Scheduling with Uncertain Processing Times in Mixed-Criticality Systems

Antonin Novak*[1,2], Premysl Sucha[2], and Zdenek Hanzalek[2]

[1]Department of Control Engineering, Faculty of Electrical Engineering,
Czech Technical University in Prague, Czech Republic
[2]Czech Institute of Informatics, Robotics, and Cybernetics,
Czech Technical University in Prague, Czech Republic

Abstract

Many scheduling problems that can be identified inside safety-critical applications, such as in autonomous cars, tend to be mixed-critical. Such scheduling problems consider tasks to have different criticalities depending on the safety levels (activation of brakes vs. activation of air-conditioning). The biggest challenge in those scheduling problems arises from the uncertainty of processing times as it disturbs the predictability of the system and thus makes the certification of the system difficult. To overcome this uncertainty, we model the tasks to have multiple processing times concerning their criticality. This approach converts these scheduling problems into a deterministic scheduling with alternative processing times.

Here, we study an $\mathcal{NP}$-hard single machine scheduling problem with makespan minimization, where the non-preemptive tasks can have multiple processing times. To solve the problem, we propose an approximation algorithm, a novel mixed-integer linear programming block formulation, and an efficient exact branch-and-price decomposition for two criticality levels. Furthermore, we demonstrate that the optimal schedules are represented as trees, which enables to formulate an exact algorithm for the problem with three criticality levels. The efficiency of the proposed method is demonstrated for difficult problem instances with up to 1000 tasks. The experimental evaluation demonstrates that our algorithms have improved the results of the best-known method by nearly two orders of magnitude.

Keywords— Scheduling, Mixed-Criticality, Uncertain Processing Time, Branch-and-Price

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

*Corresponding author: antonin.novak@cvut.cz

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 2.

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 1 (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 2 (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 $\forall i, j \in \{1, \ldots, n\}, i \neq j$:*

$$\left(s_i + p_i^{(\min\{\mathcal{X}_i, \mathcal{X}_j\})} \leq s_j\right) \vee \left(s_j + p_j^{(\min\{\mathcal{X}_i, \mathcal{X}_j\})} \leq s_i\right). \tag{1}$$

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 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 3 (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 4 (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.,*

$$\begin{aligned} &\min_{\boldsymbol{s}} C_{max} \\ \textit{subject to} \quad & \\ & \textit{feasibility conditions } (1) \\ & \boldsymbol{s} \in \mathbb{N}_0^n. \end{aligned}$$

In the three-field Graham-Blazewicz scheduling notation [18], 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 [20].

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 [37], 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 [3, 4], Burns [8, 9] and Davis [10]. This line of research mostly deals with response time analysis of different scheduling policies considering preemptive tasks in so-called event-triggered systems [22].

Often cited disadvantage of complex event-triggered systems is their inability to be certified for safety-critical applications [21, 1]. Therefore, researches have turned their attention toward static scheduling in mixed-criticality systems [22, 36, 6] that solves the problem with certification and predictability. The problem with preemptive tasks with two criticality levels was studied in [6]. They proposed a heuristic algorithm that constructs a static schedule for multiple resources while considering precedence constraints. Hanzalek *et al.* [20] 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 [28]. Dürr *et al.* [13] 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 [32] 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 [29] such as uniformly related machines with makespan minimization (i.e., $Q||C_{\max}$ [23]). 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., $P||\sum T_j$ [34]. 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 3.2. 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 [27] 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 [16], 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 [11], 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 [17].

Taking a broader perspective, the problem in this study is related to stochastic optimization [31] due to the uncertainty of processing times [19]. Moreover, it contains aspects of task disruption [30] and rejection [33] due to flexible execution of schedules, and robust scheduling [7] 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 [20, 28] only; however they lack computationally efficient exact solution method, which is presented in this paper.

1.3 Contribution and paper outline

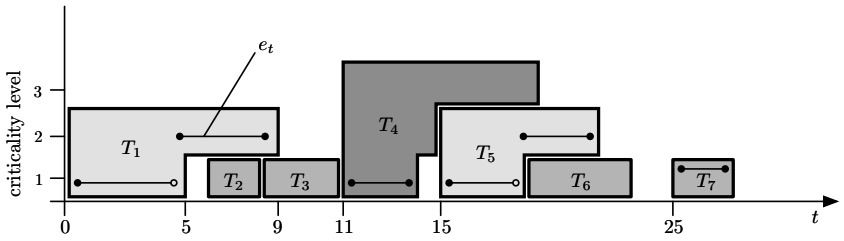
This paper focuses on the fundamental properties of F-shaped tasks that arise from scheduling problems in mixed-critical environments. We study the problem of the makespan minimization with F-shaped tasks (i.e., $1|mc = 2|C_{\max}$ and $1|mc = 3|C_{\max}$) and develop fast exact algorithms for solving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

- an approximation algorithm for the problem with two criticality levels (see Section 3.1),
- an exact efficient block MIP model that optimizes over non-isomorphic permutations (see Section 3.2),
- a branch-and-price algorithm with a pseudopolynomially solvable pricing problem (see Section 3.3),
- a structural result on optimal permutations and a generalization of the branch-and-price for more criticality levels (see Section 4.1 and Section 4.2), and
- the experimental evaluation of the proposed algorithms (see Section 5).

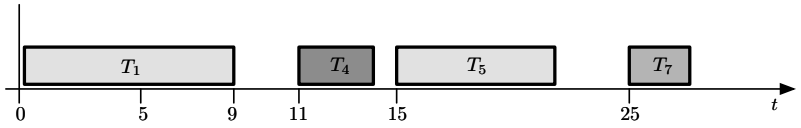
The rest of this paper is organized as follows. In Section 2, we describe our model for processing time uncertainty, explain the online execution of the schedule, and show real-life applications of the model. In Section 3, we derive a factor-two approximation algorithm for the problem with two criticality levels, unveil the structure of optimal schedules, and propose an efficient MIP formulation that optimizes over non-isomorphic permutations. In Section 4, we generalize the method for more criticality levels. The numerical experiments are described in Section 5, where we demonstrate the efficiency of our algorithms and bounds distinguishing easy instances from the difficult ones. The conclusions are drawn in Section 6.

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.

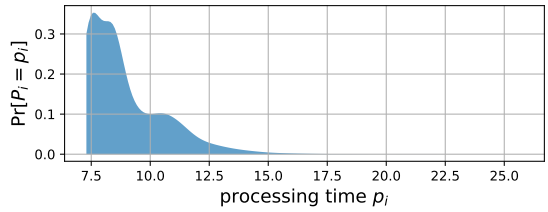


(b) Executed alternative.

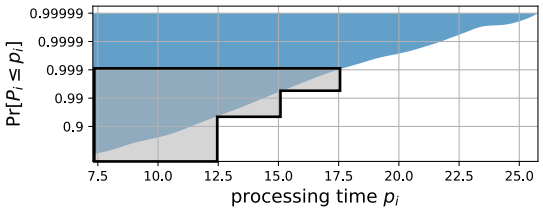
Figure 1: Schedule with F-shaped tasks and the executed alternative.

2.1 Approximation of a distribution function

The processing time uncertainty may be expressed by a *probability density function* (PDF). Figure 2a shows a real-life PDF of computational times of an algorithm used in autonomous driving. This algorithm, described in [26] 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 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 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 [21] 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 2b for the resulting F-shape.

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 1a).

An example of a static schedule of F-shaped tasks can be seen in Figure 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 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 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* [5] 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 1b.

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 [8, 36, 37], 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., [13] 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 [32], 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 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 1:

Lemma 1. *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 2. 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 1 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 5 (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.,*

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

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 2. *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 2. 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 1. *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

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

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

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.

3.1 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 1, we can assume that $L, H \neq \emptyset$. The algorithm constructively partitions tasks into the so-called *coverage sets*.

Definition 6 (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 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 [23].

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 1 shows an illustrative example of how the algorithm proceeds. The pseudocode can be seen in the (APX-MC-2) algorithm.

Table 1: Illustrative example of (APX-MC-2) algorithm.

(a) Input instance

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	$(3, -)$

(b) Iterations of the algorithm and the solution

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

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

1: let $p_1^{(1)} \geq p_2^{(1)} \geq \ldots \geq p_j^{(1)} \geq \ldots \geq p_{n_L}^{(1)}$
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: $\quad \text{cov}(T_k) \leftarrow \text{cov}(T_k) \cup \{T_j\}$
7: **end for**
8: $\pi \leftarrow ()$
9: **for** $i = 1$ **to** n_H **do**
10: $\quad \pi \leftarrow (\pi, T_i)$
11: $\quad$ **for all** $T_j \in \text{cov}(T_i)$ **do**
12: $\quad\quad \pi \leftarrow (\pi, T_j)$
13: $\quad$ **end for**
14: **end for**
15: **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 1. 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 [15] 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 (see Appendix A). 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.

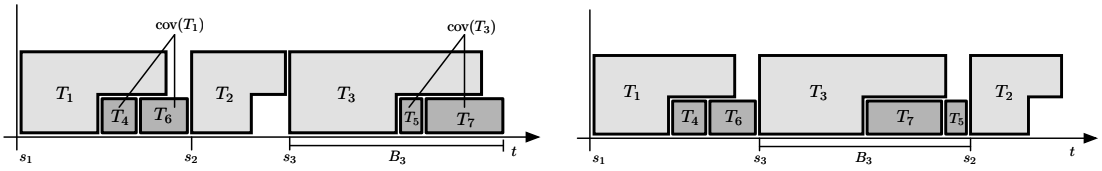
Note that (APX-MC-2) works well in practice even for the general problem, hence we use it as the initial heuristic for branch-and-price algorithm proposed in Section 3.3. In the next section, we derive the block MIP formulation that utilizes the structure of optimal permutations.

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, 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 3a. The first symmetry is broken by constraint (4), while the second symmetry is broken by constraint (3). 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: Schedule with two criticality levels.

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

subject to

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

$$B_i \geq p_i^{(1)} + \sum_{p \in P} p \cdot x_{i,p} \quad \forall T_i \in H \tag{3}$$

$$\sum_{T_i \in H} x_{i,p} = n_p \quad \forall p \in P \tag{4}$$

where

$$B_i \geq 0 \quad \forall T_i \in H \tag{5}$$

$$x_{i,p} \in \mathbb{Z}_0^+ \quad \forall (T_i, p) \in H \times P \tag{6}$$

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 3b shows one particular solution equivalent to the one in Figure 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:

$$s_q = s_{q-1} + \max \left\{ p_{q-1}^{(2)},\, p_{q-1}^{(1)} + \sum_{T_k \in \mathrm{cov}(T_{q-1})} p_k^{(1)} \right\} \quad \forall q : 1 < q \leq n_H. \tag{7}$$

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

$$s_k = s_q + p_q^{(1)} + \sum_{T_{k'} \in \mathrm{cov}(T_q) : k' < k} p_{k'}^{(1)} \quad \forall T_k \in \mathrm{cov}(T_q). \tag{8}$$

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

$$C_{\max} = s_{n_H} + \max \left\{ p_{n_H}^{(2)},\, p_{n_H}^{(1)} + \sum_{T_k \in \mathrm{cov}(T_{n_H})} p_k^{(1)} \right\}. \tag{9}$$

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

Proposition 2. *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 2 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 (7). 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 (8). For all $T_i \in H$, $T_j \in L$, there are essentially two cases. If $T_j \in \text{cov}(T_i)$, then immediately $s_j \geq s_i + p_i^{(1)}$. If $T_j \notin \text{cov}(T_i)$, then there exists some T_k such that $T_j \in \text{cov}(T_k)$. 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 (7) to makespan (9) leads to

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

Since the objective of (MIP-MC-2) is a sum of B_is and each B_i is by constraints (2) and (3) 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 $P||\sum T_j$ [34]. 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 Branch-and-price decomposition

In this section, we propose a branch-and-price decomposition algorithm [2] 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 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 $\text{cov}(T_i)$ 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) [12], 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.

3.3.1 Master problem

The master problem resolves the question, how to split a set of tasks L into coverage sets such that $L = \bigcup_{T_i \in H} \text{cov}(T_i)$ 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 $\text{cov}(T_i)$ 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 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:

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

subject to

$$\sum_{T_i \in H} \sum_{s \in S_i} a_{i,p}^{(s)} x_i^{(s)} \geq n_p \quad \forall p \in P \tag{10}$$

$$\sum_{s \in S_i} x_i^{(s)} \leq 1 \quad \forall T_i \in H \tag{11}$$

where

$$x_i^{(s)} \geq 0 \quad \forall s \in S_i, \forall T_i \in H \tag{12}$$

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 (10) ensures that each task in L is scheduled, while the constraint (11) 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

$$\sum_{p \in P} a_{i,p}^{(s)} y_p + \gamma_i \leq O_i^{(s)} \quad \forall T_i \in H, \forall s \in S_i \tag{13}$$

where

$$y_p \geq 0 \quad \forall p \in P \tag{14}$$

$$\gamma_i \leq 0 \quad \forall T_i \in H \tag{15}$$

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 (13) that is violated by the current values of $\boldsymbol{y}$ and $\boldsymbol{\gamma}$. In the next section, we will derive the pricing problem.

3.3.2 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 (13) 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 (13) with the current dual solution $\hat{\boldsymbol{y}}$, $\hat{\boldsymbol{\gamma}}$ of the master problem. This involves deciding whether the following expression

$$0 > \max\{p_i^{(1)} + \sum_{p \in P} p \cdot a_{i,p}^{(s)} - p_i^{(2)}, 0\} - \hat{\gamma}_i - \sum_{p \in P} a_{i,p}^{(s)} \hat{y}_p = \mu_i \tag{16}$$

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

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

$$a_{i,p}^{(s)} \in \mathbb{Z}_0^+ \quad \forall p \in P \tag{17}$$

Writing it down as an MIP leads to

$$\max_{\boldsymbol{a},z} \quad \sum_{p \in P} a_{i,p}^{(s)} \hat{y}_p - z \tag{18}$$

subject to

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

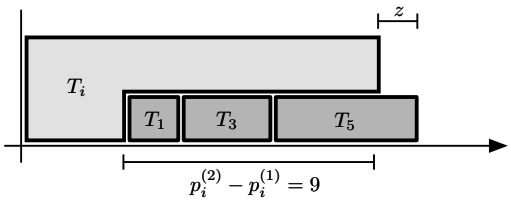


Figure 4: Optimal solution to the pricing problem instance from Table 2.

$$\sum_{p \in P} p \cdot a_{i,p}^{(s)} \leq p_i^{(2)} - p_i^{(1)} + z \tag{19}$$

where

$$z \geq 0 \tag{20}$$

$$a_{i,p}^{(s)} \in \mathbb{Z}_0 \quad \forall p \in P \tag{21}$$

which can be seen as a variant of *Knapsack Problem* [25] with items whose values are given by the current shadow prices of assignment constraints (10) 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 [11], 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 2 and the corresponding optimal solution in Figure 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.

The constraints (10) 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 3. *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:

$$U(k,j) \leftarrow \begin{cases} \max\{U(k,j-1),\, \hat{y}_j + U(k-p_j^{(1)}, j-1)\} & \text{if } p_j^{(1)} \leq k \\ U(k,j-1) & \text{otherwise} \end{cases} \tag{22}$$

and for $k > W$, as

$$U(k,j) \leftarrow \begin{cases} \max\{U(k,j-1),\, \hat{y}_j + U(k-p_j^{(1)}, j-1) - (k-W)\} & \text{if } k - p_j^{(1)} \leq W \\ \max\{U(k,j-1),\, \hat{y}_j + U(k-p_j^{(1)}, j-1) - p_j^{(1)}\} & \text{if } k - p_j^{(1)} > W \end{cases} \tag{23}$$

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 3. *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 3 is due to the influence of $z \in \mathbb{R}_0^+$ variable in MIP (18) 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 2, this rule suggests us to include tasks T_1 and T_3. Lemma 3 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 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 $\text{cov}(T_i)$. 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 3 may suggest to take some tasks that are forbidden in this branch. In this case, Lemma 3 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.

4 Problem with Three Criticality Levels

In this section, we generalize the results developed in Section 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.

4.1 Tree schedule structure

For simplicity, let us assume the problem with three criticality levels and its solution depicted in Figure 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 3.2 since any permutation of blocks achieves the same makespan.

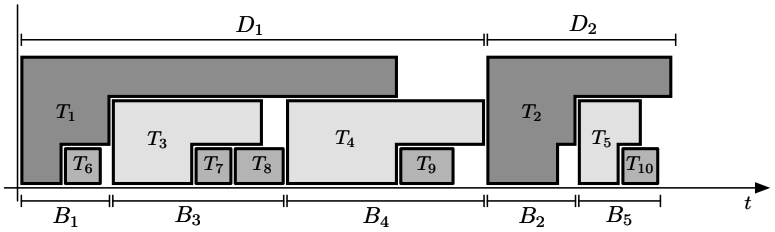
The length of the block D_1 is given by the maximum between $p_1^{(3)}$ and the sum of lengths of blocks B_1, B_3, and B_4 formed by tasks with the criticality of two. Applying the above reasoning recursively, an arbitrary order of blocks B_1, B_3, and B_4 achieves the same total length. To define the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

Definition 7. *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*

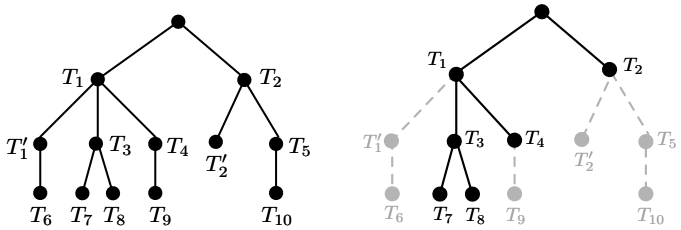
$$\mathcal{X}'_i = \mathcal{X}_i - 1, \quad \boldsymbol{P}'_i = \left(p_i^{(1)}, \ldots, p_i^{(\mathcal{X}_i - 1)} \right),$$

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

In Figure 5a, the permutation defining the order of tasks in this complete solution is given by a nested system of sets $\{\text{cov}(T_1), \text{cov}(T_2)\}$, $\text{cov}(T_1) = \{T'_1, T_3, T_4\}$, $\text{cov}(T'_1) = \{T_6\}$ and $\text{cov}(T_2) = \{T'_2, T_5\}$, $\text{cov}(T_3) = \{T_7, T_8\}$, $\text{cov}(T_4) = \{T_9\}$, $\text{cov}(T'_2) = \emptyset$, $\text{cov}(T_5) = \{T_{10}\}$.



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



(b) Tree representation.

(c) Critical subtree.

Figure 5: Schedule with three criticality levels and representation of its permutation as a tree.

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 4. *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 5b and Figure 5a, respectively. Note that swapping subtrees rooted at T_3 and T_4 in Figure 5b leads to an isomorphic graph. This transformation can be viewed as permuting B_3 and B_4 blocks inside the schedule in Figure 5a, which leads to different but an equivalent schedule. Therefore, isomorphic trees 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 8 (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 5c shows an example that highlights a critical subtree of the schedule in Figure 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 4. *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 4 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 4 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 – 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 4, 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.

4.2 Branch-and-price decomposition for MC-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 3.2. The general idea here is similar to that in Section 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

$$\sum_{T_k \in D} \sum_{s \in S_k} a_{k,p}^{(s)} x_k^{(s)} \geq n_p \quad \forall p \in P \tag{24}$$

$$\sum_{T_k \in D} \sum_{s \in S_k} b_{k,i}^{(s)} x_k^{(s)} \geq 1 \quad \forall T_i \in H \tag{25}$$

$$\sum_{s \in S_k} x_k^{(s)} \leq 1 \quad \forall T_k \in D \tag{26}$$

where

$$x_k^{(s)} \geq 0 \quad \forall s \in S_k, \forall T_k \in D \tag{27}$$

The column coefficient is given as $O_k^{(s)} = \max\left\{\sum_{T_i \in \text{cov}(T_k)} B_i - p_k^{(3)},\, 0\right\}$, where the term $\text{cov}(T_k)$ is a function of configuration s. The constant B_i denotes the length of the block given by $T_i \in \text{cov}(T_k)$, 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 (24) and (25) ensure that all tasks in $H \cup L$ are scheduled, while the constraint (26) 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.

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:

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

subject to

$$\sum_{T_i \in H \cup \{T'_k\}} B_i - p_i^{(2)}(1 - x_i) \leq p_k^{(3)} + z \tag{28}$$

$$B_i \geq p_i^{(2)} \quad \forall T_i \in H \cup \{T'_k\} \tag{29}$$

$$B_i \geq p_i^{(1)} + \sum_{p \in P} p \cdot q_{i,p} \quad \forall T_i \in H \cup \{T'_k\} \tag{30}$$

$$\sum_{T_i \in H \cup \{T'_k\}} q_{i,p} \leq n_p \quad \forall p \in P \tag{31}$$

$$\sum_{p \in P} q_{i,p} \leq n_L x_i \quad \forall T_i \in H \cup \{T'_k\} \tag{32}$$

$$x_k = 1 \tag{33}$$

where

$$z \geq 0 \tag{34}$$

$$B_i \geq 0 \quad T_i \in H \cup \{T'_k\} \tag{35}$$

$$x_i \in \{0, 1\} \quad \forall T_i \in H \cup \{T'_k\} \tag{36}$$

$$q_{i,p} \in \mathbb{Z}_0^+ \quad \forall T_i \in H \cup \{T'_k\}, \forall p \in P \tag{37}$$

The coefficients $\hat{y}_p$ are shadow prices for constraints (24) and coefficients $\hat{y}_i$ correspond to shadow prices for constraints (25). 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 (33). Finally, if the optimal objective value is greater than $-\hat{\gamma}_k$, which is the shadow price for the constraint (26) 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 3 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.

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 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 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. 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).

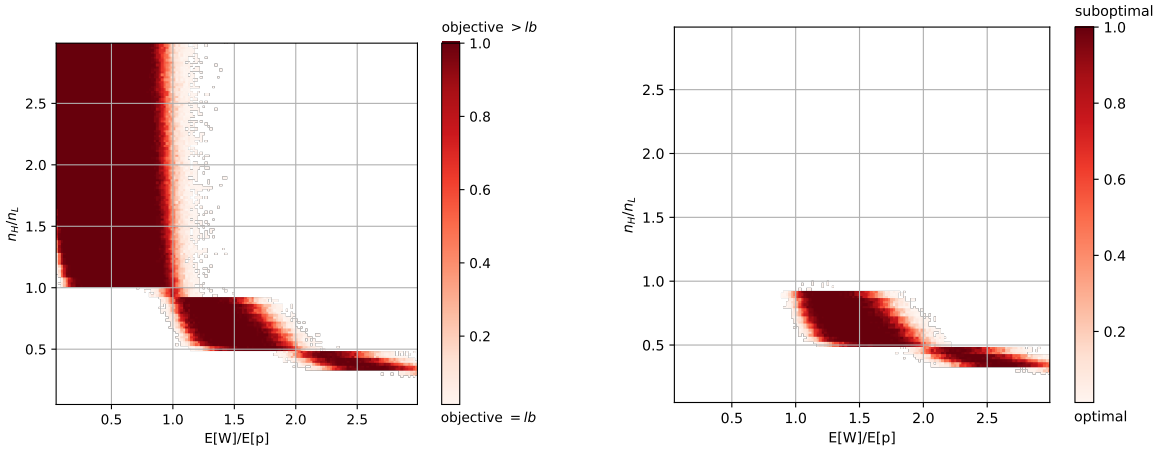
5 Computational Experiments

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.

5.1 Results of the approximation algorithm

First, we estimate the phase transition [35] 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 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 6 are obtained for the problem with $n = 50$ tasks, with each data point averaged over 75 independent samples. Figure 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 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 6 is invariant to the different values of n. The cluster of points in Figure 6b displays where the difficult instances of the MC-2 problem are located in the instance space.

5.2 Computational time for MC-2 problem

In this section, we evaluate algorithms proposed in Section 3.2 and 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 summarizes the results for instances that are generated from the distribution corresponding to the cluster of points depicted in Figure 6b, which correspond to difficult instances. We denote this dataset as *MC-2-LOP*. Table 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 [13], tasks usually have length given as a power of two [14]. 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 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 *root lb* denotes the lower bound obtained by a solver in the root node, while 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 [20]. The results of their MIP model are given in the column entitled *Relative-Order MIP*.

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

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

	(MIP-MC-2)			(BNP-MC-2)				Relative-Order MIP [20]		
n tasks	gap [%] MT/ST	root lb [–]	t [s] MT/ST	gap [%] ST	root lb [–]	gen [kcols]	t [s] ST	gap [%] MT/ST	root lb [–]	t [s] MT/ST
10	0.00 (±0.00) / 0.00 (±0.00)	113.0 (±9.5)	< 0.1 / < 0.1	0.00 (±0.00)	113.3 (±9.4)	< 0.1	0.2 (±0.0)	0.00 (±0.00) / 0.00 (±0.00)	26.7 (±3.1)	< 0.1 / 0.2 (±0.2)
15	0.00 (±0.00) / 0.00 (±0.00)	160.4 (±10.7)	< 0.1 / < 0.1	0.00 (±0.00)	160.5 (±10.6)	< 0.1	0.2 (±0.0)	0.00 (±0.00) / 0.00 (±0.00)	28.6 (±2.4)	0.9 (±1.7) / 7.5 (±28.7)
20	0.00 (±0.00) / 0.00 (±0.00)	209.8 (±11.1)	< 0.1 / < 0.1	0.00 (±0.00)	210.3 (±10.9)	< 0.1	0.2 (±0.1)	14.46 (±8.61) / 20.83 (±11.51)	29.6 (±2.4)	38.6 (±43.0) / 198.1 (±63.0)
40	0.00 (±0.00) / 0.00 (±0.00)	412.3 (±20.4)	< 0.1 / 0.1 (±0.1)	0.00 (±0.00)	412.4 (±20.3)	0.1 (±0.0)	0.6 (±0.5)	69.66 (±5.34) / 73.87 (±3.36)	30.4 (±2.5)	—
50	0.00 (±0.00) / 0.00 (±0.00)	506.1 (±17.0)	0.1 (±0.4) / 0.1 (±0.5)	0.00 (±0.00)	506.1 (±17.0)	0.1 (±0.0)	0.6 (±0.4)	79.54 (±3.09) / 80.29 (±1.80)	31.5 (±2.4)	—
100	0.00 (±0.00) / 0.00 (±0.00)	988.0 (±31.2)	0.1 (±0.1) / 0.2 (±0.1)	0.00 (±0.00)	988.0 (±31.2)	0.2 (±0.0)	1.8 (±0.7)	93.09 (±0.75) / 93.59 (±0.93)	31.5 (±1.8)	—
200	0.21 (±0.00) / 0.21 (±0.00)	1974.8 (±39.2)	0.3 (±0.2) / 0.5 (±0.5)	0.00 (±0.00)	1974.9 (±39.2)	0.4 (±0.1)	13.0 (±5.0)	97.79 (±0.32) / 98.13 (±0.12)	32.8 (±1.7)	—
400	0.00 (±0.00) / 0.00 (±0.00)	3949.5 (±44.3)	0.5 (±0.2) / 0.8 (±0.3)	0.00 (±0.00)	3949.5 (±44.3)	0.8 (±0.1)	167.4 (±64.3)	99.10 (±0.04) / 99.10 (±0.04)	33.8 (±1.3)	—
800	0.00 (±0.00) / 0.00 (±0.00)	7884.8 (±74.5)	1.7 (±1.2) / 2.6 (±0.8)	2.90 (±0.60)	7884.8 (±74.5)	1.1 (±0.0)	—	99.54 (±0.02) / 99.54 (±0.02)	34.9 (±1.5)	—
1000	0.00 (±0.00) / 0.00 (±0.00)	9864.4 (±69.8)	1.9 (±0.9) / 4.7 (±6.5)	2.68 (±0.55)	9864.4 (±69.8)	1.2 (±0.0)	—	99.63 (±0.01) / 99.63 (±0.01)	35.5 (±1.9)	—

Table 4: Computational results for MC-2 problem on *MC-2-HIP* dataset.

	(MIP-MC-2)			(BNP-MC-2)				Relative-Order MIP [20]		
n tasks	gap [%] MT/ST	root lb [–]	t [s] MT/ST	gap [%] ST	root lb [–]	gen [kcols]	t [s] ST	gap [%] MT/ST	root lb [–]	t [s] MT/ST
10	0.00 (±0.00) / 0.00 (±0.00)	153.6 (±29.8)	< 0.1 / < 0.1	0.00 (±0.00)	157.3 (±28.6)	< 0.1	0.2 (±0.0)	0.00 (±0.00) / 0.00 (±0.00)	49.9 (±9.1)	0.1 (±0.1) / 0.4 (±0.4)
15	0.00 (±0.00) / 0.00 (±0.00)	219.4 (±31.8)	< 0.1 / 0.1 (±0.2)	0.00 (±0.00)	221.5 (±31.2)	< 0.1	0.2 (±0.0)	23.38 (±7.49) / 28.44 (±10.31)	50.4 (±8.0)	16.5 (±41.4) / 18.6 (±47.6)
20	0.00 (±0.00) / 0.00 (±0.00)	279.9 (±36.8)	< 0.1 / 0.1 (±0.1)	0.00 (±0.00)	280.9 (±36.1)	< 0.1	0.2 (±0.0)	29.84 (±17.31) / 26.45 (±20.64)	56.0 (±7.7)	46.4 (±23.0) / 220.1 (±42.6)
40	0.00 (±0.00) / 0.00 (±0.00)	547.4 (±40.6)	0.3 (±0.9) / 0.5 (±1.2)	0.00 (±0.00)	550.5 (±40.2)	0.1 (±0.0)	0.3 (±0.1)	65.08 (±7.83) / 68.92 (±5.14)	57.8 (±6.2)	—
50	0.00 (±0.00) / 0.00 (±0.00)	684.5 (±49.9)	0.6 (±1.1) / 1.3 (±2.2)	0.00 (±0.00)	686.0 (±49.0)	0.1 (±0.2)	0.8 (±1.0)	74.89 (±4.58) / 76.27 (±3.81)	58.9 (±8.4)	—
100	0.08 (±0.00) / 0.08 (±0.00)	1317.2 (±69.7)	2.0 (±3.9) / 6.4 (±12.5)	0.00 (±0.00)	1318.1 (±69.4)	0.5 (±0.9)	24.4 (±84.4)	90.75 (±1.51) / 91.39 (±1.31)	62.0 (±6.3)	—
200	0.04 (±0.01) / 0.04 (±0.01)	2624.4 (±124.3)	15.2 (±41.2) / 16.2 (±34.7)	0.26 (±0.36)	2625.6 (±123.3)	0.8 (±0.8)	51.4 (±70.5)	96.60 (±0.77) / 97.24 (±0.21)	67.2 (±6.6)	—
400	0.04 (±0.05) / 0.02 (±0.01)	5225.5 (±172.6)	13.4 (±20.1) / 11.8 (±14.6)	2.10 (±0.98)	5226.8 (±174.0)	1.0 (±0.3)	139.5 (±95.9)	98.79 (±0.11) / 98.79 (±0.11)	69.9 (±5.1)	—
800	0.01 (±0.01) / 0.01 (±0.01)	10256.8 (±225.0)	30.7 (±43.0) / 31.3 (±35.4)	2.05 (±0.96)	10256.8 (±225.0)	1.0 (±0.1)	116.5 (±26.1)	99.37 (±0.04) / 99.37 (±0.04)	74.0 (±6.4)	—
1000	0.05 (±0.07) / 0.03 (±0.03)	12883.5 (±285.2)	35.8 (±41.2) / 49.2 (±59.1)	1.90 (±0.96)	12883.5 (±285.2)	1.1 (±0.0)	—	99.49 (±0.04) / 99.49 (±0.04)	74.2 (±5.7)	—

same size n are solved by (MIP-MC-2) about 10 times 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 [20] 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 [20], 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 [20] 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 [20] 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 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 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

Table 5: Computational results for MC-2 problem on *MC-2-2K* dataset.

	(MIP-MC-2)			(BNP-MC-2)				Relative-Order MIP [20]		
n tasks	gap [%] MT/ST	root lb [–]	t [s] MT/ST	gap [%] ST	root lb [–]	gen [kcols]	t [s] ST	gap [%] MT/ST	root lb [–]	t [s] MT/ST
10	0.00 (±0.00) / 0.00 (±0.00)	214.3 (±85.0)	< 0.1 / < 0.1	0.00 (±0.00)	217.8 (±84.2)	< 0.1	0.2 (±0.0)	0.00 (±0.00) / 0.00 (±0.00)	99.5 (±36.0)	0.3 (±0.4) / 1.0 (±1.4)
15	0.00 (±0.00) / 0.00 (±0.00)	329.2 (±89.7)	< 0.1 / < 0.1	0.00 (±0.00)	333.3 (±87.7)	< 0.1	0.2 (±0.0)	12.44 (±5.88) / 12.20 (±6.83)	126.0 (±19.6)	22.7 (±56.1) / 15.5 (±28.3)
20	0.00 (±0.00) / 0.00 (±0.00)	451.4 (±125.5)	< 0.1 / < 0.1	0.00 (±0.00)	454.0 (±124.3)	< 0.1	0.2 (±0.1)	23.06 (±8.44) / 22.49 (±11.01)	130.8 (±9.8)	65.1 (±58.1) / 199.2 (±66.5)
40	0.00 (±0.00) / 0.00 (±0.00)	858.7 (±168.5)	< 0.1 / < 0.1	0.00 (±0.00)	866.5 (±167.0)	< 0.1	0.3 (±0.1)	46.40 (±11.87) / 48.95 (±10.70)	132.1 (±10.9)	—
50	0.00 (±0.00) / 0.00 (±0.00)	1000.4 (±154.2)	< 0.1 / < 0.1	0.00 (±0.00)	1003.1 (±152.9)	0.1 (±0.0)	0.4 (±0.1)	56.00 (±7.38) / 59.24 (±7.27)	134.1 (±2.5)	—
100	0.00 (±0.00) / 0.00 (±0.00)	2020.8 (±277.0)	< 0.1 / < 0.1	0.00 (±0.00)	2022.5 (±275.8)	0.1 (±0.0)	0.7 (±0.2)	83.23 (±3.15) / 84.23 (±3.02)	135.2 (±1.7)	—
200	0.00 (±0.00) / 0.00 (±0.00)	3970.7 (±399.3)	0.1 (±0.0) / 0.1 (±0.1)	0.00 (±0.00)	3970.7 (±399.3)	0.2 (±0.0)	2.2 (±0.6)	93.77 (±0.54) / 95.93 (±0.56)	135.8 (±0.9)	—
400	0.00 (±0.00) / 0.00 (±0.00)	7964.0 (±630.9)	0.2 (±0.1) / 0.3 (±0.2)	0.00 (±0.00)	7964.0 (±630.9)	0.4 (±0.0)	12.4 (±2.1)	98.01 (±0.13) / 98.01 (±0.13)	136.0 (±0.0)	—
800	0.00 (±0.00) / 0.00 (±0.00)	16290.2 (±850.3)	0.7 (±0.3) / 0.9 (±1.0)	0.00 (±0.00)	16290.2 (±850.3)	0.7 (±0.0)	107.2 (±14.8)	99.02 (±0.05) / 99.02 (±0.05)	136.0 (±0.0)	—
1000	0.00 (±0.00) / 0.00 (±0.00)	19705.5 (±771.3)	0.9 (±0.8) / 1.3 (±1.0)	0.00 (±0.00)	19705.5 (±771.3)	0.9 (±0.0)	204.2 (±31.4)	99.20 (±0.02) / 99.20 (±0.02)	136.0 (±0.0)	—

pricing problem is used.

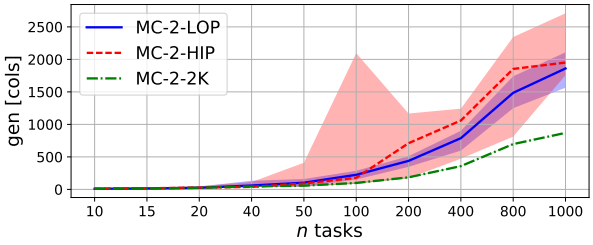


Figure 7: Median of the total number of columns generated for instances of MC-2.

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 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*.

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 6 and 7 show that scaling capabilities of [20] 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 5.2, (BNP-MC-3) can solve instances with almost twice the number of tasks than those in [20].

Table 6: Computational results for MC-3 problem on *MC-3-HIP* dataset.

	(BNP-MC-3)				Relative-Order MIP [20]		
n tasks	gap [%] MT	root lb [–]	gen [kcols]	t [s] MT	gap [%] MT	root lb [–]	t [s] MT
10	0.00 (±0.00)	191.8 (±22.4)	< 0.1	0.2 (±0.0)	0.00 (±0.00)	71.5 (±9.2)	< 0.1
15	0.00 (±0.00)	312.5 (±27.8)	< 0.1	0.3 (±0.1)	0.00 (±0.00)	72.4 (±8.9)	0.6 (±0.2)
20	0.00 (±0.00)	379.1 (±30.2)	< 0.1	0.3 (±0.2)	0.00 (±0.00)	73.3 (±7.1)	8.1 (±22.7)
40	0.00 (±0.00)	720.6 (±46.4)	0.1 (±0.0)	1.8 (±1.4)	44.40 (±8.98)	81.5 (±7.8)	—
50	0.06 (±0.00)	876.9 (±45.0)	0.2 (±0.7)	3.4 (±2.7)	61.11 (±9.22)	85.1 (±6.3)	—
100	0.02 (±0.00)	1692.0 (±72.1)	0.2 (±0.3)	18.9 (±15.3)	88.39 (±1.20)	88.5 (±5.3)	—
200	0.17 (±0.22)	3364.7 (±100.3)	0.3 (±0.3)	110.4 (±77.6)	96.20 (±0.66)	92.8 (±6.3)	—
400	0.63 (±0.53)	6712.4 (±146.4)	0.8 (±0.7)	273.0 (±185.2)	98.72 (±0.07)	96.0 (±4.8)	—
800	0.81 (±0.55)	13139.7 (±206.2)	1.0 (±0.1)	—	99.34 (±0.03)	98.6 (±3.8)	—
1000	0.82 (±0.52)	16515.2 (±250.9)	1.1 (±0.1)	—	99.47 (±0.02)	100.0 (±4.8)	—

Table 7: Computational results for MC-3 problem on *MC-3-2K* dataset.

	(BNP-MC-3)				Relative-Order MIP [20]		
n tasks	gap [%] MT	root lb [–]	gen [kcols]	t [s] MT	gap [%] MT	root lb [–]	t [s] MT
10	0.00 (±0.00)	229.2 (±80.9)	< 0.1	0.2 (±0.1)	0.00 (±0.00)	118.0 (±30.7)	0.4 (±1.3)
15	0.00 (±0.00)	289.6 (±90.4)	< 0.1	0.6 (±1.0)	8.44 (±0.00)	119.2 (±28.4)	9.7 (±21.8)
20	0.50 (±0.00)	408.9 (±86.1)	0.1 (±0.8)	0.5 (±1.0)	19.89 (±9.26)	130.9 (±13.6)	50.1 (±70.1)
40	5.20 (±5.12)	728.6 (±156.8)	0.2 (±0.6)	3.3 (±8.9)	38.99 (±11.16)	138.0 (±4.8)	—
50	9.36 (±0.00)	867.3 (±164.3)	0.4 (±2.2)	4.6 (±16.1)	50.87 (±9.95)	137.3 (±3.2)	—
100	0.00 (±0.00)	1786.8 (±237.6)	< 0.1	0.3 (±0.1)	81.79 (±2.92)	139.3 (±2.6)	—
200	0.00 (±0.00)	3372.7 (±287.2)	0.1 (±0.0)	0.9 (±0.5)	92.68 (±0.71)	141.8 (±2.7)	—
400	0.00 (±0.00)	6769.8 (±467.8)	0.1 (±0.0)	4.3 (±1.7)	98.11 (±0.12)	142.9 (±2.0)	—
800	3.33 (±0.75)	13199.3 (±661.7)	0.3 (±0.4)	29.1 (±21.6)	99.03 (±0.05)	143.9 (±0.6)	—
1000	3.60 (±0.52)	16517.5 (±941.5)	0.6 (±0.5)	81.4 (±12.5)	99.23 (±0.04)	143.9 (±0.6)	—

5.4 Discussion

For problem MC-2, both methods (MIP-MC-2) and (BNP-MC-2) proposed in this paper outperform Relative-Order MIP [20]. 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 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 [20]. 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 [24]. Hence, they still provide room for a performance improvement, but these are beyond the scope of this paper.

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.

Appendix A

Factor $3/2$ for a special case of MC-2

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

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

$$\leq 1 + \frac{\sum_{T_j \in \overline{L}} p_j^{(1)}}{n_H \cdot \overline{p} + \sum_{T_j \in \overline{L}} p_j^{(1)}} \leq 1 + \frac{\sum_{T_j \in \overline{L}} p_j^{(1)}}{2 \cdot \sum_{T_j \in \overline{L}} p_j^{(1)}} \leq \frac{3}{2},$$

where the first inequality follows from the fact that the expression in the numerator is an upper bound on the $\text{APX}(I_{\mathcal{MC}})$, 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

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

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

Acknowledgment

This work was supported by the EU and the Ministry of Industry and Trade of the Czech Republic under the Project OP PIK CZ.01.1.02/0.0/0.0/15_019/0004688 and by the European Regional Development Fund under the project AI&Reasoning (reg. no. CZ.02.1.01/0.0/0.0/15_003/0000466).

References

[1] Anand, M., Fischmeister, S., Lee, I., and Phan, L. T. (2012). State-based scheduling with tree schedules: Analysis and evaluation. *Real-Time Syst.*, 48(4):430–462.

[2] Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., and Vance, P. H. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329.

[3] Baruah, S., Bonifaci, V., D'Angelo, G., Li, H., Marchetti-Spaccamela, A., Megow, N., and Stougie, L. (2012). Scheduling real-time mixed-criticality jobs. *IEEE Transactions on Computers*, 61(8):1140–1152.

[4] Baruah, S. and Guo, Z. (2015). Mixed-criticality job models: a comparison. In *Proceedings of the 3rd International Workshop on Mixed Criticality Systems, RTSS*, pages 5–9. IEEE.

[5] Bean, J. C., Birge, J. R., Mittenthal, J., and Noon, C. E. (1991). Matchup scheduling with multiple resources, release dates and disruptions. *Operations Research*, 39(3):470–483.

[6] Behera, L. and Bhaduri, P. (2018). Time-triggered scheduling for multiprocessor mixed-criticality systems. In *Distributed Computing and Internet Technology*, pages 135–151, Cham. Springer International Publishing.

[7] Bertsimas, D., Brown, D. B., and Caramanis, C. (2011). Theory and applications of robust optimization. *SIAM review*, 53(3):464–501.

[8] Burns, A. and Davis, R. I. (2017). A survey of research into mixed criticality systems. *ACM Comput. Surv.*, 50(6):82:1–82:37.

[9] Burns, A., Davis, R. I., Baruah, S., and Bate, I. J. (2018). Robust mixed-criticality systems. *IEEE Transactions on Computers*.

[10] Davis, R. I., Altmeyer, S., and Burns, A. (2018). Mixed criticality systems with varying context switch costs. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 140–151.

[11] Delorme, M., Iori, M., and Martello, S. (2016). Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, 255(1):1 – 20.

[12] Desrosiers, J. and Lübbecke, M. E. (2005). A primer in column generation. In *Column generation*, pages 1–32. Springer.

[13] Dürr, C., Hanzálek, Z., Konrad, C., Seddik, Y., Sitters, R., Vásquez, Ó. C., and Woeginger, G. (2017). The triangle scheduling problem. *Journal of Scheduling*, pages 1–8.

[14] Dvořák, J. and Hanzálek, Z. (2016). Using two independent channels with gateway for flexray static segment scheduling. *IEEE Transactions on Industrial Informatics*, 12(5):1887–1895.

[15] Epstein, L. and Sgall, J. (2004). Approximation schemes for scheduling on uniformly related and identical parallel machines. *Algorithmica*, 39(1):43–57.

[16] Garey, M. R. and Johnson, D. S. (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.

[17] Golomb, S. W. (1996). *Polyominoes: puzzles, patterns, problems, and packings*. Princeton University Press.

[18] Graham, R. L., Lawler, E. L., Lenstra, J. K., and Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287–326.

[19] Hamaz, I., Houssin, L., and Cafieri, S. (2018). A robust basic cyclic scheduling problem. *EURO Journal on Computational Optimization*, 6(3):291–313.

[20] Hanzálek, Z., Tunys, T., and Šůcha, P. (2016). An analysis of the non-preemptive mixed-criticality match-up scheduling problem. *Journal of Scheduling*, 19(5):601–607.

[21] Jeon, S.-H., Cho, J.-H., Jung, Y., Park, S., and Han, T.-M. (2011). Automotive hardware development according to ISO 26262. In *Advanced Communication Technology (ICACT), 2011 13th International Conference on*, pages 588–592. IEEE.

[22] Kopetz, H. (1991). *Event-triggered versus time-triggered real-time systems*, pages 86–101. Springer Berlin Heidelberg, Berlin, Heidelberg.

[23] Kovács, A. (2010). New approximation bounds for LPT scheduling. *Algorithmica*, 57(2):413–433.

[24] Lübbecke, M. E. and Desrosiers, J. (2005). Selected topics in column generation. *Operations Research*, 53(6):1007–1023.

[25] Martello, S., Pisinger, D., and Toth, P. (2000). New trends in exact algorithms for the 0–1 knapsack problem. *European Journal of Operational Research*, 123(2):325 – 332.

[26] Matějka, J., Forsberg, B., Sojka, M., Hanzálek, Z., Benini, L., and Marongiu, A. (2018). Combining PREM compilation and ILP scheduling for high-performance and predictable MP-SoC execution. In *Proceedings of the 9th International Workshop on Programming Models and Applications for Multicores and Manycores*, PMAM'18, pages 11–20, New York, NY, USA. ACM.

[27] Munier, A. and Sourd, F. (2003). Scheduling chains on a single machine with non-negative time lags. *Mathematical Methods of Operations Research*, 57(1):111–123.

[28] Novak, A., Sucha, P., and Hanzalek, Z. (2017). 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*, pages 160–170.

[29] Pinedo, M. L. (2016). *Scheduling: theory, algorithms, and systems*. Springer.

[30] Qi, X., Bard, J. F., and Yu, G. (2006). Disruption management for machine scheduling: the case of SPT schedules. *International Journal of Production Economics*, 103(1):166–184.

[31] Sahinidis, N. V. (2004). Optimization under uncertainty: state-of-the-art and opportunities. *Computers & Chemical Engineering*, 28(6):971–983.

[32] Seddik, Y. and Hanzálek, Z. (2017). Match-up scheduling of mixed-criticality jobs: Maximizing the probability of jobs execution. *European Journal of Operational Research*, 262(1):46 – 59.

[33] Shabtay, D., Gaspar, N., and Kaspi, M. (2013). A survey on offline scheduling with rejection. *Journal of Scheduling*, 16(1):3–28.

[34] Shim, S.-O. and Kim, Y.-D. (2007). Scheduling on parallel identical machines to minimize total tardiness. *European Journal of Operational Research*, 177(1):135 – 146.

[35] Smith-Miles, K. and Lopes, L. (2012). Measuring instance difficulty for combinatorial optimization problems. *Computers & Operations Research*, 39(5):875 – 889.

[36] Theis, J., Fohler, G., and Baruah, S. (2013). Schedule table generation for time-triggered mixed criticality systems. *Proceedings of the 1st International Workshop on Mixed Criticality Systems, RTSS*, pages 79–84.

[37] Vestal, S. (2007). Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 239–243. IEEE.