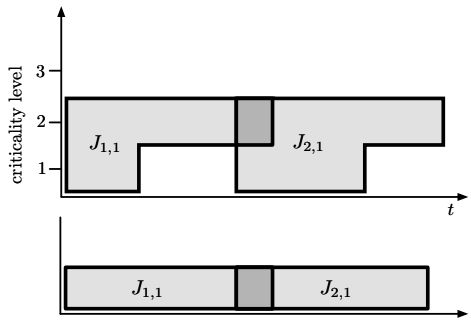


Figure 4.3: Example of an infeasible schedule. All realizations are not feasible.

The above definition of a feasible schedule comes from the strict requirement that the processing of a job cannot be affected by delays of jobs with less or equal

criticality. The motivation for it comes from the modular system design principles, where it is desired to achieve so-called *error containment* [124] which is the property of the system that guarantees that the malfunction of a single component of the system does not arbitrarily propagate through the system. This in turn facilitates *modular certification* [124], which is yet another desirable design principle. To demonstrate the meaning of the definition of the feasibility, consider the example in Figure 4.3, where we see a schedule with two jobs with two criticality levels and one of the possible execution scenarios of the schedule. If job replica $J_{1,1}$ is executed at the second level, it may correspond to an abnormal state and can be seen as an error. If any two jobs would overlap at the second criticality level (as it is shown in the figure), then this error would disrupt job replica $J_{2,1}$ with equal criticality, which is not desirable, especially if this job implements functionality that has been certified in the isolation. Thus, its certification would not be longer valid when deployed together with the functionality implemented by $J_{1,1}$. Therefore, for the above reasons, it is useful to require that in *all* execution scenarios, a job cannot be affected by delays of less or equally critical jobs.

On the other hand, the definition allows that more critical jobs can affect the execution of less critical jobs. Indeed, each schedule contains so-called *coverage sets* which specify which replicas might be rejected during the execution of the schedule.

Definition (Coverage set)**.** *We say that replica $J_{i,q}$ is covered by replica $J_{j,r}$ at level ℓ, $\mathcal{X}_j \geq \ell > 1$ in schedule $\boldsymbol{s}$, denoted by $J_{i,q} \in \mathrm{cov}_\ell(J_{j,r})$, if and only if*

$$s_{j,r} + \pi_{j,r}^{(\ell-1)} \leq s_{i,q} < s_{j,r} + \pi_{j,r}^{(\ell)}.$$

For example, in Figure 4.4a we have $J_{2,2} \in \mathrm{cov}_3(J_{1,1})$. For simplicity, we use the notation $J_{i,q} \in \mathrm{cov}(J_{j,r})$ to denote that there exists a level $\ell \in \{2, \ldots, \mathcal{X}_k\}$ such that $J_{i,q} \in \mathrm{cov}_\ell(J_{j,r})$. Note that the notation omits the dependence on schedule $\boldsymbol{s}$ as well, as it is clear from the context.

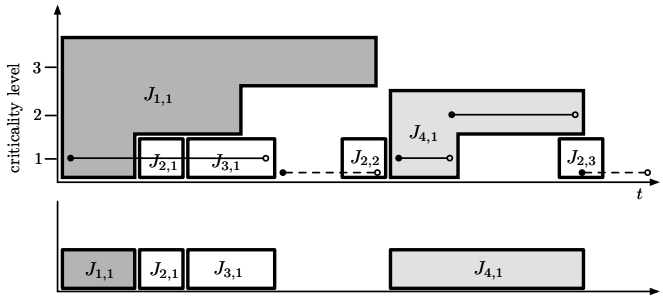
Since the processing times of jobs are uncertain, each job replica can have a different realized processing time, which is revealed during the execution of the schedule.

Definition (Execution level of a replica)**.** *We say that the execution level of replica $J_{i,q}$ is ℓ, denoted as $J_{i,q} \sim \ell$, if and only if its realized processing time $\overline{\pi}_{i,q}$ is equal to the processing time of the job J_i at ℓ-th level, i.e., $\overline{\pi}_{i,q} = \pi_i^{(\ell)}$ if and only if $J_{i,q} \sim \ell$.*

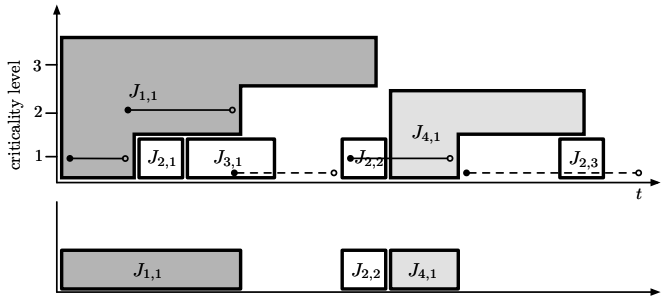
See the example in Figure 4.4. In the scenario depicted in Figure 4.4a, the realized processing time of replica $J_{1,1}$ was $\overline{\pi}_{1,1} = \pi_1^{(1)}$, hence $J_{1,1} \sim 1$. On the other hand, in the scenario in Figure 4.4b, we have that $J_{1,1} \sim 2$, thus $\overline{\pi}_{1,1} = \pi_1^{(2)}$.

Next, we describe how the mixed-criticality schedules with replicated jobs are executed. To execute replicas in a schedule with respect to their criticalities, the runtime execution of the schedule $\boldsymbol{s}$ is guided by the policy given in Algorithm 4.1. The design of the execution policy is driven by the implementation aspects of real-life embedded systems. It uses the following rules:

– the replica is started at its start time if no other replica is executed at the moment (line 3: $h(t) = 0$),



(a) The first replica of J_2 is executed, hence, the others are rejected.



(b) The second replica of J_2 is executed, hence, the last is rejected.

Figure 4.4: Schedule of mixed-criticality jobs with replication under different scenarios.

– when a replica is started, then it is completed on one of its criticality levels (lines 6–9, $\overline{e} \leftarrow \overline{e} \cup \{J_{i,q} \sim \ell\}$),

– when a replica is completed, then all of the following replicas of the job are rejected (line 3: $\nexists q', \ell : (J_{i,q'} \sim \ell) \in \overline{e}$).

The role of function $h(t)$ is to keep track of the current execution level of the schedule. See an example of the $h(t)$ function in Figure 4.4 where it is depicted on the y-axis with solid and dashed lines. The value $h(t) = 0$ denotes that the resource is available, and a replica can be executed at its start time. During the replica's execution, its realized processing time is observed, and the execution level of the schedule $h(t)$ is appropriately updated while the execution $J_{i,q} \sim \ell$ of the job replica is stored to the list of observed realizations $\overline{e}$. When the replica is completed, the execution matches up with the base value (i.e., $h(t) = 0$), signaling that the next job replica in the order can be executed.

Note that the execution policy in Algorithm 4.1 distinguishes only two states of $h(t)$ function, i.e., $h(t) = 0$ and $h(t) > 0$. Indeed, for the decision, which jobs to execute given the observed realizations of processing times, it is sufficient to observe whether the resource is available ($h(t) = 0$) at the time t if t equals to the start time of a job replica. Within the scope of this paper, we use $h(t)$ execution function only inside the execution policy in Algorithm 4.1 and to define an *execution scenario*. Essentially, any admissible execution of the schedule, i.e., the evolution of $h(t)$ function, forms an execution scenario. We define an execution scenario as a sequence of outcomes (i.e., the list of observations $\overline{e}$) that is admissible under the execution policy in Algorithm 4.1.

Algorithm 4.1: Execution policy of mixed-criticality schedules.

input : a schedule $\boldsymbol{s}$ and online observations of realized processing times
output : the sequence of decisions which replicas to execute and to reject

$t \leftarrow 0$, $h(0) \leftarrow 0$, $\overline{e} \leftarrow \emptyset$
while $t < \max \boldsymbol{s}$ **do**
  if $h(t) = 0$ **and** $\exists s_{i,q} \in \boldsymbol{s} : s_{i,q} = t$ **and** $\nexists q', \ell : (J_{i,q'} \sim \ell) \in \overline{e}$ **then**
    $\overline{\pi}_{i,q} \leftarrow$ execute $J_{i,q}$ at time $s_{i,q}$ `// observe realized processing time`
    $\overline{e} \leftarrow \overline{e} \cup \{J_{i,q} \sim \ell\}$ `// exec. level of q-th replica of job` J_i `is` ℓ
    for $k \leftarrow 1$ **to** ℓ **do** `// update exec. level of schedule` $h(t)$
      $h(t + t') \leftarrow k \quad \forall t' \in \left[0, \pi_i^{(k)}\right]$
      $t \leftarrow t + \pi_i^{(k)}$
  else
    $t \leftarrow t + 1$
  $h(t) \leftarrow 0$ `// the execution matches-up with the base value`

We note that additional functionalities of the system such as recovery policies (e.g., *graceful degradation*) may require fine-grained access to the specific execution level of the schedule $h(t)$ rather than whether the resource is busy at time t or not. These are beyond the scope of this paper but motivates us to introduce $h(t)$ function in its more general form.

Definition (Execution scenario)**.** *The execution scenario $\overline{e}$ is a sequence of replicas' execution levels $J_{i,q} \sim \ell$, i.e., specific realization of processing times, which can be realized under the execution policy given by Algorithm 4.1. We denote the set of all possible execution scenarios in schedule $\boldsymbol{s}$ by $E_{\boldsymbol{s}}$.*

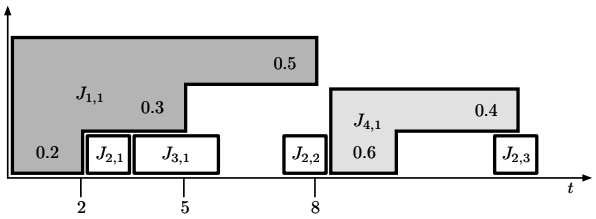
For example, the scenario corresponding to Figure 4.4b is $\bar{e} = (J_{1,1} \sim 2, J_{2,2} \sim 1, J_{4,1} \sim 1)$. Furthermore, notice that the execution policy ensures that at most one replica of each job is executed under any scenario. Consider an example in Figure 4.4, where the schedule is executed under two different scenarios. There, the solid line displays the current execution level of the schedule $h(t)$ while being dashed when it is 0. In the scenario in Figure 4.4b, $J_{1,1}$ is executed at its second level, which leads to rejection of replicas $J_{2,1}$ and $J_{3,1}$. Then, the second replica $J_{2,2}$ was executed; thus, the last replica $J_{2,3}$ is rejected. A different scenario is depicted in Figure 4.4a, where the first replica $J_{2,1}$ is executed; thus, the second (i.e., $J_{2,2}$) and the third (i.e., $J_{2,3}$) replicas are rejected.

Finally, we define the execution probability of a job as follows:

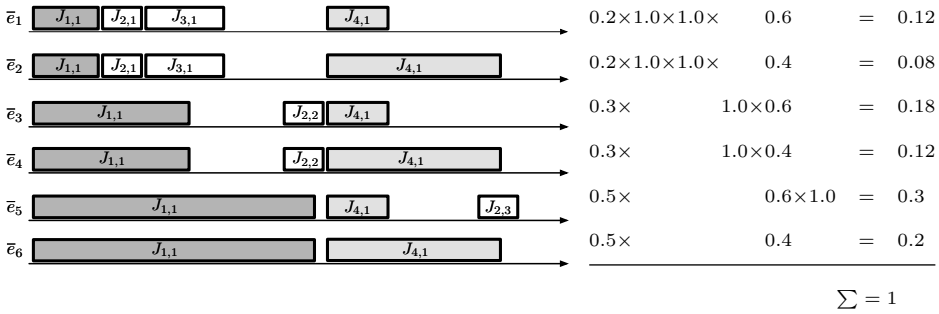
Definition (Execution probability of a job)**.** *The execution probability P_i of job J_i in schedule $\boldsymbol{s}$ is given as*

$$P_i = \sum_{\forall \overline{e} \in \{e' \in E_{\boldsymbol{s}} : \exists q, \ell : (J_{i,q} \sim \ell) \in e'\}} \quad \prod_{\forall (J_{k,r} \sim \ell) \in \overline{e}} \mu_k^{(\ell)}. \tag{4.1.1}$$

In other terms, the execution probability of a job is the sum of probabilities of scenarios where a replica of that job was executed. See the example in Figure 4.5.



(a) Schedule of mixed-criticality jobs with replication.



(b) All possible execution scenarios and their probabilities.

Figure 4.5: Example schedule and all of its possible execution scenarios.

There, all six possible scenarios and their probabilities, are depicted. Note that the lowest critical job replicas (i.e., $J_{2,1}$, $J_{2,2}$, $J_{2,3}$, and $J_{3,1}$ in Figure 4.5a) have the probability 1.0 of being completed at the first level given that they are started (see Definition). For example, the execution probability of replica $J_{3,1}$ in this schedule is $P_3 = 0.12 + 0.08 = 0.2$. This can be seen as well as from the fact that $J_{3,1}$ has probability $0.3 + 0.5 = 0.8$ of being rejected by $J_{1,1}$. Similarly, the execution probability of J_2 is $P_2 = (0.12 + 0.08) + (0.18 + 0.12) + (0.3) = 0.8$.

The aim of this paper is to compute the sum of execution probabilities for all jobs in the schedule.

Definition (Problem statement)**.** *The problem is to compute* $\sum P_i$ *of the given schedule* $\boldsymbol{s}$ *with replication, where* P_i *is the execution probability of job* $J_i \in I_{\mathcal{MC}}$*.*

We assume that the provided schedule is not trivial in the sense that at least one replica is covered; otherwise, every job would be executed with a probability of one. Indeed, it is realistic to assume that the jobs in the provided feasible schedule are constrained by deadlines and other constraints, which likely leds to non-empty coverage sets. Finally, as a remark, let us note that the definition of a mixed-criticality job assumes that $\boldsymbol{\mu}_i$ is a probability distribution, i.e., it sums to one. This assumption might not hold in cases where, e.g., the worst-case execution time cannot be bounded. On the other hand, one can always select finite values of processing times $\boldsymbol{\pi}_i$ such that the selected approximation is arbitrary close to the reality [1]. Therefore, this assumption is not crucial, but it simplifies the presentation of the main ideas.

Remark 2. *Note that in this paper, we deal with the computation of the objective function for a given schedule, but we do not touch the question of how to decide*

	replication	complexity result	algorithm
Seddik *et al.* [151]	no	finding start times of jobs with a fixed order to maximize $\sum w_i P_i$ is weakly $\mathcal{NP}$-hard	DP for $mc = 2$, MIP for $mc = \mathcal{L}$
This work	yes	computing $\sum P_i$ is $\#\mathcal{P}$-hard, fixed-parameter tractable with respect to $\max\{\mathcal{L}, \mathcal{R}\}$	inference in Bayesian networks

Table 4.1: Contributions of this paper.

which jobs to replicate and which start times to assign them. There are many possibilities to do that, e.g., in a greedy fashion, by a metaheuristic algorithm [184, 77] or by some exact method, but these require a method providing objective value for the current solution. However, the above question goes well beyond the scope of this paper.

One source of difficulty of computing the execution probability from Definition comes from the fact that the set $E_{\boldsymbol{s}}$ of all scenarios in schedule $\boldsymbol{s}$ may contain an exponential number of scenarios, i.e., $|E_{\boldsymbol{s}}| \in \mathcal{O}\left(\mathcal{L}^{n\mathcal{R}}\right)$, where $n = |I_{\mathcal{MC}}|$. Indeed, we will show that the job replication fundamentally changes (as opposed to the case without replication) the computational complexity of the problem and requires a new algorithm for the computation. Even though we will show that the exact computation of the execution probability is hard in general, we propose an efficient method for a tractable class of schedules.

4.1.4 Related work

The study of mixed-criticality systems originates from the real-time scheduling community due to its practical applications. The most widely adopted mixed-criticality scheduling model was proposed by Vestal in his seminal paper [175]. There, he proposes a model of mixed-criticality jobs that assumes that each job has an integer criticality with each criticality level associated with processing time for that level of assurance. This understanding of mixed-criticality was later adopted by the majority of follow-up works, e.g., Baruah [16], Burns [41], and Davis [54]. This line of research mostly deals with response time analysis of various scheduling policies considering preemptive jobs [86] in so-called event-triggered environment [93]. For a comprehensive review of mixed-criticality scheduling literature, we refer the reader to [40].

A known challenge for complex event-triggered systems is the difficulty of certification for safety-critical applications [13, 61]. Therefore, a new stream has emerged and turned the attention toward static scheduling in mixed-criticality systems [7, 170, 14] that simplifies the certification. A problem with preemptive jobs with two criticality levels was addressed in [20]. The authors proposed a heuristic algorithm that constructs a static schedule for multiple resources while considering precedence constraints. Novak *et al.* [7] dealt with non-preemptive mixed-criticality jobs up to three criticality levels to minimize the length of the schedule, i.e., the makespan. They proposed an approximation algorithm and a branch-and-price decomposition to solve the problem. Seddik [151] noted that makespan minimization with mixed-criticality jobs decreases the probability of the execution. Thus, instead of minimizing the length of the schedule, they proposed a non-regular scheduling criterion that maximizes the execution probability of

jobs — spreading them as much as possible under the deadline constraints. They derived a closed-form formula for the computation of the execution probability of jobs, given that the start times are fixed. Furthermore, they gave proof that finding optimal start times with the fixed permutation remains $\mathcal{NP}$-hard, and they proposed (i) a dynamic programming (DP) for the case of two criticality levels and (ii) a mixed-integer linear programming (MIP) model for the general problem. To find an optimal permutation, they developed a branch-and-bound algorithm. However, the used criterion may lead to schedules with low utilization of resources, as they did not consider the job replication.

The problem in this paper is related to stochastic scheduling due to the uncertainty of processing times [83]. There are a plethora of works focusing on processing time uncertainty, often assuming a uniform distribution, normal distribution, an uncertainty set [78, 135], or they are distributionally robust [44]. Many of these problems can be formulated as β-robust problems [53] with the goal of, e.g., maximizing the probability that all jobs are completed before the given due date. To the best of our knowledge, the existing approaches in the literature addressing stochastic and robust scheduling cannot be used to solve this problem.

The concept of job replication in scheduling is studied in works concerning parallel algorithms and communication delays [35, 125]. The idea is that replication of a job to another processor avoids extra communication and reduces system overhead. Hence, they observe that job replication can decrease the makespan of a schedule when multiple processing units are considered [85]. This is similar to our case, where we allow to consume resource time (that would not be utilized in any way) to improve the objective value by scheduling the same job more than once.

Best to our knowledge, the effect of job replication in static mixed-criticality systems has not been studied before, although it arises as a natural generalization of former mixed-criticality models, such as [175, 151]. We summarize the main contributions of this paper with respect to the most similar work [151] highlighted in Table 4.1.

In the following section, we prove the main complexity result of this paper concerning the computation of the execution probability in a mixed-criticality schedule with replication.

4.2 Time complexity of the problem

We show that the general problem where either the maximum number of criticality levels $\mathcal{L}$ or the maximum number of replicas per job $\mathcal{R}$ is bounded by a polynomial in the number of jobs and the other is equal to some chosen constant, remains $\#\mathcal{P}$-hard. We remind that $\#\mathcal{P}$ is a class of *counting problems*, i.e., a set of problems that count the number of accepting paths in a polynomial-time non-deterministic Turing machine [172]. An example of a problem contained in $\#\mathcal{P}$ is the following: What is the number of spanning trees in the given connected simple graph? A problem is said to be $\#\mathcal{P}$-hard, if for every problem in $\#\mathcal{P}$, there exists a polynomial-time counting reduction to it [51].

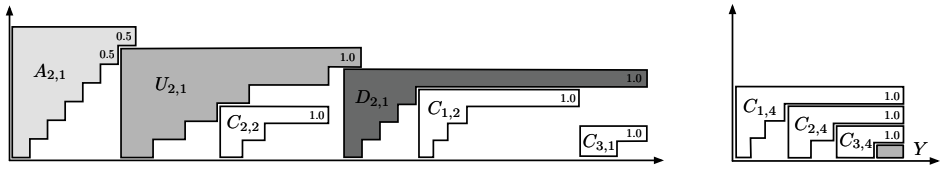
First, we show that to decide whether a job has a non-zero probability of being executed is as hard as determining whether a CNF (*conjunctive normal form*)

formula is satisfiable.

Proposition 9. *There exists a finite number of maximum replicas per job $\mathcal{R}$ such that deciding whether $P_i > 0$ for some job J_i is $\mathcal{NP}$-complete.*

Proof. First of all, it can be seen that the problem is contained in $\mathcal{NP}$. Indeed, having an execution scenario, we verify in polynomial time in the length of the input (number of all replicas and criticality levels) whether the given job J_i is executed (e.g., similarly as in Algorithm 4.1).

Next, we will show a polynomial reduction from 3-SAT to our problem. Consider a 3-CNF propositional formula $\mathcal{T} = \bigwedge_{k=1}^{m} c_k = \bigwedge_{k=1}^{m} \left(u_1^k \vee u_2^k \vee u_3^k\right)$ with m clauses and n variables $u_j, j \in \{1, \ldots, n\}$, where u_q^k denotes q-th literal of k-th clause. For such a formula, we construct the schedule in the following way. We use five sets of jobs. We have a set $U = \{U_1, \ldots, U_n\}$ that denotes jobs corresponding to variables $u_1, \ldots, u_n$ and a set $C = \{C_1, \ldots, C_m\}$ that corresponds to clauses $c_1, \ldots, c_m$. Next, we have jobs denoted as $A = \{A_1, \ldots, A_n\}$ that serve as an apriori setting of the execution of jobs U, i.e., they generate execution scenarios corresponding to all possible variable assignments of the formula $\mathcal{T}$. Furthermore, we have a set of jobs $D = \{D_1, \ldots, D_n\}$ where D_j is present if the variable u_j acts as a positive literal in some clause. Finally, we have a single job Y, whose execution probability is used to decide whether $\mathcal{T}$ is satisfiable or not. Hence, we have that $I_{\mathcal{MC}} = U \cup C \cup A \cup D \cup \{Y\}$. Job Y takes the role of job J_i, i.e., the job for which we investigate whether it will be executed with a non-zero probability.



(a) Gadget for variable u_2. (b) Gadget for AND gate.

Figure 4.6: Basic gadgets used in the reduction utilizing a constant number $\mathcal{R}$ of maximum replicas count.

Now, we will describe the parameters of the jobs. The jobs C_k have criticality $\mathcal{X}_{C_k} = m + 2 - k$, jobs U_j have criticality $\mathcal{X}_{U_j} = m + 3$, A_j have criticality $\mathcal{X}_{A_j} = m + 4$, jobs D_j have criticality $\mathcal{X}_{D_j} = m + 2$ and job Y has criticality $\mathcal{X}_Y = 1$. All jobs from sets U, D, and C have a *Dirac distribution* of probability over processing times — the probability of the execution at their top level is 1, i.e., $\boldsymbol{\mu} = (0, \ldots, 0, 1)$. Jobs A_j have probability distribution $\boldsymbol{\mu}_j = (0, \ldots, 0, 0.5, 0.5)$. Furthermore, we have that jobs in A, U, D, and job Y have a single replica while jobs in C have exactly four replicas; see Figure 4.7 where all gadgets are shown in one schedule.

The schedule consists of n copies of the gadget shown in Figure 4.6a. For each variable u_j we have one gadget containing a replica of all jobs $C_{k,\{1,2,3\}} \in \mathrm{cov}(U_{j,1})$ for clauses c_k where u_j is a negative literal and $C_{k,\{1,2,3\}} \in \mathrm{cov}(D_{j,1})$ for clauses where u_j is a positive literal. The indices $\{1, \ldots, 3\}$ of replicas $C_{k,\{1,2,3\}}$ are given according to appearance of variables u_j, $j \in \{1, \ldots, n\}$ in clauses c_k, $k \in \{1, \ldots, m\}$. The purpose of jobs $A_{j,1}$ is to ensure that for each variable u_j, job replica $U_{j,1}$ will be executed with probability 0.5. If $U_{j,1}$ is rejected, then replicas C_k, where that

u_j acts as a negative literal in clause c_k will be executed. If $U_{j,1}$ is executed, then $D_{j,1}$ is rejected; thus, all C_k in its coverage are executed since those correspond to clauses where u_j acts as a positive literal. Please note that it is crucial that jobs both in $\text{cov}(U_{j,1})$ and $\text{cov}(D_{j,1})$ do not overlap in each. The schedule is concluded with the gadget shown in Figure 4.6b, which implements AND gate, ensuring that job Y is executed if only if for each $k \in \{1, \ldots, m\}$ a replica $C_{k,\{1,2,3\}}$ was executed in the schedule before AND gate.

Consider an example formula $\mathcal{T} = c_1 \wedge c_2 \wedge c_3 = (u_1 \vee u_2 \vee u_3) \wedge (\neg u_1 \vee \neg u_2 \vee u_3) \wedge (u_2 \vee \neg u_3 \vee u_4)$. The resulting schedule derived from the reduction can be seen in Figure 4.7. The resulting schedule has $\mathcal{O}(n+m)$ replicas with at most $\mathcal{O}(m)$ criticality levels and the maximum replica count per job $\mathcal{R} = 4$.

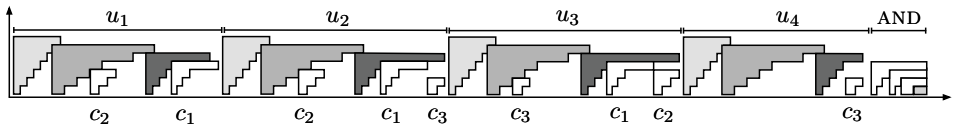


Figure 4.7: Example schedule of the reduction from $\mathcal{T} = c_1 \wedge c_2 \wedge c_3 = (u_1 \vee u_2 \vee u_3) \wedge (\neg u_1 \vee \neg u_2 \vee u_3) \wedge (u_2 \vee \neg u_3 \vee u_4)$.

Next, we need to show that in such a schedule $P_Y > 0$ if and only if $\mathcal{T}$ is satisfiable.

SAT $\implies$ $P_Y > 0$. Let ϕ be an assignment of formula $\mathcal{T}$ that is true. We will show that the probability of the execution of job Y is greater than zero, i.e., there is an execution scenario $\overline{e}$ with a non-zero probability that includes Y. Let us define the execution scenario $\overline{e}$ such that $U_{j,1}$ is executed if and only if $\phi(u_j) = \top$. This scenario occurs with probability 2^{-n}, which follows from the choice of the probability distribution of jobs A. We will show that under scenario $\overline{e}$, the job Y is executed.

If ϕ is a true assignment of $\mathcal{T}$, then $\forall c_k \in \mathcal{T}$ there exists a literal u_j that makes clause c_k satisfied. Let u_j be such literal for clause c_k:

- If u_j acts as a positive literal in c_k, then $\phi(u_j) = \top$. Thus, $U_{j,1}$ is executed, therefore $D_{j,1}$ is rejected, since from the schedule construction follows that $C_k \in \text{cov}(D_{j,1})$. Hence, a replica $C_{k,\{1,2,3\}}$ is executed.

- If u_j acts as a negative literal in c_k, then $\phi(u_j) = \bot$. From the definition of scenario $\overline{e}$ follows that $U_{j,1}$ is rejected; thus, a $C_{k,\{1,2,3\}}$ is executed since $C_k \in \text{cov}(U_{j,1})$, which follows from the construction of the schedule.

Since for all jobs C_k, one replica is executed before AND gate, then the last replica of each C_k is rejected; therefore, Y is executed. Hence, job Y is executed in at least one scenario, thus $P_Y > 0$.

$P_Y > 0 \implies$ SAT. If job Y has a non-zero probability of execution, then there is an execution scenario $\overline{e}$ such that for every C_k, one replica is executed before AND gate, otherwise Y would be rejected. We will show, that this scenario defines an assignment of formula $\mathcal{T}$ that is true.

Let us define the assignment ϕ such that $\phi(u_j) = \top$ if and only if $U_{j,1}$ is executed under scenario $\overline{e}$. We will show that ϕ is a model of $\mathcal{T}$. Let $C_{k,\{1,2,3\}}$ be a job replica

that is executed in scenario $\overline{e}$ in the schedule gadget corresponding to variable u_j. Then,

- If $C_k \in \text{cov}(D_{j,1})$, then job replica $U_{j,1}$ was executed, therefore $\phi(u_j) = \top$. From the construction of the schedule, u_j acts as the positive literal in c_k; therefore, clause c_k is satisfied.
- If $C_k \in \text{cov}(U_{j,1})$, then $U_{j,1}$ was rejected, therefore $\phi(u_j) = \bot$. From the construction of the schedule it holds that u_j acts as the negative literal in c_k; therefore, clause c_k is satisfied.

Since ϕ is true in all clauses, it is a model of $\mathcal{T}$, and, thus $\mathcal{T}$ is satisfiable. To illustrate how assignments of the formula are mapped to execution scenarios, consider the following two examples. Let us have an assignment ϕ, such that $\phi(u_1) = \top$, $\phi(u_2) = \bot$, $\phi(u_3) = \bot$ and $\phi(u_4) = \top$. The execution scenario corresponding to assignment ϕ is given as $\overline{e} = (A_{1,1} \sim 6, U_{1,1} \sim 6, C_{1,1} \sim 4, A_{2,1} \sim 7, C_{2,2} \sim 3, D_{2,1} \sim 5, A_{3,1} \sim 7, C_{3,2} \sim 2, D_{3,1} \sim 5, A_{4,1} \sim 6, U_{4,1} \sim 6, Y \sim 1)$. An another assignment ϕ' for which $\phi'(u_1) = \top$, $\phi'(u_2) = \top$, $\phi'(u_3) = \bot$ and $\phi'(u_4) = \bot$ translates into the scenario $\overline{e}' = (A_{1,1} \sim 6, U_{1,1} \sim 6, C_{1,1} \sim 4, A_{2,1} \sim 6, U_{2,1} \sim 6, C_{3,1} \sim 2, A_{3,1} \sim 7, D_{3,1} \sim 5, A_{4,1} \sim 7, D_{4,1} \sim 5, C_{2,4} \sim 3)$. □

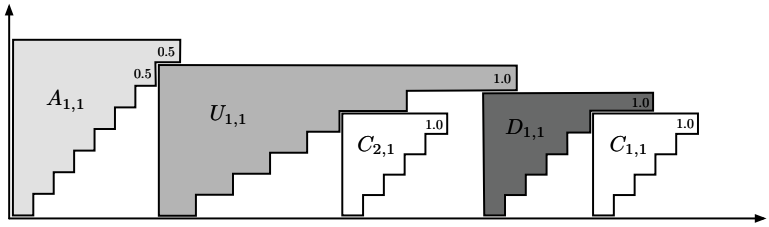
The reduction suggests that the problem remains hard even for a constant number of the maximum job replicas, i.e., $\mathcal{R} = 4$. Moreover, we will show that a non-constant number of criticality levels is not the only source of hardness. Indeed, the problem remains hard, assuming a constant number of criticality levels when the maximum number of replicas is not fixed to a constant.

Proposition 10. *There exists a finite number of criticality levels $\mathcal{L}$ such that determining whether $P_i > 0$ for some job J_i is $\mathcal{NP}$-complete.*

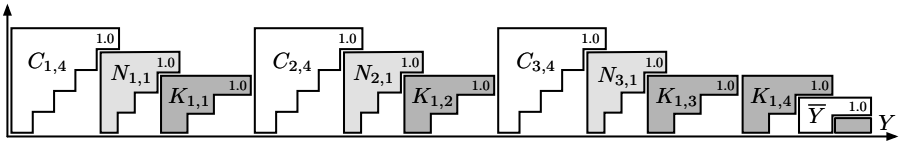
Proof. The reduction uses a similar idea as the one described in Proposition 9, where the main difference lies in the construction of AND gate. The resulting schedule again consists of two parts. The first part represents variables $u_1, \ldots, u_n$ and uses $\mathcal{L} = 8$ criticality levels. A replica of job C_k (representing clause c_k) is executed when clause c_k contains a literal u_q^k that makes the clause satisfied. The difference from the structure shown in Figure 4.6a is that here all C_k's have criticality $\mathcal{X}_{C_k} = 5$, and thus $\mathcal{X}_{D_j} = 6$, $\mathcal{X}_{U_j} = 7$, and $\mathcal{X}_{A_j} = 8$. See the example gadget corresponding to variable u_1 from formula $\mathcal{T}$ in Figure 4.8a.

The other difference lies in the second part of the schedule, i.e., AND gate, which has to be redesigned to have a constant number of criticality levels. This can be done using $\mathcal{R} \in \mathcal{O}(m)$ maximum replicas per job in a way shown in Figure 4.8b. The structure of the new AND gate introduces three additional job types: K, N and $\overline{Y}$. Job K has $m + 1$ replicas in the schedule, and the job $\overline{Y}$ a single one. Furthermore, jobs $N = \{N_1, N_2, \ldots, N_m\}$ have a single replica in the schedule for each of them. Jobs in N, K, and $\overline{Y}$ have Dirac distribution of processing times with the probability of one at their top level. The purpose of jobs N_k is to ensure that at least one of the first m replicas of job K is executed if no replica of each C_k, $k \in \{1, \ldots, m\}$ is executed before AND gate. If this happens, then we need to reject Y, which is done by executing the last replica $K_{1,m+1}$. The job replica $\overline{Y}$ ensures that Y is executed if and only if $K_{1,m+1}$ is executed.

Next, we show that job replica Y is executed if and only if a replica for every job C_k was executed in the schedule before AND gate.



(a) Gadget for variable u_1 that uses $\mathcal{L} = 8$ criticality levels.



(b) Gadget for AND gate that uses $\mathcal{L} = 5$ criticality levels.

Figure 4.8: Basic gadgets used in the reduction utilizing a constant number of criticality levels $\mathcal{L}$.

- Let us assume that for every scenario $\overline{e} \in E_{\boldsymbol{s}}$ there is a job C_k with no replica executed before AND gate. Therefore, the last replica $C_{k,4}$ is executed in AND gate. Then $N_{k,1}$ is rejected and, hence, the replica $K_{1,k}$ is executed. Therefore, in the last section of AND gate, the last replica $K_{1,m+1}$ is rejected; thus, $\overline{Y}$ is executed, which leads to rejection of Y. Furthermore, if for all scenarios $\overline{e} \in E_{\boldsymbol{s}}$ hold that there is a C_k with no replica executed before AND gate, then $P_Y = 0$.
- Next, let us assume that there is a scenario such that all C_k's are executed before AND gate. Hence, $C_{k,4}$, $k \in \{1, \ldots, m\}$ are rejected, and the corresponding $N_{k,1}$ are executed. Therefore, all replicas of job K are rejected except the last one (i.e., $K_{1,m+1}$). Thus $\overline{Y}$ is rejected, and then, finally, Y is executed; therefore, $P_Y > 0$ follows.

□

Remark 3. *Notice that jobs in sets U, C, D, N, and K in the reductions above have assumed a Dirac distribution, hence their processing times are, in fact, deterministic, which might feel a bit unnatural. However, it implies several interesting consequences. Namely, if there would be a polynomial-time algorithm that would decide whether a job can be executed, then such an algorithm either has to exploit the fact that distributions of processing times are not Dirac or it would not be sound under $\mathcal{P} \neq \mathcal{NP}$ assumption. Nevertheless, the usage of Dirac distribution for some of the jobs is not the crucial idea in the reduction. One could use a distribution that is* almost *Dirac except for some small positive ϵ. The value of ϵ would be set such that for an unsatisfiable formula the probability of execution of job Y would be non-zero, but arbitrarily small (e.g., $\ll 2^{-\mathcal{O}(poly(nm))}$) while a satisfiable formula would admit the execution of Y with probability at least $2^{-n} - f(\epsilon)$ where $f(\epsilon)$ is an arbitrarily*

small non-negative value that incorporates the sum of probabilities of false-negative *scenarios (i.e., scenarios corresponding to true valuations of the formula but do not execute Y).*

Finally, it can be seen that an algorithm computing the exact value of P_Y could be used to solve #3-SAT problem [126] (i.e., the number of satisfiable assignments for the given 3-SAT formula) as it preserves the number of YES certificates in each problem. Indeed, under the considered reduction, each true assignment of the formula $\mathcal{T}$ with n variables increases the execution probability of job Y by 2^{-n} while a false assignment does not increase it and vice versa. Since it is known that #3-SAT is $\#\mathcal{P}$-complete, we have the following corollary:

Corollary. *Computing P_i remains $\#\mathcal{P}$-hard when $\mathcal{L}$ or $\mathcal{R}$ is bounded by a polynomial in the number of jobs while the other is equal to a sufficiently large constant.*

Remark 4. *Note that it is known that* #2-SAT *is $\#\mathcal{P}$-complete as well [173], although the decision problem of* 2-SAT *is polynomially solvable. This suggests that the exact computation of execution probability is hard already for $\mathcal{R} = 3$ maximum replicas, which follows from the reduction introduced in the proof of Proposition 9. On the other hand, for the case $\mathcal{R} = 1$, a polynomial-time solution is known [151]. Hence, the complexity of the case $\mathcal{R} = 2$ stays as an open problem. For the number of criticality levels $\mathcal{L}$ we know that $\mathcal{L} = 8$ already leads to intractability if $\mathcal{R}$ is not a constant.*

In the next section, we show that the general problem (with arbitrary $\mathcal{L}$ and $\mathcal{R}$) of computing the execution probability can be reduced to probabilistic inference in Bayesian networks, which gives us an algorithm for the computation. The inference in the resulting network has complexity parametrized by the number of criticality levels $\mathcal{L}$ and by the number of maximum replicas $\mathcal{R}$. Furthermore, we show that the problem becomes tractable, when both $\mathcal{L}$ and $\mathcal{R}$ are bounded by a constant, which is often a realistic case for real-life applications.

4.3 Algorithm for computation of the execution probability

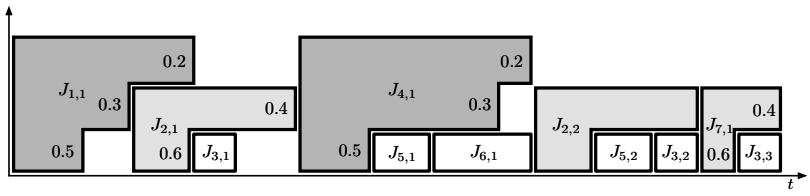
In this section, we show how the statistical properties of mixed-criticality schedules with replication can be described with Bayesian networks. A Bayesian network $G = (V, A)$ is a probabilistic directed acyclic graphical model that represents the joint distribution over the set V of random variables using conditional dependencies defined by edges A. Since the processing time of jobs is uncertain, we can view the job replicas as random variables. Then, the execution policy (i.e., Algorithm 4.1) defines a joint probability distribution $\Pr\{J_{1,1}, \ldots, J_{n,n_i}\}$ over the given schedule that assigns a probability to each execution scenario. However, the representation of the full joint distribution is costly as it has $\mathcal{O}(\mathcal{L}^{n\mathcal{R}})$ parameters. To overcome this, we use Bayesian networks (BN) [139], which can be seen as an efficient way of representing joint distributions.

We show that any schedule for scheduling problem $1|mc = \mathcal{L}, rep = \mathcal{R}|\sum P_i$ defines a Bayesian network. We use this network for the computation of the execution

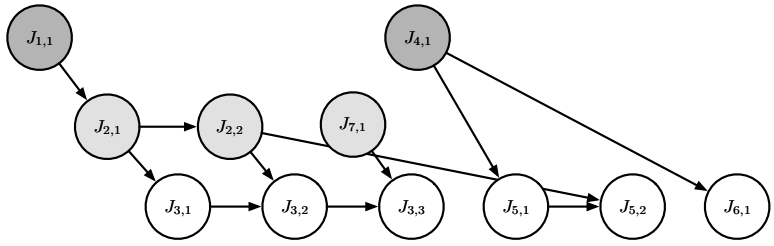
probability of jobs using the existing inference algorithms for BNs. Moreover, we utilize the theoretical framework of Bayesian networks to analyze which case of the problem admits an efficient algorithm for probability computation.

4.3.1 Reduction to Bayesian networks

Under the considered mixed-criticality model, replica $J_{i,q}$ with $\mathcal{X}_i$ criticality levels can be either: (i) executed at one of its levels (i.e., $\mathcal{X}_i$ outcomes), (ii) rejected by some other job replica, or (iii) rejected due to successfully completed previous replica of the same job. The probabilities of its outcomes are given by the start times of other replicas in the schedule. If the replica is executed, then its outcomes have probabilities given by the job specification, i.e., $\boldsymbol{\mu}_i$ distribution. When the replica is rejected, then all these outcomes have a probability of zero. However, the probability of being executed or rejected depends on the preceding jobs replicas in the schedule and their coverage.



(a) Example schedule $\boldsymbol{s}$.



(b) Corresponding Bayesian network $G(\boldsymbol{s})$.

Figure 4.9: Representation of a schedule by a Bayesian network.

4.3.1.0.1 Network structure The structure of Bayesian network $G(\boldsymbol{s}) = (V_{\boldsymbol{s}}, A_{\boldsymbol{s}})$ corresponding to a schedule $\boldsymbol{s}$ is defined in the following way. We assume that each job replica in the schedule corresponds to a single discrete random variable. Hence, the set of vertices $V_{\boldsymbol{s}}$ is equal to the set of job replicas in schedule $\boldsymbol{s}$. A random variable (i.e., a vertex) corresponding to job replica $J_{i,q}$ has outcome space given as $\{\dagger, \star, 1, \ldots, \mathcal{X}_i\}$ with the total ordering defined as $\dagger \prec \star \prec 1 \prec \ldots \prec \mathcal{X}_i$. The outcomes $\ell \succeq 1$ coincide with the execution levels of the job. That is, $J_{i,q}$ has the outcome ℓ, i.e., $J_{i,q} \sim \ell$ for $\mathcal{X}_i \succeq \ell \succeq 1$ if and only if its execution level is ℓ. The outcome $\ell = \star$ denotes that the job replica was rejected by a preceding replica of a different job with higher criticality (i.e., it was covered by it). Finally, the outcome $\ell = \dagger$ denotes that a preceding replica of the same job was completed in the schedule before. Note that we use two different outcomes for the rejection of a

job replica, i.e., $\star$ and $\dagger$. This distinction is necessary to model the fact that the execution policy of mixed-criticality schedules executes at most one replica of each job (see Algorithm 4.1). Next, as an example, let us demonstrate how the outcomes of random variables relate to possible execution scenarios in the schedule. Consider the execution scenario $\bar{e}_3$ displayed in Figure 4.5b. The sequence of outcomes corresponding to this scenario $\bar{e}_3$ is $(2, \star, \star, 1, 1, \dagger)$.

Next, we describe how vertices in the network are connected. Vertex $J_{k,r}$ is connected to $J_{i,q}$ (i.e., $J_{k,r} \to J_{i,q} \in A_{\boldsymbol{s}}$) if and only if

$$J_{i,q} \in \mathrm{cov}(J_{k,r}) \text{ or } (i = k \wedge q = r + 1)\,. \tag{4.3.1}$$

In other words, for each vertex, its parent is a vertex covering it in the schedule or its immediate preceding replica. The intuition behind connections defined like that is that a job replica needs to be connected to all other job replicas that influence its execution. See an example schedule in Figure 4.9a and the corresponding BN displayed in Figure 4.9b. Let us note that we may assume that the network $G(\boldsymbol{s})$ is connected; otherwise, one could deal with each component separately.

Conditional distributions The other parameters of BNs are *conditional probability tables* (CPTs). For each random variable, CPT defines a distribution of outcomes conditioned by the outcomes of all its parents (i.e., the *evidence*). In our case, CPTs are defined as follows. If vertex $J_{i,q}$ has no parent, then its CPT is simply the distribution over its criticality levels $\{1, \ldots, \mathcal{X}_i\}$ with the outcomes $\star$ and $\dagger$ having zero probability. If $J_{i,q}$ is the first replica of a job (i.e., $q = 1$), then for the outcomes of its parents that reject $J_{i,q}$, its CPT assigns the probability of 1 for $\star$ outcome. For the outcomes of its parents that permit its execution, the probabilities of outcomes $\{1, \ldots, \mathcal{X}_i\}$ are given by $\boldsymbol{\mu}_i$ distribution.

Consider the schedule in Figure 4.9a with the parameters of distributions for job J_1 and J_4 are $\boldsymbol{\mu}_1 = \boldsymbol{\mu}_4 = (0.5, 0.3, 0.2)$, for job J_2 and J_7 are $\boldsymbol{\mu}_2 = \boldsymbol{\mu}_7 = (0.6, 0.4)$, and for J_3, J_5 and J_6 are $\boldsymbol{\mu}_3 = \boldsymbol{\mu}_5 = \boldsymbol{\mu}_6 = (1.0)$. If $J_{1,1}$ is executed at criticality level 3, then $J_{2,1}$ will be rejected, and, given that $J_{1,1} \sim 3$, the probability of $J_{2,1}$ being rejected is 1. On the other hand, when $J_{1,1} \prec 3$, then the probability of $J_{2,1}$ being rejected is zero. More precisely, the conditional probabilities for replica $J_{i,q}$ are given as

$$\Pr\left\{ J_{i,q} \sim \ell \,\middle|\, \bigwedge_{\substack{(j,r,k):\\ J_{i,q} \in \mathrm{cov}_k(J_{j,r})}} (J_{j,r} \prec k) \wedge (J_{i,q-1} \sim \star) \right\} = \mu_i^{(\ell)},$$

$$\Pr\left\{ J_{i,q} \sim \star \,\middle|\, \left(\bigvee_{\substack{(j,r,k):\\ J_{i,q} \in \mathrm{cov}_k(J_{j,r})}} (J_{j,r} \succeq k) \right) \wedge (J_{i,q-1} \sim \star) \right\} = 1,$$

$$\Pr\left\{ J_{i,q} \sim \dagger \,\middle|\, J_{i,q-1} \succ \star \vee J_{i,q-1} \sim \dagger \right\} = 1,$$

where without loss of generality, we assume that $J_{i,0} \sim \star$.

Example CPTs for some of the replicas for the schedule in Figure 4.9a can be seen in Table 4.2. The replicas $J_{1,1}$, $J_{4,1}$, and $J_{7,1}$ do not have any evidences (i.e., the outcomes of all its parents) since their sets of parents are empty. Similarly, the blank evidence, e.g., the first row for $J_{2,2}$ in Table 4.2f, is used to denote that an arbitrary outcome (i.e., $\dagger, \star, 1, \ldots$) of its parents applies. Note that we have used a compact representation of conditional probabilities instead of full CPTs, which can be in fact exponential in $\mathcal{L}$. Therefore, BNs admit also different representations of conditional distributions (e.g., decision trees) that lead to more concise representation in some cases.

outcome					evidence
$\dagger$	$\star$	1	2	3	
0.0	0.0	0.5	0.3	0.2	$\emptyset$

(a) Replicas $J_{1,1}$ and $J_{4,1}$.

outcome				evidence
$\dagger$	$\star$	1	2	
0.0	0.0	0.6	0.4	$\emptyset$

(b) Replica $J_{7,1}$.

outcome				evidence
$\dagger$	$\star$	1	2	$J_{1,1}$
0.0	0.0	0.6	0.4	$\dagger \vee \star \vee 1 \vee 2$
0.0	1.0	0.0	0.0	3

(c) Replica $J_{2,1}$.

outcome				evidence
$\dagger$	$\star$	1	2	$J_{2,1}$
1.0	0.0	0.0	0.0	$\dagger \vee 1 \vee 2$
0.0	0.0	0.6	0.4	$\star$

(d) Replica $J_{2,2}$.

outcome			evidence
$\dagger$	$\star$	1	$J_{2,1}$
0.0	0.0	1.0	$\dagger \vee \star \vee 1$
0.0	1.0	0.0	2

(e) Replica $J_{3,1}$.

outcome			evidence	
$\dagger$	$\star$	1	$J_{3,1}$	$J_{2,2}$
1.0	0.0	0.0	$\dagger \vee 1$	
0.0	0.0	1.0	$\star$	$\dagger \vee \star \vee 1$
0.0	1.0	0.0	$\star$	2

(f) Replica $J_{3,2}$.

outcome			evidence	
$\dagger$	$\star$	1	$J_{3,2}$	$J_{7,1}$
1.0	0.0	0.0	$\dagger \vee 1$	
0.0	0.0	1.0	$\star$	$\star \vee 1$
0.0	1.0	0.0	$\star$	2

(g) Replica $J_{3,3}$.

outcome			evidence
$\dagger$	$\star$	1	$J_{4,1}$
0.0	0.0	1.0	$\dagger \vee \star \vee 1$
0.0	1.0	0.0	$2 \vee 3$

(h) Replica $J_{5,1}$.

outcome			evidence	
$\dagger$	$\star$	1	$J_{5,1}$	$J_{2,2}$
1.0	0.0	0.0	$\dagger \vee 1$	
0.0	0.0	1.0	$\star$	$\dagger \vee \star \vee 1$
0.0	1.0	0.0	$\star$	2

(i) Replica $J_{5,2}$.

outcome			evidence
$\dagger$	$\star$	1	$J_{4,1}$
0.0	0.0	1.0	$\dagger \vee \star \vee 1$
0.0	1.0	0.0	$2 \vee 3$

(j) Replica $J_{6,1}$.

Table 4.2: CPTs for the example schedule $\boldsymbol{s}$.

Marginalization To compute the probability of execution of jobs, one needs to perform marginal inference in the constructed network. There are known algorithms that can perform such inference and their implementations are widely available.

	$J_{1,1}$	$J_{2,1}$	$J_{2,2}$	$J_{3,1}$	$J_{3,2}$	$J_{3,3}$	$J_{4,1}$	$J_{5,1}$	$J_{5,2}$	$J_{6,1}$	$J_{7,1}$
$\Pr\{J_{i,q} \succeq 1\}$	1.0	0.8	0.2	0.68	0.32	0.0	1.0	0.5	0.46	0.5	1.0

Table 4.3: Execution probabilities of individual replicas in schedule $\boldsymbol{s}$.

For the exact inference in BNs, multiple algorithms exist [76], such as *variable elimination* or *junctions trees*. One can also trade the precision for the computational time and use approximate inference methods [119], such as *Markov Chain Monte Carlo* or *Gibbs sampling*. To compute the execution probability of replica $J_{i,q}$, we utilize the property of Bayesian networks [104, 139] that states

$$\Pr\{J_{i,q}\} = \sum_{\forall s_{j,r} \in \boldsymbol{s} \setminus \{s_{i,q}\}} \prod_{i'=1}^{n} \prod_{q'=1}^{n_{i'}} \Pr\{J_{i',q'} \,|\, parents(J_{i',q'})\}, \tag{4.3.2}$$

where $parents(J_{i',q'})$ denotes the set of all immediate predecessors of $J_{i',q'}$ in BN $G(\boldsymbol{s})$. The equation (4.3.2) suggests that the probability distribution $\Pr\{J_{i,q}\}$ can be computed with the marginalization over all variables except $J_{i,q}$ of the joint distribution $\Pr\{J_{1,1}, \ldots, J_{n,n_n}\}$. The joint distribution is factorized using the conditional independence relations defined by the network. The complexity of the computation is hidden in the marginalization step where we need to perform the summation over all combinations of replicas' outcomes and multiply their probabilities altogether. However, one can notice that vertices that are not ancestors of $J_{i,q}$ in $G(\boldsymbol{s})$ do not influence the distribution $\Pr\{J_{i,q}\}$ as they are marginalized out during the computation of (4.3.2). Hence, for each job in the schedule, we can define a smaller Bayesian network containing only its ancestor vertices. In fact, this is a concept related to the question of relevant nodes in Bayesian inference for a query with the given set of evidence nodes.

Definition (Bayesian network with respect to a job replica)**.** *Bayesian network with respect to job replica $J_{i,q}$, denoted by $G'_{i,q}$, is a subgraph of $G(\boldsymbol{s})$ induced by the set of vertices reachable from $J_{i,q}$ with the reversed orientation of edges in $G(\boldsymbol{s})$.*

For example, in Figure 4.9b, the Bayesian network with respect to $J_{3,2}$ has vertices $V(G'_{3,2}) = \{J_{1,1}, J_{2,1}, J_{2,2}, J_{3,1}, J_{3,2}\}$ and edges incident with $V(G'_{3,2})$. The inference is then performed in the network $G'_{i,q}$ for all replicas $J_{i,q}$ independently. Note that in specific cases, the Bayesian network with respect to some job replica might be as large as the original network $G(\boldsymbol{s})$. Indeed, for the schedule used in hardness reduction in Figure 4.7, the network with respect to job Y is identical to the whole network, i.e., $G'_{Y,1} = G(\boldsymbol{s})$. However, in the next section, we show that under realistic assumptions, the restricted networks are much smaller.

The complete algorithm that computes $\sum P_i$ is given in Algorithm 4.2.

Algorithm 4.2: Computation of $\sum P_i$ in schedule $\boldsymbol{s}$.

input : a schedule $\boldsymbol{s}$
output : the sum of execution probabilities of jobs $\sum P_i$

```
for J_i ∈ I_MC do
    for q = 1 to n_i do
        G'_{i,q} ← BN with respect to J_{i,q} in s
        P_{i,q} ← Pr{J_{i,q} ⪰ 1} in G'_{i,q}        // inference in G'_{i,q}
    P_i ← Σ_q P_{i,q}
return Σ P_i
```

The computed execution probabilities for the example in Figure 4.9a can be seen in Table 4.3.

Remark 5. *Note that there are three aspects that affect the particular values of the computed probabilities. Namely, it is the probability distribution of processing times of the jobs involved, the maximum number of replicas $\mathcal{R}$, and their actual schedule. For example, with $\mathcal{R} = 2$, it would be possible to achieve exactly the same execution probabilities as presented in Table 4.3 since the presence of $J_{3,3}$ replica does not increase the execution probability of job J_3 any further. However, with the identical set of jobs but with a different schedule, the result might be different. Therefore, it is a complex question with what value of $\mathcal{R}$ the schedules should be constructed. Another related question is whether with the increasing value of $\mathcal{R}$, the marginal improvements in the execution probability are non-increasing and if yes, how quickly the marginal improvements become negligible to the point where it is effectively meaningless to increase it further. However, in practice, the number of maximum replicas is often treated as a design parameter, not a variable to be optimized. Thus, the cases of the practical interest contain problems where $\mathcal{R}$ is fixed to a particular constant.*

4.3.2 Tractable case

In this section, we show that when both $\mathcal{L}$ and $\mathcal{R}$ are bounded by a constant independent from the input length, the computation becomes tractable. We note that this case is arguably the most practical one concerning real-world applications. Additionally, this case is also tight in the sense that when one of the $\mathcal{L}$ or $\mathcal{R}$ is not fixed, then the problem becomes intractable, as shown by Propositions 9 and 10.

The main idea is to realize that computationally the most intensive step in Algorithm 4.2 is the inference in BN $G'_{i,q}$. However, it can be shown that the size of the restricted network is limited by parameters $\mathcal{L}$ and $\mathcal{R}$. When both $\mathcal{L}$ and $\mathcal{R}$ are bounded by a constant, then the number of vertices $V(G'_{i,q})$ is independent of the number of *all* replicas in the schedule (i.e., does not depend on $n = |I_{\mathcal{MC}}|$, but only on $\mathcal{R}$ and $\mathcal{L}$).

Proposition 11. *Let $G'_{i,q}$ be a Bayesian network with respect to arbitrary $J_{i,q}$. Then it holds that*

$$\left|V(G'_{i,q})\right| \in \mathcal{O}\left((\mathcal{R}\mathcal{L})^{\mathcal{L}}\right).$$

Proof. The general idea is to follow the definition of how the networks are built in order to construct the largest possible network with the given fixed values of $\mathcal{R}$ and $\mathcal{L}$. Then, we bound the number of vertices in the resulting graph.

Let us consider a BN with respect to some job $J_{i,q}$ for the problem with $\mathcal{L}$ criticality levels, and $\mathcal{R}$ maximum replicas. Without loss of generality, let us assume that $\mathcal{X}_i = 1$. Let us organize $G'_{i,q}$ into levels, where level $L_\ell \subseteq V(G'_{i,q})$ contains all job replicas $J_{k,r}$ with criticality $\mathcal{X}_k = \ell$, such that $V(G'_{i,q}) = \bigcup_{\ell=1}^{\mathcal{L}} L_\ell$. Assuming we have at most $\mathcal{R}$ replicas per job, we have that $|L_1| \leq \mathcal{R}$. By (4.3.1), we have that every vertex has in-degree at most $(\mathcal{L}-1)+1 = \mathcal{L}$. Hence, the upper bound on the number of vertices in the second level is $|L_2| \leq \mathcal{R}^2 \times \mathcal{L}$. For level ℓ, we have that $|L_\ell| \leq \mathcal{R}^\ell \times \mathcal{L}^{\ell-1}$. Since we have $\mathcal{L}$ criticality levels, graph $G'_{i,q}$ contains at most $\left|V(G'_{i,q})\right| \leq \sum_{\ell=1}^{\mathcal{L}} |L_\ell| = \mathcal{R} + \mathcal{R}^2 \times \mathcal{L} + \ldots + \mathcal{R}^{\mathcal{L}} \times \mathcal{L}^{\mathcal{L}-1} = \mathcal{R} \times \frac{(\mathcal{R}\mathcal{L})^{\mathcal{L}}-1}{\mathcal{R}\mathcal{L}-1}$ vertices. Therefore, the cardinality of the vertex set is a function of $\mathcal{R}$ and $\mathcal{L}$ only. □

Remark 6. *Note that the upper bound suggested by Proposition 11 is overly pessimistic since we have assumed that each job replica $J_{i,q}$ is covered by $\mathcal{L}-1$ different job replicas of criticality $\mathcal{X}_i + 1$, which cannot occur. Hence, with more detailed analysis, the upper bound could be reduced.*

Next, we discuss what is the size of CPTs in $G'_{i,q}$. Since in-degree of each vertex is at most $\mathcal{L}$ by (4.3.1), then its CPT has size at most $\mathcal{O}((\mathcal{L}+2)^{\mathcal{L}})$, even under a trivial encoding given by a full CPT. Hence, the total size of the representation of BN $G'_{i,q}$ is independent of the number of jobs n. The total time complexity of Algorithm 4.2 is $\mathcal{O}(n \cdot \mathcal{R} \cdot f(\mathcal{L}, \mathcal{R}))$, where $f(\mathcal{L}, \mathcal{R})$ is complexity of the used inference algorithm in BN $G'_{i,q}$.

Finally, let us discuss inference complexity term $f(\mathcal{L}, \mathcal{R})$ in BN $G'_{i,q}$. The efficiency of exact inference algorithms is limited by the properties of conditional distributions as well as by the structure of networks. It is known that networks satisfying *local variance bound* (LVB), i.e., a requirement that forbids extreme conditional distributions, admit a subexponential deterministic inference algorithm [52]. Unfortunately, in our case, conditional distributions do not satisfy LVB due to the execution policy (a replica is rejected with the probability of one when the previous replica is executed). Concerning the structural properties of the network, it is known that they are mostly related to the treewidth of a graph. Informally, the treewidth of a graph is a quantity related to its connectivity. For example, the treewidth of the least possible connected graph (i.e., a tree) is equal to 1 whereas the complete graph with n vertices has treewidth equal to $n-1$. The result of [101] suggests that under reasonable assumptions, no polynomial algorithm exists for networks with unbounded treewidth. A similar idea to avoid intractability was applied in [147], where they focus on tree networks. In our case, the networks have bounded treewidth by a function of $\mathcal{L}$ and $\mathcal{R}$ which avoids the dependence on the total number of jobs n. Thus, its complexity does not depend on the number of jobs n. What is more, the practical experience suggests that, in an average case of mixed-criticality schedules, the treewidth of our networks achieves much smaller values.

Finally, we note that the current implementations of Bayesian network solvers easily handle computations in networks with hundreds to thousands of vertices

(i.e., replicas) within seconds [146]. Hence, we see the proposed method as a computationally efficient way of solving the problem.

4.4 Conclusion

In this work, we have introduced the replication of jobs as a mechanism for increasing the execution probability in mixed-criticality schedules. We have studied the complexity of the computation of execution probability in the given static schedule. Our main result shows that there are two primary parameters that influence the complexity — the maximum number of replicas per job $\mathcal{R}$ and the number of criticality levels $\mathcal{L}$. We have shown that although the replication significantly improves the execution probability in mixed-criticality schedules, it introduces additional complexity to the problem and the exact computation becomes $\#\mathcal{P}$-hard. In fact, the problem remains hard if either the maximum number of replicas $\mathcal{R} \geq 3$ or the number of criticality levels $\mathcal{L} \geq 8$ is fixed while the other quantity is bounded by a polynomial in the number of jobs.

To solve the problem, we have proposed a reduction to probabilistic inference in Bayesian networks, showing an interesting connection between schedules with uncertain execution and probabilistic graphical models. The analysis of the resulting networks shows that for the practical case when both $\mathcal{R}$ and $\mathcal{L}$ are bounded by a constant, the computation becomes tractable. Considering available implementations of exact and approximate inference algorithms for Bayesian networks, the problem can be efficiently solved in practice as well, offering a viable choice for improving the efficiency of static schedules for mixed-criticality systems.

Chapter

Scheduling jobs with normally distributed processing times on parallel machines

5

Antonín Novák, Premysl Sucha, Matej Novotny, Richard Stec, and Zdenek Hanzalek. "Scheduling jobs with normally distributed processing times on parallel machines". In: *European Journal of Operational Research* 297.2 (2022), pp. 422–441. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2021.05.011

5.1 Introduction

This work focuses on the problem of parallel machine scheduling, where the processing time of each job is an uncertain parameter and is given by a normal probability distribution. The optimal solution is a schedule that maximizes the probability that all jobs are finished before the common due date.

A practical example of such a scheduling model is the scheduling in a production stage operated by human workers. The workers are supposed to put various parts together to create a piece of product using the tools available at each workbench. A single assembly may take up to a few dozens of minutes but the precise time cannot be set in advance as the human aspect introduces uncertainty related to the complexity of the given piece. The presence of a common due date is typically related to the "end-of-the-day" targets, such as fulfilling a certain assembly quota or preparing boxes of the finished products to be dispatched. The above scheduling model finds also other applications, e.g., in scheduling of operations in operation theaters [179] or the allocation of control algorithms to computational units for unmanned aerial vehicles [9].

5.1.1 Problem statement

Consider a set of m identical parallel machines $M = \{M_1, \ldots, M_m\}$ and a set of n jobs $J = \{J_1, \ldots, J_n\}$. Let us denote j-th job as J_j with a processing time π_j which is an uncertain parameter and is described by a normal probability distribution. In other words, we associate processing time π_j of each job with a normal distribution, which is given by its mean μ_j and variance σ_j^2:

$$\pi_j \sim \mathcal{N}(\mu_j, \sigma_j^2), \quad \mu_j, \sigma_j^2 \in \mathbb{N}. \tag{5.1.1}$$

Let us denote the vector of all means as $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_n) \in \mathbb{N}^n$ and vector of variances $\boldsymbol{\sigma^2} = (\sigma_1^2, \ldots, \sigma_n^2) \in \mathbb{N}^n$. Finally, we are given a common due date $\delta \in \mathbb{N}_0$, before which all the jobs should be completed. The objective is to find an assignment

of jobs to machines that *maximizes the probability that we process all jobs before due date* δ. Note that without loss of generality, the assumption on integer parameters of the given distributions may be lifted to rational numbers, since all parameters may be multiplied by the least common multiplier of their denominators.

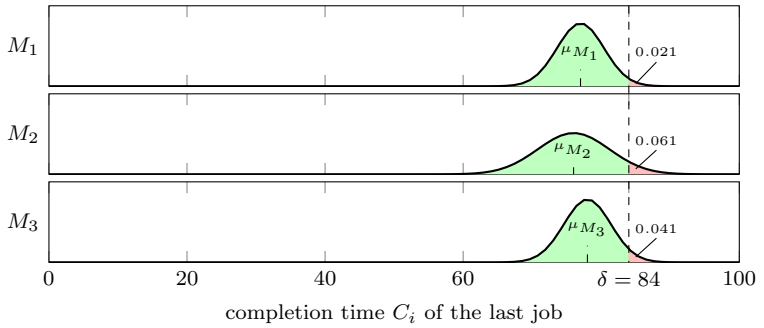


Figure 5.1: Example solution of a problem instance with three machines.

A solution to this problem is an assignment of jobs to machines. Let us consider a set of jobs $S_i \subseteq J$ to be scheduled on machine M_i represented by characteristic vector $\boldsymbol{s}_i = (s_{i,1}, \ldots, s_{i,n})$ with $s_{i,j}$ being the j-th element of $\boldsymbol{s}_i$. Characteristic vector $\boldsymbol{s}_i$ for set S_i means that $J_j \in S_i$ if and only if $s_{i,j} = 1$, otherwise $s_{i,j} = 0$. When the jobs are independent, it can be shown that the completion time C_i of the last job on machine M_i is a normally distributed random variable

$$C_i = \sum_{J_j \in S_i} \pi_j \sim \mathcal{N}(\mu_{M_i}, \sigma^2_{M_i}), \quad \mu_{M_i} = \boldsymbol{\mu}^\intercal \boldsymbol{s}_i, \sigma^2_{M_i} = \boldsymbol{\sigma}^{2\intercal} \boldsymbol{s}_i$$

where μ_{M_i} and $\sigma^2_{M_i}$ are the total mean and total variance of jobs assigned to M_i, respectively. The probability that machine M_i completes the assigned jobs before the due date δ (i.e., $\Pr[C_i \leq \delta]$) can be expressed by the cumulative normalized normal distribution function Φ as

$$\Pr[M_i \text{ finishes before } \delta] = \Phi\left(\frac{\delta - \mu_{M_i}}{\sqrt{\sigma^2_{M_i}}}\right). \tag{5.1.2}$$

Note that Φ does not have an analytical form and needs to be approximated in practice. An example illustrating a problem instance with three machines is shown in Figure 5.1. The green area represents the probability that machine M_i finishes the assigned jobs S_i before δ. Under the assumption that processing times of jobs are independent, the probability that all machines finish processing before the due date δ is

$$\Pr[\text{all jobs in } J \text{ are finished before } \delta] = \prod_{i=1}^{m} \Pr[M_i \text{ finishes before } \delta], \tag{5.1.3}$$

which can be rewritten as

$$\prod_{i=1}^{m} \Phi\left(\frac{\delta - \mu_{M_i}}{\sqrt{\sigma^2_{M_i}}}\right). \tag{5.1.4}$$

The problem defined above is known as a (β-robust) *stochastic parallel machine scheduling problem* [135]. With Graham's three-field scheduling notation [75] the problem can be classified as $P|\pi_j \sim \mathcal{N}(\mu_j, \sigma_j^2)|\Pr[C_{\max} \leq \delta]$, and can be shown to be NP-hard by the reduction from 3-Partition problem. Let us note that for the case where $n \leq m$, i.e., the number of machines is larger or equal to the number of jobs to schedule, the solution is trivial since we can schedule one job per machine, potentially leaving some machine empty. Hence, without loss of generality from now on, we will assume that $n > m > 1$; therefore, each machine schedules at least one job.

Example An example of a problem instance considering 10 jobs is summarized in Table 5.1. Let $\delta = 84$ and the number of machines $m = 3$. Then, the optimal

J_j	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9	J_{10}
μ_j	32	12	2	20	19	35	23	21	26	41
σ_j^2	2	2	1	9	2	4	13	3	8	7

Table 5.1: Jobs parameters for an example problem instance.

solution is

$$S_1 = \{J_1, J_5, J_9\},\ S_2 = \{J_2, J_4, J_7, J_8\},\ S_3 = \{J_3, J_6, J_{10}\}$$

with the objective value being

$$\Phi\left(\frac{84-(32+19+26)}{\sqrt{2+2+8}}\right) \times \Phi\left(\frac{84-(12+20+23+21)}{\sqrt{2+9+13+3}}\right) \times \Phi\left(\frac{84-(2+35+41)}{\sqrt{1+4+7}}\right) = 0.8796.$$

A visualization of the solution shown in Figure5.1 illustrates the probability density function of completion times on each machine. Red areas represent the probability that the last job on the specific machine exceeds the common due date δ.

5.1.2 Non-linear model

The problem defined above can be described by a mixed-integer non-linear program (MINLP) [135] as follows. The model uses a binary assignment variables $s_{i,j}$ with the meaning

$$s_{i,j} = \begin{cases} 1 & \text{if job } J_j \text{ is scheduled on machine } M_i, \\ 0 & \text{otherwise.} \end{cases}$$

Subsequently, the whole model can be stated as

$$\max \prod_{i=1}^{m} \Phi\left(\frac{\delta - \mu_{M_i}}{\sqrt{\sigma^2_{M_i}}}\right) \tag{5.1.5}$$

$$\text{subject to:} \quad \sum_{j=1}^{n} s_{i,j} \cdot \mu_j = \mu_{M_i} \quad \forall i \in \{1, \ldots, m\}, \tag{5.1.6}$$

$$\sum_{j=1}^{n} s_{i,j} \cdot \sigma_j^2 = \sigma^2_{M_i} \quad \forall i \in \{1, \ldots, m\}, \tag{5.1.7}$$

$$\sum_{i=1}^{m} s_{i,j} = 1 \quad \forall j \in \{1, \ldots, n\}, \tag{5.1.8}$$

$$\text{where:} \quad s_{i,j} \in \{0, 1\} \quad \forall i \in \{1, \ldots, m\}, \forall j \in \{1, \ldots, n\} \tag{5.1.9}$$

$$\mu_{M_i}, \sigma^2_{M_i} \geq 0 \quad \forall i \in \{1, \ldots, m\}. \tag{5.1.10}$$

Constraints (5.1.6) and (5.1.7) compute the corresponding means and variances on each machine, while (5.1.8) represents the constraint that forces the scheduling of each job exactly once. An obstacle lies in expressing the objective (5.1.5) since the cumulative normal distribution Φ does not have an analytical form. Hence, it is necessary to approximate it by one of the known approximations [174]. Then, the problem can be solved by a non-linear mixed-integer programming solver with the model (5.1.5)–(5.1.10). However, as it will be shown later, even with some improvements to the above MINLP, it scales only about up to four machines and 15 jobs. Hence, further in the paper, we will develop much more efficient algorithms.

5.1.3 Related work

The methodology of solving problems on parallel machines with uncertain parameters most typically falls into one of the four main categories: (i) reactive scheduling, (ii) robust optimization (RO), (iii) stochastic programming (SP), and (iv) distributionally robust optimization (DRO).

Reactive scheduling assumes an online scheduling policy that reacts to the observed values of parameters during the execution (or breakdown) of the schedule [161, 142]. Simulation optimization [91] is an alternative approach that also does not assume any specific analytical form of uncertainty in advance. For example, in [145], discrete-event simulation is used inside a heuristic optimization routine to estimate uncertain inventory function at specific points of interest. Although these methods have obvious advantages for uncertainties that are hard to describe analytically, it is challenging to ensure their provably safe behavior or their ability to account for the worst-case scenarios.

RO paradigm assumes that unknown parameters belong to a given uncertainty set with the aim to optimize the worst-case utility function or ensure that constraints hold for all possible realization of parameters within uncertainty set [22]. Uncertainty sets often have a polyhedral [79] or conic representation [32] and, thus, are efficiently solvable in practice. A notable related study was done by [96] who devise a branch-and-bound method to solve the two-machine flow shop robust scheduling

problem with uncertain processing times. They assume that processing times of all jobs are drawn from an uncertainty set containing scenarios either with discrete processing time or continuous intervals. Another related problem, but with a different criterion and the form of uncertainty, was studied in [165]. The authors study a robust scheduling problem, minimizing the number of identical machines required to completing all the given jobs before a deadline. They solved the problem using a branch-and-price approach. The processing time uncertainty was defined using intervals and the stability radius, i.e., the number of jobs that can deviate from their nominal processing times is defined as a parameter. A notion of multiple levels of robustness was introduced by authors of [7], where they consider finite discrete processing time uncertainty set and construct schedules that comply with requirements of jobs, which may vary according to their criticality. An often-cited disadvantage of RO is that the solution does not utilize the full information about the distribution of parameters that might be obtained, e.g., from empirical data. Thus the price for robustness might be too costly, and in some settings, it is advantageous to optimize the problem in terms of expected outcomes rather than the worst-case scenarios.

SP treats problems by optimizing the expected utility function of the systems with respect to a known probabilistic distribution of parameters. In general, SP problems can also be treated as a single, two-stage [114] or multi-stage optimization problems [34] depending on how the uncertainty is unfolded. The problem is often solved by a translation into a tractable finite-sized mathematical program. For example, Skutella and Uetz [160] studied parallel machine scheduling problem with precedences and release times with the minimization of the expected weighted completion time. They provided several approximation results based on LP relaxations for various variants of the problems, but their applicability to our problem is not clear as they consider expectation criterion rather than the probability of completion.

Although the solutions obtained with SP show a good performance in the long term, they can suffer from sudden performance fluctuations. To combat this, the β-robust approach was coined by [53]. The goal is to create a schedule that optimizes a utility function with respect to target level β, e.g., maximize the probability that a certain threshold is met. In the case of parallel machine scheduling problems, the research focuses almost exclusively on the total flow time, as the β-robust objective. Single-machine variants were studied by, e.g., [53] and [107], who introduced sequence-dependent setup times to the problem. Alimoradi *et al.* [2] solve β-robust scheduling problem with parallel machines and the total flow time criterion. They described several problem properties and proposed a branch-and-bound method to solve the problem. As their criterion is simpler than ours, they were able to solve instances with up to 45 jobs and 5 machines in less than 20 minutes. A similar problem was addressed by [130]. The authors maximize the probability that the sum of job completion times will be lower than a given bound. The paper proposes a mixed-integer non-linear programming model, which is solved by a surrogate mixed-integer programming model; however, it does not guarantee to find the optimal solution to the problem.

Finally, the same problem as in this paper was studied in [135]. They solve problem $P|\pi_j \sim \mathcal{N}(\mu_j, \sigma_j^2)|\Pr[C_{\max} \leq \delta]$ which requires maximization of the joint

probability that the completion time of all machines is less than δ. The problem is more complex than previously studied total-flow time expectation models since the objective function does require the product operator and an additional non-linearity introduced by the cumulative normal distribution function. The authors devised two branch-and-bound algorithms that differ in their branching schemes and are capable of solving instances with about 20 jobs. The main limitation of this approach is the symmetries induced by the branching scheme. The authors mitigate this issue by a suitable solution encoding and by a dominance rule; however, the results indicate that the CPU time sharply grows with an increasing number of machines. We note that this problem is also related to the concept of chance constraints [42] when one requires that a single constraint is satisfied within some defined probability. However, as we consider multiple machines, we would require that constraints are not satisfied separately, but rather in a joint probabilistic sense. As it is incredibly complex to account for joint chance constraints, conservative over approximations such as Bonferroni bounds [183] are typically used to guarantee the satisfaction of constraints, but for the price of reducing solution space leading to sub-optimal results. Moreover, these approaches are designed for continuous problems and are used to handle feasibility rather than optimization. An another closely related problem is Extensible Bin Packing (EBP) [56, 103]. This variant of bin packing problem considers a fixed number of bins with a given capacity, which might be enlarged for a certain cost. The goal is to pack all items into the bins such that the total cost of packing is minimized. In stochastic setting of EBP problem, the sizes of items are can be subject to uncertainty or the capacity itself can be revealed after the choice of packing is commissioned [120]. Recently, Sagnol and Schmidt [143] have considered Stochastic EBP, where the objective to minimize the expected value of the sum of costs subject to uncertain item sizes. The family considered distributions is not particularly limited. They studied approximation algorithms and derived guarantees for LEPT (longest expected processing time) rule with tight ratio $1 + e^{-1} \approx 1.368$ and even stronger results for distributions that have bounded coefficient of variation. A branch-and-price approach for a similar problem was proposed in [186]. They considered a chance-constrained Stochastic EBP with the goal of minimizing the expected cost of the extension. The proposed branch-and-price approach was able to solve instances with three bins and about 28 items in about thousand seconds. Finally, we note that all above problems are also closely related to the scheduling with the total tardiness and common due dates [95].

Distributionally robust optimization was introduced by Scarf [148] back in 1958, but it started to receive attention recently. DRO aims to minimize the worst-case expectation with respect to the uncertainty of the underlying distribution of the parameters, i.e., the so-called ambiguity set. This new wave of interest in DRO was sparked by recent advancements of mathematical programming solvers and tractable formulations of ambiguity sets [30]. The problem is typically solved by a reformulation to a computationally tractable mathematical programming problem [158], whose complexity depends on the used formulation of the ambiguity set. In [179], the authors solve surgery blocks to operating rooms (ORs). Processing times of surgeries are subject to a probability contained in ambiguity sets defining bounds on mean values and mean absolute deviations. The reformulation leads

to a mixed-integer linear program of exponential size in the number of ORs. The approach was able to solve problems with about 15 surgery blocks within an hour. In [44], a distributionally robust variant of the identical parallel machine problem with the minimization of the worst-case expected total flow time is investigated. The processing times of jobs are subject to uncertainty belonging to the ambiguity set defined by the moment uncertainty. The problem is reduced to a second-order cone integer program and later solved by an exact algorithm which explores all solutions satisfying optimality conditions. The proposed approach was able to solve instances with 100 jobs and 5 machines within several seconds.

5.1.4 Contribution and outline

In this paper, we build on the work of Ranjbar *et al.* [135]. We apply integer programming methodology to solve problem $P|\pi_j \sim \mathcal{N}(\mu_j, \sigma_j^2)|\Pr[C_{\max} \leq \delta]$ treating it as a β-robust scheduling problem maximizing the probability that all jobs are completed within due date δ. We solve the problem with a branch-and-price approach, and we propose efficient heuristic algorithms. The efficiency of our method is based on the proof of so-called aggregated machines upper bound on the objective that can be used, e.g., for the computation of the minimum and the maximum number of jobs that each machine can schedule. The used solution methodology also reveals connections to static stochastic Knapsack problem [116, 115]. Furthermore, we use a constraint branching mechanism to mitigate symmetries in solution space, and we exploit the concavity result for the case of a problem with two machines to solve instances with up to 500 jobs.

The main contributions of this work are:

(i) a list scheduling and a genetic algorithm, which provide better lower bounds than the heuristics described in [135],

(ii) a new upper bound on the objective based on the concept of aggregated machines,

(iii) a branch-and-price decomposition of the problem with a branching mechanism which mitigates symmetries and pricing problem admitting a fast solution algorithm,

(iv) complexity proof of the pricing problem,

(v) an exact algorithm for the special case of two machines based on the concavity of the relaxed problem,

(vi) experimental results which show significant improvements over the algorithms published in [135].

This paper extends our previous conference paper [168] by describing a new initial heuristics, the proposition and proof of aggregated machines upper bound on the objective, and an algorithm for the case of the two machines utilizing the concavity of the relaxed problem. The paper also provides a detailed evaluation and analysis of the proposed approaches.

The outline of the paper is the following. In Section 5.2, we propose an upper bound on the objective function, and we describe a computational method for obtaining bounds for the number of jobs scheduled on each machine. Next, we present two heuristics to solve the problem. In Section 5.3, we introduce the branch-and-price algorithm. We show the complexity of the pricing problem, and we derive an efficient solution method for it. In Section 5.4, we solve a special case of the problem with two machines. Finally, we provide computational experiments, including results for an MINLP model in Section 5.5, and the paper is concluded by Section 5.6.

5.2 Problem bounds

The purpose of this section is to establish problem properties and bounds used thorough the rest of the paper. The outline of this section is the following. First, we propose two new heuristic algorithms that provide a feasible solution that acts as a lower bound. Next, we propose an upper bound on the objective, which is based on the notion of aggregated machines – a relaxation of the problem in terms of the number of machines and machine-specific due dates. Finally, based on the proposed upper bound, we derive bounds on the number of jobs that each machine has to schedule.

5.2.1 Lower bound heuristics

In work [135], the authors mention that their initial heuristic has a large optimality gap, and they argued towards the development of new and more efficient algorithms. Hence, in this section, we propose two heuristic scheduling algorithm.

The first algorithm we propose is a list scheduling algorithm analogical to the LPT rule (*longest processing time* first) for $P||C_{max}$ problem. It sorts jobs in non-increasing order of sums of means and variances $\mu_j + \sigma_j^2$ with ties broken by larger μ_j. In each step, a job is taken and assigned to the machine $i \in \{1, \ldots, m\}$ with the currently largest probability of completion before δ, denoted as $c^{(i)}$. We denote the objective value of a feasible solution with m machines with the common due date δ as $\mathrm{LB}((\delta)_{i=1,\ldots,m})$ and it defines the lower bound to the studied problem. The corresponding solution is defined in terms of job-machine assignment vectors $\boldsymbol{s}_i = (s_{i,1}, \ldots, s_{i,j}, \ldots, s_{i,n})$. We call this list scheduling algorithm as *Large Sum Allocated First* (LSAF) and is shown in Algorithm 5.1. The experiments on our test instances revealed that LSAF gives better results than the lower bound heuristics introduced in [135] and [168]. We have tried several other ways how to sort the list of jobs, e.g., product of means and variances or the *coefficient of variation* [160]. However, all of them have performed worse than LSAF. A similar algorithm for Stochastic Extensible Bin Packing (SEBP) problem was proposed in [143]. Their LEPT (longest expected processing time) first rule essentially resembles LSAF. Although LEPT was shown to be $1 + e^{-1} \approx 1.368$ approximation algorithm for SEBP, the question whether LSAF can have similar guarantees for our problem as well remains open.

A second algorithm is a genetic meta-heuristic implemented using Nevergrad [136] library. We denote it as DF-DOPO. The solution is represented with

Algorithm 5.1: LSAF list scheduling.

let $\mu_1 + \sigma_1^2 \geq \mu_2 + \sigma_2^2 \geq \ldots \geq \mu_n + \sigma_n^2$
$c^{(i)} \leftarrow 1$, $\boldsymbol{s}_i \leftarrow \mathbf{0} \quad \forall i \in \{1, \ldots, m\}$
for $j \leftarrow 1$ **to** n **do**
  $i^\star \leftarrow \arg\max_{i \in \{1,\ldots,m\}} c^{(i)}$
  $s_{i^\star,j} \leftarrow 1$
  $c^{(i)} \leftarrow \Phi\left(\frac{\delta - \boldsymbol{\mu}^\intercal \boldsymbol{s}_{i^\star}}{\sqrt{\boldsymbol{\sigma}^{2\intercal} \boldsymbol{s}_{i^\star}}}\right)$
return $LB((\delta)_{i=1,\ldots,m})$

a real vector of dimension n. Each entry corresponds to a job and its assignment on a machine, which is encoded using *ordered discrete* representations, i.e., as a real number on the interval $[0, 1]$, which is equidistantly divided into m segments. Each segment then corresponds to a specific machine. The algorithm is *Discrete One Plus One* genetic optimizer [150] with the heavy-tailed mutation operator [60] (i.e., *double fast genetic algorithm* – DF-DOPO). We observed that this variant performed the best among the other choices of representations and optimizers.

After DF-DOPO finishes, we further improve its solution by applying an additional local search which reoptimizes jobs assigned along all pair of machines. Essentially, every pair of machines and the set of jobs that is allocated to them resembles an instance of the problem with $m = 2$ machines. As it will be shown in Section 5.4, the problem with two machines can be solved up to the optimality very efficiently in practice. Thus, the intensification local search procedure reoptimizes all pairs of machines optimally. We denote the variant of DF-DOPO where we apply the two-machine intensification procedure twice as DF-DOPO+TM. A comparison of LSAF, DF-DOPO+TM, and the heuristic from [135] can be found in Section 5.5.2.

5.2.2 Upper bound on objective with aggregated machines

In this section, we propose a relaxation of the problem in terms of the number of machines and their due dates. We use the concept of so-called *aggregated machines*. The main idea is to replace the problem with m machines by a problem with $k + 1$ machines for some $m > k > 0$. In the resulting problem, k machines are preserved, and the last machine *aggregates* all remaining machines by increasing its machine-specified due date. For example, it will be shown that the optimal objective value for a problem with 3 machines with a common due date δ can be upper bounded by an optimal solution for two machines, with one machine having the due date set to 2δ. We show that the optimal solution of such a problem acts as an upper bound on the objective. The upper bound is stated by Proposition 13; however, first, we develop several useful inequalities before we prove the result.

Proposition 12. *Let $a, b \in \mathbb{R}$ and $c, d \in \mathbb{R}_{>0}$. Then*

$$\Phi\left(\frac{a}{\sqrt{c}}\right) \times \Phi\left(\frac{b}{\sqrt{d}}\right) \leq \Phi\left(\frac{a+b}{\sqrt{c+d}}\right).$$

Proof. Pick $a, b \in \mathbb{R}$ and $c, d > 0$ and denote $\frac{a}{\sqrt{c}} = u$, $\frac{b}{\sqrt{d}} = v$ and $\alpha = \frac{\sqrt{c}}{\sqrt{c+d}}$, $\beta = \frac{\sqrt{d}}{\sqrt{c+d}}$. As $\frac{a+b}{\sqrt{c+d}} = \alpha u + \beta v$, it suffices to prove that

$$\Phi(u) \times \Phi(v) \leq \Phi(\alpha u + \beta v).$$

Let X, Y be independent random variables with $X \sim \mathcal{N}(0,1)$ and $Y \sim \mathcal{N}(0,1)$. We have that

$$\Phi(u) \times \Phi(v) = \Pr[X \leq u] \times \Pr[Y \leq v] = \int_{-\infty}^{u} \int_{-\infty}^{v} f_{X,Y}(x,y)\,\mathrm{d}x\mathrm{d}y,$$

where $f_{X,Y}$ is the joint probability density function of (X, Y). Note that $\alpha^2+\beta^2 = 1$, which means $\|(\alpha, \beta)\| = 1$ and therefore $(\alpha u+\beta v)\cdot(\alpha, \beta)$ is the orthogonal projection of the vector (u, v) on to the linear subspace generated by (α, β). Indeed, we have

$$(u, v)^\intercal(\alpha, \beta) \cdot \frac{(\alpha, \beta)}{\|(\alpha, \beta)\|} = (\alpha u + \beta v) \cdot (\alpha, \beta).$$

Apart from that, note that $(-\infty, u]\times(-\infty, v] \subseteq \{(x, y) |\, x, y \in \mathbb{R},\ \alpha x+\beta y \leq \alpha u+\beta v\}$ (see Figure 5.2 for an illustration). From that and from the non-negativity of $f_{X,Y}$

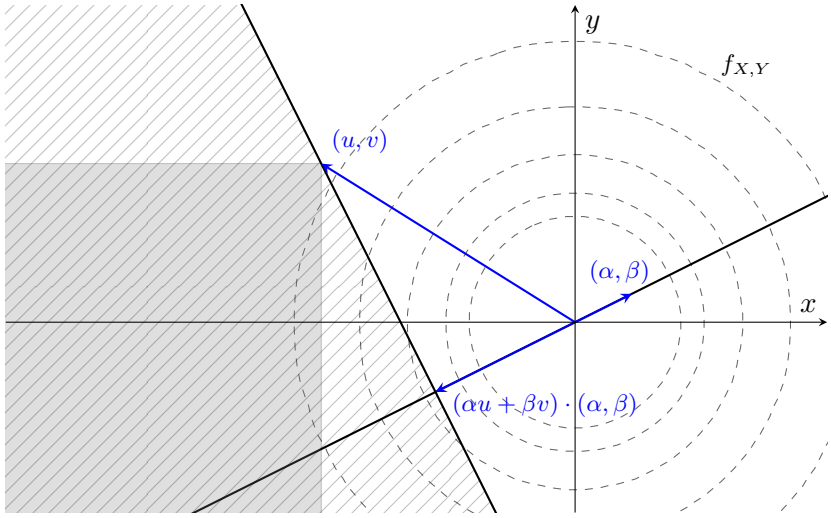


Figure 5.2: Geometric illustration of the integration bounds, the projection of (u, v) and contours of $f_{X,Y}$.

follows

$$\int_{-\infty}^{u} \int_{-\infty}^{v} f_{X,Y}(x,y)\,\mathrm{d}x\mathrm{d}y \leq \iint\limits_{\substack{(x,y)\in\mathbb{R}^2 \\ \alpha x+\beta y\leq \alpha u+\beta v}} f_{X,Y}(x,y)\,\mathrm{d}x\mathrm{d}y.$$

The bounds of the last integral can be simplified by rotation of the coordinate system around the origin by the angle between vectors $(1, 0)$ and (α, β). The corresponding

substitution of variables is $x = \alpha t - \beta k$ and $y = \beta t + \alpha k$, applying which we get

$$\iint\limits_{\substack{(x,y)\in\mathbb{R}^2 \\ \alpha x+\beta u\le\alpha u+\beta v}} f_{X,Y}(x,y)\,\mathrm{d}x\mathrm{d}y = \int_{-\infty}^{+\infty}\int_{-\infty}^{\alpha u+\beta v} f_{X,Y}(\alpha t-\beta k,\beta t+\alpha k)\det\begin{bmatrix}\alpha & -\beta\\ \beta & \alpha\end{bmatrix}\mathrm{d}t\mathrm{d}k.$$

As the value of $f_{X,Y}$ is invariant to rotation around the origin, the last integral equals

$$\int_{-\infty}^{\alpha u+\beta v}\int_{-\infty}^{+\infty} f_{X,Y}(t,k)\,\mathrm{d}k\mathrm{d}t = \Phi(\alpha u+\beta v),$$

which completes the proof. □

The following lemma shows that Proposition 12 can be generalized to the case with the product of m terms:

Lemma 7. *Let $m \ge 1$, $\delta \in \mathbb{N}_0$, $\boldsymbol{\mu}, \boldsymbol{\sigma^2} \in \mathbb{N}^n$ and $\forall i \in \{1,\ldots,m\} : \boldsymbol{s}_i \in \{0,1\}^n$, $\boldsymbol{s}_i \neq \mathbf{0}$ then*

$$\Phi\left(\frac{\delta m - \boldsymbol{\mu}^\intercal(\boldsymbol{s}_1+\ldots+\boldsymbol{s}_m)}{\sqrt{\boldsymbol{\sigma^2}^\intercal(\boldsymbol{s}_1+\ldots+\boldsymbol{s}_m)}}\right) \ge \prod_{i=1}^{m}\Phi\left(\frac{\delta-\boldsymbol{\mu}^\intercal\boldsymbol{s}_i}{\sqrt{\boldsymbol{\sigma^2}^\intercal\boldsymbol{s}_i}}\right). \tag{5.2.1}$$

Proof. By induction over m. The base case $m = 1$ can be verified by routine calculations. Let us denote the statement (5.2.1) as $V(m)$, i.e., induction hypothesis (I.H.). Now we prove that $\forall m \ge 1 : V(m) \implies V(m+1)$. By expanding the right hand side of $V(m+1)$ we have that

$$\begin{aligned}\prod_{i=1}^{m+1}\Phi\left(\frac{\delta-\boldsymbol{\mu}^\intercal\boldsymbol{s}_i}{\sqrt{\boldsymbol{\sigma^2}^\intercal\boldsymbol{s}_i}}\right) &= \prod_{i=1}^{m}\Phi\left(\frac{\delta-\boldsymbol{\mu}^\intercal\boldsymbol{s}_i}{\sqrt{\boldsymbol{\sigma^2}^\intercal\boldsymbol{s}_i}}\right)\times\Phi\left(\frac{\delta-\boldsymbol{\mu}^\intercal\boldsymbol{s}_{m+1}}{\sqrt{\boldsymbol{\sigma^2}^\intercal\boldsymbol{s}_{m+1}}}\right) \overset{\text{I.H.}}{\le}\\ &\le \Phi\left(\frac{\delta m-\boldsymbol{\mu}^\intercal(\boldsymbol{s}_1+\ldots+\boldsymbol{s}_m)}{\sqrt{\boldsymbol{\sigma^2}^\intercal(\boldsymbol{s}_1+\ldots+\boldsymbol{s}_m)}}\right)\times\Phi\left(\frac{\delta-\boldsymbol{\mu}^\intercal\boldsymbol{s}_{m+1}}{\sqrt{\boldsymbol{\sigma^2}^\intercal\boldsymbol{s}_{m+1}}}\right) \overset{\text{Prop. 12}}{\le}\\ &\le \Phi\left(\frac{\delta(m+1)-\boldsymbol{\mu}^\intercal(\boldsymbol{s}_1+\ldots+\boldsymbol{s}_{m+1})}{\sqrt{\boldsymbol{\sigma^2}^\intercal(\boldsymbol{s}_1+\ldots+\boldsymbol{s}_{m+1})}}\right).\end{aligned}$$

□

Now, we will state an upper bound on the objective for the problem with m machines in terms of the optimal solution of a problem with $k+1 < m$ machines. Let us denote an optimal objective value of the problem with m machines and machine-specified due dates $\delta_1, \delta_2, \ldots, \delta_m$ as $\mathrm{OPT}(\delta_1,\delta_2,\ldots,\delta_m)$. Then, we have the following bound:

Proposition 13. *Let $m > 1$ be the number of machines and $k \in \mathbb{N}$, $m > k > 0$. Then,*

$$OPT((\delta)_{i=1,\ldots,k}, \delta\cdot(m-k)) \ge OPT((\delta)_{i=1,\ldots,m}).$$

Proof. Let us denote characteristic vectors of optimal job-machine assignments for the problem with m machines as $\boldsymbol{s}_1^\star, \ldots, \boldsymbol{s}_m^\star$. We have that

$$\mathrm{OPT}((\delta)_{i=1,\ldots,m}) = \Phi\left(\frac{\delta - \boldsymbol{\mu}^\intercal \boldsymbol{s}_1^\star}{\sqrt{\boldsymbol{\sigma}^{2\intercal} \boldsymbol{s}_1^\star}}\right) \times \cdots \times \Phi\left(\frac{\delta - \boldsymbol{\mu}^\intercal \boldsymbol{s}_m^\star}{\sqrt{\boldsymbol{\sigma}^{2\intercal} \boldsymbol{s}_m^\star}}\right) \overset{\text{Lemma 7}}{\leq}$$

$$\leq \prod_{i=1}^{k} \Phi\left(\frac{\delta - \boldsymbol{\mu}^\intercal \boldsymbol{s}_i^\star}{\sqrt{\boldsymbol{\sigma}^{2\intercal} \boldsymbol{s}_i^\star}}\right) \times \Phi\left(\frac{\delta(m-k) - \boldsymbol{\mu}^\intercal (\boldsymbol{s}_{k+1}^\star + \ldots + \boldsymbol{s}_m^\star)}{\sqrt{\boldsymbol{\sigma}^{2\intercal} (\boldsymbol{s}_{k+1}^\star + \ldots + \boldsymbol{s}_m^\star)}}\right) =$$

$$= \mathrm{LB}((\delta)_{i=1,\ldots,k}, \delta \cdot (m-k)) \leq \mathrm{OPT}((\delta)_{i=1,\ldots,k}, \delta \cdot (m-k)).$$

The first inequality follows from Lemma 7; the equality from the fact that the term represents the objective value of a feasible solution for $k+1$ machines with due dates $\delta_1 = \ldots = \delta_k = \delta$ and $\delta_{k+1} = \delta(m-k)$ defined by job-machine assignments $\boldsymbol{s}_1^\star, \ldots, \boldsymbol{s}_k^\star$ and $\boldsymbol{s}_{k+1}^\star + \ldots + \boldsymbol{s}_m^\star$. The last inequality follows from the fact that the optimal solution for this problem is at least as good as any feasible one. □

Please note that computing just $\mathrm{OPT}(\delta, \delta \cdot (m-1))$ is already an NP-hard problem for $m = 2$ by the reduction from 2-Partition problem. However, solving it gives a non-trivial upper bound on the objective of the problem with m machines. The practicality of the proposed bound follows from the fact that the problem with two machines can be solved efficiently in practice even for a large number of jobs, which will be demonstrated in Section 5.4.

In the next section, we propose an algorithm for computing the minimum and the maximum number of jobs that any machine can schedule in an optimal solution. The approach described below utilizes the upper bound $\mathrm{OPT}(\delta, \delta \cdot (m-1))$, which can be parameterized by requiring that the first machine schedules some given number of jobs.

5.2.3 Bounds on the number of jobs on a machine

In work [135], the authors introduce a lower bound on the number of jobs that each machine has to process, denoted as x_{min}, i.e., the number of jobs that has to be scheduled on each machine. Here, we show a new approach for computing x_{min}. Furthermore, we also introduce a new bound x_{max} on the maximal number of jobs to be processed on any machine. Our approach presented here is based on the upper bound on the objective function introduced in Section 5.2.2. The core idea is to utilize the upper bound stated by Proposition 13 with the case $k = 1$. For this case, the algorithm computing $\mathrm{OPT}(\delta, \delta \cdot (m-1))$ can be implemented in such a way that we can specify the number of jobs to be scheduled on the first (non-aggregated) machine. The details of this implementation are given in Section 5.4.3.

Let us define $\mathrm{OPT}_q(\delta, \delta \cdot (m-1))$ as the optimal objective value for the problem with $\delta_1 = \delta$, $\delta_2 = \delta \cdot (m-1)$ with the first machine having exactly q jobs assigned. It can be shown (likewise in Proposition 13) that $\mathrm{OPT}_q(\delta, \delta \cdot (m-1))$ acts as an upper bound on $\mathrm{OPT}_q(\delta_1, \delta_2, \ldots, \delta_m)$ with $\delta_1 = \delta_2 = \ldots = \delta_m = \delta$. The idea is to find a lower bound x_{min} and an upper bound x_{max} on value q such that the value of the upper bound parametrized by q does not contradict a provably achievable objective $\mathrm{LB}((\delta)_{i=1,\ldots,m})$. See the pseudocode in Algorithm 5.2.

Algorithm 5.2: Computing x_{min} and x_{max}.

$x_{min} \leftarrow 1$, $x_{max} \leftarrow n - m + 1$
compute $\mathrm{LB}((\delta)_{i=1,\dots,m})$
while $LB((\delta)_{i=1,\dots,m}) > OPT_{x_{min}}(\delta, \delta \cdot (m-1))$ **do** `// See Section 5.4.3.`
  $x_{min} \leftarrow x_{min} + 1$
while $LB((\delta)_{i=1,\dots,m}) > OPT_{x_{max}}(\delta, \delta \cdot (m-1))$ **do**
  $x_{max} \leftarrow x_{max} - 1$
return x_{min}, x_{max}

Example For the example instance in Table 5.1, assume that a heuristic algorithm finds a lower bound $\mathrm{LB}(\delta, \delta, \delta) = 0.8796$. Then, it can be computed that $x_{min} \geq 2$ and $x_{max} \leq 5$ with the computation taking less than 250 ms on commodity hardware.

5.3 Branch-and-price algorithm

In this section, we propose a branch-and-price algorithm [11] to solve the problem. In general, the branch-and-price algorithm consists of two support models: a master problem and a pricing problem. The master problem is a reformulation of the original problem where the job-machine assignment variables are replaced by indicator variables whether some subset of jobs is scheduled on some machine. We refer to such sets of jobs as to *patterns*. The model selects m patterns such that they maximize the objective function given the constraint that each job is scheduled. Since there is an exponential number of all possible patterns, the linear relaxation of the formulation is solved lazily via *column generation* [58], and an integer solution is ensured by a branching mechanism.

In the column generation procedure, the current values of dual variables of the master problem are passed on to the pricing problem. The pricing problem acts as a separation problem for the dual formulation. When its optimal solution has a so-called *negative reduced cost* (i.e., the objective value), then a new pattern is added to the master problem. The existing set of patterns is updated, and the master problem is resolved, yielding new values of dual variables. The procedure is repeated, while a pattern with a negative reduced cost can be found.

The master problem relaxes the pattern indicator variables and may not provide an integer solution. In that case, we need to apply a branching scheme that creates two copies of the original problem, one with the imposed constraint that some two jobs must be scheduled together and the other with the constraint that some two jobs cannot be scheduled together. When one of them does not yield an integer solution, the branching is performed on that problem again, which creates a branching tree. Once all the leaves of the branching tree are closed, we take the best solution, which represents the optimal solution to the given instance.

The main advantages of the branch-and-price reformulation are the more efficient mitigation of symmetries between identical machines compared to the branch-and-bound from [135]. What is more, it was recently documented by [186] that extensible bin packing type-of problems with stochastic items display structure which favours

to branch-and-price solution methods. Especially, the employed decomposition allows us to displace the non-linearity presented in the objective to the pricing problem that can be solved by a dedicated algorithm.

In the following sections, we describe each part of the algorithm in details.

5.3.1 Master problem

The master problem lifts up the original variable space and works with indicator variables of all possible sets of jobs. The solution selects m of such sets so that the objective function is maximized subject to the constraint that every job is scheduled. We represent each set of jobs as a characteristic vector $\boldsymbol{p}^{(k)} = \{0,1\}^n$. We refer to such a vector as a *pattern*. Let us define the value of $p_j^{(k)}$ as

$$p_j^{(k)} = \begin{cases} 1 & \text{if job } J_j \text{ is contained in pattern } \boldsymbol{p}^{(k)} \\ 0 & \text{otherwise.} \end{cases} \tag{5.3.1}$$

We apply the log transformation to the original objective function to transform the product of probabilities into their sum. Hence, the cost of the k-th pattern, denoted as $\log\left(c^{(k)}\right)$, is calculated as the logarithm of the probability that jobs scheduled in the pattern are completed before δ.

5.3.1.0.1 Example Consider an instance with 4 jobs given by deadline $\delta = 35$ and $\boldsymbol{\mu} = (12, 18, 7, 13)$ and $\boldsymbol{\sigma^2} = (7, 2, 3, 5)$. One of the possible patterns is $\boldsymbol{p}^{(0)} = (1, 1, 0, 0)$, which indicates that jobs J_1 and J_2 are scheduled together on some machine. The cost $\log\left(c^{(0)}\right)$ of pattern $\boldsymbol{p}^{(0)}$ is then

$$\log\left(c^{(0)}\right) = \log\Phi\left(\frac{\delta - \boldsymbol{\mu}^\intercal \boldsymbol{p}^{(0)}}{\sqrt{\boldsymbol{\sigma^2}^\intercal \boldsymbol{p}^{(0)}}}\right) = \log\Phi\left(\frac{35 - 12 - 18}{\sqrt{7+2}}\right) = \log(0.95).$$

Let us denote the set of all patterns by P. The complete representation of P would be costly as it contains 2^n patterns. Hence, the column generation starts with an initial subset of patterns $P' \subseteq P$ and generates more patterns lazily as needed. To prove the optimality of a solution, the algorithm often does not need to generate the whole set P since only a small fraction of P is typically needed. The master problem is given as

$$\max \sum_{k=1}^{|P'|} y_k \cdot \log\left(c^{(k)}\right) \tag{5.3.2}$$

$$\text{subject to:} \quad \sum_{k=1}^{|P'|} y_k \cdot p_j^{(k)} \geq 1 \quad \forall j \in \{1, \ldots, n\}, \tag{5.3.3}$$

$$\sum_{k=1}^{|P'|} y_k \leq m \tag{5.3.4}$$

$$\text{where:} \quad y_k \in \mathbb{R}_0^+ \quad \forall k \in \{1, \ldots, |P'|\} \tag{5.3.5}$$

The y_k is a decision variable that indicates if we choose the k-th pattern to be selected or not. The constraint (5.3.4) limits the number of used resources. Note that the master problem is the linear relaxation of its integer counterpart where the complete set of patterns P was replaced by a subset $P' \subseteq P$. Since it is a linear program, we can derive its dual formulation

$$\min \sum_{j=1}^{n} \psi_j + m \cdot \gamma \tag{5.3.6}$$

$$\text{subject to:} \quad \sum_{j=1}^{n} \psi_j \cdot p_j^{(k)} + \gamma \geq \log\left(c^{(k)}\right) \quad \forall k \in \{1, \ldots, |P'|\}, \tag{5.3.7}$$

$$\text{where:} \quad \gamma \in \mathbb{R}_0^+, \tag{5.3.8}$$

$$\psi_j \in \mathbb{R}_0^- \quad \forall j \in \{1, \ldots, n\}. \tag{5.3.9}$$

From the theory of strong duality, it can be shown that patterns that can improve the current objective of the master problem are those violating the dual constraint (5.3.7).

5.3.2 Pricing problem

In this section, we derive a pricing problem that suggests which pattern to generate. We derive the objective function from the constraint (5.3.7) of the dual master problem. We look for a pattern $\boldsymbol{p}^{(k)}$ that violates constraint (5.3.7) as much as possible. Let us introduce a decision variable $x_j \in \{0, 1\}$, which indicates if job J_j is scheduled in the new pattern. Note that in this case, we consider only a single machine and the variables x_j are represented with a single vector $\boldsymbol{x}$, so the goal is

$$\min_{\boldsymbol{x} \in \{0,1\}^n} -\boldsymbol{\psi}^\intercal \boldsymbol{x} + \gamma - \log \Phi\left(\frac{\delta - \boldsymbol{\mu}^\intercal \boldsymbol{x}}{\sqrt{\boldsymbol{\sigma}^{2\intercal} \boldsymbol{x}}}\right), \tag{5.3.10}$$

where we substituted the definition of c_k by its definition, replaced $p_j^{(k)}$ coefficients with the x_j assignment variables and let $\boldsymbol{\psi} = (-\psi_1, -\psi_2, \ldots, -\psi_n) \in \mathbb{R}_{\geq 0}^n$. The dual prices $\boldsymbol{\psi}$ for constraints (5.3.3) express the need to include the given job in the new pattern. Since the dual price γ for constraint (5.3.4) acts as a constant in the pricing problem, we can omit γ from the objective and write the pricing problem as

$$\boldsymbol{x}^\star = \arg \max_{\boldsymbol{x} \in \{0,1\}^n} \log \Phi\left(\frac{\delta - \boldsymbol{\mu}^\intercal \boldsymbol{x}}{\sqrt{\boldsymbol{\sigma}^{2\intercal} \boldsymbol{x}}}\right) + \boldsymbol{\psi}^\intercal \boldsymbol{x}. \tag{5.3.11}$$

We note that without loss of generality, we may assume that $\boldsymbol{\sigma}^{2\intercal} \boldsymbol{x} \geq 1$ as in any optimal solution at least one job is allocated on each machine, and variances are non-negative integers. What is typical in branch-and-price algorithms is that the space of feasible patterns $\boldsymbol{x} \in \{0, 1\}^n$ will become progressively constrained as the solution progresses. This is due to the fact that the sequence of pricing problems solves only a mere continuous relaxation of the master problem rather than its integer counterpart, which needs to be obtained by branching decisions. What will be explained in Section 5.3.4, is the branching mechanism introduces so-called

conflict pairs in form of $x_i + x_j \leq 1$ between individual jobs J_i, J_j, forming edges $B_{\leq}$ of the conflict graph.

Let us discuss what the pricing problem intuitively does. It can be seen that the pricing problem balances the gain $\psi_j \geq 0$ from taking job $J_j \in J$ into the new pattern and the loss (non-linearly) proportional to μ_j and σ_j^2 while admitting imposed conflict pairs $B_{\leq}$. Hence, it acts as a kind of Knapsack Problem [90] with a non-linear capacity (soft) constraint and conflict pairs [74, 128]. This connection has inspired us to prove the complexity of the pricing problem by the reduction from Knapsack Problem with a conflict graph:

Definition (Knapsack Problem with Conflict Graph)**.** *The instance of the problem and the solution is given as follows:*

Input: $S = \{1, \ldots, n\}$, $E \subseteq S \times S$, $\boldsymbol{v} = (v_1, \ldots, v_n) \in \mathbb{N}^n$, $\boldsymbol{w} = (w_1, \ldots, w_n) \in \mathbb{N}^n$ *and* $C, k \in \mathbb{N}$.

Output: *Is there* $S' \subseteq S$ *such that* $\sum_{i \in S'} v_i \geq k$, $\sum_{i \in S'} w_i \leq C$ *and* $\forall i, j \in S' : (i, j) \notin E$*?*

Proposition 14. *Pricing problem is strongly NP-hard.*

Proof. See [8]. □

5.3.3 Pricing algorithm

To find an exact solution to the pricing problem, we adopt a trick used for solving stochastic knapsack problems [116]. For ease of exposition, let us assume that the current set of conflict pairs $B_{\leq} = \emptyset$. We observe that solving (5.3.11) is equivalent to the following

$$\max_{v \in V} \max_{\boldsymbol{x} \in \{0,1\}^n} \log \Phi \left(\frac{\delta - \boldsymbol{\mu}^\intercal \boldsymbol{x}}{\sqrt{v}} \right) + \boldsymbol{\psi}^\intercal \boldsymbol{x} \tag{5.3.12}$$

subject to

$$v = \boldsymbol{\sigma^2}^\intercal \boldsymbol{x} \tag{5.3.13}$$

$$x_{max} \geq \boldsymbol{x}^\intercal \mathbf{1} \geq x_{min} \tag{5.3.14}$$

where $V = \{\underline{v}, \underline{v} + 1, \ldots, \overline{v} - 1, \overline{v}\}$ such that $\underline{v}$ and $\overline{v}$ is a lower and upper bound on $\boldsymbol{\sigma^2}^\intercal \boldsymbol{x}$, respectively. The advantage of this reformulation is that for any fixed v, the expressions $\delta/\sqrt{v}$ and $\boldsymbol{\mu}/\sqrt{v}$ become constants. Hence, for every $v \in V$, we optimize

$$\max_{\boldsymbol{x} \in Q(v)} \log \Phi \left(\frac{\delta}{\sqrt{v}} - \frac{1}{\sqrt{v}} \boldsymbol{\mu}^\intercal \boldsymbol{x} \right) + \boldsymbol{\psi}^\intercal \boldsymbol{x}$$

where $Q(v) = \{\boldsymbol{x} \in \{0,1\}^n \mid \boldsymbol{\sigma^2}^\intercal \boldsymbol{x} = v \wedge \boldsymbol{x}^\intercal \mathbf{1} \in [x_{min}, x_{max}]\}$. It can be formulated as a MIP where $\log \Phi(\cdot)$ is given by a piece-wise linear function. The bounds $\underline{v}$ and $\overline{v}$ can be set as $\underline{v} = \sum_{k=1}^{x_{min}} \overline{\sigma}_k^2$ and $\overline{v} = \sum_{k=n-x_{max}+1}^{n} \overline{\sigma}_k^2$, where $\overline{\sigma}_k^2$ is the k-th smallest element of $\boldsymbol{\sigma^2}$. The formulation can be solved by a specialized simplex method for piece-wise linear functions implemented in Gurobi solver [**gurobi**]. When the set of conflict pairs $B_{\leq}$ is not empty, then $Q(v)$ is simply intersected by $\{\boldsymbol{x} \in \{0,1\}^n \mid \forall \{x_i, x_j\} \in B_{\leq} : x_i + x_j \leq 1\}$.

The objective value (5.3.11) of an optimal solution $\boldsymbol{x}^\star$ is compared with γ. When the objective value of (5.3.11) is greater than γ, the pattern is added to the master problem, which is consequently resolved. When it is less or equal to γ, the master problem is solved optimally. In practice, we solve the pricing problem as follows. First, we solve the model (5.3.15)–(5.3.28), which is a MIP formulation of the pricing problem based on a non-trivial linearization of the objective function.

For ease of exposition, let us assume that the set of conflict pairs $B_{\leq} = \emptyset$. It uses decision variables $\boldsymbol{x}$ to decide which jobs are considered in the new pattern. The division of the two variables is eliminated with *Charnes-Cooper transformation* [48] used in linear-fractional programming. It introduces a new continuous variable t with the meaning $t = 1/\sqrt{\boldsymbol{\sigma}^{2\intercal}\boldsymbol{x}}$. The boundary points of the domain of t are given by the square root of the lower and upper bounds on the minimum and maximum total variance on any machine, e.g., $t_{min}^{-1} = \sqrt{\overline{v}}$ and $t_{max}^{-1} = \sqrt{\underline{v}}$.

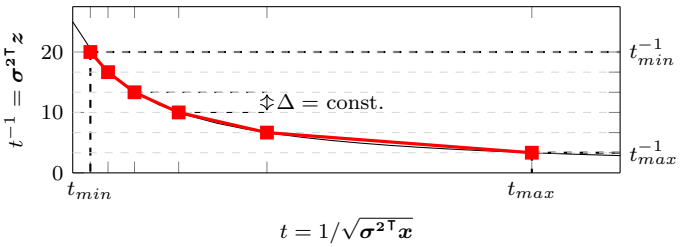


Figure 5.3: Approximation with $L = 6$.

The formulation uses an efficient *special order set of type 2* constraint to model the non-linear relation between $\boldsymbol{\sigma}^{2\intercal}\boldsymbol{z}$ and t. The relation is approximated with a set of points $\{t^{(l)}, y^{(l)}\}_{l=1}^{L}$, such that $y^{(l)} = 1/t^{(l)}$. We choose points such that $y^{(l)}$ are equidistantly distributed on the interval $[t_{max}^{-1}, t_{min}^{-1}]$. See a visualization in Figure 5.3. The constraint $\boldsymbol{\omega} \in \text{SOS2}$ ensures that at most two consecutive elements of $\boldsymbol{\omega}$ have a non-zero value. Moreover, the actual implementation uses indicator constraints rather than the big $\mathbf{M}$ presented in (5.3.17)–(5.3.18). Finally, the term $\log \Phi(\cdot)$ in the objective is approximated using Gurobi's piece-wise linear functions on the interval $\left[\Phi^{-1}(\text{LB}(\delta,\ldots,\delta)), \frac{\delta-\mu}{\sqrt{v^\star}}\right]$ equidistantly in the function range by K points. The quantities involved in the expression are: $\Phi^{-1}(\cdot)$ is the quantile function of the cumulative normalized normal distribution, $\underline{\mu}$ is a lower bound on the total mean for any machine, and $v^\star$ equals to $\underline{v}$ if $\delta - \underline{\mu} \geq 0$ and $\overline{v}$ otherwise. In the experiments, we have used values $L = \lfloor 45 + 30 \cdot \max\{\frac{n}{14} - 1, 0\} \rfloor$ and $K = \lfloor 60 + 30 \cdot \max\{\frac{n}{14} - 1, 0\} \rfloor$.

$$\max \log \Phi\left(\delta \cdot t - \boldsymbol{\mu}^{\intercal}\boldsymbol{z}\right) + \boldsymbol{\psi}^{\intercal}\boldsymbol{x} \tag{5.3.15}$$

$$\text{subject to: } \frac{\delta - \mu}{\sqrt{v^\star}} \geq \delta \cdot t - \boldsymbol{\mu}^{\intercal}\boldsymbol{z} \geq \Phi^{-1}(\text{LB}(\delta,\ldots,\delta)) \tag{5.3.16}$$

$$t \cdot \mathbf{1} - (\mathbf{1} - \boldsymbol{x}) \cdot \mathbf{M} \leq \boldsymbol{z} \leq t \cdot \mathbf{1} + (\mathbf{1} - \boldsymbol{x}) \cdot \mathbf{M} \tag{5.3.17}$$

$$\boldsymbol{z} \leq \boldsymbol{x} \cdot \mathbf{M} \tag{5.3.18}$$

$$\boldsymbol{z}^{\intercal}\mathbf{1} \geq t \tag{5.3.19}$$

$$x_{max} \geq \boldsymbol{x}^{\intercal}\mathbf{1} \geq x_{min} \tag{5.3.20}$$

$$t = \sum_{l=1}^{L} \omega_l \cdot t^{(l)} \tag{5.3.21}$$

$$\boldsymbol{\sigma}^{2\intercal}\boldsymbol{z} = \sum_{l=1}^{L} \omega_l \cdot y^{(l)} \tag{5.3.22}$$

$$\boldsymbol{\omega}^{\intercal}\mathbf{1} = 1 \tag{5.3.23}$$

$$\boldsymbol{\omega} \in \mathrm{SOS2} \tag{5.3.24}$$

where: $$t \in [t_{min}, t_{max}]\,, \tag{5.3.25}$$

$$\boldsymbol{z} \in [0, t_{max}]^n \tag{5.3.26}$$

$$\boldsymbol{\omega} \in \mathbb{R}^L_{\geq 0} \tag{5.3.27}$$

$$\boldsymbol{x} \in \{0,1\}^n \tag{5.3.28}$$

The model is solved with Gurobi solver utilizing the solution pool, which is the solver's feature that allows to obtain several solutions from a single run of optimization model with almost no additional cost. Moreover, we use model (5.3.15)–(5.3.28) with the imposed time limit, as we do not need to spend time with proving optimality. Hence, it serves mainly as a heuristics that provides several (potentially suboptimal) patterns at once and occasionally might fail to find a pattern with a negative reduced cost. Hence, in these cases, we additionally solve the exact formulation (5.3.12)–(5.3.14) afterward to verify that there is indeed no such pattern or insert it into the master problem, if there is one.

Remark 1 Let us describe an efficient method of how to significantly reduce the cardinality of V using the aggregated machines upper bound on the objective. This is especially important, as the running time of the pricing algorithm depedends on the cardinality of V, since it solve one subproblem for each fixed $v \in V$. It first computes values of $\mathrm{OPT}(\delta, \delta \cdot (m-1))$ for all possible fixed values of the total variance on the first machine. If the value of the upper bound with the fixed total variance is smaller than a lower bound on the objective obtained by some heuristic solution, then any pattern with such total variance cannot be included in an optimal integer solution of the original problem; hence, it does not need to be considered by the pricing algorithm. See Figure 5.4 for the numerical demonstration of the variance set reduction related to the example from Table 5.1. There, you see the values of the parametric upper bound for different values of variance on the first machine together with a heuristic objective value. The values of the total variance v below the heuristically computed lower bound $\mathrm{LP}(\delta, \delta, \delta)$ will not be considered by the pricing algorithm. For this example, set V can be reduced to just $|V| = 23$ with $\underline{v} = 7$ and $\overline{v} = 31$ wheres the choice of bounds described above in the definition of $Q(v)$ would lead to larger size $|V| = 38$ with $\underline{v} = 3$ and $\overline{v} = 41$.

Remark 2 Note that all information about the probability distribution related to the selected jobs on some machine is contained solely in the pricing problem. Hence, it can be directly extended to handle other distributions closed under convolution, i.e., the distribution of their sum is of the same family as the individual independent

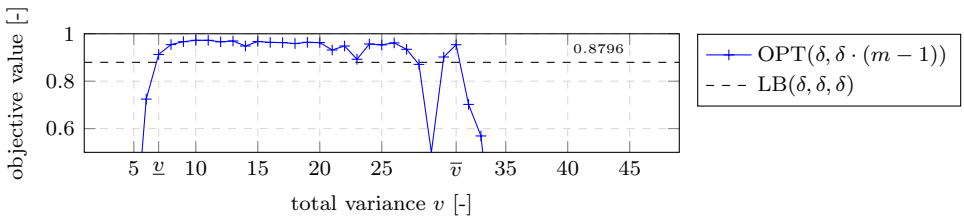


Figure 5.4: Reduction of variance set V using the upper bound on the objective with aggregated machines.

random variables. Some of these distributions include Cauchy(a_j, γ_j), Gamma(α_j, β) with a fixed rate β or Chi-squared distribution $\chi^2(r_j)$.

In the next section, we describe a branching procedure, its impact on the pricing problem, and the rest of the branch-and-price algorithm.

5.3.4 Branching scheme

The branching occurs when the master problem is solved optimally, but its solution is fractional. Hence, to find an integer solution, a branching needs to be applied. Here, we use Ryan-Forster's branching scheme [141] to eliminate symmetries between identical machines. The branching creates a branching tree, where each node represents a single master problem with constraints on patterns given by the path from the root node to the current node. In each node, a new constraint is introduced on a selected pair of jobs. It enforces decision that the two jobs i and j have to be scheduled together on some machine (i.e., $\exists k \in P' : p_i^{(k)} = p_j^{(k)} = 1$) or that two jobs cannot be scheduled together on any machine (i.e., $\forall k \in P' : p_i^{(k)} \neq p_j^{(k)}$). The constraints must be respected by the pricing problem when generating a new pattern, but also the set P' assumed in a particular master problem cannot be in a contradiction with them.

Let us denote B as a set of all pairs of jobs with some constraint imposed on them in the given branching node. Let us denote all pairs of jobs that cannot be scheduled together as $B_{\leq} \subseteq B$ and the jobs that are scheduled together as $B_{=} \subseteq B$. All constraints B have to be respected by the pricing problem when generating a new pattern. We demonstrate the meaning of these sets in the example of a branching tree in Figure 5.5. Constraints of type $x_i = x_j$ indicates that the jobs J_i and J_j have to be scheduled together on the same machine or none of them is scheduled on this machine. Constraints $x_i + x_j \leq 1$ state that they both cannot be assigned to the same machine. The node labeled as $\emptyset$ indicates the root node where no branching constraints are enforced. Thus, for the node labelled $x_5 = x_6$ it holds that $B_{=} = \{\{J_5, J_6\}\}$ and $B_{\leq} = \{\{J_1, J_2\}, \{J_3, J_4\}\}$.

The branching mechanism selects a pair of jobs to branch on as follows. Let R be a subset of patterns P' with a non-zero value y_k (i.e., the value of its corresponding indicator variables) in the current master problem solution and let hamming(i, j) be Hamming distance between vectors $(p_i^{(k)})_{k \in R}$ and $(p_j^{(k)})_{k \in R}$. Then, the pair of

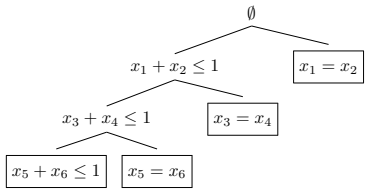


Figure 5.5: Example branching tree with highlighted four open nodes.

jobs $(i, j)^\star$ to branch on, is selected as

$$(i,j)^\star = \arg\min_{(i,j)\notin B} \left| \text{hamming}(i,j) - \frac{|R|}{2} \right|. \tag{5.3.29}$$

The rule (5.3.29) is chosen in order to create child nodes with similarly disrupted sets of active patterns; however, this is our design choice. Any selection mechanism that is able to select all possible pairs applies as well.

Note that the selection of an arbitrary pair of jobs may lead to two possible issues: (i) we may try to impose a constraint that already exists, (ii) the introduced constraint is in the contradiction with previous ones. The first issue can be checked by examining B, and if it is the case, we replace the second job with the other one in the order of given by the objective in (5.3.29). If the situation repeats, we again take the next following job in the order. Eventually, we may try all pairs of jobs.

The second issue suggests that there might be conflicts in the generated constraints. Indeed, consider an example where $B_= = \{\{J_1, J_2\}, \{J_2, J_3\}\}$ and $B_\leq = \{\{J_1, J_3\}\}$. Then the set of decision $B = B_= \cup B_\leq$ is contradictory considering that at least one of the jobs J_1, J_2, and J_3 has to be scheduled in some pattern. To mitigate conflicts in B, we introduce a conflict graph. A job is represented as a vertex, and each constraint of type $B_\leq$ represents an edge. Pairs of vertices in $B_=$ are contracted to a single vertex. The selection of the pair of jobs to branch on is done with jobs that are not in the same connected component of the conflict graph, which serves as a heuristics to avoid possible contradictions. However, eventually, it might become necessary to select jobs from the same component. The selection procedure described above does not guarantee that one of the new branching decisions will not cause a contradiction with the previous decisions. Nevertheless, such situations are exceptional, and they are handled by the so-called *recovery model* described in the next section.

After the branching, all existing patterns that violate the current branching constraints B are removed from P'. This, in turn, can cause infeasibility of the master problem since it might end up with an insufficient set of patterns that cannot satisfy the condition of scheduling each job at least once. Therefore, we need to check the feasibility of the master problem and, eventually, we need to regenerate additional patterns with respect to the given constraints B. The feasibility check of the master problem and pattern regeneration is also done with the recovery model.

5.3.5 Recovery model

The purpose of the recovery model is to check consistency with the current set of branching decisions B and potentially add new patterns to P' to guarantee the

feasibility of the master model. If a conflict in the branching decisions is detected, the recovery model turns infeasible, and we cut the node off in the branching tree.

Let $\lambda_j^{(k)}$ be a binary assignment variable for $k \in \{1,\dots,m\}, j \in \{1,\dots,n\}$ denoting whether job J_j is presented in pattern k. We formulate the objective function of the recovery model as the minimization of the sum across all these variables subject to branching decisions $B = B_= \cup B_\leq$ while introducing the constraint stating that each pattern has to schedule between x_{min} and x_{max} jobs. The recovery model can be described with the following MIP:

$$\min \sum_{k=1}^{m}\sum_{j=1}^{n} \lambda_j^{(k)} \tag{5.3.30}$$

$$\text{subject to:} \quad \sum_{k=1}^{m} \lambda_j^{(k)} = 1 \quad \forall j \in \{1,\dots,n\}, \tag{5.3.31}$$

$$x_{min} \leq \sum_{j=1}^{n} \lambda_j^{(k)} \leq x_{max} \quad \forall k \in \{1,\dots,m\}, \tag{5.3.32}$$

$$\lambda_a^{(k)} = \lambda_b^{(k)} \quad \forall k \in \{1,\dots,m\}, \forall \{a,b\} \in B_=, \tag{5.3.33}$$

$$\lambda_a^{(k)} + \lambda_b^{(k)} \leq 1 \quad \forall k \in \{1,\dots,m\}, \forall \{a,b\} \in B_\leq, \tag{5.3.34}$$

$$\text{where:} \quad \lambda_j^{(k)} \in \{0,1\} \quad \forall k \in \{1,\dots,m\}, \forall j \in \{1,\dots,n\}. \tag{5.3.35}$$

Note that the recovery model does not consider the exact costs of generated patterns; it just focuses on their feasibility. Nevertheless, the cost of regenerated patterns can be improved by imposing the constraint (5.3.32) that each generated pattern needs to contain between x_{min} and x_{max} jobs. However, for the consistency check itself, this constraint can be dropped. It can be seen that the formulation (5.3.30)–(5.3.35) is essentially tantamount to Graph Coloring problem. Indeed, the feasibility problem can be reduced to a graph with at most n vertices and the set of edges defined by $B_\leq$ with the question of whether it can be colored with exactly m colors. Each vertex corresponds to a single job except the pairs of jobs in set $B_=$ which are replaced by a single vertex for each pair. The constraint (5.3.31) represents the fact that each vertex has to be colored with one of the m colors. Although the graph coloring is NP-complete for $m \geq 3$ colors (i.e., machines), its solution time is negligible compared to the rest of the algorithm.

5.4 Problem with two machines

In this section, we introduce an algorithm for a special case of the problem with two machines. Our motivation for the study of this special case is mainly two-fold. First of all, we will show that the problem with two machines can be solved very efficiently in practice. This enables the practical use of the aggregated machines upper bound proposed in Proposition 13 with $k = 1$. For the second, efficient solutions of two-machine cases of scheduling problems also find applications for solving the general problems, where they act as subroutines which repeatably perform fast optimizations over search neighborhood defined by selected two machines [106].

In our case, we utilize it as an intensification step for DF-DOPO metaheuristic described in Section 5.2.1.

We first reformulate the problem statement for this special case. Then, we show that the continuous relaxation of this problem leads to the maximization of a concave function over a single variable. Finally, we show that we can obtain the optimal schedule from the solution of the relaxed problem by solving two simple sub-problems.

5.4.1 Reformulation for two machines

Let $\boldsymbol{x}_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,n})$ be the vector of job-machine assignment variables $x_{i,j} \in \{0,1\}$ for machine $i \in \{1,2\}$. We derive the objective function for the case of two machines. By applying the logarithm and expanding the sum, we write that

$$\max_{\boldsymbol{x}_1, \boldsymbol{x}_2} \log \Phi \left(\frac{\delta - \boldsymbol{\mu}^\intercal \boldsymbol{x}_1}{\sqrt{\boldsymbol{\sigma}^{2\intercal} \boldsymbol{x}_1}} \right) + \log \Phi \left(\frac{\delta - \boldsymbol{\mu}^\intercal \boldsymbol{x}_2}{\sqrt{\boldsymbol{\sigma}^{2\intercal} \boldsymbol{x}_2}} \right). \tag{5.4.1}$$

Since the sum of the total means on both machines must add up to the sum of all available means, we can write that

$$\boldsymbol{\mu}^\intercal \mathbf{1} = \boldsymbol{\mu}^\intercal \boldsymbol{x}_1 + \boldsymbol{\mu}^\intercal \boldsymbol{x}_2,$$

where $\mathbf{1}$ is the unit vector. The variances obey the identical relation:

$$\boldsymbol{\sigma}^{2\intercal} \mathbf{1} = \boldsymbol{\sigma}^{2\intercal} \boldsymbol{x}_1 + \boldsymbol{\sigma}^{2\intercal} \boldsymbol{x}_2.$$

Let $\underline{v} = \sum_{k=1}^{x_{min}} \overline{\sigma}_k^2$ with $\overline{\sigma}_k^2$ being the k-th smallest element of $\boldsymbol{\sigma}^2$, $\overline{v} = \boldsymbol{\sigma}^{2\intercal} \mathbf{1}$, $V = \left\{ \underline{v}, \underline{v}+1, \ldots, \lceil \frac{\overline{v}}{2} \rceil \right\}$ and for simplicity let $\boldsymbol{x} = \boldsymbol{x}_1$. Then, we reformulate the problem as

$$\max_{v \in V} \max_{\boldsymbol{x}} \log \Phi \left(\frac{\delta - \boldsymbol{\mu}^\intercal \boldsymbol{x}}{\sqrt{v}} \right) + \log \Phi \left(\frac{\delta - \boldsymbol{\mu}^\intercal \mathbf{1} + \boldsymbol{\mu}^\intercal \boldsymbol{x}}{\sqrt{\overline{v} - v}} \right) \tag{5.4.2}$$

$$\text{subject to:} \quad \boldsymbol{\sigma}^{2\intercal} \boldsymbol{x} = v, \tag{5.4.3}$$

$$\text{where:} \quad \boldsymbol{x} \in \{0,1\}^n. \tag{5.4.4}$$

We solve the problem similarly as in Section 5.3.3. For each fixed $v \in V$, we solve the problem only in terms of $\boldsymbol{x}$ variables. It is sufficient to test values of v up to $\lceil \frac{\overline{v}}{2} \rceil$ due to the symmetry of the two machines (i.e., the first machine is the machine with smaller or equal of the total variance).

5.4.2 Solving the relaxation

In this section, we deal with the solution of the relaxed subproblem parametrized by a value of v (i.e., the total variance of the jobs on the first machine), represented by the inner maximization problem in (5.4.2). We will use the substitutions

$$a = \delta - \boldsymbol{\mu}^\intercal \boldsymbol{x},$$
$$c = -2 \cdot \delta + \boldsymbol{\mu}^\intercal \mathbf{1},$$

and the identity $\Phi(z) = 1 - \Phi(-z)$, $\forall z \in \mathbb{R}$. The relaxation is done by allowing a to take any real value. Then, we write the relaxed subproblem for a fixed value of v as

$$\max_a g_v(a) \tag{5.4.5}$$

$$g_v(a) = \log \Phi\left(\frac{a}{\sqrt{v}}\right) + \log\left(1 - \Phi\left(\frac{c+a}{\sqrt{\overline{v}-v}}\right)\right), \tag{5.4.6}$$

$$\text{where:} \quad a \in \mathbb{R}. \tag{5.4.7}$$

It can be shown that such problem has a single extreme.

Proposition 15. *For any fixed $v \in [\underline{v}, \overline{v} - 1]$, $g_v(a)$ is concave on $\mathbb{R}$.*

Proof. The result can be shown with operations that preserve concavity. It is known that the logarithm of the cumulative normal distribution is concave on $\mathbb{R}$, see, e.g., [38, p. 104]. The same arguments apply for the second term of $g_v(a)$. Finally, since the sum of concave functions is a concave function, we have that $g_v(a)$ is concave on $\mathbb{R}$. □

The general approach to the analytic solution of (5.4.5)–(5.4.7) would be to find $a^\star$ where $g'_v(a^\star) = 0$. Using the definition of the normal cumulative distribution function, we get that

$$g_v(a) = \log\left(\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\frac{a}{\sqrt{v}}} e^{-\frac{t^2}{2}}\, dt\right) + \log\left(1 - \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\frac{c+a}{\sqrt{\overline{v}-v}}} e^{-\frac{t^2}{2}}\, dt\right). \tag{5.4.8}$$

Note that $g_v(a)$ is a function of variable a only, and it is present in the bound of

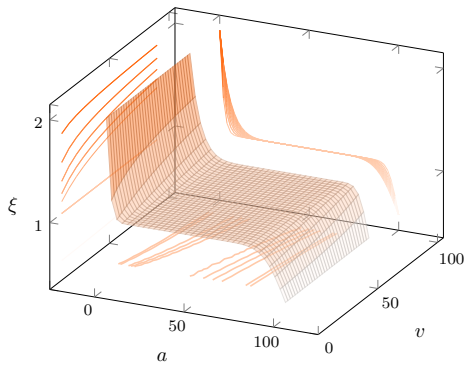


Figure 5.6: Correction factor ξ with $\overline{v} = 100$ and $c = -100$.

the two integrals. To find the derivative of $g_v(a)$ with respect to a, we use Leibniz integral rule which leads to

$$\frac{\mathrm{d}g_v}{\mathrm{d}a} = \xi \cdot \frac{f\left(\frac{a}{\sqrt{v}}\right)}{\sqrt{v}} - \frac{f\left(\frac{c+a}{\sqrt{\overline{v}-v}}\right)}{\sqrt{\overline{v}-v}} = 0, \tag{5.4.9}$$

where f is the probability density function of $\mathcal{N}(0,1)$ and

$$\xi = \frac{1 - \Phi\left(\frac{c+a}{\sqrt{\overline{v}-v}}\right)}{\Phi\left(\frac{a}{\sqrt{v}}\right)}$$

is a correction factor. We conjecture that (5.4.9) is analytically unsolvable due to the presence of a both in the cumulative distribution function Φ and the probability density function f. Hence, instead of solving (5.4.9), our approach is to find a surrogate equation that we can solve analytically. The true global extreme of the original function could be then recovered using the gradient descent procedure if needed. The efficiency of this approach depends on the choice of the surrogate equation. We have noticed that for optimal solutions often hold that $\xi \approx 1$, which suggests to set $\xi = 1$ and obtain the surrogate equation in the form of

$$\frac{1}{\sqrt{v}} \cdot f\left(\frac{a}{\sqrt{v}}\right) - \frac{1}{\sqrt{\bar{v}-v}} \cdot f\left(\frac{c+a}{\sqrt{\bar{v}-v}}\right) = 0. \tag{5.4.10}$$

We note that the surrogate equation (5.4.10) is remarkably similar to the original one for almost all sensible values a and v, which can be demonstrated by the plot of the correction factor ξ in Figure 5.6. Coincidentally, the equation (5.4.10) has a closed-form solution, since it leads to a quadratic equation over a single variable. The roots are given as

$$a^*_{\mp} = \frac{\frac{c}{\bar{v}-v} \mp \sqrt{\frac{c^2}{(\bar{v}-v)\cdot v} - \left(\frac{1}{v} - \frac{1}{\bar{v}-v}\right) \cdot \log\left(\frac{v}{\bar{v}-v}\right)}}{\frac{1}{v} - \frac{1}{\bar{v}-v}}. \tag{5.4.11}$$

The true global optimum of (5.4.5) up to the integer precision needs to be computed when ξ is not exactly 1. This can be done by the gradient descent procedure on the original function with a starting point given by the solution of the surrogate equation. In the next section, we will explain how the relaxed solution of a subproblem (5.4.5)–(5.4.7) is used to obtain the optimal solution of the original problem.

5.4.3 Complete algorithm

The surrogate equation (5.4.10) gives us two real solutions. The root a^*_- of (5.4.10) is meaningful, while a^*_+ is the result of the introduced approximation of ξ. We set a^*_- as the starting point of the gradient descent procedure since it is typically very close to the true global optimum of (5.4.5).

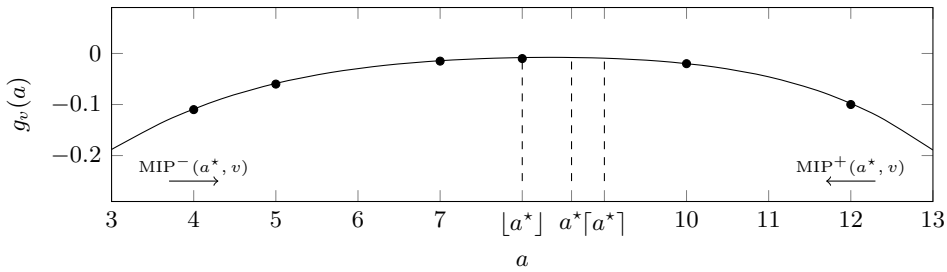


Figure 5.7: Example of the objective function with highlighted optimal relaxed solution $a^\star$.

To find the optimal solution of the original (discrete) problem, we utilize the following observation. For the given value of v, the objective function (5.4.2) and its domain is the same as the relaxed subproblem (5.4.5) except that (5.4.2) is defined only for some integer values of a that are attainable in the given problem instance (e.g., denoted by $\bullet$ in Figure 5.7). Since we know that the relaxed objective function is concave and where the maximum of its relaxation is located, we can solve the original problem without modeling the objective function explicitly. This allows us to solve the problem much more efficiently.

The idea is the following. We construct two MIPs: the first one minimizes the value of a subject to constraints: (i) value a is greater or equal to $\lceil a^\star \rceil$, (ii) value a is attainable in the given problem instance. The second MIP maximizes a subject to a is smaller or equal to $\lfloor a^\star \rfloor$ and the attainability condition. See the complete description of those MIPs below.

$$
\begin{array}{lll}
\mathrm{MIP}^+(a^\star, v): & & \\
 & \min a & \\
\text{subject to:} & a \geq \lceil a^\star \rceil, & \\
 & a = \delta - \boldsymbol{\mu}^\intercal \boldsymbol{x}, & \\
 & v = \boldsymbol{\sigma^2}^\intercal \boldsymbol{x}, & \\
\text{where:} & a \in \mathbb{Z}, & \\
 & \boldsymbol{x} \in \{0,1\}^n. &
\end{array}
\qquad
\begin{array}{lll}
\mathrm{MIP}^-(a^\star, v): & & \\
 & \max a & \\
\text{subject to:} & a \leq \lfloor a^\star \rfloor, & \\
 & a = \delta - \boldsymbol{\mu}^\intercal \boldsymbol{x}, & \\
 & v = \boldsymbol{\sigma^2}^\intercal \boldsymbol{x}, & \\
\text{where:} & a \in \mathbb{Z}, & \\
 & \boldsymbol{x} \in \{0,1\}^n. &
\end{array}
$$

The output of the two MIPs is two job-machines assignments $\boldsymbol{x}^\star_+, \boldsymbol{x}^\star_-$. We evaluate both $\boldsymbol{x}^\star_+, \boldsymbol{x}^\star_-$ in terms of the original objective (5.4.1) with $\boldsymbol{x}_1 = \boldsymbol{x}^\star_+$ ($\boldsymbol{x}^\star_-$) and $\boldsymbol{x}_2 = \mathbf{1} - \boldsymbol{x}_1$ and select the better one. The whole algorithm is shown in Algorithm 5.3, and we refer to it as to TM (two-machines) algorithm.

Let us note that for the most challenging problem instances, the gradient descent performs just a few iterations in a vast majority of cases. We want to point out a great advantage of TM algorithm, which is a possibility of an easy parallelization. The idea is to allocate a pool of worker threads that will process subproblems for each value of $v \in V$ independently. In Section 5.5.6, we demonstrate speedups of TM algorithm achievable by this parallelism.

Finally, we describe modifications of TM algorithm needed to compute the value of $\mathrm{OPT}_q(\delta, \delta(m-1))$, which is involved in the bounds x_{min} and x_{max} presented in Section 5.2. First, it is necessary to set $c = -\delta m + \boldsymbol{\mu}^\intercal \mathbf{1}$ and $V = \{0, \ldots, \boldsymbol{\sigma^2}^\intercal \mathbf{1}\}$. Next, we impose constraint $\boldsymbol{x}^\intercal \mathbf{1} = q$ to both models $\mathrm{MIP}^+(a^\star, v)$ and $\mathrm{MIP}^-(a^\star, v)$. Finally, note that one has to check boundary conditions $v = 0$ and $v = \boldsymbol{\sigma^2}^\intercal \mathbf{1}$ separately in Algorithm 5.3.

5.5 Experiments

5.5.1 Experimental setup

To compare our proposed algorithms, we implemented the better of the two branch-and-bound algorithms proposed by [135] (named B&B1) as a reference. The

Algorithm 5.3: TM algorithm.

$g_{\text{best}} \leftarrow -\infty$, $\boldsymbol{x}_{\text{best}} \leftarrow \emptyset$
for $v \in V$ **do**
  $a^*_- \leftarrow \left(\frac{c}{\bar{v}-v} - \sqrt{\frac{c^2}{(\bar{v}-v)\cdot v} - \left(\frac{1}{v} - \frac{1}{\bar{v}-v}\right) \cdot \log\left(\frac{v}{\bar{v}-v}\right)} \right) \cdot \left(\frac{1}{v} - \frac{1}{\bar{v}-v}\right)^{-1}$
  while $g_v(a^\star_-) < \max\left\{g_v(a^\star_- - 1), g_v(a^\star_- + 1)\right\}$ **do** `// discrete gradient descent`
    if $g_v(a^\star_-) < g_v(a^\star_- - 1)$ **then**
      $a^\star_- \leftarrow a^\star_- - 1$
    else
      $a^\star_- \leftarrow a^\star_- + 1$
  $a^\star \leftarrow a^\star_-$
  $\boldsymbol{x}^\star_+ \leftarrow \text{MIP}^+(a^\star, v)$, $\boldsymbol{x}^\star_- \leftarrow \text{MIP}^-(a^\star, v)$
  $\boldsymbol{x}^\star \leftarrow \arg\max\left\{g_v(\delta - \boldsymbol{\mu}^\intercal \boldsymbol{x}^\star_+), g_v(\delta - \boldsymbol{\mu}^\intercal \boldsymbol{x}^\star_-)\right\}$ `// recovering an integer optimum`
  if $g_v(\delta - \boldsymbol{\mu}^\intercal \boldsymbol{x}^\star) > g_{best}$ **then**
    $g_{\text{best}} \leftarrow g_v(\delta - \boldsymbol{\mu}^\intercal \boldsymbol{x}^\star)$, $\boldsymbol{x}_{\text{best}} \leftarrow \boldsymbol{x}^\star$
return $g_{best}, \boldsymbol{x}_{best}$

branch-and-bound method was implemented in C++ wheres the other methods were implemented in Python 3.7. We used Gurobi 9.1.1 solver to solve MIP models. The MINLP (5.1.5)–(5.1.10) was solved using SCIP 6.0.0 solver. Note that the most computationally intensive parts of Python codes are in fact implemented in C++ as well, since they are offloaded to external libraries, such as Numpy, Nevergrad or a MIP solver. Thus, we would expect only negligible speed-ups of the Python codes, if the their whole codebase would be migrated to C++ as well. All experiments were run on a computer with two Intel Xeon E5-2620 v4 processors, each having 28 threads, 252 GB RAM memory and a 64-bit operating system. For branch-and-price experiments, at most 28 threads were allowed to use. MINLP and branch-and-bound methods do not offer parallel execution; thus, they use a single CPU core.

For each tested algorithm, we set the timeout of five hours (if not specified otherwise), after which if the program did not finish its computation, it has been stopped. Such instances, where an algorithm was not able to find an optimal solution in the given time, were not included in the mean runtime and expanded nodes calculation.

The experiments were carried out on several datasets where the means of jobs $\boldsymbol{\mu}$ are drawn from a normal distribution with mean equal to 20 and standard deviation equal to 9. For a particular value of μ_j, the variance of a job σ_j^2 is generated from $[1, 0.1 \cdot \eta \cdot \mu_j^2]$ uniformly. Parameter η influences the variance of the jobs [135]. In this dataset it was set to either 0.25 or 0.75. The MINLP was tested using a set of instances with 2, 3, and 4 machines and $n \in \{10, \ldots, 15\}$ jobs. Ten instances were generated for each combination (m, n, η), with a total of 240 instances in the

whole test set. The set of instances where $m = 2$ was used for the comparison of the branch and bound and TM algorithm. This set was extended with instances having 16, 18, 20, 40, 80, 200 and 500 jobs, which were also solved by TM algorithm and branch-and-bound to demonstrate their scalability.

A separated set of instances for branch-and-price experiments was created for comparability with [135]. All instances were generated with the same parameters as used in [135], i.e., the mean values of jobs are drawn from the normal distribution with the mean equal to 20 and standard deviation equal to 3 while the distribution of jobs' variances is kept the same. Moreover, the η parameter was set to either 0.25, 0.50 and 0.75. With this setting, we have created instances with 3, 4, 5, 6 and 7 machines and 14, 16, 18, 20, 22 and 24 jobs. Ten instances for each (m, n, η) tuple were generated.

The rest of this section is structured as follows. First, we compare heuristic algorithms, and then we evaluate the performance of the MINLP model and branch-and-price method. Finally, we benchmark TM algorithm for the special case with two machines.

5.5.2 Heuristic solution quality

Figure 5.8 shows a histogram of objective values calculated using the heuristic proposed in [135], values obtained using our LSAF list scheduling, and DF-DOPO also proposed in this paper. The computation of DF-DOPO is not deterministic; hence, we report median objective value across 21 runs for each instance. The standard deviation of objective values among different runs is 0.01 on average. The DF-DOPO utilizes budget equal to 1500 (i.e., the number of evaluations of the fitness function). After DF-DOPO finishes, a local search intensification procedure by TM algorithm is performed. We refer to such combination of the two algorithms as DF-DODO+TM.

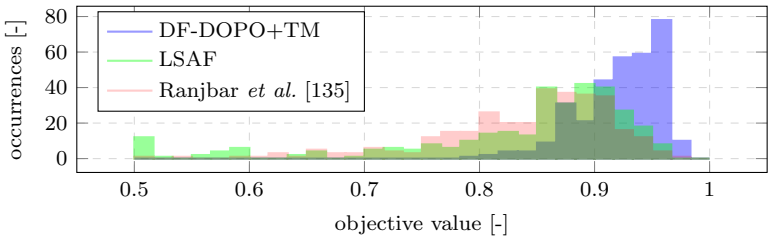


Figure 5.8: Histogram of objective values produced by initial heuristics.

The comparison among list scheduling algorithms shows that the heuristic of [135] has significantly better worst-case performance than LSAF. On the other hand, LSAF performs better in the majority of cases – LSAF found a strictly better solution in 59.6% of cases. The runtime of both methods is negligible. Hence, considering low running times of both methods, we recommend running both and taking the better solution of the two. As expected, DF-DOPO+TM is superior both on average and the worst-case but at the expense of longer runtime which is in orders of seconds on commodity hardware.

5.5.3 MINLP performance

Table 5.2 shows a comparison of the MINLP model against the reference branch-and-bound [135]. For a comparison, we have reimplemented the branch-and-bound algorithm [135] as the authors did not test in instances with 10 and 11 jobs or instances with $m = 2$ machines. The basic MINLP model was enhanced with a few improvements. First of all, we have observed that the original model suffers from the symmetry of the solution space. To mitigate this, we introduced a symmetry-breaking constraint stating that the sum of mean values on the i-th machine has to be greater or equal than the sum of means on the $(i + 1)$-th machine, i.e., $\mu_{M_i} \geq \mu_{M_{i+1}}, \forall i \in \{1, \ldots, m-1\}$. Furthermore, we have imposed bounds on the objective and bounds on the minimum and the maximum variance each machine can contain in order to improve the performance and the numerical stability of the solver. The first two columns in Table 5.2 show the performance of the MINLP model with and without the symmetry-breaking constraint, respectively. The last column of Table 5.2 shows results for the our reimplementation of the reference branch-and-bound algorithm. However, note that it should be meant for illustration purposes only, as our reimplementation may not be as efficient as the original one.

For $m \geq 3$ machines, the symmetry-breaking constraint has a noticeable effect on the performance of the MINLP model. On the largest tested instances the speed up was approximately five-fold. This is also reflected in a lower number of expanded nodes for each instance. MINLP model with symmetry breaking is competitive with our reimplementation of the branch-and-bound algorithm. It scales similarly, but it is about 2–3 times slower. On the smallest instances (i.e., $m = 2, n \leq 12$), the reference algorithm was able to solve them in less than 100 milliseconds. On the largest instances ($m = 3, n = 15$ and $m = 4, n = 15$), the reference algorithm outperformed the MINLP model, although the number of expanded nodes is still significantly higher.

		MINLP		MINLP (no symmetry break)		branch-and-bound [135]	
machines	jobs	runtime [s]	nodes [–]	runtime [s]	nodes [–]	runtime [s]	nodes [–]
$m = 2$	$n = 10$	1.1 (±0.4)	76.0 (±44.1)	1.4 (±0.4)	232.8 (±191.6)	0.0 (±0.0)	503.8 (±64.2)
	$n = 11$	1.0 (±0.4)	108.3 (±53.5)	1.8 (±0.6)	353.1 (±242.1)	0.0 (±0.0)	1.0K (±126.6)
	$n = 12$	1.3 (±0.5)	160.2 (±97.1)	1.9 (±0.4)	393.6 (±199.0)	0.0 (±0.0)	1.9K (±246.9)
	$n = 13$	1.3 (±0.7)	255.8 (±200.8)	2.9 (±1.0)	1.0K (±609.3)	0.1 (±0.0)	4.0K (±495.4)
	$n = 14$	1.5 (±0.8)	294.6 (±268.1)	2.8 (±1.1)	1.1K (±1.2K)	0.2 (±0.0)	8.3K (±803.5)
	$n = 15$	2.1 (±1.1)	746.4 (±735.3)	3.7 (±1.3)	1.3K (±610.8)	0.4 (±0.0)	16.4K (±1.2K)
$m = 3$	$n = 10$	3.8 (±1.4)	503.8 (±261.0)	6.6 (±2.2)	2.3K (±1.3K)	0.1 (±0.0)	2.5K (±613.5)
	$n = 11$	5.1 (±1.4)	977.9 (±509.1)	13.0 (±8.6)	4.7K (±4.0K)	0.2 (±0.0)	8.5K (±1.6K)
	$n = 12$	8.5 (±2.9)	2.1K (±863.2)	25.4 (±11.9)	9.5K (±4.9K)	0.6 (±0.1)	24.1K (±4.7K)
	$n = 13$	14.9 (±5.3)	4.5K (±2.4K)	55.7 (±28.2)	21.9K (±12.8K)	1.8 (±0.4)	71.2K (±14.2K)
	$n = 14$	35.5 (±28.9)	14.8K (±13.0K)	207.3 (±188.0)	66.7K (±50.9K)	6.0 (±1.3)	236.5K (±53.1K)
	$n = 15$	70.4 (±41.3)	29.0K (±18.5K)	586.5 (±534.3)	139.5K (±101.5K)	18.5 (±3.8)	729.9K (±157.1K)
$m = 4$	$n = 10$	5.9 (±2.5)	1.1K (±649.0)	19.8 (±9.3)	6.5K (±3.2K)	0.1 (±0.0)	2.7K (±971.0)
	$n = 11$	9.7 (±2.2)	2.6K (±1.1K)	58.9 (±36.8)	19.1K (±12.6K)	0.3 (±0.1)	9.4K (±2.5K)
	$n = 12$	20.3 (±11.0)	6.9K (±5.0K)	129.6 (±107.9)	40.1K (±34.2K)	1.3 (±0.4)	42.1K (±13.1K)
	$n = 13$	50.0 (±38.4)	16.9K (±12.5K)	390.8 (±319.6)	123.5K (±98.0K)	5.3 (±1.6)	172.1K (±56.0K)
	$n = 14$	112.2 (±75.5)	37.7K (±24.0K)	750.9 (±552.2)	204.1K (±164.7K)	23.3 (±7.2)	766.5K (±237.8K)
	$n = 15$	267.0 (±131.8)	79.0K (±40.2K)	1.3K (±979.3)	304.4K (±149.6K)	91.9 (±18.0)	3.0M (±589.1K)

Table 5.2: Comparison of MINLP and branch-and-bound.

5.5.4 Branch-and-price performance

First, we perform experiments to assess the dependency of runtimes to the parameters of the instance. Namely, we are interested in how the values of job processing time variances affect the performance of the branch-and-price algorithm. To determine that, we have generated a dataset with the fixed values $m = 3$ and $n = 18$. To obtain instances with different variances of the processing time, we assumed five different values $\eta \in \{0.1, 0.3, 0.5, 0.7\}$ and we generated 20 instances for each η. The results are presented in Figure 5.9, where each box plot reports 0.1, 0.25, 0.5, 0.75 and 0.9 quantilies of running times.

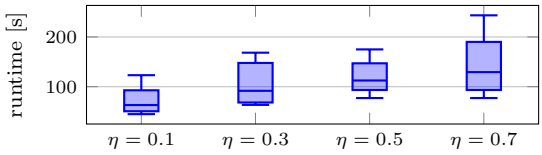


Figure 5.9: The effect of job variances to the performance of branch-and-price algorithm.

The results show that with the increasing value of η, both the median, variance and the worst-case runtime are increasing as well. Since the complexity of the pricing algorithm depends on the maximum value of σ_j^2 present in the input instance, it is expected that its value affects the overall runtime as well. In fact, the observation that the runtime decreases for smaller values of variances seems to be a natural property of the branch-and-price algorithm. Note that the limiting case of the studied problem, when variances approach zero, is the deterministic parallel machines scheduling problem with a common due date. Nevertheless, this kind of problem can be solved very efficiently in practice. Thus, it is expected that the problem with decreasing η gets easier.

As a next experiment, we evaluate the performance of the branch and price with varying m and n. The results for the branch-and-price algorithm and for the reference branch and bound [135] are summarized in Table 5.3. The results for the branch-and-price algorithm in each row show the aggregated values over different values of $\eta \in \{0.25, 0.5, 0.75\}$ with 10 instances for each η, i.e., 30 instances per row. Table 5.3 provides the runtime, percentage of instances that were not solved in the time limit of 8 hours, and the mean number of expanded nodes supplemented with the standard deviation in parentheses. The instances that exceed the time limit were excluded from these quantities. Note that, as we have seen in Figure 5.9, the runtime of branch-and-price algorithm is sensitive to the value of η. Particularly here, the median is lower than mean value which susceptible to outliers. Thus, we report in the runtime column both median and mean value of runtime with its standard deviation in parenthesis for more robust statistical estimate.

For sake of comparison, we present the results of the better of the two branch and bound algorithms presented in [135]. The runtimes from [135] were rescaled to reflect different processors used. The authors present results up to $n \leq 20$ with $m \leq 5$ that shows showing that the runtime of their branch and bound increases 10–25 times for each two additional jobs in the instance. Thus, we projected the runtimes and the number of expanded nodes up to $n = 24$ and we report them

machines	jobs	branch-and-price runtime [s]	branch-and-price nodes [–]	branch-and-price timeouts [%]	branch-and-bound [135] runtime [s]	branch-and-bound [135] nodes [–]
$m = 3$	$n = 14$	19.4 (20.4 ± 4.3)	1.8 (±2.9)	0	< 0.1	76K
	$n = 16$	37.5 (49.7 ± 33.0)	5.3 (±11.1)	0	0.3	559K
	$n = 18$	144.5 (158.0 ± 76.7)	4.4 (±7.7)	0	3.9	7767K
	$n = 20$	334.4 (459.3 ± 290.2)	3.3 (±4.1)	0	40.6	78124K
	$n = 22$	642.1 (1368.1 ± 1285.6)	9.0 (±12.4)	0	*388.5*	*778M*
	$n = 24$	823.4 (2163.7 ± 2899.4)	12.2 (±24.0)	0	*3739.8*	*7776M*
$m = 4$	$n = 14$	17.5 (18.1 ± 3.8)	1.5 (±2.9)	0	0.1	120K
	$n = 16$	25.1 (97.7 ± 156.0)	81.9 (±183.9)	0	0.8	1899K
	$n = 18$	41.4 (197.1 ± 421.5)	81.0 (±229.7)	0	15.2	35730K
	$n = 20$	1030.5 (2341.4 ± 2881.5)	395.8 (±685.4)	0	319.7	686100K
	$n = 22$	3101.9 (4490.7 ± 4907.6)	206.7 (±307.0)	7	*3636.4*	*9286M*
	$n = 24$	863.1 (3758.5 ± 5772.5)	40.5 (±80.9)	57	*41679.3*	*126474M*
$m = 5$	$n = 14$	17.9 (21.5 ± 14.9)	6.4 (±23.5)	0	< 0.1	53K
	$n = 16$	21.1 (34.2 ± 37.8)	11.8 (±34.4)	0	0.7	1840K
	$n = 18$	27.5 (67.3 ± 119.6)	22.3 (±62.3)	0	24.4	61228K
	$n = 20$	94.4 (859.9 ± 1344.8)	261.7 (±482.0)	0	608.7	1400605K
	$n = 22$	813.0 (2038.9 ± 3083.2)	272.9 (±697.2)	0	*6666.1*	*20099M*
	$n = 24$	711.1 (4608.9 ± 6360.4)	178.8 (±383.9)	63	*73493.0*	*290298M*
$m = 6$	$n = 14$	19.8 (19.9 ± 0.6)	1.1 (±0.4)	0	–	–
	$n = 16$	22.1 (29.5 ± 28.4)	9.4 (±33.9)	0	–	–
	$n = 18$	24.9 (75.1 ± 123.9)	39.4 (±101.3)	0	–	–
	$n = 20$	33.3 (141.5 ± 236.7)	52.4 (±148.4)	0	–	–
	$n = 22$	132.0 (1469.0 ± 2145.0)	445.3 (±949.9)	0	–	–
	$n = 24$	4493.7 (7387.2 ± 7801.3)	940.0 (±1500.5)	27	–	–
$m = 7$	$n = 14$	21.9 (22.1 ± 1.2)	1.7 (±3.6)	0	–	–
	$n = 16$	23.6 (23.7 ± 0.6)	1.0 (±0.0)	0	–	–
	$n = 18$	26.5 (50.4 ± 62.1)	19.9 (±55.4)	0	–	–
	$n = 20$	31.9 (246.7 ± 466.5)	175.9 (±427.7)	0	–	–
	$n = 22$	47.8 (891.1 ± 2164.3)	326.6 (±940.7)	0	–	–
	$n = 24$	113.7 (805.0 ± 1552.4)	83.9 (±231.5)	27	–	–

Table 5.3: Comparison of branch-and-price and branch-and-bound algorithms.

printed in italics. Note that we did not projected entries for instances with different number of machines as we did not have enough data to do so.

One can observe that with the increasing number of machines, the problem becomes easier for the branch and price, but more complex for the branch and bound [135]. There can be identified two causes for that. First, increasing the number of machines can be seen for branch-and-price algorithm as a form of relaxation (see equation (5.3.4)). Moreover, the similar behavior was observed for related deterministic problems as well, e.g., [98]. The only exception for the branch and price are instances with $m = 3$ which can be also solved efficiently, due to the tighter aggregated machines upper bound proposed Section 5.2.2.

On contrary, the branch and bound has more difficulties as it has difficulties to exploit problem symmetries in terms of identical machines. Regarding different values of η, we have observed that instances with $\eta = 0.75$ are particularly challenging for the branch and price — majority of time outs and large values of the running times are observed for these instances.

To summarize, the results suggest that the branch and price is better for instances with $m \geq 4$, especially with the increasing number of jobs. On the other hand, we recommend using the branch and bound [135] for small-sized instances and instances containing jobs with large variance values. In the following section, we propose a new scheme of generating instances and perform sensitivity with respect to different correlation values between means and variances of the jobs.

5.5.5 Correlated instances

In the experiments above, we have observed that the scheme of generating instances as described in Section 5.5.4 and used in [135] has two limitations. First of all, it only generates jobs with positively correlated mean and variance. Although it is a reasonable property, we believe that it is useful to have also parameter $\rho \in [-1, 1]$ that controls the dependence between the mean and variance, so it would allow even negative correlations between those two. Second, since the variances are generated from a uniform distribution whose range depends quadratically on the job's mean, then the former scheme occasionally generates a large variance for a job such that it effectively admits the realizations of negative processing times in a considerable number of cases.

Thus, in this experiment, we have proposed a different scheme of generating instances. Essentially, the parameters of jobs are generated as samples from a multivariate normal distribution $(\mu_j, \sigma_j^2) \sim \mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})$ with parameters $\hat{\boldsymbol{\mu}} = (20, 2)$ and $\hat{\boldsymbol{\Sigma}} = \begin{bmatrix} 10 & 2\rho \\ 2\rho & 1 \end{bmatrix}$. The particular values of the used constants were chosen such that the ranges of generated parameters are comparable to those produced by scheme of [135] and such that $\hat{\boldsymbol{\Sigma}}$ is a positive semidefinite matrix for a $\rho \in [-1, 1]$. We refer to these instances as to correlated instances with ρ parameter. See Figure 5.10 for samples from such a distribution of job parameters for different values of ρ.

Bellow we investigate the effect of ρ parameter on the computational time of the branch-and-price algorithm. Hence, we have fixed the number of machines to $m = 3$ and generated 10 instances for each combination of $n \in \{14, 16, \ldots, 24\}$ and $\rho \in \{-1, -0.7, 0, 0.7, 1\}$. The timeout was to 5 hours. The results are displayed in Table 5.4.

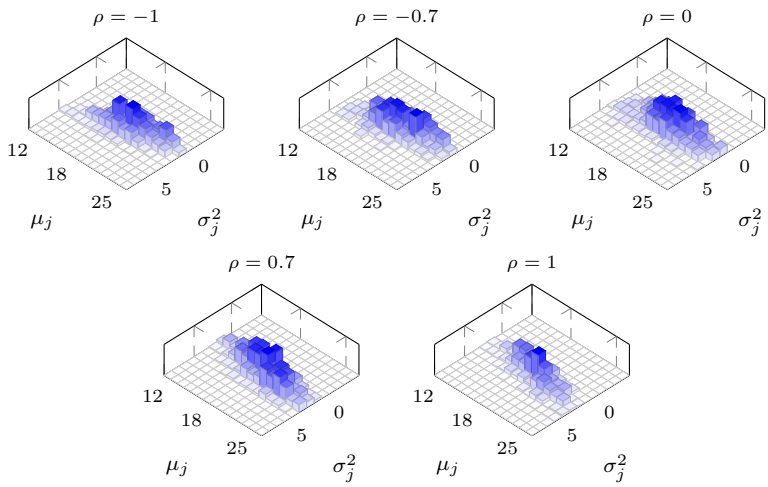


Figure 5.10: Effect of parameter ρ on the distribution of jobs' parameters.

Compared to the instances from Section 5.5.4, correlated instances appear to be harder than the ones generated by the former scheme. Concerning the effect of specific values of ρ, large positive correlations ρ generate hardest ones. This is reflected especially in the larger variance of the computational times, suggesting

correlation	jobs	branch-and-price runtime [s]	nodes [–]	timeouts [%]
$\rho = -1.0$	$n = 14$	18.6 (20.6 ± 7.1)	3.0 (±5.4)	0
	$n = 16$	45.7 (65.4 ± 37.2)	13.8 (±15.7)	0
	$n = 18$	85.0 (210.4 ± 309.0)	56.2 (±151.1)	0
	$n = 20$	206.7 (669.0 ± 624.2)	18.2 (±24.1)	0
	$n = 22$	625.0 (1450.1 ± 2310.1)	19.0 (±38.6)	0
	$n = 24$	513.6 (922.5 ± 878.5)	6.5 (±11.3)	20
$\rho = -0.7$	$n = 14$	18.6 (19.3 ± 2.6)	1.6 (±1.3)	0
	$n = 16$	33.0 (41.3 ± 15.7)	4.4 (±5.7)	0
	$n = 18$	86.8 (104.4 ± 64.3)	7.2 (±11.9)	0
	$n = 20$	472.6 (938.1 ± 1004.4)	31.2 (±42.9)	0
	$n = 22$	1750.1 (4184.6 ± 4984.1)	92.8 (±173.8)	20
	$n = 24$	1557.8 (3515.0 ± 3131.2)	26.1 (±26.8)	10
$\rho = 0.0$	$n = 14$	17.8 (17.9 ± 1.4)	1.2 (±0.6)	0
	$n = 16$	32.1 (41.1 ± 30.1)	5.0 (±11.3)	0
	$n = 18$	61.2 (131.8 ± 139.4)	11.4 (±26.8)	0
	$n = 20$	681.5 (1064.6 ± 1134.3)	39.4 (±51.4)	0
	$n = 22$	1014.3 (3159.0 ± 5321.6)	52.6 (±107.7)	10
	$n = 24$	4282.9 (5059.1 ± 4307.1)	53.9 (±54.0)	10
$\rho = 0.7$	$n = 14$	17.4 (18.0 ± 2.7)	1.0 (±0.0)	0
	$n = 16$	32.0 (40.5 ± 27.1)	5.8 (±12.5)	0
	$n = 18$	121.8 (106.8 ± 42.5)	5.6 (±3.9)	0
	$n = 20$	168.0 (331.6 ± 329.2)	10.6 (±18.5)	0
	$n = 22$	2293.3 (2899.5 ± 2739.0)	48.4 (±54.4)	0
	$n = 24$	7455.8 (6891.6 ± 5549.6)	94.1 (±101.7)	30
$\rho = 1.0$	$n = 14$	16.6 (17.1 ± 1.6)	1.0 (±0.0)	0
	$n = 16$	27.4 (32.1 ± 14.9)	3.4 (±6.6)	0
	$n = 18$	94.3 (116.8 ± 76.6)	18.6 (±23.4)	0
	$n = 20$	375.6 (1541.7 ± 2546.2)	301.0 (±795.9)	0
	$n = 22$	2598.2 (3568.7 ± 3460.9)	63.0 (±73.6)	20
	$n = 24$	6534.8 (5459.1 ± 3552.0)	55.3 (±44.9)	40

Table 5.4: Sensitivity with respect to ρ parameter for instances with $m = 3$.

that instances with larger ρ values contains more outliers in terms of runtime. This observation is in line with sensitivity analysis in Figure 5.9, where large values of η produce large variances of runtimes as well. Indeed, the instances with large values of η can be seen as positively correlated instances with large value of ρ which were proved to be the most difficult ones for the branch-and-price algorithm.

5.5.6 TM algorithm performance

Table 5.5 shows the mean running time and its standard deviation of the TM algorithm compared to the branch and price and our reimplementation of branch-and-bound algorithm for instances with $m = 2$ machines. We can divide the used instances into the two sets. The first set contains small instances with sizes $n \in \{14, 16, 18, 20\}$. To benchmark the scalability of the methods, we introduce the second set with $n \in \{40, 80, 200, 500\}$. We present results using a single CPU core and four CPU cores to demonstrate the parallelism capabilities of TM algorithm. The time limit for both methods was set to one hour.

For an illustrative comparison, we have used the reimplemented branch-and-bound algorithm from [135], as the paper do not contain experiments for instances with two machines. Although our reimplementation may not be as efficient as the original one, it displays the same scaling trend. It can be clearly seen that TM algorithm is comparable to the branch and bound on the instances with the smallest number of jobs but is significantly better as the number of jobs increases.

		TM algorithm (1 CPU)		TM algorithm (4 CPUs)		branch-and-bound [135]	
machines	jobs	runtime [s]	timeouts [%]	runtime [s]	speed up [-]	runtime [s]	timeouts [%]
$m = 2$	$n = 14$	1.1 (±0.8)	0	0.3 (±0.2)	3.6 (±0.4)	0.2 (±0.0)	0
	$n = 16$	1.0 (±0.9)	0	0.3 (±0.3)	3.3 (±0.3)	0.7 (±0.1)	0
	$n = 18$	1.3 (±1.2)	0	0.4 (±0.3)	3.2 (±0.4)	3.0 (±0.3)	0
	$n = 20$	1.5 (±1.1)	0	0.4 (±0.3)	3.7 (±0.4)	12.3 (±0.9)	0
	$n = 40$	2.5 (±1.9)	0	0.7 (±0.6)	3.6 (±0.3)	–	100
	$n = 80$	5.4 (±4.2)	0	1.5 (±1.2)	3.6 (±0.3)	–	100
	$n = 200$	14.6 (±8.9)	0	3.9 (±2.4)	3.7 (±0.4)	–	100
	$n = 500$	65.1 (±37.3)	0	16.9 (±9.7)	3.8 (±0.4)	–	100

Table 5.5: Comparison of TM algorithm and branch-and-bound algorithm.

5.5.7 Summary

From the results, we can observe the following:

(i) DF-DOPO+TM outperforms both list scheduling algorithms in terms of quality, but is more computationally demanding,

(ii) considering the simplicity of implementation, the MINLP with symmetry breaking constrains is competitive for instances with up to 15 jobs (especially with a low number of machines), but does not scale up well,

(iii) the case of two machines can be solved efficiently by the TM algorithm which is very fast even for hundreds of jobs and achieves almost linear parallel speedups,

(iv) for the general case, the proposed branch-and-price outperforms the branch-and-bound algorithm [135] on instances with larger number of machines and jobs, but is more sensitive to the presence of large values of job variances.

5.6 Conclusion

In this paper, we have revisited a parallel machines scheduling problem where the processing time of each job is given by a normal distribution from integer programming perspective. The problem is formulated as a β-robust scheduling problem where the objective is to maximize the probability the schedule is completed before a given common due date.

We have proposed a new relaxation of the problem based on the concept of aggregated machines. The upper bound on the objective is computed via the solution of a problem with a smaller number of machines with one machine having a modified due date. The practicality of the proposed bound follows from the existence of a very fast and parallel algorithm for the two-machines case. For this case, we have devised a specialized algorithm that utilizes the concavity of the relaxed objective function and a fast recovery procedure that converts optimal relaxed solution into an optimal integer one. We showed that our method is able to solve instances with up to 500 jobs in several seconds while having almost linear parallel speedups.

To solve the general problem, we have proposed a branch-and-price algorithm. The resulting pricing problem reveals a connection to the stochastic knapsack problem and can be efficiently solved in practice. Additionally, the efficiency of our algorithm lies in the use of Ryan-Forster's constraint branching, which mitigates symmetries in the solution space and good quality of initial solutions provided by the genetic algorithm with local search intensification. Furthermore, the branch and price utilizes bounds on the number of jobs that can be assigned to each machine, computed by the aggregated machines upper bound.

The experiments have demonstrated that the performance of our initial heuristic provides better lower bounds and that the proposed algorithms scales better than the branch-and-bound algorithms proposed in [135], especially with the increasing number of machines and jobs. The main drawback of the proposed branch-and-price remains in the weaker performance for the instances with the presence of very large variance values.

As for the future work, we note some extensions that might be incorporated into the proposed branch-and-price algorithm: (i) additional machine-related constraints (e.g., jobs incompatibility) can be embedded into the pricing problem without significant modifications of the rest of the algorithm, and (ii) the decomposition allows extending the approach to distributions closed under the convolution or even into the distributionally robust setting by employing existing results on distributionally robust knapsack problems. Finally, we would like to note that there are several interesting questions left open. Namely, does a specific list scheduling method have approximation guarantees, or can it be arbitrarily bad or whether the instances with agreeable jobs (e.g., smaller mean time implies smaller variance) could be solved faster?

Chapter 6

Conclusion

In this work, we have studied different means of integrating full distributional knowledge of processing times into scheduling problems. One group of methods proposed in this work is inspired by techniques used in statistical machine learning. For example, we have investigated how different vector norms affect the regularization term in a distributionally robust scheduling problem and its effect on the price and stability of solutions. The other core idea is that often, a suitable problem formulation is already a part of its successful solution. For example, we have transformed an inherently stochastic problem arising in message scheduling into a deterministic scheduling problem with alternatives. This modeling allowed us to efficiently solve the problem while avoiding the complexities of stochastic formulations.

In Chapter 2, we have used in-sample data to estimate the parameters of an ambiguity set to obtain a distributionally robust solution via different regularization techniques. In Chapter 3, we have proposed how to model jobs with uncertain processing times with an abstraction called F-shape, which discretizes the distribution function of processing times. This abstraction leads to schedules with alternatives, which allows achieving trade-offs between efficiency and robustness of the schedule. The resulting problem formulation is essentially a kind of packing problem that can be solved efficiently in practice. In Chapter 4, we have extended this model by job replication, which improves the execution probability of jobs in a schedule. Despite that, in the general case, the problem of calculation of the execution probability becomes hard while the cases of practical interest remain tractable. Finally, in Chapter 5, we have studied the problem where the distribution of processing times can be approximated by a normal distribution. The full distributional knowledge is utilized in the risk-averse objective function that maximizes the probability that all jobs are completed before a given common due date.

6.1 Fulfillment of the goals

Below we describe how the given goals were fulfilled.

1. *Study the related scheduling literature in robust, stochastic, and distributionally robust optimization fields. Identify new challenges, promising approaches, and possible improvements to existing modeling methods, problem formulations, and algorithms.*

   We surveyed the optimization paradigms for uncertain problems and their properties from a high-level perspective in Section 1.1. Focusing on particular challenges, in Section 2.2, we described some of the key properties of the expressiveness of the commonly used ambiguity sets in distributionally robust scheduling. These properties led us to a question of how crucial it is to solve problems with certain types of ambiguity sets optimally when the protection against the (unrealistic) worst-case distribution is not required. In Section 3.1.2 and Section 4.1.4, we have identified the lack of models

and scalable computational methods for non-preemptive static scheduling in mixed-criticality environments. Furthermore, we noted that the robustness of static mixed-criticality schedules should be further improved by considering job replication, which we have proposed in [122]. Considering the stochastic programming scheduling problems with normally distributed processing times, in Section 5.1.3 we have followed the suggestion proposed by [135] in designing more efficient heuristics for this problem. Moreover, it appeared to us that this kind of non-linear problem might benefit from applying the integer programming framework due to constantly improving solver technology.

2. *Develop formal descriptions of the deduced problems. Formulate the uncertainty models, constraints, and objective functions considering the distributional knowledge.*

   In Chapter 2, we dealt with the idea of solving distributionally robust problems in terms of different solution variance regularizations. A new renormalization technique for a fine-tuned control of the price and robustness trade-off was proposed in Section 2.4.2. Next, we have proposed to model the processing time uncertainty as a discretization of the distribution function in Chapter 3 where we have demonstrated it for the solution of a makespan scheduling problem. In [123], the above model was used to describe and solve a jitter minimization problem in periodic environments. In Chapter 4, we have introduced job replication as a mechanism for increasing the execution probability of jobs in the schedule. We have observed that the problem of computing the execution probability becomes closely connected to the probabilistic inference in suitably constructed Bayesian networks. Finally, in Chapter 5, we studied a parallel machine scheduling problem with a risk-averse objective function that involves the product of normal cumulative distribution functions.

3. *Propose heuristic and exact algorithms that compute robust schedules with respect to the processing time uncertainty.*

   We have proposed a scalable algorithm for a distributionally robust problem with independent jobs in Section 2.3.3, and we have extended it for dependent jobs in Section 2.3.5. In the case mixed-criticality environments, we have proposed an approximation algorithm in Section 3.3.1, an efficient mixed-integer linear programming model for the problem with two criticality levels in Section 3.3.2 and a branch-and-price decomposition for the problem with three criticality levels in Section 3.4. A heuristic algorithm for periodic mixed-criticality environments was proposed in [123]. For the computation of the execution probability in the presence of the job replication, we have proposed the reduction to the inference in Bayesian networks in Section 4.3.1. For the problem with a normal distribution of processing times, we have proposed a new list scheduling algorithm and a powerful genetic algorithm in Section 5.2.1. The genetic algorithm uses an intensification step implemented by an efficient algorithm for the special case of two machines, which is proposed in Section 5.4. To solve the general problem, we have proposed a branch-and-price decomposition in Section 5.3.

4. *Benchmark the developed algorithms by numerical experiments. Discuss the*

obtained results from the perspective of quality and time complexity. Demonstrate the scalability of the proposed approaches and discuss the robustness of their solutions.

We have compared the price and robustness trade-offs for our distributionally robust algorithms in Section 2.5.3. We have observed that the regularization in terms of ℓ_1 achieves almost identical out-of-sample trade-offs as ℓ_2 formulation initially proposed by [44] while being much faster. Furthermore, in Section 2.5.4, we have conducted the experiments to assess the benefits of utilizing general covariance matrices in the case of dependent jobs in an imperfect-knowledge setting. In Section 3.5.1, we have studied the average-case instance complexity of a mixed-criticality problem, showing a narrow phase transition where the difficult problem instances appear. In Section 3.5.2 and in Section 3.5.3 we benchmarked computational times of the proposed algorithms showing to outperform the former mixed-integer linear programming model proposed by [81]. For the problem with jobs with normally distributed processing times, we showed in Section 5.5.2 that our heuristic achieves better objective values if more computational resources are given. In Section 5.5.3, we study the scaling performance of the non-linear integer programming formulation. Furthermore, we tested the sensitivity with respect to instance parameters and the scalability of our branch-and-price algorithm for the general problem in Section 5.5.4 and Section 5.5.5. The results showed that our method scales better than the former branch-and-bound [135], especially with the increasing number of machines. Finally, the performance of the algorithm for two machines is measured in Section 5.5.6. We have shown its scalability for instances up to hundreds of jobs and the ability to achieve almost linear parallel speed-up.

6.2 Future work

It seems to us that applying objective regularization techniques similar to what we showed in Chapter 2 might be viable for other scheduling problems as well. The point being is that for many instances of the specific problem, the solution obtained with different regularization terms might be almost as good as an optimal one for the original problem statement derived from the chosen uncertainty model. What is more, one could even sample more solutions from the surrogate problem and evaluate them by means of the original formulation and choose the best one. This idea is also connected with our observation that in the existing literature, we have identified a gap between the study of tractable but conservative over-approximations of ambiguity sets, e.g., [181, 30], and the design of heuristic algorithms of exact intractable formulations of the ambiguity. It appears to us that the relation between an exact solution of conservative over-approximations and approximate solutions of exact formulations of ambiguity sets is still not well-researched. Hence we propose a systematic study and benchmarking of heuristic approaches to (distributionally) robust scheduling with expressive ambiguity sets. In that respect, an attractive option seems to be Wasserstein ambiguity sets [70].

Bibliography

[1] I. Agirre et al. "Fitting Software Execution-Time Exceedance into a Residual Random Fault in ISO-26262". In: *IEEE Transactions on Reliability* 67.3 (Sept. 2018), pp. 1314–1327. DOI: 10.1109/TR.2018.2828222.

[2] S. Alimoradi, M. Hematian, and G. Moslehi. "Robust scheduling of parallel machines considering total flow time". In: *Computers & Industrial Engineering* 93 (2016), pp. 152–161. ISSN: 0360-8352.

[3] Madhukar Anand et al. "State-based Scheduling with Tree Schedules: Analysis and Evaluation". In: *Real-Time Syst.* 48.4 (July 2012), pp. 430–462. ISSN: 0922-6443. DOI: 10.1007/s11241-012-9151-3.

[4] **Antonín Novák**, Andrzej Gnatowski, and Premysl Sucha. "Distributionally robust scheduling algorithms for total flow time minimization on parallel machines using norm regularizations". In: *European Journal of Operational Research* 302.2 (2022), pp. 438–455. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2022.01.002.

[5] **Antonín Novák**, Zdenek Hanzalek, and Premysl Sucha. "Scheduling of safety-critical time-constrained traffic with F-shaped messages". In: *2017 IEEE 13th International Workshop on Factory Communication Systems (WFCS)*. 2017, pp. 1–9. DOI: 10.1109/WFCS.2017.7991948.

[6] **Antonín Novák**, Premysl Sucha, and Zdenek Hanzalek. "Exact Approach to the Scheduling of F-shaped Tasks with Two and Three Criticality Levels". In: *Proceedings of the 6th International Conference on Operations Research and Enterprise Systems - ICORES,* 2017, pp. 160–170. ISBN: 978-989-758-218-9. DOI: 10.5220/0006198101600170.

[7] **Antonín Novák**, Premysl Sucha, and Zdenek Hanzalek. "Scheduling with uncertain processing times in mixed-criticality systems". In: *European Journal of Operational Research* 279.3 (2019), pp. 687–703. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2019.05.038.

[8] **Antonín Novák** et al. "Scheduling jobs with normally distributed processing times on parallel machines". In: *European Journal of Operational Research* 297.2 (2022), pp. 422–441. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2021.05.011.

[9] Tomas Baca et al. "Autonomous landing on a moving vehicle with an unmanned aerial vehicle". In: *Journal of Field Robotics* 36.5 (2019), pp. 874–891.

[10] N. Balakrishnan and Bruno Scarpa. "Multivariate measures of skewness for the skew-normal distribution". In: *Journal of Multivariate Analysis* 104.1 (2012), pp. 73–87. ISSN: 0047-259X.

[11] Cynthia Barnhart et al. "Branch-and-Price: Column Generation for Solving Huge Integer Programs". In: *Operations Research* 46.3 (1998), pp. 316–329. DOI: 10.1287/opre.46.3.316.

[12] S. Baruah et al. "Scheduling Real-Time Mixed-Criticality Jobs". In: *IEEE Transactions on Computers* 61.8 (Aug. 2012), pp. 1140–1152. ISSN: 0018-9340. DOI: 10.1109/TC.2011.142.

[13] Sanjoy Baruah. "Predictability Issues in Mixed-Criticality Real-Time Systems". In: *Principles of Modeling: Essays Dedicated to Edward A. Lee on the Occasion of His 60th Birthday*. Ed. by Marten Lohstroh, Patricia Derler, and Marjan Sirjani. Cham: Springer International Publishing, 2018, pp. 77–87. ISBN: 978-3-319-95246-8. DOI: 10.1007/978-3-319-95246-8_5.

[14] Sanjoy Baruah and Gerhard Fohler. "Certification-cognizant time-triggered scheduling of mixed-criticality systems". In: *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*. IEEE. 2011, pp. 3–12.

[15] Sanjoy Baruah and Zhishan Guo. "Mixed-criticality job models: a comparison". In: *Proceedings of the 3rd International Workshop on Mixed Criticality Systems, RTSS*. IEEE. 2015, pp. 5–9.

[16] Sanjoy Baruah et al. "Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems". In: *Journal of the ACM (JACM)* 62.2 (2015), p. 14.

[17] Güzin Bayraksan and David K Love. "Data-driven stochastic programming using phi-divergences". In: *The Operations Research Revolution*. INFORMS, 2015, pp. 1–19.

[18] Güzin Bayraksan, David P Morton, and Amit Partani. "Simulation-based optimality tests for stochastic programs". In: *Stochastic Programming*. Springer, 2010, pp. 37–55.

[19] James C Bean et al. "Matchup scheduling with multiple resources, release dates and disruptions". In: *Operations Research* 39.3 (1991), pp. 470–483.

[20] Lalatendu Behera and Purandar Bhaduri. "Time-Triggered Scheduling for Multiprocessor Mixed-Criticality Systems". In: *Distributed Computing and Internet Technology*. Cham: Springer International Publishing, 2018, pp. 135–151. ISBN: 978-3-319-72344-0.

[21] Ron Bell. "Introduction to IEC 61508". In: *Proceedings of the 10th Australian workshop on Safety critical systems and software-Volume 55*. Australian Computer Society, Inc. 2006, pp. 3–12.

[22] Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust optimization*. Vol. 28. Princeton University Press, 2009.

[23] Aharon Ben-Tal and Eithan Hochman. "More bounds on the expectation of a convex function of a random variable". In: *Journal of Applied Probability* 9.4 (1972), pp. 803–812.

[24] Aharon Ben-Tal and Arkadi Nemirovski. "Robust convex optimization". In: *Mathematics of operations research* 23.4 (1998), pp. 769–805.

[25] Aharon Ben-Tal and Arkadi Nemirovski. "Robust solutions of linear programming problems contaminated with uncertain data". In: *Mathematical programming* 88.3 (2000), pp. 411–424.

[26] Aharon Ben-Tal and Arkadi Nemirovski. "Robust solutions of uncertain linear programs". In: *Operations research letters* 25.1 (1999), pp. 1–13.

[27] Aharon Ben-Tal et al. “Robust solutions of optimization problems affected by uncertain probabilities”. In: *Management Science* 59.2 (2013), pp. 341–357.

[28] Yael Berstein and Shmuel Onn. “Nonlinear bipartite matching”. In: *Discrete Optimization* 5.1 (2008), pp. 53–65. ISSN: 1572-5286.

[29] Dimitris Bertsimas, David B Brown, and Constantine Caramanis. “Theory and applications of robust optimization”. In: *SIAM review* 53.3 (2011), pp. 464–501.

[30] Dimitris Bertsimas, Vishal Gupta, and Nathan Kallus. “Data-driven robust optimization”. In: *Mathematical Programming* 167.2 (2018), pp. 235–292. DOI: 10.1007/s10107-017-1125-8.

[31] Dimitris Bertsimas and Melvyn Sim. “The price of robustness”. In: *Operations research* 52.1 (2004), pp. 35–53.

[32] Dimitris Bertsimas, Melvyn Sim, and Meilin Zhang. “Adaptive distributionally robust optimization”. In: *Management Science* 65.2 (2019), pp. 604–618.

[33] Hans-Georg Beyer and Bernhard Sendhoff. “Robust optimization–a comprehensive survey”. In: *Computer methods in applied mechanics and engineering* 196.33-34 (2007), pp. 3190–3218.

[34] John R Birge and Francois Louveaux. *Introduction to stochastic programming*. Springer Science & Business Media, 2011.

[35] Jacek Blazewicz et al. *Handbook on scheduling: from theory to applications*. Springer Science & Business Media, 2007.

[36] Michal Bouska et al. “Machine learning-driven scheduling algorithm for a single machine problem minimizing the total tardiness”. In: *European Journal of Operational Research* ().

[37] Michal Bouška et al. “Data-driven Algorithm for Scheduling with Total Tardiness”. In: *Proceedings of the 9th International Conference on Operations Research and Enterprise Systems - ICORES*. 2020, pp. 59–68. ISBN: 978-989-758-396-4. DOI: 10.5220/0008915300590068.

[38] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[39] Peter Brucker. *Scheduling Algorithms*. Springer Berlin Heidelberg, 2007. ISBN: 978-3-540-69516-5.

[40] Alan Burns and Robert I. Davis. “A Survey of Research into Mixed Criticality Systems”. In: *ACM Comput. Surv.* 50.6 (Nov. 2017), 82:1–82:37. ISSN: 0360-0300. DOI: 10.1145/3131347.

[41] Alan Burns et al. “Robust mixed-criticality systems”. In: *IEEE Transactions on Computers* 67.10 (2018), pp. 1478–1491.

[42] Giuseppe Carlo Calafiore and Laurent El Ghaoui. “On distributionally robust chance-constrained linear programs”. In: *Journal of Optimization Theory and Applications* 130.1 (2006), pp. 1–22.

[43] Eugenia Ana Capota et al. "Towards Fully Jitterless Applications: Periodic Scheduling in Multiprocessor MCSs Using a Table-Driven Approach". In: *Applied Sciences* 10.19 (2020), p. 6702.

[44] Zhiqi Chang, Jian-Ya Ding, and Shiji Song. "Distributionally robust scheduling on parallel machines under moment uncertainty". In: *European Journal of Operational Research* 272.3 (2019), pp. 832–846. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2018.07.007.

[45] Zhiqi Chang et al. "Distributionally robust single machine scheduling with risk aversion". In: *European Journal of Operational Research* 256.1 (2017), pp. 261–274.

[46] Moses Charikar and Amit Sahai. "Dimension reduction in the ℓ_1 norm". In: *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proc.* IEEE. 2002, pp. 551–560.

[47] Abraham Charnes and William W Cooper. "Chance-constrained programming". In: *Management science* 6.1 (1959), pp. 73–79.

[48] Abraham Charnes and William W Cooper. "Programming with linear fractional functionals". In: *Naval Research logistics quarterly* 9.3-4 (1962), pp. 181–186.

[49] Jianqiang Cheng, Erick Delage, and Abdel Lisser. "Distributionally robust stochastic knapsack problem". In: *SIAM Journal on Optimization* 24.3 (2014), pp. 1485–1506.

[50] Sung-Jin Chung. "NP-completeness of the linear complementarity problem". In: *Journal of optimization theory and applications* 60.3 (1989), pp. 393–399.

[51] Nadia Creignou and Miki Hermann. "On P completeness of some counting problems". PhD thesis. INRIA, 1993.

[52] Paul Dagum and Michael Luby. "An optimal approximation algorithm for Bayesian inference". In: *Artificial Intelligence* 93.1-2 (1997), pp. 1–27.

[53] Richard L Daniels and Janice E Carrillo. "β-Robust scheduling for single-machine systems with uncertain processing times". In: *IIE transactions* 29.11 (1997), pp. 977–985.

[54] R. I. Davis, S. Altmeyer, and A. Burns. "Mixed Criticality Systems with Varying Context Switch Costs". In: *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. Apr. 2018, pp. 140–151. DOI: 10.1109/RTAS.2018.00024.

[55] Erick Hans Delage. "Distributionally robust optimization in context of data-driven problems". PhD thesis. Stanford University, 2009.

[56] Paolo Dell'Olmo et al. "A 13/12 approximation algorithm for bin packing with extendable bins". In: *Information Processing Letters* 65.5 (1998), pp. 229–233.

[57] Maxence Delorme, Manuel Iori, and Silvano Martello. "Bin packing and cutting stock problems: Mathematical models and exact algorithms". In: *European Journal of Operational Research* 255.1 (2016), pp. 1–20. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2016.04.030.

[58] Guy Desaulniers, Jacques Desrosiers, and Marius M Solomon. *Column generation*. Vol. 5. Springer Science & Business Media, 2006.

[59] Jacques Desrosiers and Marco E Lübbecke. "A primer in column generation". In: *Column generation*. Springer, 2005, pp. 1–32.

[60] Benjamin Doerr et al. "Fast genetic algorithms". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2017, pp. 777–784.

[61] Stefan Draskovic, Pengcheng Huang, and Lothar Thiele. "On the Safety of Mixed-Criticality Scheduling". In: *Proceedings of the 4th International Workshop on Mixed Criticality Systems, RTSS*. IEEE. Porto, Portugal, Nov. 2016, pp. 19–24.

[62] Jitka Dupačová. "On minimax solutions of stochastic linear programming problems". eng. In: *Časopis pro pěstování matematiky* 091.4 (1966), pp. 423–430.

[63] Jitka Dupačová. "The minimax approach to stochastic programming and an illustrative application". In: *Stochastics: An International Journal of Probability and Stochastic Processes* 20.1 (1987), pp. 73–88.

[64] Christoph Dürr et al. "The triangle scheduling problem". In: *Journal of Scheduling* (2017), pp. 1–8.

[65] Jan Dvořák and Zdeněk Hanzálek. "Using Two Independent Channels With Gateway for FlexRay Static Segment Scheduling". In: *IEEE Transactions on Industrial Informatics* 12.5 (Oct. 2016), pp. 1887–1895. ISSN: 1551-3203. DOI: 10.1109/TII.2016.2571667.

[66] Michael TM Emmerich and André H Deutz. "A tutorial on multiobjective optimization: fundamentals and evolutionary methods". In: *Natural computing* 17.3 (2018), pp. 585–609.

[67] Leah Epstein and Jiri Sgall. "Approximation Schemes for Scheduling on Uniformly Related and Identical Parallel Machines". In: *Algorithmica* 39.1 (May 2004), pp. 43–57. ISSN: 1432-0541. DOI: 10.1007/s00453-003-1077-7.

[68] Peyman Mohajerin Esfahani and Daniel Kuhn. "Data-driven distributionally robust optimization using the Wasserstein metric: Performance guarantees and tractable reformulations". In: *Mathematical Programming* 171.1 (2018), pp. 115–166.

[69] Claudio Gambella, Bissan Ghaddar, and Joe Naoum-Sawaya. "Optimization Problems for Machine Learning: A Survey". In: *European Journal of Operational Research* (2020). ISSN: 0377-2217.

[70] Rui Gao and Anton J Kleywegt. "Distributionally robust stochastic optimization with Wasserstein distance". In: *arXiv preprint arXiv:1604.02199* (2016).

[71] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990. ISBN: 0716710455.

[72] Dongdong Ge, Xiaoye Jiang, and Yinyu Ye. "A note on the complexity of ℓ_p minimization". In: *Mathematical programming* 129.2 (2011), pp. 285–299.

[73] Solomon W Golomb. *Polyominoes: puzzles, patterns, problems, and packings*. Princeton University Press, 1996.

[74] Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*. Elsevier, 2004.

[75] Ronald L Graham et al. "Optimization and approximation in deterministic sequencing and scheduling: a survey". In: *Annals of discrete mathematics* 5 (1979), pp. 287–326.

[76] Haipeng Guo and William Hsu. "A survey of algorithms for real-time Bayesian network inference". In: *Join Workshop on Real Time Decision Support and Diagnosis Systems*. 2002.

[77] Racha El-Hajj et al. "A PSO based algorithm with an efficient optimal split procedure for the multiperiod vehicle routing problem with profit". In: *Annals of Operations Research* (2020), pp. 1–36.

[78] Idir Hamaz, Laurent Houssin, and Sonia Cafieri. "A Branch-and-Bound Procedure for the Robust Cyclic Job Shop Problem". In: *International Symposium on Combinatorial Optimization*. Springer. 2018, pp. 228–240.

[79] Idir Hamaz, Laurent Houssin, and Sonia Cafieri. "A robust basic cyclic scheduling problem". In: *EURO Journal on Computational Optimization* 6.3 (Sept. 2018), pp. 291–313. ISSN: 2192-4414. DOI: 10.1007/s13675-018-0100-3.

[80] Zdenek Hanzalek and Premysl Sucha. "Time symmetry of resource constrained project scheduling with general temporal constraints and take-give resources". In: *Annals of Operations Research* 248.1-2 (2017), pp. 209–237.

[81] Zdenek Hanzalek, Tomas Tunys, and Premysl Sucha. "An analysis of the non-preemptive mixed-criticality match-up scheduling problem". In: *Journal of Scheduling* 19.5 (Oct. 2016), pp. 601–607. ISSN: 1099-1425. DOI: 10.1007/s10951-016-0468-y.

[82] Lukas Hejl et al. "Minimizing the weighted number of tardy jobs on a single machine: Strongly correlated instances". In: *European Journal of Operational Research* 298.2 (2022), pp. 413–424. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2021.07.002.

[83] Willy Herroelen and Roel Leus. "Project scheduling under uncertainty: Survey and research potentials". In: *European Journal of Operational Research* 165.2 (2005). Project Management and Scheduling, pp. 289–306. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2004.04.002.

[84] Piotr Indyk. "Stable distributions, pseudorandom generators, embeddings and data stream computation". In: *Proceedings 41st Annual Symposium on Foundations of Computer Science*. IEEE. 2000, pp. 189–197.

[85] Ishfaq Ahmad and Yu-Kwong Kwok. "On exploiting task duplication in parallel program scheduling". In: *IEEE Transactions on Parallel and Distributed Systems* 9.9 (Sept. 1998), pp. 872–892. ISSN: 1045-9219. DOI: 10.1109/71.722221.

[86] Fernando Jaramillo, Busra Keles, and Murat Erkoc. "Modeling single machine preemptive scheduling problems for computational efficiency". In: *Annals of Operations Research* 285.1 (2020), pp. 197–222.

[87] Seo-Hyun Jeon et al. "Automotive hardware development according to ISO 26262". In: *Advanced Communication Technology (ICACT), 2011 13th International Conference on*. IEEE. 2011, pp. 588–592.

[88] Xi Jin et al. "Mixed-criticality Industrial Data Scheduling on 5G NR". In: *IEEE Internet of Things Journal* (2021).

[89] Roy Jonker and Anton Volgenant. "A shortest augmenting path algorithm for dense and sparse linear assignment problems". In: *Computing* 38.4 (1987), pp. 325–340.

[90] Hans Kellerer, Ulrich Pferschy, and David Pisinger. "Introduction to NP-Completeness of knapsack problems". In: *Knapsack problems*. Springer, 2004, pp. 483–493.

[91] Burcu B. Keskin, Sharif H. Melouk, and Ivan L. Meyer. "A simulation-optimization approach for integrated sourcing and inventory decisions". In: *Computers & Operations Research* 37.9 (2010), pp. 1648–1661. ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2009.12.012.

[92] Jaroslav Klapálek et al. "Car Racing Line Optimization with Genetic Algorithm using Approximate Homeomorphism". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021, pp. 601–607. DOI: 10.1109/IROS51168.2021.9636503.

[93] H. Kopetz. "Event-triggered versus time-triggered real-time systems". In: *Operating Systems of the 90s and Beyond: International Workshop Dagstuhl Castle, Germany, July 8–12 1991 Proceedings*. Ed. by Arthur Karshmer and Jurgen Nehmer. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 86–101. ISBN: 978-3-540-46630-7. DOI: 10.1007/BFb0024530.

[94] Samuel Kotz, Tomaz J Kozubowski, and Krzysztof Podgórski. "Asymmetric multivariate Laplace distribution". In: *The Laplace distribution and generalizations*. Springer, 2001, pp. 239–272.

[95] Christos Koulamas. "The single-machine total tardiness scheduling problem: Review and extensions". In: *European Journal of Operational Research* 202.1 (2010), pp. 1–7. ISSN: 0377-2217.

[96] Panos Kouvelis, Richard L Daniels, and George Vairaktarakis. "Robust scheduling of a two-machine flow shop with uncertain processing times". In: *IIE Transactions* 32.5 (2000), pp. 421–432.

[97] Annamária Kovács. "New Approximation Bounds for LPT Scheduling". In: *Algorithmica* 57.2 (June 2010), pp. 413–433. ISSN: 1432-0541. DOI: 10.1007/s00453-008-9224-9.

[98] Daniel Kowalczyk and Roel Leus. "A branch-and-price algorithm for parallel machine scheduling using ZDDs and generic branching". In: *INFORMS Journal on Computing* 30.4 (2018), pp. 768–782.

[99] Arthur Kramer, Mauro Dell'Amico, and Manuel Iori. "Enhanced arc-flow formulations to minimize weighted completion time on identical parallel machines". In: *European Journal of Operational Research* 275.1 (2019), pp. 67–79. ISSN: 0377-2217.

[100] Daniel Kuhn et al. "Wasserstein distributionally robust optimization: Theory and applications in machine learning". In: *Operations Research & Management Science in the Age of Analytics*. INFORMS, 2019, pp. 130–166.

[101] Johan Kwisthout, Hans L Bodlaender, and Linda C van der Gaag. "The Necessity of Bounded Treewidth for Efficient Inference in Bayesian Networks." In: *ECAI*. Vol. 215. 2010, pp. 237–242.

[102] Stefano Leonardi and Danny Raz. "Approximating total flow time on parallel machines". In: *Journal of Computer and System Sciences* 73.6 (2007), pp. 875–891. ISSN: 0022-0000.

[103] Asaf Levin. *Approximation schemes for the generalized extensible bin packing problem*. 2019. arXiv: 1905.09750 [cs.DS].

[104] Yan-Feng Li et al. "Reliability analysis of multi-state systems with common cause failures based on Bayesian network and fuzzy probability". In: *Annals of Operations Research* (2019), pp. 1–15.

[105] Jeff Linderoth, Alexander Shapiro, and Stephen Wright. "The empirical behavior of sampling methods for stochastic programming". In: *Annals of Operations Research* 142.1 (2006), pp. 215–241.

[106] Helena Ramalhinho Lourenco. "Job-shop scheduling: Computational study of local search and large-step optimization methods". In: *European Journal of Operational Research* 83.2 (1995), pp. 347–364.

[107] Chung-Cheng Lu, Shih-Wei Lin, and Kuo-Ching Ying. "Robust scheduling on a single machine to minimize total flow time". In: *Computers & Operations Research* 39.7 (2012), pp. 1682–1691.

[108] Marco E. Lübbecke and Jacques Desrosiers. "Selected Topics in Column Generation". In: *Operations Research* 53.6 (2005), pp. 1007–1023. DOI: 10.1287/opre.1050.0234.

[109] Satya N. Majumdar and Arnab Pal. *Extreme value statistics of correlated random variables*. 2014. arXiv: 1406.6768 [cond-mat.stat-mech].

[110] O.L. Mangasarian and R.R. Meyer. "Absolute value equations". In: *Linear Algebra and its Applications* 419.2 (2006), pp. 359–367. ISSN: 0024-3795.

[111] H.M. Markowitz. "Portfolio selection". In: *Journal of Finance* 7 (1952), pp. 77–91.

[112] Silvano Martello, David Pisinger, and Paolo Toth. "New trends in exact algorithms for the 0–1 knapsack problem". In: *European Journal of Operational Research* 123.2 (2000), pp. 325–332. ISSN: 0377-2217. DOI: http://dx.doi.org/10.1016/S0377-2217(99)00260-X.

[113] Joel Matějka et al. “Combining PREM Compilation and ILP Scheduling for High-performance and Predictable MPSoC Execution”. In: *Proceedings of the 9th International Workshop on Programming Models and Applications for Multicores and Manycores*. PMAM’18. Vienna, Austria: ACM, 2018, pp. 11–20. ISBN: 978-1-4503-5645-9. DOI: 10.1145/3178442.3178444.

[114] Shumail Mazahir and Amir Ardestani-Jaafari. “Robust global sourcing under compliance legislation”. In: *European Journal of Operational Research* 284.1 (2020), pp. 152–163. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2019.12.017.

[115] Yasemin Merzifonluoğlu, Joseph Geunes, and H. Edwin Romeijn. “The static stochastic knapsack problem with normally distributed item sizes”. In: *Mathematical Programming* 134.2 (Sept. 2012), pp. 459–489. ISSN: 1436-4646. DOI: 10.1007/s10107-011-0443-5.

[116] David P Morton and R Kevin Wood. “On a stochastic knapsack problem and generalizations”. In: *Advances in computational and stochastic optimization, logic programming, and heuristic search*. Springer, 1998, pp. 149–168.

[117] Alix Munier and Francis Sourd. “Scheduling chains on a single machine with non-negative time lags”. In: *Mathematical Methods of Operations Research* 57.1 (Apr. 2003), pp. 111–123. ISSN: 1432-5217. DOI: 10.1007/s001860200242.

[118] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[119] Kevin P Murphy, Yair Weiss, and Michael I Jordan. “Loopy belief propagation for approximate inference: An empirical study”. In: *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc. 1999, pp. 467–475.

[120] Alfredo Navarra and Cristina M. Pinotti. “Online knapsack of unknown capacity: How to optimize energy consumption in smartphones”. In: *Theoretical Computer Science* 697 (2017), pp. 98–109. ISSN: 0304-3975.

[121] Shengsheng Niu et al. “Distributionally robust single machine scheduling with the total tardiness criterion”. In: *Computers & Operations Research* 101 (2019), pp. 13–28. ISSN: 0305-0548.

[122] **Antonín Novák** and Zdenek Hanzalek. “Computing the execution probability of jobs with replication in mixed-criticality schedules”. In: *Annals of Operations Research* (2022). DOI: 10.1007/s10479-021-04445-x.

[123] **Antonín Novák**, Premysl Sucha, and Zdenek Hanzalek. “Efficient Algorithm for Jitter Minimization in Time-Triggered Periodic Mixed-Criticality Message Scheduling Problem”. In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. RTNS ’16. Brest, France: Association for Computing Machinery, 2016, pp. 23–31. ISBN: 9781450347877. DOI: 10.1145/2997465.2997481.

[124] Roman Obermaisser et al. “Error containment in the time-triggered system-on-a-chip architecture”. In: *Embedded System Design: Topics, Techniques and Trends*. Springer, 2007, pp. 339–352.

[125] Christos H Papadimitriou and Mihalis Yannakakis. "Towards an architecture-independent analysis of parallel algorithms". In: *SIAM journal on computing* 19.2 (1990), pp. 322–328.

[126] R. Paredes et al. "Principled network reliability approximation: A counting-based approach". In: *Reliability Engineering & System Safety* 191 (2019), p. 106472. ISSN: 0951-8320. DOI: https://doi.org/10.1016/j.ress.2019.04.025.

[127] Zhi Pei et al. "Target-based distributionally robust optimization for single machine scheduling". In: *European Journal of Operational Research* 299.2 (2022), pp. 420–431. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2021.08.034.

[128] Ulrich Pferschy and Joachim Schauer. "The Knapsack Problem with Conflict Graphs." In: *J. Graph Algorithms Appl.* 13.2 (2009), pp. 233–249.

[129] Michael L Pinedo. *Scheduling: theory, algorithms, and systems*. Springer, 2016.

[130] Akram Pishevar and R Tavakkoi. "β-robust parallel machine scheduling with uncertain durations". In: *Universal Journal of Industrial and Business Management* 2.3 (2014), pp. 69–74.

[131] Warren B Powell. "A unified framework for stochastic optimization". In: *European Journal of Operational Research* 275.3 (2019), pp. 795–821.

[132] Warren B Powell. "Clearing the jungle of stochastic optimization". In: *Bridging data and decisions*. Informs, 2014, pp. 109–137.

[133] Xiangtong Qi, Jonathan F Bard, and Gang Yu. "Disruption management for machine scheduling: the case of SPT schedules". In: *International Journal of Production Economics* 103.1 (2006), pp. 166–184.

[134] Hamed Rahimian and Sanjay Mehrotra. "Distributionally robust optimization: A review". In: *arXiv preprint arXiv:1908.05659* (2019).

[135] Mohammad Ranjbar, Morteza Davari, and Roel Leus. "Two branch-and-bound algorithms for the robust parallel machine scheduling problem". In: *Computers & Operations Research* 39.7 (2012), pp. 1652–1660. ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2011.09.019.

[136] J. Rapin and O. Teytaud. *Nevergrad - A gradient-free optimization platform*. https://GitHub.com/FacebookResearch/Nevergrad. 2018.

[137] R Tyrrell Rockafellar. "Coherent approaches to risk in optimization under uncertainty". In: *OR Tools and Applications: Glimpses of Future Technologies*. Informs, 2007, pp. 38–61.

[138] R Tyrrell Rockafellar, Stanislav Uryasev, et al. "Optimization of conditional value-at-risk". In: *Journal of Risk* 2 (2000), pp. 21–42.

[139] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 2016.

[140] Andrzej Ruszczyński and Alexander Shapiro. "Stochastic Programming Models". In: *Stochastic Programming*. Vol. 10. Handbooks in Operations Research and Management Science. Elsevier, 2003, pp. 1–64. DOI: https://doi.org/10.1016/S0927-0507(03)10001-1.

[141] DM Ryan and EA Foster. "An integer programming approach to scheduling". In: *Computer Scheduling of Public Transport* (1981), pp. 269–280.

[142] Ihsan Sabuncuoglu and M Bayız. "Analysis of reactive scheduling problems in a job shop environment". In: *European Journal of operational research* 126.3 (2000), pp. 567–586.

[143] Guillaume Sagnol et al. "Stochastic Extensible Bin Packing". In: *arXiv preprint arXiv:2002.00060* (2020).

[144] Nikolaos V Sahinidis. "Optimization under uncertainty: state-of-the-art and opportunities". In: *Computers & Chemical Engineering* 28.6 (2004), pp. 971–983.

[145] Ahmed Saif and Samir Elhedhli. "Cold supply chain design with environmental considerations: A simulation-optimization approach". In: *European Journal of Operational Research* 251.1 (2016), pp. 274–287. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2015.10.056.

[146] Tian Sang, Paul Bearne, and Henry Kautz. "Performing Bayesian Inference by Weighted Model Counting". In: *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 1*. AAAI'05. Pittsburgh, Pennsylvania: AAAI Press, 2005, pp. 475–481. ISBN: 1-57735-236-x.

[147] José A Santiváñez and Emanuel Melachrinoudis. "Reliable maximin–maxisum locations for maximum service availability on tree networks vulnerable to disruptions". In: *Annals of Operations Research* 286.1 (2020), pp. 669–701.

[148] Herbert Scarf. "A min-max solution of an inventory problem". In: *Studies in the mathematical theory of inventory and production* (1958).

[149] Rüdiger Schultz. "Stochastic programming with integer variables". In: *Mathematical Programming* 97.1 (2003), pp. 285–309.

[150] M Schumer and Kenneth Steiglitz. "Adaptive step size random search". In: *IEEE Transactions on Automatic Control* 13.3 (1968), pp. 270–276.

[151] Yasmina Seddik and Zdenek Hanzalek. "Match-up scheduling of mixed-criticality jobs: Maximizing the probability of jobs execution". In: *European Journal of Operational Research* 262.1 (2017), pp. 46–59. ISSN: 0377-2217. DOI: http://dx.doi.org/10.1016/j.ejor.2017.03.054.

[152] Dvir Shabtay, Nufar Gaspar, and Moshe Kaspi. "A survey on offline scheduling with rejection". In: *Journal of Scheduling* 16.1 (2013), pp. 3–28.

[153] Chao Shang and Fengqi You. "Distributionally robust optimization for planning and scheduling under uncertainty". In: *Computers & Chemical Engineering* 110 (2018), pp. 53–68. ISSN: 0098-1354.

[154] Alexander Shapiro. "Monte Carlo sampling methods". In: *Handbooks in operations research and management science* 10 (2003), pp. 353–425.

[155] Alexander Shapiro. "On Duality Theory of Conic Linear Problems". In: *Semi-Infinite Programming: Recent Advances*. Ed. by Miguel Á. Goberna and Marco A. López. Boston, MA: Springer US, 2001, pp. 135–165. ISBN: 978-1-4757-3403-4.

[156] Alexander Shapiro and Arkadi Nemirovski. "On complexity of stochastic programming problems". In: *Continuous optimization*. Springer, 2005, pp. 111–146.

[157] Karmel S Shehadeh. "Data-Driven Distributionally Robust Surgery Planning in Flexible Operating Rooms Over a Wasserstein Ambiguity". In: *arXiv preprint arXiv:2103.15221* (2021).

[158] Karmel S. Shehadeh, Amy E.M. Cohn, and Ruiwei Jiang. "A distributionally robust optimization approach for outpatient colonoscopy scheduling". In: *European Journal of Operational Research* 283.2 (2020), pp. 549–561. ISSN: 0377-2217.

[159] Sang-Oh Shim and Yeong-Dae Kim. "Scheduling on parallel identical machines to minimize total tardiness". In: *European Journal of Operational Research* 177.1 (2007), pp. 135–146. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2005.09.038.

[160] Martin Skutella and Marc Uetz. "Stochastic Machine Scheduling with Precedence Constraints". In: *SIAM Journal on Computing* 34.4 (2005), pp. 788–802. DOI: 10.1137/S0097539702415007.

[161] Stephen F Smith. "Reactive scheduling systems". In: *Intelligent scheduling systems*. Springer, 1995, pp. 155–192.

[162] Kate Smith-Miles and Leo Lopes. "Measuring instance difficulty for combinatorial optimization problems". In: *Computers & Operations Research* 39.5 (2012), pp. 875–889. ISSN: 0305-0548. DOI: http://dx.doi.org/10.1016/j.cor.2011.07.006.

[163] Karuturi Sneha and Gowda M Malle. "Research on software testing techniques and software automation testing tools". In: *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*. 2017, pp. 77–81.

[164] Christian Sohler and David P Woodruff. "Subspace embeddings for the l1-norm with applications". In: *Proceedings of the forty-third annual ACM symposium on Theory of computing*. 2011, pp. 755–764.

[165] Guopeng Song, Daniel Kowalczyk, and Roel Leus. "The robust machine availability problem–bin packing under uncertainty". In: *IISE Transactions* 50.11 (2018), pp. 997–1012.

[166] Allen L Soyster. "Convex programming with set-inclusive constraints and applications to inexact linear programming". In: *Operations research* 21.5 (1973), pp. 1154–1157.

[167] Seçil Sözüer and Aurélie C Thiele. "The state of robust optimization". In: *Robustness analysis in decision aiding, optimization, and analytics*. Springer, 2016, pp. 89–112.

[168] Richard Stec et al. “Scheduling Jobs with Stochastic Processing Time on Parallel Identical Machines”. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 5628–5634. DOI: 10.24963/ijcai.2019/781.

[169] Liangde Tao, Lin Chen, and Guochuan Zhang. “Scheduling Stochastic Jobs-Complexity and Approximation Algorithms”. In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 31. 2021, pp. 367–375.

[170] Jens Theis, Gerhard Fohler, and Sanjoy Baruah. “Schedule table generation for time-triggered mixed criticality systems”. In: *Proceedings of the 1st International Workshop on Mixed Criticality Systems, RTSS* (2013), pp. 79–84.

[171] Roman Václavík et al. “Accelerating the Branch-and-Price Algorithm Using Machine Learning”. In: *European Journal of Operational Research* 271.3 (2018), pp. 1055–1069. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2018.05.046.

[172] L.G. Valiant. “The complexity of computing the permanent”. In: *Theoretical Computer Science* 8.2 (1979), pp. 189–201. ISSN: 0304-3975. DOI: https://doi.org/10.1016/0304-3975(79)90044-6.

[173] L.G. Valiant. “The Complexity of Enumeration and Reliability Problems”. In: *SIAM Journal on Computing* 8.3 (1979), pp. 410–421. DOI: 10.1137/0208032.

[174] Hector Vazquez-Leal et al. “High accurate simple approximation of normal distribution integral”. In: *Mathematical problems in engineering* 2012 (2012).

[175] Steve Vestal. “Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance”. In: *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*. IEEE. 2007, pp. 239–243.

[176] Marek Vlk, **Antonín Novák**, and Zdenek Hanzalek. “Makespan Minimization with Sequence-dependent Non-overlapping Setups”. In: *Proceedings of the 8th International Conference on Operations Research and Enterprise Systems - ICORES,* 2019, pp. 91–101. ISBN: 978-989-758-352-0. DOI: 10.5220/0007362700910101.

[177] Marek Vlk et al. “Non-overlapping Sequence-Dependent Setup Scheduling with Dedicated Tasks”. In: *Operations Research and Enterprise Systems*. Ed. by Greg H. Parlier, Federico Liberatore, and Marc Demange. Cham: Springer International Publishing, 2020, pp. 23–46. ISBN: 978-3-030-37584-3. DOI: 10.1007/978-3-030-37584-3_2.

[178] Ruosong Wang and David P. Woodruff. *Tight Bounds for ℓ_p Oblivious Subspace Embeddings*. 2018. arXiv: 1801.04414 [cs.DS].

[179] Yu Wang, Yu Zhang, and Jiafu Tang. “A distributionally robust optimization approach for surgery block allocation”. In: *European Journal of Operational Research* 273.2 (2019), pp. 740–753.

[180] Zizhuo Wang, Peter W Glynn, and Yinyu Ye. "Likelihood robust optimization for data-driven problems". In: *Computational Management Science* 13.2 (2016), pp. 241–261.

[181] Wolfram Wiesemann, Daniel Kuhn, and Melvyn Sim. "Distributionally robust convex optimization". In: *Operations Research* 62.6 (2014), pp. 1358–1376.

[182] Guoliang Xue and Yinyu Ye. "An Efficient Algorithm for Minimizing a Sum of p-Norms". In: *SIAM Journal on Optimization* 10.2 (2000), pp. 551–579.

[183] Wenzhuo Yang and Huan Xu. "Distributionally robust chance constraints for non-linear uncertainties". In: *Mathematical Programming* 155.1-2 (2016), pp. 231–265.

[184] Cheng-Ta Yeh. "Binary-state line assignment optimization to maximize the reliability of an information network under time and budget constraints". In: *Annals of Operations Research* 287.1 (2020), pp. 439–463.

[185] Lu Yin, Junlong Zhou, and Jin Sun. "A stochastic algorithm for scheduling bag-of-tasks applications on hybrid clouds under task duration variations". In: *Journal of Systems and Software* 184 (2022), p. 111123.

[186] Zheng Zhang, B. Denton, and X. Xie. "Branch and Price for Chance-Constrained Bin Packing". In: *INFORMS J. Comput.* 32 (2020), pp. 547–564.

Chapter A

Curriculum Vitae

Antonín Novák was born in Brandys nad Labem, Czech Republic, in 1991. He received a bachelor's degree with honors in 2013 in Cybernetics and Robotics at Czech Technical University in Prague, followed by a master's degree with honors in Computer Science in 2015. His work towards Ph.D. started the same year at the Department of Control Engineering in the area of scheduling under uncertainty.

Antonín's research interests span the area of scheduling with uncertain parameters and the connections of machine learning with stochastic and robust optimization problems at various levels. He has published six papers in impacted journals and several conference papers at spotlight venues such as IJCAI, IROS, and others. Since 2018 he regularly helped to review scheduling papers for ICAPS conference.

Antonín participated in several national and international projects funded by US Navy and the EU. He has served as the principal investigator of projects Factory of Future and Connected Motor Starter funded by the EU, the Ministry of Industry and Trade of the Czech Republic, and Eaton Corporation. Moreover, he worked on several occasions with industrial partners such as Skoda Auto, Porsche Engineering, and Beckman Coulter, where he applied his academic knowledge for the projects at the borders of machine learning and mathematical optimization.

Antonín is also known for his passion for teaching activities — he has created many new teaching materials and served as the head of labs of Combinatorial Optimization course since 2016, he received Dean's award for the best lab teacher in 2018, and in 2022, he has begun to develop and teach labs of Combinatorial Algorithms course. Two of his former master students have received Dean's award for outstanding master thesis.

Antonín Novák
Prague, August 2022

Chapter B

List of Author's Publications

Publications in Journals with Impact Factor

Michal Bouska, Premysl Sucha, **Antonín Novák**, and Zdenek Hanzalek. "Machine learning-driven scheduling algorithm for a single machine problem minimizing the total tardiness". In: *European Journal of Operational Research* ()
Coauthorship 25%, Under minor revision

Antonín Novák, Andrzej Gnatowski, and Premysl Sucha. "Distributionally robust scheduling algorithms for total flow time minimization on parallel machines using norm regularizations". In: *European Journal of Operational Research* 302.2 (2022), pp. 438–455. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2022.01.002
Coauthorship 45%, 0 citations (WoS/Scopus, without self-citations)

Antonín Novák and Zdenek Hanzalek. "Computing the execution probability of jobs with replication in mixed-criticality schedules". In: *Annals of Operations Research* (2022). DOI: 10.1007/s10479-021-04445-x
Coauthorship 70%, 0 citations (WoS/Scopus, without self-citations)

Antonín Novák, Premysl Sucha, Matej Novotny, Richard Stec, and Zdenek Hanzalek. "Scheduling jobs with normally distributed processing times on parallel machines". In: *European Journal of Operational Research* 297.2 (2022), pp. 422–441. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2021.05.011
Coauthorship 40%, 1 citations (WoS/Scopus, without self-citations)

Lukas Hejl, Premysl Sucha, **Antonín Novák**, and Zdenek Hanzalek. "Minimizing the weighted number of tardy jobs on a single machine: Strongly correlated instances". In: *European Journal of Operational Research* 298.2 (2022), pp. 413–424. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2021.07.002
Coauthorship 25%, 0 citations (WoS/Scopus, without self-citations)

Antonín Novák, Premysl Sucha, and Zdenek Hanzalek. "Scheduling with uncertain processing times in mixed-criticality systems". In: *European Journal of Operational Research* 279.3 (2019), pp. 687–703. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2019.05.038
Coauthorship 33%, 3 citations (WoS/Scopus, without self-citations)

Roman Václavík, **Antonín Novák**, Přemysl Šůcha, and Zdeněk Hanzálek. "Accelerating the Branch-and-Price Algorithm Using Machine Learning". In: *European Journal of Operational Research* 271.3 (2018), pp. 1055–1069. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2018.05.046
Coauthorship 25%, 9 citations (WoS/Scopus, without self-citations)

International Conferences

Jaroslav Klapálek, **Antonín Novák**, Michal Sojka, and Zdeněk Hanzálek. "Car Racing Line Optimization with Genetic Algorithm using Approximate Homeomorphism". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021, pp. 601–607. DOI: 10.1109/IROS51168.2021.9636503
Coauthorship 25%, 0 citations (WoS/Scopus, without self-citations)

Michal Bouška, **Antonín Novák**, Přemysl Šůcha, István Módos, and Zdeněk Hanzálek. "Data-driven Algorithm for Scheduling with Total Tardiness". In: *Proceedings of the 9th International Conference on Operations Research and Enterprise Systems - ICORES*. 2020, pp. 59–68. ISBN: 978-989-758-396-4. DOI: 10.5220/0008915300590068
Coauthorship 20%, 2 citations (WoS/Scopus, without self-citations)

Marek Vlk, **Antonín Novák**, Zdenek Hanzalek, and Arnaud Malapert. "Non-overlapping Sequence-Dependent Setup Scheduling with Dedicated Tasks". In: *Operations Research and Enterprise Systems*. Ed. by Greg H. Parlier, Federico Liberatore, and Marc Demange. Cham: Springer International Publishing, 2020, pp. 23–46. ISBN: 978-3-030-37584-3. DOI: 10.1007/978-3-030-37584-3_2
Coauthorship 40%, 1 citation (WoS/Scopus, without self-citations)

Richard Stec, **Antonín Novák**, Premysl Sucha, and Zdenek Hanzalek. "Scheduling Jobs with Stochastic Processing Time on Parallel Identical Machines". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 5628–5634. DOI: 10.24963/ijcai.2019/781
Coauthorship 45%, 2 citations (WoS/Scopus, without self-citations)

Marek Vlk, **Antonín Novák**, and Zdenek Hanzalek. "Makespan Minimization with Sequence-dependent Non-overlapping Setups". In: *Proceedings of the 8th International Conference on Operations Research and Enterprise Systems - ICORES,*. 2019, pp. 91–101. ISBN: 978-989-758-352-0. DOI: 10.5220/0007362700910101
Coauthorship 30%, the best student paper award, 1 citation (WoS/Scopus, without self-citations)

Antonín Novák, Zdenek Hanzalek, and Premysl Sucha. "Scheduling of safety-critical time-constrained traffic with F-shaped messages". In: *2017 IEEE 13th International Workshop on Factory Communication Systems (WFCS)*. 2017, pp. 1–9. DOI: 10.1109/WFCS.2017.7991948
Coauthorship 33%, 2 citations (WoS/Scopus, without self-citations)

Antonín Novák, Premysl Sucha, and Zdenek Hanzalek. "Exact Approach to the Scheduling of F-shaped Tasks with Two and Three Criticality Levels". In: *Proceedings of the 6th International Conference on Operations Research and Enterprise Systems - ICORES,*. 2017, pp. 160–170. ISBN: 978-989-758-218-9. DOI: 10.5220/0006198101600170
Coauthorship 33%, 1 citation (WoS/Scopus, without self-citations)

Antonín Novák, Premysl Sucha, and Zdenek Hanzalek. "Efficient Algorithm for Jitter Minimization in Time-Triggered Periodic Mixed-Criticality Message Scheduling Problem". In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. RTNS '16. Brest, France: Association for Computing Machinery, 2016, pp. 23–31. ISBN: 9781450347877. DOI: 10.1145/2997465.2997481
Coauthorship 33%, 7 citations (WoS/Scopus, without self-citations)

Antonín Novák
Prague, August 2022

This thesis studies scheduling problems with processing time uncertainty described by their distributions. The aim is the design of new modeling approaches, the study of their complexity, and the development of scalable algorithms that utilize full distributional knowledge for efficient and robust decision-making.

The main contributions are:

- We model a discretization of the cumulative distribution of the processing time with an F-shaped job.
- We propose the job replication for Mixed-Criticality scheduling to increase the execution probability of jobs.
- We characterize the complexity of distributionally robust scheduling problem with total flow time minimization in terms of different norm regularizations of the solution variance.
- We propose efficient exact and heuristic algorithms for the problems of Mixed-Criticality scheduling, scheduling with normally distributed processing times, and distributionally robust total flow time minimization with dependent jobs.

CTU

CZECH TECHNICAL
UNIVERSITY
IN PRAGUE