Multithread Interval Scheduling with Flexible Machine Availabilities: Complexity and Efficient Algorithms

Mariia Anapolska[a,1,*], Tabea Brandt[a,1], Christina Büsing[a,1,2,*], Tobias Mömke[b,3]

[a] *Teaching and Research Area Combinatorial Optimization, RWTH Aachen University, Ahornstr. 55, 52074 Aachen, Germany, {anapolska,brandt,buesing}@combi.rwth-aachen.de*
[b] *Department of Computer Science, University of Augsburg, 86135 Augsburg, Germany, moemke@informatik.uni-augsburg.de*

Abstract

In the known Interval Scheduling problem with Machine Availabilities (ISMA), each machine has a contiguous availability interval and each job has a specific time interval which has to be scheduled. The objective is to schedule all jobs such that the machines' availability intervals are respected or to decide that there exists no such schedule. We extend ISMA by introducing machine capacities and flexible machine end times. Using machine capacities we model parallel processing of multiple jobs per machine, which leads to the Multithread Interval Scheduling with Machine Availabilities (MISMA). Limited machine availabilities are usually due to maintenance. Time slots for maintenance at the end of a processing period are often predetermined by staff schedules before the slots are assigned to specific machines. This motivates a variant of MISMA in which the end times of the machines' availability intervals can be permuted, the Flexible Multithread ISMA (FlexMISMA).

In this paper, we determine a tight classification of conditions that are required for obtaining a polynomial-time algorithm for both MISMA and FlexMISMA. More specifically, we show that FlexMISMA is at least as hard as MISMA. For FlexMISMA, we present polynomial-time algorithms for instances (i) with at most two available machines at a time, and (ii) with constantly many parallel jobs at each point in time, which both also solve MISMA; (iii) with arbitrarily many machines of capacity one each, in which case MISMA is known to be NP-hard; and (iv) with jobs having length one or two, for which the complexity of MISMA remains open. Furthermore, we complement result (i) by showing that both problems are NP-hard already for instances with three machines as a special case of the Vertex-Disjoint Paths problem. In contrast to (iii), we prove that increasing the capacity of machines from one to two renders FlexMISMA NP-hard as well for arbitrarily many machines.

Keywords: Interval Scheduling, Algorithms, Complexity

1. Introduction

Interval scheduling problems frequently appear both in theory and in practice [20]. However, the classical interval scheduling problem has only limited power to model real world applications, and already slight generalizations are NP-hard [14, 5]. One example of a real world application is the inclusion of maintenance intervals for machines, which is known to be NP-hard [5].

We propose two generalizations of the Interval Scheduling problem with Machine Availabilities (ISMA) [20], which, in turn, is a generalization of the well-studied Interval Coloring problem[4] [15, 26]. The first generalization is Multithread Interval Scheduling with Machine Availabilities

*Corresponding author

[1] This work was supported by the Freigeist-Fellowship of the Volkswagen Stiftung and by the German research council (DFG) Research Training Group 2236 UnRAVeL.

[2] This work was partially supported by the German Federal Ministry of Education and Research (grant no. 05M16PAA) within the project "HealthFaCT - Health: Facility Location, Covering and Transport".

[3] Partially supported by the DFG Grant 439522729 (Heisenberg-Grant).

[4] Recall that coloring of interval graphs is equivalent to interval scheduling.

(MISMA). MISMA is a natural scheduling problem where we are given m machines and n jobs. Each machine i has an availability interval $[s_i, f_i)$ and an integer capacity C_i. Each job j has a demand of one and a processing interval $[a_j, b_j)$, which has to be scheduled. The task is to schedule all jobs, i.e., assign jobs to machines, or to decide that no such schedule exists.

If the machines all have the same capacity, then their processing capabilities are the same, so the machines are equivalent. In this case, the start and end times provide information about when and how the number of available machines changes. In other words, we may assume that the machines' end times are interchangeable. In the setting of maintenance schedules, this would mean that the maintenance team fixes the time slots for servicing the machines, but does not decide which machine they service in which slot. Therefore, the second generalization is a version of MISMA in which it is allowed to permute the end times of machines. However, we may still assume without loss of generality that every machine is preassigned a start time. The task is to assign every machine an end time and to schedule all jobs, or to decide that there is no such end time assignment and schedule. Due to the increased flexibility, we call the problem FlexMISMA.

Besides the maintenance scheduling, FlexMISMA finds application in planning the usage of shared resources: when renting third-party production capacities, the number of production units available for rent varies over time due to already existing bookings. Another application is integrated production and transport planning, where the transport of manufactured parts is scheduled in advance, and a sufficient number of jobs must be finished before each transport.

1.1. Related work

The Interval Scheduling problem and its various extensions have been studied in the last decades. Reviews [20] and [21] give an overview of the early research on Interval Scheduling.

There have been several approaches to extend Interval Scheduling by restricted machine availabilities, see [20]. Brucker and Nordman [6] introduce a variant of Interval Scheduling, the k-track assignment problem, in which every machine is available only for a given time interval. The authors consider both identical machines and a generalization where machines can process only given subsets of jobs. Later, Kolen et al. [20] studied this problem using the name Interval Scheduling with Machine Availabilities (ISMA). They show that the problem is NP-complete if the number of machines is part of the input but polynomially solvable for a fixed number of machines.

Restricted periods of availability of machines is a special case of incompatibilities between some of the jobs and machines. Such incompatibilities can also be caused by specialized machines being able to process only a subset of jobs. This model finds application in aircraft maintenance scheduling, staff planning and booking management. Kolen and Kroon [19] analyze the computational complexity of the optimization version of the problem with the objective of minimizing the number of machines (TFISP). They show the NP-hardness of the problem for three or more distinct machine types, and provide an exact polynomial algorithm for instances with two types of machines. A generalization called Personnel Task Scheduling Problem (PTSP) includes both temporal availabilities of machines, which stand for workers' shifts, and job-machine incompatibilities, representing workers' skills. Various versions of PTSP were surveyed by Krishnamoorthy and Ernst in [22]

Allowing for multitasking machines in Interval Scheduling has also been considered by several authors. Mertzios et al. [24] consider a variant, called Interval Scheduling with Bounded Parallelism, in which all machines can concurrently process up to a given number of jobs. The authors consider two objectives: minimizing the total busy time of the machines needed to process all jobs, and maximizing the number of processed jobs given a budget of machine busy time. FlexMISMA can be seen as a special case of the decision version of this problem, since the machines' busy time in FlexMISMA can start and end only at the time points specified in the problem input. Interval Scheduling with Bounded Parallelism is known to be NP-hard already for machine capacity of two for both optimization versions [30, 24], and several approximation algorithms are known, e.g., [12, 24]. A slightly different version of the machine busy time minimization is considered in [13], where the costs for busy time incur only if the number of necessary machines exceeds a given value, and only for the additional machines. This models acquiring third-party machines for the periods when the company-own machines do not suffice.

Another approach to allow for multitasking machines was presented by Angelelli and Filippi [2], extending the Resource Allocation Problem [10] by considering multiple machines. They introduce

Interval Scheduling with a Resource Constraint (FISRC), where every machine and every job is additionally characterized by a resource supply or demand; they observe the relation of FISRC to ISMA and show the NP-hardness of the decision version of the problem in some special cases. For the optimization version of the problem studied in [1], the authors show the polynomial-time approximability, present an exact column generation approach and compare multiple greedy and enumeration heuristics.

Generalizations of interval scheduling include the well-studied Unsplittable Flow Problem on a path (UFP). In UFP, instead of machines we have a resource capacity that can be used by all scheduled jobs and jobs have individual demands. While it is easy to decide whether all jobs can be scheduled, the optimization problem where we have to select a maximum cardinality or maximum weight subset of jobs is NP-hard (generalizing Knapsack) and the currently best result is a 5/3-approximation algorithm [17]. UFP has a geometric version called the Storage Allocation Problem (SAP) [3, 25], where all scheduled jobs have to be drawn as non-overlapping axis-parallel rectangles. SAP with uniform job demands corresponds to a version of MISMA in which for each pair of machines either the availability interval of one machine is contained in the interval of the other or the two intervals are disjoint. A further generalization of SAP, the Flexible SAP, introduced in [29] and studied in [27, 18], considers the job resource demands as upper bounds, so that assigning fewer units of resource to a job is allowed at a cost of decreased profit.

Pre-defined end times of machines in FlexMISMA can be interpreted as fixed transportation times at which the manufactured goods are shipped to the customer. This makes FlexMISMA a special case of the problem of integrated job scheduling and transportation planning; see [8] for an overview.

Another important direction of recent research is dedicated to online versions of Interval Scheduling, where the jobs become known at their start time, as well as to stochastic variants of the problem, in which the finish times of jobs are subject to uncertainty. These variants find application in particular in the area of booking services and cloud computing, see e.g. [16, 28, 4]. We leave this field of study out of the scope of this contribution.

To the best of our knowledge, our particular extension of the Interval Scheduling problem, though related to already studied problems, has not been considered in the prior work.

1.2. Our contribution

In this paper, we introduce FlexMISMA and determine a classification of conditions that are required for obtaining a polynomial time algorithm for FlexMISMA. We first show that FlexMISMA is at least as hard as MISMA, i.e., for each MISMA instance, we can construct an equivalent FlexMISMA instance that is feasible if and only if the original MISMA instance is feasible. Subsequently, it is generally sufficient to show all hardness results for MISMA (Sec. 3) and all algorithmic results for FlexMISMA (Sec. 4), which implies that all provided results hold for both FlexMISMA and MISMA. There are, however, the following caveats. For one thing, the reduction does not preserve some properties of the job set, such as job length relations. Furthermore, the transformation of a MISMA instance to the equivalent FlexMISMA instance increases the machine capacities by one. This increase cannot be avoided for the following reason: it is known that already the special version of MISMA with unit capacities is NP-hard [20]; for FlexMISMA, however, we show in Theorem 3 that for unit capacities the problem essentially boils down to solving an interval coloring instance and is thus solvable in polynomial time. We complement the result by showing in Theorem 1 that FlexMISMA is NP-hard for a machine capacity of two.

We continue with an analysis of the problem's hardness depending on the number of machines, and show in Section 3.2 that MISMA is NP-complete already for three machines, and, consequently, also for any greater fixed number of machines. The hardness proof consists of two steps. First, in Lemma 2, we show that MISMA, and therefore FlexMISMA, is at least as hard as a specific partition permutation problem (PPP), introduced in the same section. PPP is related to the similar problem on permutation composition that was introduced by Garey et al. [14] in order to show the NP-hardness of Circular-Arc-Graph Coloring. We then show in Lemma 3 that PPP is NP-hard. While the main ideas of the proof were already used by Garey et al. [14], we have to take care of some small but important differences.

In Section 4, we present various special cases that are solvable in polynomial time and provide corresponding algorithms. In Section 4.2, we show that both MISMA and FlexMISMA can be solved in polynomial time if the number of machines is at most two. The general idea is to start by computing a schedule for only one machine using a MaxFlow Algorithm to find vertex-disjoint paths in a special graph. In a second step, the found paths are transformed to a feasible solution for both machines. Furthermore, in Section 4.3 we provide a polynomial algorithm for FlexMISMA with a constant number of threads, extending an analogous known result for MISMA. Finally, in Section 4.4 we consider the special case of FlexMISMA with uniform jobs, i.e., with jobs of identical length, for which the aforementioned relation between MISMA and FlexMISMA does not hold. We propose a linear-time algorithm for jobs of length two, and discuss why our approach does not work for job length of three or greater.

2. Problem formulation

Interval Scheduling with Machine Availabilities (ISMA) incorporates a machine availability constraint into the Interval Scheduling problem, thus ISMA assumes that every machine has a fixed availability period. Throughout this work, we denote by $[n]$ the set $\{1, 2, \ldots, n\} \subseteq \mathbb{N}$ for any $n \in \mathbb{N}$. Kolen et al. define ISMA as follows [20].

Definition 1 (ISMA). Given m machines that are available in periods $[s_i, f_i)$ for $i \in [m]$, and n jobs that require processing in the periods $[a_j, b_j)$ for $j \in [n]$, ISMA asks for a schedule that respects the availability of each machine and schedules no two jobs with overlapping processing intervals onto the same machine.

ISMA assumes that every machine processes at most one job at a time. We extend the problem formulation to allow for *multithread* machines that can process several jobs simultaneously.

Definition 2 (Multithread ISMA (MISMA)). Given m machines that are available in periods $[s_i, f_i)$ and have capacity C_i for $i \in [m]$, and n jobs that require processing in the periods $[a_j, b_j)$ for $j \in [n]$, MISMA asks for a schedule that respects the availability period of each machine and schedules at all times no more than C_i jobs simultaneously per machine $i \in [m]$.

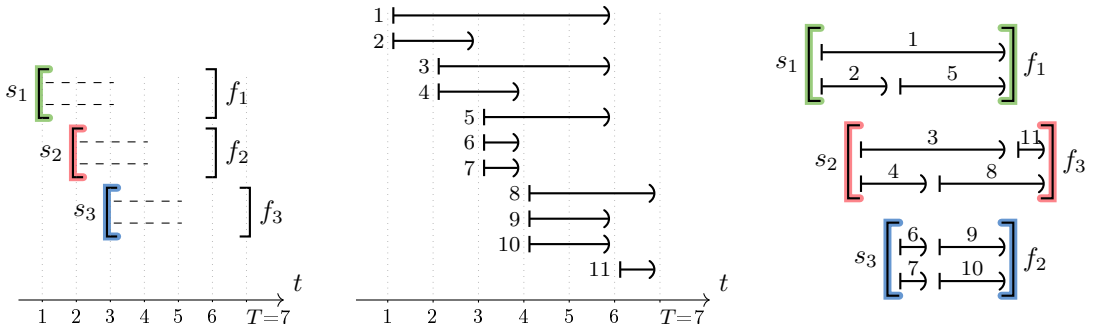
If all machines have equal capacity, they can be considered to be equivalent, and their end times interchangeable. Interchangeability of the machines' end times leads to the following variant of Interval Scheduling where each machine has an assigned start time, and the end times are fixed but not preassigned to the machines.

Definition 3 (The Flexible Multithread ISMA problem (FlexMISMA)). An instance of the Flexible Multithread ISMA (FlexMISMA) problem is given by m machines, their capacity $C \in \mathbb{N}$, start times $(s_i)_{i\in[m]}$ for every machine, m end times $(f_i)_{i\in[m]}$ that still have to be assigned to a machine, n jobs, and the jobs' processing intervals $[a_j, b_j)$ for $j \in [n]$. FlexMISMA asks for two assignments: a bijective assignment $\tau\colon [m] \to [m]$ of machines to end times with $s_i \leqslant f_{\tau(i)}$ for all $i \in [m]$, and an assignment $\alpha\colon [n] \to [m]$ of jobs to machines such that every machine $i \in [m]$ processes at most C jobs simultaneously and only between its start and end time, i.e., $\left|\{j \in \alpha^{-1}(i) \mid t \in [a_j, b_j)\}\right| \leqslant C$ for all $t \in [s_i, f_{\tau(i)})$, and $s_i \leqslant a_j < b_j \leqslant f_{\tau(i)}$ for all $j \in \alpha^{-1}(i)$.

We assume all input values to be integers. Without loss of generality, we assume that the earliest start time is 1, that is, $1 = \min_{i\in[m]} s_i$, and we denote the latest end time as $T := \max_{i\in[m]} f_i$. Observe that we can consider every multithread machine of capacity C as a group of C single-thread machines that are required to have the same start and end times. Hence, FlexMISMA is an extension of ISMA in which the machines are partitioned into groups by availability periods, and the end times must be equal within each group but can be permuted between the groups.

Figure 1 shows an exemplary instance of FlexMISMA. This instance is given by three machines with capacity $C = 2$, and by the job set with start and end times as displayed in the figure. One feasible solution for the example instance is presented in Figure 1c.

Note that we can check in polynomial time whether, at some point in time, more jobs need to be processed than there is machine capacity available, as the number of available machines at time



(a) start and end times (b) processing periods of jobs (c) feasible solution with $\tau(2) = 3$ and $\tau(3) = 2$

Figure 1: An example of a FlexMISMA instance with $m = 3$ machines of $C = 2$.

point can be computed from the start and end times: the number of machines available at time point t is

$$m_t := |\{i \in [m] \mid s_i \leqslant t\}| - |\{i \in [m] \mid f_i \leqslant t\}| .$$

If at some point in time more than $m_t \cdot C$ jobs need to be processed, then the instance is *trivially infeasible*. We thus assume in the following that at no point in time more jobs need to be processed than there are machine threads available; we call this number *available capacity*. This is, in general, not sufficient for feasibility, as the following small example demonstrates. Consider the instance of FlexMISMA shown in Figure 2, with two machines of capacity two. The set of four jobs displayed in the figure satisfies the available capacity limit at any time point, but cannot be assigned to machines respecting the start and end times.

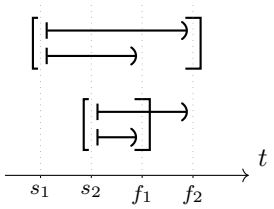


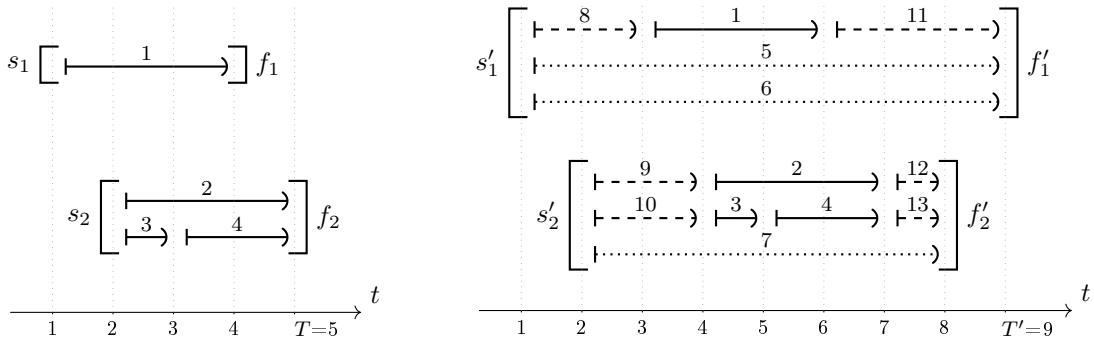
Figure 2: An infeasible instance of FlexMISMA that respects the available machine capacity.

Observe further that if there exist $i, k \in [m]$ so that $s_i = f_k$, we can remove these two values and reduce the number of machines by one, obtaining an equivalent instance. Indeed, without reducing the number of machines, depending on the assignment of end times τ, we end up either with one machine with an empty availability period or with two machines with adjoining availability periods. In the latter case, we can then combine those two machines to one machine with a longer availability period. Thus, we assume in the following $s_i \neq f_k$ for all $i, k \in [m]$.

Finally, we assume without loss of generality that all start and end times are integers no greater than $2n + 2m$, i.e., $T \leqslant 2n + 2m$. Note that $2n + 2m$ is the maximum number of distinct time points which are start or finish times of jobs or machines. If some time point $t \in [1, T]$ is neither start nor end point of a job or a machine (as, for instance, the time point 5 in the instance in Figure 1) then it can be merged with the next time point $t+1$, and the time scale can be shortened by one. This transformation preserves the overlap relation between jobs and machine availability periods and does not change the solution space.

3. Complexity results for FlexMISMA and MISMA

The size of an instance of FlexMISMA is defined by three variables: the number of jobs n, the number of machines m and the machine capacity C. If the number of jobs is smaller than the total number of machine threads, then the instance is easily solvable via first-fit approach. Hence, in the non-trivial instances the number of jobs is bounded from below by the total machine capacity:



(a) MISMA instance with $C_1 = 1$, $C_2 = 2$

(b) the resulting FlexMISMA instance with $C' = 3$

Figure 3: Transformation of MISMA to FlexMISMA for an instance with two machines.

$n \geqslant m \cdot C$. We observe that the number of jobs is the main determinant of the size of an instance of FlexMISMA. In the following complexity study, we will focus on cases differentiated by values of parameters m and C, while the number of jobs remains unbounded. If the number of jobs is bounded by a constant, then the instance is trivially solvable in polynomial time via enumeration.

3.1. Constant machine capacity

In this section, we study MISMA and FlexMISMA in the case of a fixed machine capacity C. We show that the problem is already NP-complete for fixed capacities $C \geqslant 2$. To this end, we first investigate the relation between the problems ISMA, MISMA and FlexMISMA.

We start with the correspondence of ISMA and MISMA: on the one hand, we can interpret every ISMA instance as a MISMA instance with machines of capacity one; on the other hand, we can interpret every MISMA instance as an ISMA instance by treating every thread as a separate machine. Thus, ISMA and MISMA are equivalent. Next, we establish the less obvious relation to FlexMISMA.

Lemma 1. *For every* MISMA *instance with n jobs and m machines of capacities C_i for $i \in [m]$, there exists an equivalent* FlexMISMA *instance with m machines of capacity $C' \coloneqq 1 + \max_{i\in[m]} C_i$ and with $n' \coloneqq n + mC' + \sum_{i\in[m]} C_i$ jobs.*

Proof. Suppose that an instance of MISMA with m machines with capacities C_i available in periods $[s_i, f_i)$ for $i \in [m]$, and with n jobs with processing intervals $[a_j, b_j)$ for $j \in [n]$ is given. We denote the total capacity of this MISMA instance by $k \coloneqq \sum_{i\in[m]} C_i$ and the time horizon by $T \coloneqq \max_{i\in[m]} f_i$.

We construct an instance of FlexMISMA with m machines with start times $s'_i \coloneqq i$ for $i \in [m]$. We set the capacity of all machines to $C' \coloneqq 1 + \max_{i\in[m]} C_i$, and define end times $f'_i \coloneqq T + 2m + 1 - i$ for $i \in [m]$. Note that by construction all start (end) times are pairwise different, and every machine has at least one thread more than its counterpart in MISMA. Let $k' \coloneqq \sum_{i\in[m]}(C' - C_i)$ denote the number of additional threads in the instance of FlexMISMA. We further construct $n' \coloneqq n + k' + 2k$ jobs with the following processing intervals. We first shift the periods of all jobs of the MISMA instance by m, obtaining jobs $[a'_j, b'_j) \coloneqq [a_j + m, b_j + m)$ for $j \in [n]$. Then, we add k' *long* jobs with processing periods $[s'_i, f'_i)$, $i \in [m]$, to fully occupy the additional threads. Finally, we add $2k$ jobs, two for each thread in the original instance, to pad the increased availability periods of machines. Overall, we obtain the set of n' jobs defined as follows:

$$[a'_j, b'_j) \coloneqq \begin{cases} [a_j + m, b_j + m), & j \in [n], \\ [s'_i, f'_i), & i \in [m],\ n + \sum_{l=1}^{i-1}(C' - C_l) < j \leqslant n + \sum_{l=1}^{i}(C' - C_l), \\ [s'_i, s_i + m), & i \in [m],\ n + k' + \sum_{l=1}^{i-1}(C_l) < j \leqslant n + k' + \sum_{l=1}^{i}(C_l), \\ [f_i + m, f'_i), & i \in [m],\ n + mC' + \sum_{l=1}^{i-1}(C_l) < j \leqslant n + mC' + \sum_{l=1}^{i}(C_l). \end{cases}$$

An example of this transformation is given in Figure 3.

It remains to prove that the constructed FlexMISMA instance is feasible if and only if the original MISMA instance is feasible. Let $\alpha : [n] \to [m]$ be a feasible solution to the MISMA instance. We extend this solution to a solution for FlexMISMA as follows: choose the end time assignment $\tau = \mathrm{id}$, where $\mathrm{id}\colon [m] \to [m]$ denotes the identity mapping, and extend assignment α to $\alpha' : [n'] \to [m]$ by filling the additional threads and extended availability periods with the additionally created jobs. By construction, the assignment

$$\alpha'(j) := \begin{cases} \alpha(j), & j \in [n], \\ i \in [m], & a'_j = s'_i \text{ or } b'_j = f'_i, \end{cases}$$

with time assignment $\tau = \mathrm{id}$ yields a feasible solution for the FlexMISMA instance.

Conversely, let $\alpha' : [n'] \to [m]$ and $\tau\colon [m] \to [m]$ represent a solution for the FlexMISMA instance. By construction, all start and end times of machines are distinct. Moreover, for each $i \in [m]$ there are C jobs in $[n']$ starting at time s_i, and C jobs finishing at time f_i. Therefore, all jobs j with $a'_j = s'_i$ or $b'_j = f'_{\tau(i)}$ for an $i \in [m]$ are necessarily assigned to machine i. Remark that, by construction, these are exactly the jobs $j \in [n'] \setminus [n]$. In particular, the long jobs with availability periods $[s'_i, f'_i)$ guarantee that $\tau = \mathrm{id}$. Note that there exists at least one such job for every $i \in [m]$. Furthermore, these jobs occupy their assigned machine during its complete availability period. After all the jobs in $[n'] \setminus [n]$ are assigned, a machine $i \in [m]$ has remaining capacity of $C' - (C' - C_i) = C_i$ threads, which are still unoccupied only in the period $[s_i + m, f_i + m)$. This corresponds one to one to the availability periods and machine capacities and jobs of the original MISMA instance. Thus, $\alpha'|_{[n]}$ is a feasible solution for the MISMA instance. □

As the construction in the proof of Lemma 1 can be performed in polynomial time, FlexMISMA is at least as hard as MISMA.

Observe that MISMA with all machine capacities equal to one is equivalent to the original ISMA problem. Kolen et al. proved that ISMA is NP-complete [20]. Thus, setting all machine capacities to one in the instance of MISMA in Lemma 1 immediately proves the following result.

Theorem 1. FlexMISMA *is NP-complete if machine capacity is equal to* 2.

This result concludes the discussion of the hardness of FlexMISMA with fixed machine capacity. In the next subsection, we consider the case where the number of machines is constant.

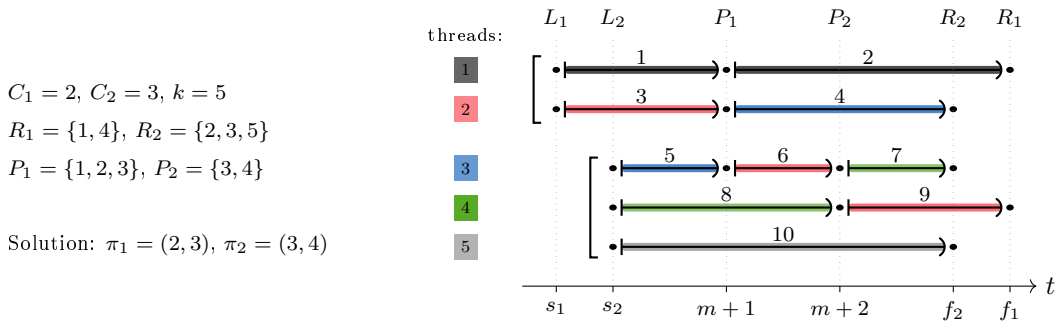
3.2. Constant number of machines

In the previous section, we have seen that FlexMISMA is NP-complete even for a constant machine capacity of two. This section is devoted to FlexMISMA's complexity in case the number of machines is constant, and their capacity unbounded.

More specifically, we show that MISMA, and thus also FlexMISMA, are NP-complete for three or more machines, even if the capacities of all machines are equal. We prove the NP-completeness by a two-staged reduction. First, we show that MISMA is at least as hard as the Partition Permutation problem (PPP). Next, we prove the NP-completeness of PPP with three partition classes by a reduction from the Directed Vertex-Disjoint Paths problem. As we already proved in Lemma 1 that FlexMISMA is at least as hard as MISMA with the same number of machines, the NP-completeness of FlexMISMA for $m \geqslant 3$ follows.

Beforehand, we present the *Partition Permutation problem*, which is a decision problem on symmetric groups and is inspired by the *Word Problem for Products of Symmetric Groups* introduced by Garey et al. [14]. In the following, we denote the symmetric group on k elements by S_k. We denote the point-wise stabilizer of a subset $U \subseteq [k]$, i.e., the subgroup of all permutations from S_k that keep every element from U fixed, by $\mathrm{Stab}(U) := \{\pi \in S_k \mid \pi(i) = i \text{ for all } i \in U\}$. We also consider all groups to be left multiplicative and we use the common notation $G_2 \circ G_1 := \{\tau \circ \pi \mid \pi \in G_1,\ \tau \in G_2\} \subseteq S_k$ for the multiplication of subgroups $G_1, G_2 \leqslant S_k$.

Definition 4 (Partition Permutation problem (PPP)). Let m integers $C_i \in \mathbb{N}$, $i \in [m]$, a target partition $\mathcal{R} := \{R_1, \ldots, R_m\}$ of $[k]$, where $k = \sum_{i \in [m]} C_i$ and $|R_i| = C_i$, as well as r



(a) PPP instance with $m = 2$ (b) resulting MISMA instance with a color-coded job assignment

Figure 4: Transformation of PPP to MISMA.

arbitrary subsets $P_u \subseteq [k]$ for $u \in [r]$ be given. We define the start partition $\mathcal{L} := \{L_1, \ldots, L_m\}$ of $[k]$ via

$$L_i := \{x \in \mathbb{N} \mid \sum_{l=1}^{i-1} C_l < x \leqslant \sum_{l=1}^{i} C_l\} \subseteq [k]$$

for $i \in [m]$. Finally, we define the embedded permutation groups $G_u := \mathrm{Stab}([k] \setminus P_u) \subseteq S_k$ for $u \in [r]$. PPP then asks whether there exists a permutation $\pi \in G_r \circ \ldots \circ G_1$ such that $\pi(L_i) = R_i$ for all $i \in [m]$.

Remark that a permutation from G_u, $u \in [r]$, operates only on the elements of P_u and keeps all other elements fixed.

Lemma 2. PPP *polynomially reduces to* MISMA.

Proof. Given a PPP instance in the above notation, we construct a MISMA instance with m machines. Machine $i \in [m]$ has capacity C_i and an availability period $[s_i := i, f_i := 2m + r + 1 - i)$. We construct k job sequences with a total of $n := k + \sum_{u \in [r]} |P_u|$ jobs. Every sequence consists of jobs with pairwise disjoint but adjacent processing periods. The start times of the first jobs and the end times of last jobs in the sequences represent the two partitions $\mathcal{L}$ and $\mathcal{R}$: The first job of sequence number $l \in [k]$ has start time s_i for index $i \in [m]$ such that $l \in L_i$. Respectively, the last job of sequence number $l \in [k]$ has end time f_i for $i \in [m]$ such that $l \in R_i$. The intermediate jobs' start and end times represent the P-sets: for every $u \in [r]$ and every $l \in P_u$, we construct a job in sequence l that ends at time point $m+u$. For easier enumeration of the jobs, let $A^l := (u)_{u \in [r],\ l \in P_u}$ be the ascending sequence of indices of P-sets that contain $l \in [k]$, which become the end points of the jobs in the sequence l. Note that $\sum_{l \in [k]} |A^l| = \sum_{u \in [r]} |P_u|$. Then we define an index mapping

$$I\colon \{(l, z) \mid l \in [k], z \in [|A^l| + 1]\} \twoheadrightarrow [n], \quad I(l, z) := \sum_{\lambda=1}^{l-1} (1 + |A_\lambda|) + z,$$

which allows us to formalize the constructed jobs as follows: for all $l \in [k]$ and $z \in [|A^l| + 1]$ the MISMA instance contains a job with processing interval

$$[a_{I(l,z)}, b_{I(l,z)}) := \begin{cases} [s_i, f_{i'}), & i, i' \in [m] \colon l \in L_i,\ l \in R_{i'},\ \text{if } |A^l| = 0, \\ [s_i, m + (A^l)_z), & i \in [m] \colon l \in L_i,\ \text{if } |A^l| > 0 \text{ and } z = 1, \\ [m + (A^l)_{z-1}, m + (A^l)_z), & \text{if } 2 \leqslant z \leqslant |A^l|, \\ [m + (A^l)_{z-1}, f_{i'}), & i' \in [m] \colon l \in R_{i'},\ \text{if } |A^l| > 0 \text{ and } z = |A^l| + 1. \end{cases}$$

We show constructively that this MISMA instance is feasible if and only if the original PPP instance is feasible. In this proof, we consider α as an assignment of jobs not to machines but rather to specific threads, which are numbered consecutively, see Figure 4. Such a more detailed assignment can be easily derived from an assignment to machines using a first fit interval scheduling algorithm that assigns jobs assigned to a machine to single machine threads.

First, assume that a solution $\pi = \pi_r \circ \ldots \circ \pi_1 \in G_r \circ \ldots \circ G_1$ for the original PPP instance is given. Then assignment $\alpha\colon [n] \to [k]$, defined for all $l \in [k]$ and $z \in [|A^l| + 1]$ as

$$\alpha(I(l,z)) := \begin{cases} l, & \text{if } z = 1, \\ (\pi_u \circ \ldots \circ \pi_1)^{-1}(l), & u = a_{I(l,z)} - m, \text{ if } 2 \leqslant z \leqslant |A^l| + 1, \end{cases}$$

is a feasible assignment of jobs to machines. Before we give a formal technical proof, let us provide the intuition behind the construction above. We constructed a job sequence for each thread such that the start times of the first jobs fit the start times of the corresponding machines, which are given by the partition $\mathcal{L}$. Each permutation π_u, for $u \in [r]$, operates only on those sequences where a job ends at the corresponding time unit $m + u$, and represents the fact that the remainders of those sequences are moved to threads according to π_u. Thus, the machines' capacities are respected at all times, and $R_i = \pi(L_i)$ ensures that all threads of machine $i \in [m]$ finish at the same time.

Observe that, by construction, every originally constructed sequence contains a unique job processed at time $m + u$, for $u \in [r]$. Clearly, all jobs processed at time $m + u$ have to be assigned to pairwise different threads. Remark that $(\pi_u \circ \ldots \circ \pi_1)(l)$ gives us the number of the job sequence that contains the job processed on thread l at time $m + u$. Accordingly, $(\pi_u \circ \ldots \circ \pi_1)^{-1}(l)$ is the thread to which the job of sequence l at time $m + u$ is assigned.

Now we formally show that the thread assignment α is feasible. Clearly, every job is assigned to exactly one thread. It remains to show that the jobs assigned to the same thread do not overlap, and that the availability periods are respected.

First, let $j = I(l,z)$ and $j' = I(l',z')$ be two jobs with $\alpha(j) = \alpha(j') = h$ for some thread $h \in [k]$. We denote $(A^l)_0 := s_i - m$ for any $l \in [k]$ and $i \in [m]$ such that $l \in L_i$. With this notation, we have $a_j - m = (A^l)_{z-1} =: u$ and $a_{j'} - m = (A^{l'})_{z'-1} =: u'$ by construction of jobs, and

$$l = (\pi_u \circ \ldots \circ \pi_1)(h), \quad l' = (\pi_{u'} \circ \ldots \circ \pi_1)(h)$$

by the definition of the assignment α.

Assume that the jobs j and j' start simultaneously, i.e., $a_j = a_{j'}$. Then $u = u'$, and consequently $l = l'$ due to equations above. Moreover, from the equality of the sequence elements $(A^l)_{z-1} = (A^l)_{z'-1}$ follows the equality of the indices, and hence $z = z'$ and $j = j'$. Thus, no two distinct jobs assigned to the same thread start simultaneously.

Next, let $a_j < a_{j'}$ and assume that the jobs overlap, i.e., $a_{j'} < b_j$. For the corresponding elements of the index sequences it follows that

$$(A^l)_{z-1} < (A^{l'})_{z'-1} < (A^l)_z.$$

Hence, for any $w \in [u+1, u']$ we have $l \notin P_w$, so $\pi_w(l) = (l)$. Consequently,

$$l' = (\pi_{u'} \circ \ldots \circ \pi_{u+1})\big((\pi_u \circ \ldots \circ \pi_1)(h)\big) = (\pi_{u'} \circ \ldots \circ \pi_{u+1})(l) = l.$$

So sequences A^l and $A^{l'}$ coincide, and between two consecutive elements of the sequence A^l there is a further element $(A^l)_{z'-1}$, which leads to a contradiction. Therefore, no two jobs assigned to the same thread overlap.

Next, we show that the assignment α respects the availability periods of the machines. It suffices to show that for any machine $i \in [m]$ and for each thread $h \in L$ of the machine, there is a job with start time s_i and a job with end time f_i assigned to the thread h. Then the rest of the jobs assigned to h start after time m and finish before time $r + m$, and so no job violates the availability period of the thread.

Let job $j \in [n]$ be such that $a_j = s_i$ for some machine $i \in [m]$. Then $j = I(l,1)$ for some $l \in L_i$, and thus $\alpha(j) \in L_i$; that is, job j is assigned to one of the thread of machine i. Next, consider job $j \in [n]$ with $b_j = f_i$ for some $i \in [m]$. Then $j = I(l,z)$ with $l \in R_i$ and $z = |A^l| + 1$. If $|A^l| = 0$, then $z = 1$ and $\alpha(j) = l$. Furthermore, since in this case the set A^l is empty, for any $u \in [r]$ we have $l \notin P_u$, and thus $\pi^{-1}(l) = l$. In particular, from $l \in R_i$ follows $\alpha(j) = l \in L_i$. Otherwise, if $|A^l| > 0$, then, by the definition of the assignment, $\alpha(j) = (\pi_u \circ \ldots \circ \pi_1)^{-1}(l)$ holds for $u = (A^l)_{|A^l|}$.

Since u is the last element of the sequence A^l, for all $w > u$ we have $l \notin P_w$ and $\pi_w^{-1}(l) = l$. Hence,

$$\alpha(j) = (\pi_u \circ \ldots \circ \pi_1)^{-1}(l) = (\pi_u \circ \ldots \circ \pi_1)^{-1}\big((\pi_{u+1}^{-1} \circ \ldots \circ \pi_r^{-1})(l)\big) = (\pi_r \circ \ldots \circ \pi_1)^{-1}(l) = \pi^{-1}(l),$$

and since $l \in R_i$, we obtain that $\alpha(j) \in L_i$. So, also every job ending at time f_i is assigned to a thread of machine i.

Since all first (last) jobs of the sequences overlap pairwise, and since we have shown that jobs on the same thread do not overlap, we conclude that every machine i gets assigned the exactly C_i jobs that start at time s_i (end at time f_i.) Hence, assignment α respects the machine capacity and is thus a feasible solution for MISMA.

Second, assume that a solution $\alpha\colon [n] \to [k]$ for the constructed MISMA instance is given. Without loss of generality, we further assume that the first job of sequence l is assigned to thread l, i.e., $\alpha(I(l,1)) = l$ for all $l \in [k]$, as otherwise we simply renumber the sequences. We then define $\tau_u(l) \in [k]$ for $u \in [r]$ as the thread to which the job of sequence $l \in [k]$ at time point $m + u$ is assigned, i.e., $\tau_u \in S_k$ with

$$\tau_u(l) := \begin{cases} \tau_{u-1}(l), & \text{if } l \notin P_u, \\ \alpha(I(l, v+1)), & \text{if } l \in P_u, \text{ with } v \in [\big|A^l\big|] \text{ s.t. } u = (A^l)_v. \end{cases}$$

Additionally, we set $\tau_0 := \mathrm{id} \in S_k$.

Now define $\pi_u := \tau_u^{-1} \circ \tau_{u-1}$ for $u \in [r]$. Then $\pi_u(l) = \tau_u^{-1}(\tau_{u-1}(l)) = l$ for any $l \notin P_u$, and thus $\pi_u \in G_u$. It remains to prove that $\pi(L_i) = R_i$. By construction, exactly the jobs $I(l, \big|A^l\big|+1)$ with $l \in R_i$ have the same end time and are assigned to threads in L_i, i.e., $\tau_r(R_i) = L_i$. Thus,

$$\pi(L_i) = (\pi_r \circ \ldots \circ \pi_1)(L_i) = \left(\tau_r^{-1} \circ \tau_{r-1} \circ \tau_{r-1}^{-1} \circ \ldots \circ \tau_1^{-1} \circ \tau_1\right)(L_i) = \tau_r^{-1}(L_i) = R_i.$$

Hence, the instances are equivalent.

To obtain an instance of MISMA with machines of equal capacity, we proceed as in the proof of Theorem 1: we set the desired machine capacity to $C = \max_{i \in [m]} C_i$, and add $C - C_i$ threads and $C - C_i$ dummy jobs with processing periods $[s_i, f_i)$ for every machine i. Clearly, the obtained MISMA instance with constant machine capacity is feasible if and only if the original MISMA instance is feasible, since in any feasible solution, the dummy jobs will be assigned to the machines they correspond to.

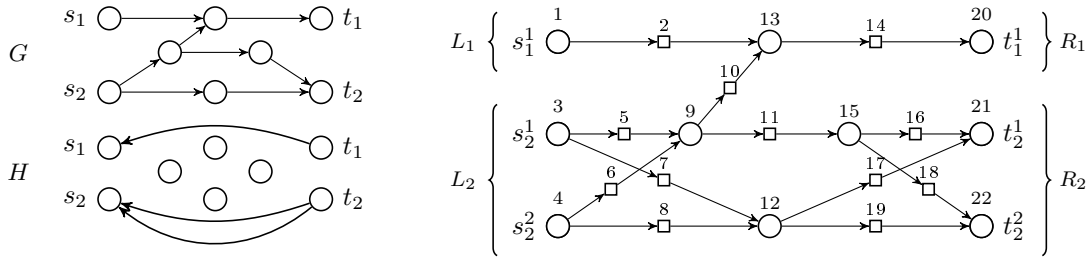
Observing that the reduction can be constructed in polynomial time concludes the proof. □

Next, we prove that PPP is also NP-complete by a polynomial reduction from the Directed Vertex-Disjoint Paths problem, which is inspired by the NP-hardness proof of Garey et al. [14].

Lemma 3. PPP *with three partition sets is* NP*-complete.*

Proof. An instance of the directed Vertex-Disjoint Paths problem is given by a pair of directed graphs (G, H) over the same vertex set V, where H is a multigraph. The task is to find a set of internally vertex-disjoint paths $\{\mathcal{P}_a \subseteq V \mid a = (t, s) \in A(H) \text{ and } \mathcal{P}_a \text{ is an } s\text{-}t\text{-path in } G\}$. The directed Vertex-Disjoint Paths problem is known to be NP-hard even if graph G is acyclic and the set $A(H)$ contains only arcs between two different pairs of vertices [11]. Let such an instance (G, H) of the vertex-disjoint paths problem on an acyclic graph G be given. Further, let H consist of k_1 parallel edges (t_1, s_1) and k_2 parallel edges (t_2, s_2), with s_1, s_2, t_1, t_2 all distinct.

We begin by constructing an auxiliary graph G' in two steps. First, we copy the vertices s_1 and t_1 exactly k_1 times including all in- and outgoing arcs, and vertices s_2, t_2 respectively k_2 times. We set $L_i := \{s_i^1, \ldots, s_i^{k_i}\}$ and $R_i := \{t_i^1, \ldots, t_i^{k_i}\}$ for $i \in \{1, 2\}$. Second, we subdivide every arc a



(a) original graphs G and H

(b) resulting graph with subdivided arcs

Figure 5: Transformation of a Disjoint Paths instance.

in the resulting graph with a new vertex v_a. We obtain $G' := (V', A')$ with

$$\begin{aligned} V' :=& (V \setminus \{s_1, s_2, t_1, t_2\}) \cup L_1 \cup L_2 \cup R_1 \cup R_2 \\ & \cup \{v_a \mid a \in A(G - \{s_1, s_2, t_1, t_2\})\} \\ & \cup \{v_a \mid a \in (L_i \times N^+(s_i)) \cup (N^-(t_i) \times R_i),\ i \in \{1,2\}\},\ \text{and} \\ A' :=& \{(v, v_a), (v_a, w) \mid a = (v, w) \in A(G)\} \\ & \cup \{(s, v_a), (v_a, w) \mid a = (s, w) \in L_i \times N^+(s_i),\ i \in \{1,2\}\} \\ & \cup \left\{(w, v_a), (v_a, t) \mid a = (w, t) \in N^-(t_i) \times R_i,\ i \in \{1,2\}\right\}, \end{aligned}$$

where $N^+(s_i) := \{v \in V(G) \mid (s_i, v) \in A(G)\}$ denotes the outgoing neighborhood of s_i in the original graph G and $N^-(t_i) := \{v \in V(G) \mid (v, t_i) \in A(G)\}$ the ingoing neighborhood, respectively. Remark that any set of $k_1 + k_2$ pairwise strictly vertex disjoint paths that start in L_i and end in R_i, for $i \in \{1,2\}$, translates directly to a solution for the original Vertex-Disjoint Paths instance. Moreover, as G' is also acyclic, we can enumerate the $k := |V'|$ vertices of G' according to the topological order, i.e., so that $v < u$ for all $(v, u) \in A'$, see Figure 5. In the following, we identify vertices of graph G' with their numbers in $[k]$.

Next, we construct a PPP instance with $m = 3$ as follows. Let $C_1 := k_1$, $C_2 := k_2$, and $C_3 := k - k_1 - k_2$. We already defined L_1 and L_2, as well as R_1 and R_2. We set $L_3 := [k] \setminus (L_1 \cup L_2)$ and $R_3 := [k] \setminus (R_1 \cup R_2)$. Further, we set $r = k$ and define

$$P_u := \{u\} \cup \{v \in [k] \mid (v, u) \in A'\} \text{ for } u \in [k].$$

We claim that the constructed PPP instance is feasible if and only if the Vertex-Disjoint Paths instance is feasible. On the one hand, let a set $\{\mathcal{P}_i^j \mid i \in \{1,2\},\ j \in [k_i],\ \mathcal{P}_i^j \text{ is an } s_i^j\text{-}t_i^j\text{-path}\}$ of pairwise vertex-disjoint paths be given. We construct a family of permutations $\{\pi_u\}_{u \in [k]}$ as follows. For each path $\mathcal{P}_i^j = (v_1, \ldots, v_r)$ in the solution of PPP, we define permutations $\pi_{v_l} \in G_{v_l}$, where $G_{v_l} = \mathrm{Stab}([k] \setminus P_{v_l})$, via $\pi_{v_l} = (v_{l-1},\ v_l)$ for $2 \leqslant l \leqslant r$ and $\pi_{v_1} = \mathrm{id}$. For all $u \in [k]$ that are not part of any vertex-disjoint path, we also set $\pi_u = \mathrm{id}$. Then, $(\pi_{v_l} \circ \ldots \circ \pi_{v_1})(v_1) = v_l$ gives us the l-th vertex on the path from p_1 to p_r; in particular, also for the terminal vertex $v_r = (\pi_{v_r} \circ \ldots \circ \pi_{v_1})(v_1)$. Moreover, an additional composition with any other permutation π_u has no effect on this property, as every vertex on path $\mathcal{P}$ is kept fixed by any π_u with $u \notin \mathcal{P}$. Thus, for

$$\pi := \pi_k \circ \pi_{k-1} \circ \ldots \circ \pi_2 \circ \pi_1$$

we have $\pi(s_i^j) = t_i^j$ for all $i \in \{1,2\}$ and $j \in [k_i]$. Hence $\pi(L_i) = R_i$ for $i \in \{1,2\}$, and, since π is bijective, also $\pi(L_3) = R_3$ follows. Therefore, the constructed permutations π_u, $u \in [k]$, are a feasible solution for the PPP instance.

On the other hand, let a feasible solution $\pi = (\pi_k \circ \ldots \circ \pi_1)$ of the PPP instance be given. Without loss of generality, we assume $\pi(s_i^j) = t_i^j$. Otherwise, we simply rename the target vertices accordingly, as all t_1^j are equivalent for $j \in [k_1]$, and all $t_2^{j'}$, respectively.

First, we prove that the sequence $\left(s_i^j,\ \pi_1(s_i^j),\ (\pi_2 \circ \pi_1)(s_i^j), \ldots,\ \pi(s_i^j) = t_i^j\right)$ represents an

s_i^j-t_i^j-path. To this end, we show by induction over $u \in [k]$ that

$$(\pi_u \circ \ldots \circ \pi_1)(s_i^j) = \begin{cases} \pi_u(v) \in [k] \text{ with } (v, \pi_u(v)) \in A', \text{ or} \\ (\pi_{u-1} \circ \ldots \circ \pi_1)(s_i^j) \end{cases} \tag{1}$$

for all $i \in \{1, 2\}$ and $j \in [k_i]$.

We start with the case $u = k$. We already know that $\pi(s_i^j) = (\pi_k \circ \ldots \circ \pi_1)(s_i^j) = t_i^j$. Furthermore, $\pi_k \in G_k$ with $P_k = \{k\} \cup \{j \mid (j,k) \in A'\}$. Moreover, by construction of the vertex enumeration, vertex k has no outgoing edges and is thus not a subdivision vertex. Now, consider two cases: If $t_i^j \neq k$, then $(t_i^j, k) \notin A'$ since both are not subdivision vertices; hence $t_i^j \notin P_k$ and thus $\pi_k(t_i^j) = t_i^j$. Therefore, $(\pi_{k-1} \circ \ldots \circ \pi_1)(s_i^j) = t_i^j$ and (1) is satisfied. If $t_i^j = k$, then $\pi(s_i^j) = k$, and so $\pi(k) \neq k$. By construction, for any $u \in [k]$ and for any $w \in P_u$ holds $w \leqslant u$. Consequently, $k \notin P_u$ for $u \neq k$, since $u < k$; hence k is a fixed point of any permutation π_u for $u \neq k$. Therefore, $\pi(k) \neq k$ implies $\pi_k(k) \neq k$; so there exists a vertex $v \in P_k \setminus \{k\}$ with $\pi_k(v) = k$. By definition of P_k, we have $(v, k) \in A'$ and, since $\pi(s_i^j) = k$, also $v = (\pi_{k-1} \circ \ldots \circ \pi_1)(s_i^j)$.

Next, let (1) be true for all $u < l \leqslant k$, and let $v = (\pi_{u-1} \circ \ldots \circ \pi_1)(s_i^j)$. By the induction hypothesis, the sequence $(\pi_u(v), \pi_{u+1}(\pi_u(v)), \ldots, (\pi_k \circ \ldots \circ \pi_{u+1})(\pi_u(v)) = t_i^j)$ represents a path from $\pi_u(v)$ to t_i^j in G'. We prove that (1) holds also for u, which would imply that the sequence $(v, \pi_{u-1}(v), (\pi_u \circ \pi_{u-1})(v), \ldots, t_i^j)$ represents a path from v to t_i^j in G'. Specifically, we have to show that either $\pi_u(v) = v$ or $\pi_u(v) = w \in [k]$ with $(v, w) \in A'$.

To begin with, recall that, by construction, for any $x, y \in [k]$ from $y \in P_x$ follows $y \leqslant x$.

If $v \notin P_u$, then it immediately follows that $\pi_u(v) = v$, and thus (1) holds for u. So suppose $v \in P_u$, and thus either $v = u$ or $(v, u) \in A'$.

If $v = u$, then for any $w < u$ follows $v \notin P_w$. In particular,

$$s_i^j = (\pi_{u-1} \circ \ldots \circ \pi_1)^{-1}(v) = v,$$

and, since $u = S_i^j$ has no ingoing arcs, we have $P_u = \{u\}$ and $\pi_u(v) = v$.

It remains to consider the case $v \neq u$ and $(v, u) \in A'$. Now, consider the node $\pi_u(v) \in P_u$. If $\pi_u(v) = v$ or $\pi_u(v) = u$, we are done. Otherwise, (v, u) and $(\pi_u(v), u)$ are two distinct arcs in A'. Then, vertex u with two ingoing edges is not a subdivision vertex, and hence v and $\pi_u(v)$ are both subdivision vertices by construction of G'. Since subdivision vertices have only one outgoing edge, for any $w > u$ we have $v, \pi_u(v) \notin P_w$. Hence,

$$t_i^j = (\pi_k \circ \ldots \circ \pi_{u+1})(\pi_u(v)) = \pi_u(v),$$

which is a contradiction to $\pi_u(v)$ being a subdivision vertex.

In conclusion, (1) holds true for all $u \in [k]$ by the principle of induction and the sequence $(s_i^j, \pi_1(s_i^j), \ldots, (\pi_k \circ \ldots \circ \pi_1)(s_i^j))$ represents a path from s_i^j to t_i^j.

Second, we prove that all paths represented by π are vertex disjoint. We observe that if two paths have a common vertex, they also have a common vertex v of in-degree at least two such that the two paths enter v over different arcs (u, v) and (w, v) with $u, w < v$. However, we have either $\pi_v(u) = v$ or $\pi_v(w) = v$ for permutation π_v. Without loss of generality, let $\pi_v(u) = v$ and thus $\pi_v(w) =: w' < v$. Since u, w and w' are all subdivision vertices, none of them can be part of any P_x with $x > v$. Thus, w cannot be contained in an s_i^j-t_i^j-path, and all the constructed paths are vertex disjoint. □

The NP-hardness of MISMA already for three machines follows directly from Lemma 2 and Lemma 3. Thus, FLEXMISMA is also NP-complete by Lemma 1. In summary, we obtain the following theorem.

Theorem 2. *Both* MISMA *and* FLEXMISMA *are* NP*-complete if the number of machines is fixed and greater than two.*

4. Polynomially solvable cases

As demonstrated in the previous section, MISMA and FlexMISMA are NP-complete in a wide range of special cases. However, there are still cases in which FlexMISMA can be solved efficiently. These cases are presented in this section.

4.1. Machines with unit capacity

In Section 3.1, we saw that FlexMISMA with machines having two or more threads is NP-complete. It remains to consider FlexMISMA with machine capacity equal to one. We show that in this case, the problem can be solved efficiently.

Theorem 3. FlexMISMA *with unit machine capacity is solvable in time linear in the number of jobs.*

Proof. In the case that every machine can process only one job at a time, FlexMISMA can be formulated as the Interval Coloring problem. Given an FlexMISMA instance with n jobs and m machines, we transform it into an instance of Interval Coloring with $N \coloneqq n + 2m$ intervals by representing start times s_i with intervals $(0, s_i)$ and end times f_i with intervals $(f_i, T + 1)$ for all $i \in [m]$.

Interval Coloring is solved by a greedy algorithm in time linear in the number of intervals, provided that the endpoints of the intervals are sorted [7]. All interval endpoints in the instance of Interval Coloring are non-negative integers not greater than $T + 1$. Therefore, the counting sort with runtime in $\mathcal{O}(2N + T + 1) = \mathcal{O}(n)$ [9] can be applied to sort the intervals. □

Remark that FlexMISMA with single-thread machines differs from ISMA only by the fact that we are allowed to permute the end times of machines. We observe that weakening this one constraint transforms the NP-complete ISMA into a polynomially solvable problem.

4.2. Two machines of arbitrary capacity

In this section, we present a further complementary result to a hardness proof from the previous section, this time considering the number of machines. Having seen that FlexMISMA is NP-complete for three machines, we now present a polynomial algorithm for solving the problem with two machines.

Beforehand, we introduce an additional assumption on the problem input, which can be made without loss of generality. We assume that all machines are utilized to full capacity; that is, at any point in time there are exactly as many jobs to be processed as threads available. This property can be verified in linear time. If for some time period there are fewer jobs than available machine threads, then, following the technique in [20], we transform the instance by adding auxiliary jobs with processing intervals of length one for the respective time periods. This transformation has no influence on the feasibility of a solution and needs only polynomially many auxiliary jobs. Thus, in this section we assume without loss of generality that every machine is utilized to full capacity in its availability period. In the following, this assumption is called the *full-load assumption*.

The full-load assumption implies that in any feasible solution for FlexMISMA, every machine processes exactly C jobs at any time point of its availability period. We call a non-empty subset $\mathcal{J}$ of jobs *fitting for machine* $i \in [m]$ *and end time* $f \in [1, T]$, if for every time point $s_i \leqslant t < f$ the set $\mathcal{J}$ contains exactly C jobs that are in process at time t , i.e., if for all $t \in \mathbb{N}$ holds

$$|\{j \in \mathcal{J} \mid t \in [a_j, b_j)\}| = \begin{cases} C, & \text{if } s_i \leqslant t < f, \\ 0, & \text{otherwise.} \end{cases}$$

Note that assigning an end time and a fitting set of jobs for this end time to a machine satisfies all constraints of FlexMISMA for this machine. In particular, if the instance of FlexMISMA consists of one machine, then the satisfied full-load assumption implies feasibility of the instance.

In general, finding a fitting job set for one machine and some end time is not sufficient to solve FlexMISMA, as we will see in the example in Figure 7 later. However, it is sufficient if an instance of FlexMISMA has only two machines.

Lemma 4. *For a feasible instance of* FlexMISMA *with $m = 2$ machines, n jobs and end times (f_1, f_2), let the subset $\mathcal{J}_1 \subseteq [n]$ of jobs be fitting for machine 1 with end time $f_i \in \{f_1, f_2\}$. Denote by $f_{i'}$ the unique remaining end time, where $i' \neq i$. Then the set $\mathcal{J}_2 := [n] \setminus \mathcal{J}_1$ is fitting for machine 2 with end time $f_{i'}$.*

Proof. By construction, assignment $\tau\colon \{1 \mapsto i,\ 2 \mapsto i'\}$ is a bijection and

$$\alpha\colon [n] \to [m],\ j \mapsto \begin{cases} 1, & \text{if } j \in \mathcal{J}_1, \\ 2, & \text{if } j \notin \mathcal{J}_1, \end{cases}$$

assigns all jobs to one of the machines. It remains to show that τ and α satisfy the constraints of FlexMISMA for machine 2, i.e., that $\mathcal{J}_2$ is feasible for machine 2 with end time $f_{i'}$. To prove the feasibility of the solution, we calculate the number of jobs from the set $\mathcal{J}_2$ that are in process at an arbitrary time point $t \in \mathbb{N}$. Throughout this proof, we consider the following two cases: $s_2 < f_1$ and $s_2 > f_1$, as $s_2 = f_1$ is excluded by the assumptions in Section 2.

Let $J(t)$, $J_1(t)$ and $J_2(t)$ denote the number of jobs from the sets $[n]$, $\mathcal{J}_1$ and $\mathcal{J}_2$, respectively, that are in process at time $t \in \mathbb{N}$. By definition, the job set $\mathcal{J}_1$ fully utilizes the availability period of machine 1, that is,

$$J_1(t) = \begin{cases} C, & \text{if } s_1 \leqslant t < f_i, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Furthermore, assuming the full utilization of machines, the total number of jobs $J(t)$ in the given instance expresses as follows.

$$\text{If } s_2 < f_1, \text{ then } \quad J(t) = \begin{cases} 2C, & \text{if } t \in [s_2, f_1), \\ C, & \text{if } t \in [s_1, s_2) \text{ or } t \in [f_1, f_2), \\ 0, & \text{otherwise.} \end{cases} \tag{3a}$$

$$\text{If } s_2 > f_1, \text{ then } \quad J(t) = \begin{cases} C, & \text{if } t \in [s_1, f_1) \text{ or } t \in [s_2, f_2), \\ 0, & \text{otherwise.} \end{cases} \tag{3b}$$

Using these equations, we derive the number $J_2(t)$ of jobs from the relation $\mathcal{J}_2 = [n] \setminus \mathcal{J}_1$. First, consider the case $s_2 < f_1$. We subtract Equation (2) from Equation (3a) and differentiate two subcases with respect to the value of f_i. Specifically, for $t \in [f_1, f_2)$ the value $J_1(t)$ equals 0 if $i = 1$, and it equals C if $i = 2$. This yields the following equalities:

$$J_2(t) = J(t) - J_1(t) = \begin{cases} 0 - 0, & \text{if } t < s_1 \text{ or } t \geqslant f_2, \\ C - C, & \text{if } s_1 \leqslant t < s_2, \\ 2C - C, & \text{if } s_2 \leqslant t < f_1, \\ C - 0, & \text{if } f_1 \leqslant t < f_2 \text{ and } i = 1, \\ C - C, & \text{if } f_1 \leqslant t < f_2 \text{ and } i = 2, \end{cases} = \begin{cases} 0, & \text{if } t < s_2 \text{ or } t \geqslant f_2, \\ C, & \text{if } s_2 \leqslant t < f_{i'}, \\ 0, & \text{if } f_{i'} \leqslant t < f_2. \end{cases}$$

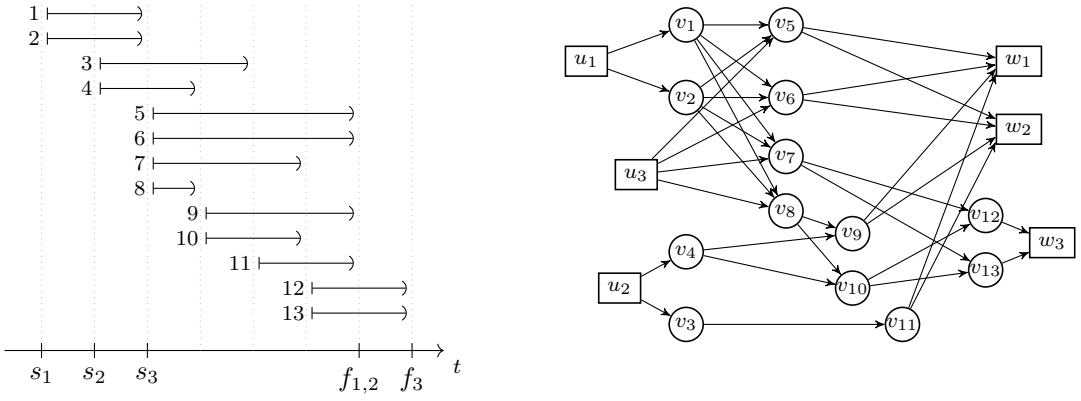
Note that the interval $[f_{i'}, f_2)$ of the last case is empty if $i = 1$.

In case $f_1 < s_2$, Equations (2) and (3b), together with the fact that $J_1(t) \leqslant J(t)$, imply that $f_i = f_1$ and $f_{i'} = f_2$. We thus obtain that

$$J_2(t) = J(t) - J_1(t) = \begin{cases} C - C, & \text{if } s_1 \leqslant t < f_1, \\ C - 0, & \text{if } s_2 \leqslant t < f_2, \\ 0 - 0, & \text{otherwise} \end{cases} \overset{f_{i'} = f_2}{=} \begin{cases} C, & \text{if } s_2 \leqslant t < f_{i'}, \\ 0, & \text{if } t < s_2 \text{ or } t \geqslant f_{i'}. \end{cases}$$

In both cases, assigning to machine 2 the job set $\mathcal{J}_2$ and the end time $f_{i'}$ satisfies the constraints of FlexMISMA. □

Lemma 4 can be generalized to an arbitrary number of machines, as long as the number of available machines m_t is never greater than two. For brevity, we call the maximal number of simultaneously available machines $\max_{t \in [1,T]} m_t$ *the maximum overlap*.



(a) job set, start and end times of machines

(b) successor graph

Figure 6: Successor graph for the FlexMISMA instance with $m = 3$ and $C = 2$.

Lemma 5. *Let $\mathcal{I}$ be a feasible instance of* FlexMISMA *satisfying the full-load assumption and with maximum overlap of two. Let $\mathcal{J}_1 \subseteq [n]$ be a fitting set for machine 1 and some machine end time $f \in (f_i)_{i\in[m]}$. Then instance $\overline{\mathcal{I}}$ resulting from $\mathcal{I}$ by deleting machine 1, the end time f and the job subset $\mathcal{J}_1$ is also feasible.*

Note that the reverse implication is trivially true.

We provide the complete proof of Lemma 5 in Appendix A. The idea of the proof is as follows: we consider a feasible solution for instance $\mathcal{I}$, distinguish three cases depending on the end time assigned to machine 1 by this solution, and show for each case how to transform a solution for instance $\mathcal{I}$ to a feasible solution for the reduced instance $\overline{\mathcal{I}}$.

As a consequence of Lemma 5, FlexMISMA with maximum overlap of two can be solved by iteratively finding a fitting job set for one machine and end time and removing the found job set from the instance. Therefore, we continue by proposing a method for finding such fitting job sets, which is based on network flows.

First, observe that the full-load assumption implies that every job is immediately followed by another job, unless the former ends at some machine's end time; that is, for a job $j \in [n]$ either $b_j = f_i$ holds for some $i \in [m]$, or there exists a job j' with $a_{j'} = b_j$. If the latter is true, we call job j' a *successor* of j.

We use this relation between jobs to construct a directed graph $G = (V, A)$ that represents the FlexMISMA instance. This graph is called the *successor graph* and contains three types of nodes: a source vertex u for every machine, a target vertex w for every end time, and a transit vertex v for every job; that is,

$$V := \{u_i, w_i \mid i \in [m]\} \cup \{v_j \mid j \in [n]\}.$$

The arcs of the network G reflect the succession relationship: For machine $i \in [m]$, we construct arcs between the source vertex u_i and all vertices v_j whose corresponding jobs start at the same time as i, i.e., for jobs with $s_i = a_j$. For an end time number $i \in [m]$, we construct arcs between the target vertex w_i and every vertex v_j whose corresponding job ends at f_i, i.e., for jobs j with $b_j = f_i$. For every two transit vertices v_j and $v_{j'}$, we construct an arc from v_j to $v_{j'}$ if and only if j' is a successor of j, i.e., $b_j = a_{j'}$. Therefore,

$$\begin{aligned} A :=& \{(u_i, v_j) \mid i \in [m],\ j \in [n],\ s_i = a_j\} \\ &\cup \{(v_j, w_i) \mid i \in [m],\ j \in [n],\ b_j = f_i\} \\ &\cup \{(v_j, v_{j'}) \mid j, j' \in [n],\ b_j = a_{j'}\}. \end{aligned}$$

An exemplary FlexMISMA instance and its corresponding successor graph are shown in Figure 6. Remark that the successor graph is always acyclic and consists of $|V| = 2 \cdot m + n$ vertices and $|A| = \mathcal{O}(n^2 + m \cdot n)$ arcs. Therefore, its construction requires time polynomial in the size of the underlying FlexMISMA instance.

We use the successor graph to construct fitting job sets for the machines by computing a family of vertex-disjoint u_i-$w_{i'}$-paths. We use the term *disjoint* for internally vertex-disjoint paths.

Lemma 6. *Let C vertex-disjoint u_i-$w_{i'}$-paths in the successor graph of a* FlexMISMA *instance be given, where u_i is a source node and $w_{i'}$ is a target node. Then the set of jobs corresponding to the nodes that are traversed by these paths is a fitting set for machine $i \in [m]$ and end time $f_{i'}$. Conversely, if there is a fitting job set for machine i and end time $f_{i'}$, then the successor graph contains C disjoint u_i-$w_{i'}$-paths.*

Proof. Let $\mathcal{P}_1, \ldots, \mathcal{P}_C$ be C vertex-disjoint u_i-$w_{i'}$-paths in the successor graph, where $i, i' \in [m]$. For each path $\mathcal{P}_l$, with $l = 1, \ldots, C$, let $J_l \subseteq [n]$ denote the set of corresponding jobs, that is, $J_l = \{j \in [n] \mid v_j \in \mathcal{P}_l\}$. Let $\mathcal{J}$ denote the union of all those job sets:

$$\mathcal{J} \coloneqq \bigcup_{l=1}^{C} J_l.$$

We show that the job set $\mathcal{J}$ satisfies the conditions of a fitting set for machine i. Since the paths are vertex-disjoint, the job sets J_l are pairwise disjoint as well. By construction of the successor graph, for each $l = 1, \ldots, C$, the jobs in J_l have disjoint processing intervals that cover in total exactly the interval $[s_i, f_{i'})$, i.e., for all $s_i \leqslant t < f_{i'}$ there exists exactly one job $j \in J_l$ with $t \in [a_j, b_j)$ and $[a_j, b_j) \subseteq [s_i, f_{i'})$ for all $j \in J_l$. As a result, for the union set $\mathcal{J}$ we have

$$|\{j \in \mathcal{J} \mid t \in [a_j, b_j)\}| = \begin{cases} C, & \text{if } s_i \leqslant t < f_{i'}, \\ 0, & \text{otherwise,} \end{cases}$$

hence $\mathcal{J}$ is a fitting set for machine i.

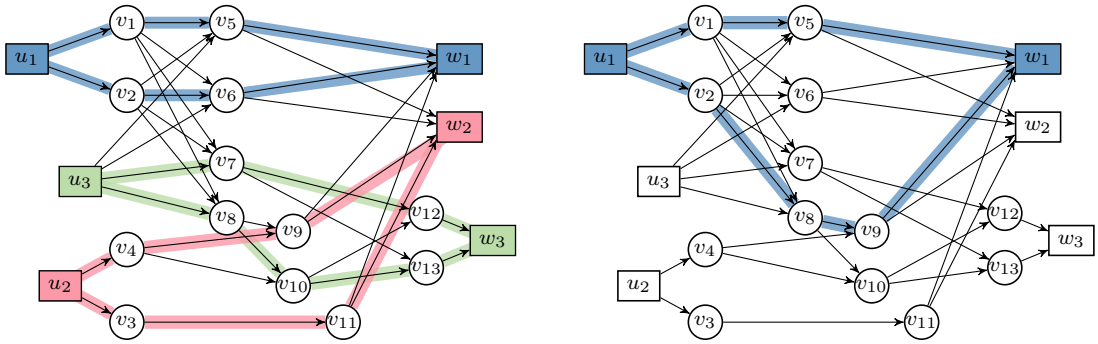
Conversely, let $\mathcal{J} \subseteq [n]$ be a fitting job set for machine $i \in [m]$ and end time $f_{i'}$. We explicitly construct the corresponding disjoint paths. First, we color the jobs in $\mathcal{J}$ with C colors so that jobs with intersecting processing intervals have different colors. Such a coloring exists by definition of a fitting job set. Next, we color the corresponding transit vertices in the successor graph accordingly. In addition, for easier notation, we assign all C colors simultaneously to vertices u_i and $w_{i'}$.

Now we show that every color class J_l, where $l \in [C]$, yields a u_i-$w_{i'}$-path. Notice that by definition of a fitting set for machine i, for every time point t between s_i and $f_{i'}$, the set $\mathcal{J}$ contains C jobs that are in process at time t, and no other jobs. Therefore, at any time point of the machine's availability period and for each color $l \in [C]$, there is a job in process that is colored with color l. Thus, in the successor graph, every transit vertex of color l is adjacent to exactly two other vertices of color l, to one by an outgoing and to one by an incoming arc. Additionally, the source vertex u_i and the target vertex $w_{i'}$ are adjacent to exactly one transit vertex of color l. As a result, the vertices corresponding to job set J_l form a u_i-$w_{i'}$-path in the successor graph. Since the color classes are pairwise disjoint, so are the paths constructed from distinct color classes of the set $\mathcal{J}$. □

Lemma 4 and Lemma 6 imply that to solve FlexMISMA with two machines, it suffices to find a family of C disjoint paths with common source and target vertex in the successor graph, or to show that no such family exists. Analogously, for FlexMISMA with maximum overlap of two, according to Lemma 5, it suffices to find a disjoint collection of such families for all but one source vertex.

We suggest using a MaxFlow algorithm to search for such families of disjoint paths. Computing a MaxFlow in a graph in a common way considers only edge capacities; hence, setting edge capacities to 1, and assuming to construct an integer MaxFlow, results in a family of merely edge-disjoint paths. We use the commonly known transformation in order to ensure that the paths are also vertex disjoint: we split every transit vertex v_j into two vertices v_j^- and v_j^+ connected by an arc (v_j^-, v_j^+), and all incoming arcs of the original vertex become incident to v_j^-, whereas the outgoing arcs of the original vertex become incident to v_j^+.

After the successor graph is constructed and transformed, the procedure solving FlexMISMA with maximum overlap of two on instances that are not trivially infeasible works as follows. We use



(a) a feasible solution for FlexMISMA

(b) the highlighted fitting set disrupts fitting sets for the remaining machines

Figure 7: Iterative MaxFlow approach for three machines.

a subroutine that, given two vertices u and w and an integer C, finds a u-w flow of value exactly C. It can be easily derived from MaxFlow solvers and runs in polynomial time. Consider the sources in the order of their indices. For the source u_i with the currently smallest index, iterate over the targets that were not yet removed from the graph in the order of their indices. For each target $w_{i'}$, ask for a u_i-$w_{i'}$ flow of value C. If there is no such flow, proceed with the next target. If all targets have been considered and no flow has been found, abort — the instance is infeasible. Once a u_i-$w_{i'}$ flow ϕ_i of value C was found for some $i' \in [m]$, assign the end time $f_{i'}$ and all jobs $j \in [n]$ for which the corresponding node v_j was traversed by the flow ϕ_i to machine i. Finally, remove all vertices of flow ϕ_i, including terminals, from the network, and proceed with the next source.

In this manner, the procedure not only finds disjoint families of disjoint paths in the successor graph, but also directly constructs a solution for FlexMISMA.

The runtime of the procedure for m machines is determined by the runtime of the MaxFlow subroutine, which is called at most $\mathcal{O}(m^2)$ times. Hence, the procedure runs in polynomial time, and its correctness follows from Lemmata 5 and 6.

In Section 3.2 we showed that FlexMISMA is NP-hard for three machines, and thus whenever the maximum overlap is at least 3. Hence, the presented method cannot be applied to solve FlexMISMA in general. An intuitive reason for this is the following: in the first iteration, the algorithm may choose a "wrong" fitting job set for the first machine, making it impossible to assign all remaining jobs to the remaining machines; an example is presented in Figure 7. Hence, the procedure would abort, even if the input instance were feasible. In case of the maximum overlap of two, Lemma 5 explicitly guarantees that any fitting set for the first machine is "correct", i.e., does not disrupt possible fitting sets of the remaining machines, as long as the entire instance is feasible.

4.3. Constant number of threads

To complete the complexity overview with respect to machine parameters, we show that FlexMISMA is linear-time solvable if both the number of machines and their capacity are fixed. To prove this, we make use of the related result for ISMA: instances with m machines and n jobs are solvable in $\mathcal{O}((m+n)m! \cdot m \log m)$ time [20].

The algorithm for ISMA suggested by Kolen et al. [20] works as follows. First, it considers the machine start times and the job set, and enumerates all end time configurations that can be realized with the given job set. Next, it verifies whether the obtained set of configurations contains the fixed end time configuration given by the instance.

Hence, the algorithm can also be used to solve a variant of ISMA that allows for a set of feasible end time assignments instead of a single fixed configuration, and thus to solve FlexMISMA: given an instance of FlexMISMA with m machines of capacity C and job set $[n]$, we apply the algorithm to an ISMA instance with the same job set $[n]$ and with $k \coloneqq m \cdot C$ machines, one for each thread in FlexMISMA, with the according start times, i.e., for each $i \in [m]$ all machines

$1 + C(i-1) \leqslant l \leqslant Ci$ have the same start time $s_l = s_i$. Furthermore, we copy each end time exactly C times so that we have k end times in total.

After the algorithm determines the realizable set of end time assignments $S^* \subseteq S_k$, we check if one of them preserves the correspondence to the original machines, i.e., we check if there exists $\tau^* \in S^*$ such that for each $i \in [m]$, threads $1 + C(i-1) \leqslant l \leqslant Ci$ are assigned the same end time $f_{\tau^*(l)}$. If this is the case, the found solution is also feasible for the original FLEXMISMA instance. If no such element in S^* exists, the original FLEXMISMA instance is infeasible.

The last check for feasibility can be performed for every element of S^* in linear time. Thus, the runtime of the entire algorithm is still determined by the enumeration procedure, which runs in $\mathcal{O}((k+n)k!k\log k)$ time, where $k = mC$ is fixed. Therefore, FLEXMISMA is solvable in linear time if the number of threads is constant.

4.4. Uniform jobs

In this section, we change the perspective and consider the complexity of FLEXMISMA not with respect to machines, but with respect to jobs. In particular, we consider a further special case of the problem, in which all jobs have equal length $\ell \in \mathbb{N}$, which we denote by ℓ-FLEXMISMA. Note that if all jobs have length 1, then the problem is solved by a trivial first-fit algorithm. Hence, we consider the job length $\ell \geqslant 2$.

For 2-FLEXMISMA, we show the following surprising result.

Theorem 4. *Any instance of* 2-FLEXMISMA *that satisfies the capacity restriction is feasible, and a solution can be constructed in linear time.*

We prove the theorem by presenting a greedy algorithm that constructs a solution for any not trivially infeasible instance. Before we describe the algorithm, let us introduce the necessary notation and definitions.

Let n_t be the number of jobs starting at time t. Note that all these jobs finish simultaneously at time $t+2$. We call jobs that start and end simultaneously *equivalent* jobs.

In the following, we distinguish between the *time point* t, at which the jobs can start or finish, and the *time period* t, which is the period between time points t and $t+1$. We assume that jobs with start time a actually start at time $a + \varepsilon$ for a sufficiently small ε; in other words, at time point t the jobs starting at t are not yet processed.

The idea of the greedy algorithm described below is to assign identical jobs in groups of size C as often as possible. The algorithm keeps track of, and updates at each iteration, the following three machine sets: the set of empty machines E, the set of full machines F, and the set of half-full machines H. A machine is called *empty* at time t if at the beginning of time period t no jobs on this machine are in execution. In other words, all jobs that have been assigned to this machine finished at time t the latest, or will start at time t or later. We say that a machine is *full* at time t if there are C jobs running on this machine at time t. Note that for the uniform job length $\ell = 2$, this means that exactly C jobs starting at time $t-1$ were assigned to the considered machine. If a machine is neither full nor empty, we say it is *half-full*.

For the job length $\ell = 2$, only the following transitions between the three machine states are possible when proceeding to the next time period. A full machine cannot be assigned any further jobs in the current iteration. Since its jobs have been already processed for one time period, at the end of the current period the jobs will end. Hence, every full machine at time t becomes empty at time $t+1$. A half-full machine becomes either empty as well, if no new jobs are assigned to it, or stays half-full, if new jobs are assigned; in both cases, the currently executed jobs will end by end of the next period. An empty machine can stay empty, if no jobs are assigned to it, or become full or half-full depending on the number of newly assigned jobs.

The algorithm proceeds chronologically and considers every time period between 1 and T. At the beginning of every time period t, first the set of available machines is updated. The machines with start time t are added to the set E of empty machines. Next, the number x_t of machines that are to be *closed*, i.e., that become unavailable, by time t is calculated. Arbitrary x_t machines are chosen from the set E of currently empty machines and are assigned each an individual end time from the set of end times $\{f_i \mid i \in [m], f_i = t\}$. Afterwards, the set $\bar{J}(t)$ of $n_t \coloneqq \left|\bar{J}(t)\right|$ jobs starting at time t is considered. Let $n_t = k_t \cdot C + r_t$ be the result of division with remainder, i.e., $r_t \in [0, C-1] \cap \mathbb{N}$. The set $\bar{J}(t)$ is partitioned into k_t sets $J_t^1, \ldots, J_t^{k_t}$ of size C and, if $r_t > 0$, an

additional *remainder* set R_t. Every complete set J_t^l, $l \in [k_t]$, is assigned to an individual empty machine from set E. These k_t machines are full at the beginning of the next time period, and are hence moved to the set F.

Finally, the remaining r_t jobs have to be assigned. If there is a half-full machine h in set H, and if it contains not more than $C - r_t$ jobs, then the jobs of set R_t are assigned to machine h, and this machine remains half-full for the next iteration. If machine h already processes too many jobs, then the jobs of set R_t are all assigned to a further empty machine $e \in E$. It makes sense not to split the equivalent jobs of the set R_t, as this ensures that an additional machine – the currently half-full one – becomes empty for the next iteration. In the next iteration, machine h will become empty and is hence moved to set E; machine e, in contrast, is half-full for the next iteration and is thus added to set H.

The iteration is completed by updating the sets of machines. All machines that started the current iteration as full are moved to set E, since they will become empty by the beginning of the next period. At any step of the described iteration, if there are not enough empty machines available, then the algorithm aborts. The described steps are summarized in Algorithm 1 below. A complete listing is given in Appendix B.

The main property of algorithm, which is key to proving its correctness, is the following: at the beginning of every time period, there is at most one machine that is half-full. To underline this fact, in the listing below we keep track of only one half-full machine $h \in [m] \cup \{\texttt{null}\}$ instead of a set H.

Algorithm 1: Greedy algorithm for 2-FLEXMISMA

Input: number of machines m, capacity C, start times $(s_i)_{i\in[m]}$, end times $(f_i)_{i\in[m]}$, jobs $[n]$ with $b_j = a_j + 2$ for all $j \in [n]$
Output: machine assightment $\alpha\colon [n] \to [m]$, end time assignment $\tau\colon [m] \to [m]$

```
Set E := ∅, F := ∅, h := null;
for t = 1, ..., T do
    // 1. adjust the machine set
    add machines i with s_i = t to set E;
    let X := {i ∈ [m] | f_i = t}; // |X| is the number of machines to be closed
    choose |X| machines from set E; if not possible, STOP;
    assign to each chosen machine an end time f_i for a unique i ∈ X;
    remove closed machines from E;
    // 2. assing starting jobs
    let J be the set of jobs starting at t, let n_t := |J|;
    partition set J into sets of cardinality C and an additional set R_t;
    assign each set of cardinality C to an empty machine;
    mark chosen machines as "full";
    if R_t ≠ ∅ then
        if there is a half-full machine h then
            if h has r_t free threads then
                assign jobs in set R_t to h;
        else
            choose an empty machine e from E; if not possible, STOP;
            assign jobs in set R_t to e;
            mark the former half-full machine h, if existing, as "empty";
            h := e; // e is the new half-full machine
    else
        mark the half-full machine h, if existing, as "empty";
        h := null;
    // 3. update machine sets
    mark all machines that were full before at the start of the iteration as "empty";
```

Clearly, all steps of Algorithm 1 can always be executed, except for choosing a required number

of machines in set E. We assume that the algorithm aborts if there are not enough empty machines.

Next, we show that Algorithm 1 is correct, i.e., it terminates if and only if the underlying instance is feasible, and in that case the constructed assignments are feasible.

Lemma 7. *If Algorithm 1 terminates, then the constructed assignments are feasible.*

Proof. The end-time assignment τ is bijective: every machine is assigned at most one end time, as the closed machines never enter the set E again, and every end time is considered and assigned at least once, when the algorithm iterates over the corresponding time point. Also, end times are assigned only to machines that started until that time point, hence $s_i \leqslant f_{\tau(i)}$ is satisfied for all machines.

The machine assignment α respects machine capacity, since either a group of at most C jobs is assigned to an empty machine, or the capacity is explicitly verified when assigning to a half-full machine. It also respects the availability periods of machines, since machines become available by being added to set E only upon their start time, and are closed when empty.

This argumentation assumes that all machines in the set E are actually empty at any time when some machines from E are requested. It is easy to see from the listing that it is indeed the case. □

It remains to show that the algorithm always terminates on feasible instances, that is, it never aborts due to the lack of required empty machines.

Lemma 8. *If an instance of* 2-FlexMISMA *is feasible, then in every iteration of Algorithm 1 there are enough empty machines.*

Proof. Let m_t again denote the number of machines available for the time period t. Recall that it can be immediately derived from the problem input. The number of jobs starting at time t (and hence finishing at time $t+2$) is denoted by n_t. Observe that the number of jobs in process during the time period t is $n_t + n_{t-1}$; hence, an instance is trivially infeasible, if for some time period t holds

$$n_{t-1} + n_t > m_t \cdot C.$$

We denote with $|E_t|$ and $|F_t|$ the cardinality of the sets E and F respectively at the beginning of the job assignment phase in t^{th} iteration, that is, after adjusting the set of available machines for time period t. Next, we record several properties of the machine sets considered by the algorithm.

P1 At the beginning of any time period, there is at most one half-full machine — this follows immediately from the algorithm.

P2 For the total number of available machines holds $m_t \in \{|E_t| + |F_t|\,,\ |E_t| + |F_t| + 1\}$, where 1 stands for the optional empty machine.

P3 If there is a half-full machine at the beginning of iteration t (at time point t), then it contains exactly r_{t-1} jobs that are in process; these are exactly the jobs of the remainder set R_{t-1} of the previous iteration.

P4 Jobs starting at time t occupy at most $\lceil n_t/C \rceil$ machines. Jobs in process at time t occupy at most $\lceil (n_t + n_{t-1})/C \rceil$ machines:
indeed, either $\lceil n_t/C \rceil + \lceil n_{t-1}/C \rceil = \lceil (n_t + n_{t-1})/C \rceil$, or $r_t + r_{t-1} \leqslant C$; in the latter case the remainder sets of both time periods share one machine (P3), so that only

$$\lceil (n_t + n_{t-1})/C \rceil = \lceil n_t/C \rceil + \lceil n_{t-1}/C \rceil - 1$$

machines are occupied.

P5 There are no full machines at the beginning, so $|F_1| = 0$, and for any time period $t \geqslant 2$ we have $|F_t| = k_{t-1}$, where $k_{t-1} = \lfloor n_{t-1}/C \rfloor$ is the number of job groups of size C starting in the previous period. Indeed, machines that are full in time period t are exactly those that were assigned a set of C identical jobs in the previous iteration.

Now, we consider an arbitrary iteration t and show that if there are not enough empty machines, then the instance is trivially infeasible.

First, we show that there are enough empty machines to close. Let $x := |\{i \in [m] \mid f_i = t\}|$ be the number of machines to close at time t, and hence the required number of empty machines. At the end of time period $t-1$, there are m_{t-1} machines available, and m_t machines must be still available during the time period t. As t is not a start time due to the assumption that $s_i \neq f_{i'}$ for all $i, i' \in [m]$, no additional machine becomes available, so we have exactly $m_{t-1} = m_t + x$. Furthermore, at most $C \cdot m_t$ jobs can be being processed at time point t without violating the capacity for time period t; these jobs occupy at most m_t machines (P4). Hence, there are at least $m_{t-1} - m_t = x$ empty machines at time point t.

Next, we show that there are enough empty machines to assign the starting jobs. Since the number of required machines depends on the number of starting jobs and on the existence of a half-full machine, we have to distinguish several cases. We call the jobs of the set R_t remainder jobs.

First, suppose there is a half-full machine h. Then the number of available machines is given by $m_t = |E_t| + |F_t| + 1$, and machine h contains r_{t-1} jobs (P3). If additionally $r_{t-1} + r_t \leqslant C$, i.e., if the half-full machine can accommodate the remainder jobs of set R_t, then k_t empty machines are required. Suppose there are actually fewer empty machines, i.e., $|E_t| \leqslant k_t - 1$. Then we obtain the following relation:

$$\begin{aligned} n_{t-1} + n_t &= C \cdot (k_{t-1} + k_t) + r_{t-1} + r_t \\ &\geqslant C \cdot (|E_t| + |F_t| + 1) + r_{t-1} + r_t \qquad \text{(P5, } |E_t| \leqslant k_t - 1) \\ &> C \cdot (|E_t| + |F_t| + 1) \\ &= C \cdot m_t, \end{aligned}$$

which implies the trivial infeasibility of the instance. Otherwise, if the remainder jobs cannot be all assigned to machine h, i.e., if $r_{t-1} + r_t > C$, then an additional empty machine is required, i.e., $k_t + 1$ machines are needed. Suppose that $E_t \leqslant k_t$; then, by an analogous calculation, we obtain

$$n_{t-1} + n_t \geqslant C \cdot (|E_t| + |F_t|) + (r_{t-1} + r_t) > C \cdot (|E_t| + |F_t|) + C = C \cdot m_t,$$

which again implies the infeasibility of the instance.

Conversely, if there is no half-full machine, then $m_t = |E_t| + |F_t|$. Again, if there are no remainder jobs, then only k_t empty machines are necessary. For $|E_t| < k_t$ we obtain

$$n_{t-1} + n_t \geqslant C \cdot (|E_t| + |F_t| + 1) > C \cdot (|E_t| + |F_t|) = C \cdot m_t;$$

if, in contrast, there are $r_t > 0$ remainder jobs, then $k_t + 1$ empty machines are necessary. If $|E_t| \leqslant k_t$, then

$$n_{t-1} + n_t \geqslant C \cdot (|E_t| + |F_t|) + r_{t-1} + r_t > C \cdot (|E_t| + |F_t|).$$

In both cases, if follows that the input instance is trivially infeasible.

To conclude, for any time period t the set of empty machines contains enough elements to assign the jobs starting at time t, assuming that the previous jobs were assigned densely according to Algorithm 1. □

Lemmata 7 and 8 show that the linear-time Algorithm 1 finds a feasible solution for any instance of 2-FlexMISMA satisfying the capacity restriction, and thus prove Theorem 4.

Observe that the relation between MISMA and FlexMISMA established in Lemma 1 does not hold any more for the uniform case, since the reduction does not preserve the uniformity. Consequently, the existence of a polynomial algorithm for 2-FlexMISMA unfortunately does not imply any complexity result for the problem version with fixed end times.

Moreover, the property stated in Theorem 4 does not hold for machines with fixed availability periods, as in MISMA: there are instances of 2-MISMA that satisfy the capacity restriction at all time points but are infeasible, as, e.g., the instance shown in Figure 8. In fact, if the job length ℓ is unbounded, then ℓ-MISMA is NP-complete already for unit machine capacity. This result follows from the proof of NP-completeness of the Precoloring Extension on unit interval graphs presented in [23].

A natural further step is to attempt to generalize the efficient algorithm for 2-FlexMISMA

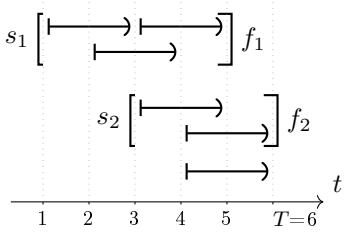


Figure 8: An infeasible instance of 2-MISMA satisfying the capacity restriction.

to work on instances with greater job length. Clearly, our solution approach is not applicable for $\ell > 2$ as is: the half-full machines do not become empty after two time periods, and hence our method produces several, up to $\ell - 1$, half-full machines. This leads to unused threads on the half-full machines that cannot be used without splitting the sets of equivalent jobs. For example, consider an instance of 3-FlexMISMA with two machines of capacity $C = 3$ and start times 1 and 2, and with two jobs starting at each time point. In the first two time periods, the new starting jobs will occupy a new machine. At time $t = 3$, both machines are half-full, and neither has enough free capacity to accommodate the whole set of the starting jobs.

Moreover, even the main idea consisting in assigning equivalent jobs to a minimal number of different machines turns out to be misleading for job length $\ell > 2$, as an instance shown in Figure 9 demonstrates: in any feasible solution for this instance, all pairs of equivalent jobs are split and assigned to different machines.

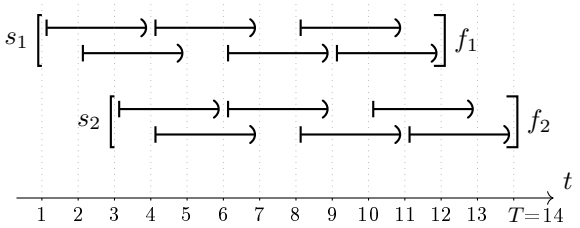


Figure 9: An instance of 3-FlexMISMA and its feasible solution. All pairs of equivalent jobs are split.

Consequently, a significantly different approach is necessary for a future study of complexity of ℓ-FlexMISMA for $\ell \geqslant 3$.

5. Conclusion

In this paper, we presented the interval scheduling extensions MISMA and FlexMISMA, and provided a tight classification of their hardness w.r.t. the number of machines and their capacity. Additionally, we considered the influence of the jobs' lengths on the problem's complexity.

For a fixed number of machines, we have shown that FlexMISMA is NP-complete for three or more machines, and polynomially solvable for up to two machines. If the machine capacity is fixed, then MISMA is NP-complete for any machine capacity, whereas FlexMISMA is NP-complete only for proper multithread machines with capacity two or greater. We have further shown that bounding both the number of machines and their capacity makes FlexMISMA tractable. For all polynomial-time solvable cases, constructive algorithms were provided. In addition, we introduced the special case of FlexMISMA with constant job length and provided a polynomial-time algorithm for jobs of length two.

While the complexity of MISMA and FlexMISMA with respect to machine parameters was completely analyzed, we only initiated the study of specific job set and their influence on the complexity of the problem. For instance, the complexity of ℓ-FlexMISMA for $\ell \geqslant 3$, as well as of MISMA with fixed job length, were not covered in this contribution and will be addressed in future work. Further special job sets are an equally interesting direction for future research.

Furthermore, due to the machine capacities, it is sensible to consider jobs with individual demands, leading to a new problem closely related to the Storage Allocation Problem.

Declarations of interest: none.

References

[1] E. Angelelli, N. Bianchessi, and C. Filippi. Optimal interval scheduling with a resource constraint. *Computers and Operations Research*, 51:268–281, 2014.

[2] E. Angelelli and C. Filippi. On the complexity of interval scheduling with a resource constraint. *Theoretical Computer Science*, 412(29):3650–3657, 2011.

[3] R. Bar-Yehuda, M. Beder, and D. Rawitz. A constant factor approximation algorithm for the storage allocation problem. *Algorithmica*, 77(4):1105–1127, 2017.

[4] M. Bender, C. Thielen, and S. Westphal. Online interval scheduling with a bounded number of failures. *Journal of Scheduling*, 20(5):443–457, Jan. 2017.

[5] M. Biró, M. Hujter, and Z. Tuza. Precoloring extension. i. interval graphs. *Discret. Math.*, 100(1-3):267–279, 1992.

[6] P. Brucker and L. Nordmann. The k-track assignment problem. *Computing*, 52(2):97–122, 1994.

[7] M. C. Carlisle and E. L. Lloyd. On the k-coloring of intervals. *Discrete Applied Mathematics*, 59(93):225–235, 1995.

[8] Z.-L. Chen. Integrated production and outbound distribution scheduling: Review and extensions. *Operations Research*, 58(1):130–148, Feb. 2010.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction To Algorithms*, chapter 8.2. MIT Press, Cambridge, 2 edition, 2001.

[10] A. Darmann, U. Pferschy, and J. Schauer. Resource allocation with time intervals. *Theoretical Computer Science*, 411(49):4217–4234, 2010.

[11] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4):691–703, 1976.

[12] M. Flammini, G. Monaco, L. Moscardelli, H. Shachnai, M. Shalom, T. Tamir, and S. Zaks. Minimizing total busy time in parallel scheduling with application to optical networks. *Theoretical Computer Science*, 411(40-42):3553–3562, Sept. 2010.

[13] I. Fridman, M. Y. Kovalyov, E. Pesch, and A. Ryzhikov. Fixed interval scheduling with third-party machines. *Networks*, 77(3):361–371, Aug. 2020.

[14] M. R. Garey, D. S. Johnson, G. L. Miller, and C. H. Papadimitriou. The Complexity of Coloring Circular Arcs and Chords. *SIAM Journal on Algebraic Discrete Methods*, 1980.

[15] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*, chapter 8 – Interval Graphs, pages 171 – 202. Academic Press, 1980.

[16] S. Goyal and D. Gupta. The online reservation problem. *Algorithms*, 13:241, 09 2020.

[17] F. Grandoni, T. Mömke, A. Wiese, and H. Zhou. A $(5/3 + \epsilon)$-approximation for unsplittable flow on a path: placing small tasks into boxes. In *STOC*, pages 607–619. ACM, 2018.

[18] D. Katz, B. Schieber, and H. Shachnai. Flexible resource allocation to interval jobs. *Algorithmica*, 81(8):3217–3244, May 2019.

[19] A. W. Kolen and L. G. Kroon. License class design: complexity and algorithms. *European Journal of Operational Research*, 63(3):432–444, 1992.

[20] A. W. Kolen, J. K. Lenstra, C. H. Papadimitriou, and F. C. Spieksma. Interval scheduling: A survey. *Naval Research Logistics (NRL)*, 54(5):530–543, 2007.

[21] M. Kovalyov, C. Ng, and T. C. E. Cheng. Fixed interval scheduling: Models, applications, computational complexity and algorithms. *European Journal of Operational Research*, 178:331–342, 04 2007.

[22] M. Krishnamoorthy and A. Ernst. *The Personnel Task Scheduling Problem*, pages 343–368. 01 2001.

[23] D. Marx. Precoloring extension on unit interval graphs. *Discrete Applied Mathematics*, 154(6):995–1002, 2006.

[24] G. B. Mertzios, M. Shalom, A. Voloshin, P. W. Wong, and S. Zaks. Optimizing busy time on parallel machines. *Theoretical Computer Science*, 562:524–541, 2015.

[25] T. Mömke and A. Wiese. Breaking the barrier of 2 for the storage allocation problem. In *ICALP*, volume 168 of *LIPIcs*, pages 86:1–86:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[26] S. Olariu. An optimal greedy heuristic to color interval graphs. *Information Processing Letters*, 37(1):21–25, 1991.

[27] H. Shachnai, A. Voloshin, and S. Zaks. Optimizing bandwidth allocation in elastic optical networks with application to scheduling. *Journal of Discrete Algorithms*, 45:1–13, July 2017.

[28] M. Shalom, A. Voloshin, P. W. H. Wong, F. C. C. Yung, and S. Zaks. Online optimization of busy time on parallel machines - (extended abstract). In *Theory and Applications of Models of Computation*, 2012.

[29] M. Shalom, P. W. H. Wong, and S. Zaks. Profit maximization in flex-grid all-optical networks. In *Structural Information and Communication Complexity*, pages 249–260. Springer International Publishing, 2013.

[30] P. Winkler and L. Zhang. Wavelength assignment and generalized interval graph coloring. page 830 – 831, 2003.

Appendix A. Proof of Lemma 5

Lemma 5. *Let $\mathcal{I}$ be a feasible instance of* FlexMISMA *satisfying the full-load assumption and with maximum overlap of two. Let $\mathcal{J}_1 \subseteq [n]$ be a fitting set for machine 1 and some machine end time $f \in (f_i)_{i\in[m]}$. Then instance $\overline{\mathcal{I}}$ resulting from $\mathcal{I}$ by deleting machine 1, the end time f and the job subset $\mathcal{J}_1$ is also feasible.*

Before we provide the proof of Lemma 5, let us make the following auxiliary observation.

Remark 1. *An instance of* FlexMISMA *with at most one available machine at any point in time, i.e., with maximum overlap of 1, is feasible if and only if it is not trivially infeasible.*

Proof of Lemma. For any time point $t \in \mathbb{N}$, we denote by $\mathcal{J}(t)$ the set of jobs in process at time t.

First, observe that since at most two machines are available simultaneously, the start and end times of machines must satisfy $f_i < s_{i+2}$ for any $i \leqslant m-2$. Besides, without loss of generality we can assume that in the time period between s_1 and f_m there is always at least one machine available – otherwise the instance can be split at the time point with no machines into two smaller instances, which can be solved independently. Consequently, we assume that $f_i > s_{i+1}$ for all $i \leqslant m-1$. In particular, this implies that $s_i \neq s_{i+1}$ for $i > 1$ and $f_{i-1} \neq f_i$ for $i < m$.

Let the fitting end time be $f = f_i$ for some $i \in [m]$. Then the instance $\overline{\mathcal{I}}$ has $m-1$ machines, denoted $2, \dots, m$ for convenience, with start times s_μ, $2 \leqslant \mu \leqslant m$, and end times $f_1, \dots, f_{i-1}, f_{i+1}, \dots, f_m$. Observe that the instance $\overline{\mathcal{I}}$ has one machine fewer available in period $[s_1, f_i)$ compared to instance $\mathcal{I}$; denoting the number of available machines in instance $\overline{\mathcal{I}}$ by $\overline{m}_t$ we write

$$\overline{m}_t = \begin{cases} m_t - 1, & \text{if } t < f_i, \\ m_t, & \text{if } t \geqslant f_i. \end{cases}$$

Furthermore, the job set $\overline{\mathcal{J}} \coloneqq [n] \setminus \mathcal{J}_1$ of instance $\overline{\mathcal{I}}$ is such that $\overline{\mathcal{I}}$ also satisfies the full-load assumption.

In particular, if $1 < i < m$, then for any $t \in [f_{i-1}, s_{i+1})$ we have $\overline{m}_t = 0$ and $\overline{\mathcal{J}}(t) = \emptyset$, i.e., in instance $\overline{\mathcal{I}}$ there are no machines available and no jobs to process in this time period (this can be verified by calculating the number of start and end times of machines up to time f_{i-1}). Consequently, instance $\overline{\mathcal{I}}$ can be split into two instances $\mathcal{I}'$ and $\mathcal{I}''$, containing jobs and start and end times before the time point f_{i-1} and after the time point s_{i+1}, respectively. Note that both sub-instances still satisfy the full-load assumption. If $i = 1$ or $i = m$, then one of the instances is empty.

First, consider the instance $\mathcal{I}'$. Since $\overline{m}_t = m_t - 1 \leqslant 1$ for all time points $t \leqslant f_{i-1}$, the instance $\mathcal{I}'$ is feasible by Remark 1. It remains to show that instance $\mathcal{I}''$ also has a feasible solution. In that case also the composed instance $\overline{\mathcal{I}}$ is clearly feasible.

Let the pair of assignments (τ, α) be a feasible solution for the instance $\mathcal{I}$. We distinguish three cases by the value of the end time $\tau(1)$ assigned to machine 1 by the feasible solution, and explicitly construct solutions (φ, β) to the instance $\mathcal{I}''$, where

$$\begin{aligned} &\varphi\colon \{i+1, \dots, m\} \to \{i+1, \dots, m\} \quad \text{and} \\ &\beta\colon \mathcal{J}'' \to \{i+1, \dots, m\} \quad \text{with} \quad \mathcal{J}'' \coloneqq \{j \in [n] \setminus \mathcal{J}_1 \mid a_j \geqslant s_{i+1}\} \end{aligned}$$

are the end time assignment and the machine assignment respectively.

Beforehand, recall that a feasible end time assignment τ ensures that $s_\mu < f_{\tau(\mu)}$ for all machines $\mu \in [m]$; from the observations at the beginning of the proof it follows that

$$\tau(\mu) \geqslant \mu - 1$$

for all machines $\mu \in [m]$.

Case 1: $\tau(1) = i$. We define

$$\varphi \coloneqq \tau\,|_{\{i+1,\dots,m\}}, \qquad \beta\colon j \mapsto \begin{cases} \alpha(j), & \text{if } b_j > f_i, \\ i+1, & \text{if } b_j \leqslant f_i. \end{cases}$$

Both mappings are well-defined. Since $\tau(\mu) \geqslant \mu - 1$ and $\tau^{-1}(i) = 1$, the mapping τ is bijective on the set $\{i+1,\ \ldots,\ m\} = [m] \setminus [i]$; hence, φ has range $\{i+1,\ \ldots,\ m\}$ and is bijective and feasible for FLEXMISMA. Assignment β assigns each jobs exactly once, and for any job j holds $\beta(j) \geqslant i+1$: if $b_j > f_i$, then machine $\alpha(j)$ should have an end time later than f_i, that is $f_{\tau(\alpha(j))} \geqslant f_{i+1}$ and $\tau(\alpha(j)) \geqslant i+1$, and, since the mapping τ is bijective on the set $i+1, \ldots, m$, we obtain $\alpha(j) \geqslant i+1$. It remains to show that assignment β satisfies the capacity constraint and machine availability period given by assignment φ. To this end, analogously to the proof of Lemma 4, we consider the set of jobs assigned to a machine μ that are in process at time t, denoted by$\beta^{-1}(\mu) \cap \mathcal{J}(t)$, and show that

$$\left|\beta^{-1}(\mu) \cap \mathcal{J}(t)\right| = \begin{cases} C, & \text{if } s_\mu \leqslant t \leqslant f_{\varphi(\mu)}, \\ 0, & \text{otherwise,} \end{cases} \qquad (\star)$$

holds for all machines $\mu \in [m] \setminus [i]$ and all time points $t \in \mathbb{N}$.

For $\mu > i+1$, we have $\beta^{-1}(\mu) = \alpha^{-1}(\mu)$, and hence the property $(\star)$ follows from the feasibility of the solution (τ, α). For $\mu = i+1$, we have

$$\beta^{-1}(\mu) = \left(\alpha^{-1}(i+1) \cap \{j \in \mathcal{J}'' \mid b_j > f_i\}\right) \dot\cup \{j \in \mathcal{J}'' \mid b_j \leqslant f_i\}.$$

For $t \geqslant f_i$, the jobs in process are only those with $b_j > t$, i.e., with $b_j > f_i$. Hence, in this case we have

$$\beta^{-1}(i+1) \cap \mathcal{J}(t) \subseteq \alpha^{-1}(i+1) \cap \mathcal{J}(t).$$

On the other hand, for any job $j \in [n]$, from $b_j > f_i$ follows that $j \in \mathcal{J}''$ and $j \notin \mathcal{J}_1$. Hence for $t \geqslant f_i$ we have $\alpha^{-1}(i+1) \cap \mathcal{J}(t) = \beta^{-1}(i+1) \cap \mathcal{J}(t)$ and

$$\left|\beta^{-1}(i+1) \cap \mathcal{J}(t)\right| = \left|\alpha^{-1}(i+1) \cap \mathcal{J}(t)\right| = \begin{cases} C, & \text{if } t < f_{\tau(i+1)} = f_{\varphi(i+1)}, \\ 0, & \text{if } t \geqslant f_{\varphi(i+1)}. \end{cases}$$

Next, consider a time point $t \in [s_{i+1}, f_i)$ (note that the interval is not empty). If job j is in process at time t, then it ends later than at time f_{i-1}, so $j \notin \mathcal{J}'$ and thus $j \in \mathcal{J}_1 \dot\cup \mathcal{J}''$; that is, $\mathcal{J}(t) \subseteq \mathcal{J}_1 \dot\cup \mathcal{J}''$. Furthermore, for the job's start we have $a_j < s_{i+2}$, hence $\alpha(j) \leqslant i+1$.

Now consider the subset $\beta^{-1}(i+1) \cap \mathcal{J}(t)$ of jobs assigned to machine $i+1$ and in process at time t. By definition of β, we have

$$\beta^{-1}(i+1) \cap \mathcal{J}(t) = \left\{j \in \mathcal{J}'' \cap \mathcal{J}(t) \mid b_j \leqslant f_i\right\} \dot\cup \left\{j \in \mathcal{J}'' \cap \mathcal{J}(t) \mid b_j > f_i \text{ and } \alpha(j) = i+1\right\}.$$

Since we have $\alpha(j) \geqslant i+1$ for any j with $b_j > f_i$ and $\alpha(j) \leqslant i+1$ for any $j \in \mathcal{J}(t)$, we conclude that for any $j \in \mathcal{J}'' \cap \mathcal{J}(t)$ from $b_j > f_i$ follows $\alpha(j) = i+1$. Hence

$$\begin{aligned} \beta^{-1}(i+1) \cap \mathcal{J}(t) &= \{j \in \mathcal{J}'' \cap \mathcal{J}(t) \mid b_j \leqslant f_i\} \dot\cup \{j \in \mathcal{J}'' \cap \mathcal{J}(t) \mid b_j > f_i\} \\ &= \mathcal{J}'' \cap \mathcal{J}(t). \end{aligned}$$

As $\mathcal{J}(t) = \left(\mathcal{J}_1 \cap \mathcal{J}(t)\right) \dot\cup \left(\mathcal{J}'' \cap \mathcal{J}(t)\right)$, we have

$$\begin{aligned} \left|\beta^{-1}(i+1) \cap \mathcal{J}(t)\right| &= |\mathcal{J}'' \cap \mathcal{J}(t)| \\ &= |\mathcal{J}(t)| - |\mathcal{J}_1 \cap \mathcal{J}(t)| \\ &= C \cdot m_t - C && \text{(full-load assumption, } \mathcal{J}_1 \text{ is fitting)} \\ &= C. && (m_t = 2 \text{ for } s_{i+1} \leqslant t < f_i) \end{aligned}$$

Finally, for $t < s_{i+1}$, the jobs in process at time t do not belong to set $\mathcal{J}''$ per definition; hence $\left|\beta^{-1}(i+1) \cap \mathcal{J}(t)\right| = 0$ for $t < s_{i+1}$. Overall, property $(\star)$ is satisfied for the assignments φ and β constructed for Case 1.

Case 2: $u := \tau(1) > i$. We define assignments

$$\varphi\colon \mu \mapsto \begin{cases} u, & \text{if } \mu = i+1, \\ \tau(\mu), & \text{if } \mu > i+1, \end{cases} \qquad \beta\colon j \mapsto \begin{cases} \alpha(j), & \text{if } \alpha(j) \neq 1, \\ i+1, & \text{if } \alpha(j) = 1. \end{cases}$$

To show that these assignments respect the indicated codomains, we first make the following observation.

Remark 2. *If $\tau(1) = u > 1$, then for all machines $k \in \{2, \dots, u\}$ we have $\tau(k) = k - 1$.*

Proof. For any machine index $\mu \in [m]$ we have that $s_{\mu+2} > f_\mu$; hence, machine μ can only be assigned end times greater than $f_{\mu-2}$, i.e., $\tau(\mu) \geqslant \mu - 1$ and conversely $\tau^{-1}(\mu) \leqslant \mu + 1$. Consequently, for the first $u - 1$ machines we have

$$\tau^{-1}(\{1, \dots, u-1\}) \subseteq \{1, \dots, u\},$$

but since $\tau^{-1}(u) = 1$ and τ is bijective, it follows that $\tau^{-1}(\{1, \dots, u-1\}) \subseteq \{2, \dots, u\}$. Thus $\tau^{-1}(1) = 2$ and, by induction, $\tau^{-1}(k) = k + 1$ for all $k \leqslant u - 1$. ■

We verify that assignment φ has values only in the set $[m] \setminus [i]$. By Remark 2, $\tau(\mu) = \mu - 1$, and thus $\tau(\mu) \geqslant i + 1$ holds for any μ with $i + 1 < \mu \leqslant u$. Furthermore, Remark 2 implies that for $\mu > u$ also $\tau(\mu) > u$, so in particular $\tau(\mu) > i + 1$. To sum up, $\varphi(\mu) \geqslant i + 1$ for any $\mu > i + 1$. In addition, $\varphi(i + 1) = u \geqslant i + 1$, since $u > i$. Hence, $\varphi(\mu) \geqslant i + 1$ for all $\mu \geqslant i + 1$.

Since τ is bijective, $\tau(1) = u$ implies that $\tau(\mu) \neq u$ for all $\mu \geqslant i+1$, and thus φ is injective and hence bijective. Finally, we show that φ is a feasible end time assignment, i.e., $s_\mu < f_{\varphi(\mu)}$ for all $\mu \geqslant i+1$. For $\mu > i+1$ this property follows from the feasibility of the assignment τ; for $\mu = i+1$, we have $\varphi(\mu) \geqslant i + 1$, and so $f_{\varphi(\mu)} \geqslant f_{i+1} > s_{i+1}$. Overall, we have shown that assignment φ is well-defined, bijective and feasible for the FlexMISMA instance $\mathcal{I}''$.

For the assignment β, we also first show that the image of β is included in the set $[m] \setminus [i]$. To this end, we recall that any job $j \in \mathcal{J}''$ starts at time $a_j \geqslant s_{i+1}$ or later, and thus machine $\alpha(j)$ should have an end time $f_{\tau(\alpha(j))} > s_{i+1}$, or $f_{\tau(\alpha(j))} \geqslant f_i$, hence $\tau(\alpha(j)) \geqslant i$. By Remark 2 implies that $\tau(\mu) = \mu - 1 < i$ for $1 < \mu < i + 1$. Hence $\alpha(j) = 1$ or $\alpha(j) \geqslant i + 1$. Consequently, for any job $j \in \mathcal{J}''$ we have $\beta(j) \geqslant i + 1$.

Next, we show feasibility of β and φ as a solution for FlexMISMA, by again verifying property $(\star)$. For $\mu \notin \{1, i + 1\}$, we have $\beta^{-1}(\mu) = \alpha^{-1}(\mu)$ and $\varphi(\mu) = \tau(\mu)$, so $(\star)$ follows from the feasibility of the assignments τ and α. For $\mu = i + 1$ we have

$$\beta^{-1}(i+1) = \left(\alpha^{-1}(i+1) \mathbin{\dot\cup} \alpha^{-1}(1)\right) \cap \mathcal{J}''.$$

For any time point $t \geqslant s_{i+1}$, if $j \in \mathcal{J}(t)$, then $j \in \mathcal{J}'' \mathbin{\dot\cup} \mathcal{J}_1$, so $\mathcal{J}'' \cap \mathcal{J}(t) = \mathcal{J}(t) \setminus \mathcal{J}_1$. Therefore

$$\begin{aligned} \beta^{-1}(i+1) \cap \mathcal{J}(t) &= \left(\alpha^{-1}(i+1) \mathbin{\dot\cup} \alpha^{-1}(1)\right) \cap \mathcal{J}'' \cap \mathcal{J}(t) \\ &= \left(\left(\alpha^{-1}(i+1) \mathbin{\dot\cup} \alpha^{-1}(1)\right) \cap \mathcal{J}(t)\right) \setminus \mathcal{J}_1. \end{aligned}$$

Hence the cardinality can be expressed as

$$\left|\beta^{-1}(i+1) \cap \mathcal{J}(t)\right| = \left|\left(\alpha^{-1}(i+1) \mathbin{\dot\cup} \alpha^{-1}(1)\right) \cap \mathcal{J}(t)\right| - \left|\mathcal{J}_1 \cap \mathcal{J}(t) \cap \left(\alpha^{-1}(i+1) \mathbin{\dot\cup} \alpha^{-1}(1)\right)\right|.$$

Next, we simplify the second term by showing that $\mathcal{J}_1 \cap \mathcal{J}(t) \subseteq \alpha^{-1}(i + 1) \mathbin{\dot\cup} \alpha^{-1}(1)$. Recall that for any job $j \in \mathcal{J}(t)$, the job is present after the time point s_{i+1} or later; hence $\alpha(j) = 1$ or $\alpha(j) \geqslant i + 1$. On the other hand, any job $j \in \mathcal{J}_1$ ends before the time point $f_i < s_{i+2}$, hence $\alpha(j) \leqslant i + 1$. Overall, for any job $j \in \mathcal{J}_1 \cap \mathcal{J}(t)$ holds $\alpha(j) \in \{1,\ i + 1\}$. Hence, for $t \geqslant f_{i+1}$ we obtain

$$\begin{aligned} \left|\beta^{-1}(i+1) \cap \mathcal{J}(t)\right| &= \left|\left(\alpha^{-1}(i+1) \mathbin{\dot\cup} \alpha^{-1}(1)\right) \cap \mathcal{J}(t)\right| - \left|\mathcal{J}_1 \cap \mathcal{J}(t)\right| \\ &= \left|\alpha^{-1}(i+1) \cap \mathcal{J}(t)\right| + \left|\alpha^{-1}(1) \cap \mathcal{J}(t)\right| - \left|\mathcal{J}_1 \cap \mathcal{J}(t)\right|. \end{aligned}$$

Recalling that the instance satisfies the full-load assumption, and so every job set assigned to a machine is a fitting set, we obtain

$$\left|\alpha^{-1}(i+1)\cap\mathcal{J}(t)\right| = \begin{cases} C, & \text{if } t\in[s_{i+1},f_{\tau(i+1)}) = [s_{i+1},f_i),\\ 0, & \text{otherwise;}\end{cases}$$

$$\left|\alpha^{-1}(1)\cap\mathcal{J}(t)\right| = \begin{cases} C, & \text{if } t\in[s_1,f_{\tau(1)}) = [s_1,f_u),\\ 0, & \text{otherwise;}\end{cases}$$

$$\left|\mathcal{J}_1\cap\mathcal{J}(t)\right| = \begin{cases} C, & \text{if } t\in[s_1,f_i),\\ 0, & \text{otherwise.}\end{cases}$$

Adding the three expressions, we obtain

$$\left|\beta^{-1}(i+1)\cap\mathcal{J}(t)\right| = \begin{cases} C, & \text{if } t\in[s_{i+1},f_i),\\ C, & \text{if } t\in[f_i,f_u,\\ 0, & \text{otherwise.}\end{cases}$$

For $t < s_{i+1}$ clearly holds that $\beta^{-1}(i+1)\cap\mathcal{J}(t) = \emptyset$, so property $(\star)$ holds for assignments φ and β and for all machines in $[m]\setminus[i]$ and all time points $t\in\mathbb{N}$.

Case 3: $u \coloneqq \tau(1) < i$.

We start with observing that any job $j\in\mathcal{J}''$ finishes later than at time f_{i-1}. This implies that $\tau(\alpha(j)) \geqslant i$, so

$$\alpha(j)\in\tau^{-1}(\{i,\ i+1,\dots,m\}).$$

Since τ is a bijection with $\tau(\mu)\geqslant\mu-1$, we conclude that there exists an $\ell\in[m]$ with $\ell\leqslant i$ such that

$$\tau^{-1}(\{i,\ i+1,\dots,m\}) = \{\ell\}\,\dot\cup\,\{i+1,\dots,m\}. \tag{A.1}$$

To define the assignments for instance $\mathcal{I}''$, we distinguish two cases $\tau(i+1)=i$ vs. $\tau(i+1)>i$. If $\tau(i+1)>i$, then the set $\{i+1,\dots,m\}$ is closed under τ and $\tau(\ell)=i$. Otherwise, $\tau(i+1)=i$ and hence $\tau(\ell)\geqslant i+1$. We define

$$\varphi\colon\mu\mapsto\begin{cases}\tau(i+1), & \text{if } \mu=i+1 \text{ and } \tau(i+1)\geqslant i+1,\\ \tau(\ell), & \text{if } \mu=i+1 \text{ and } \tau(i+1)=i,\\ \tau(\mu), & \text{if } \mu>i+1,\end{cases}\qquad \beta\colon j\mapsto\begin{cases}\alpha(j), & \text{if } \alpha(j)\geqslant i+1,\\ i+1, & \text{if } \alpha(j)=\ell.\end{cases}$$

Feasibility of assignment φ follows from the feasibility of τ. The image of assignment β is clearly in the set $[m]\setminus[i]$, and from (A.1) follows that β is defined for all jobs in $\mathcal{J}''$.

The proof of feasibility is similar to that in Case 1. For machines $\mu > i+1$ property $(\star)$ follows immediately from the feasibility of the solution τ and α. For $\mu = i+1$, we express the set of assigned jobs as

$$\beta^{-1}(i+1) = \left(\alpha^{-1}(i+1)\cap\mathcal{J}''\right)\ \dot\cup\ \left(\alpha^{-1}(\ell)\cap\mathcal{J}''\right).$$

For $t\geqslant f_i$ we have $\mathcal{J}(t)\subseteq\mathcal{J}''$; hence,

$$\begin{aligned}\beta^{-1}(i+1)\cap\mathcal{J}(t) &= \left(\alpha^{-1}(i+1)\cap\mathcal{J}(t)\right)\ \dot\cup\ \left(\alpha^{-1}(\ell)\cap\mathcal{J}(t)\right)\\ &= \begin{cases}\alpha^{-1}(i+1)\cap\mathcal{J}(t), & \text{if } \tau(\ell)=i,\\ \alpha^{-1}(\ell)\cap\mathcal{J}(t), & \text{if } \tau(i+1)=i,\end{cases}\end{aligned}$$

since for μ with $\tau(\mu) = i$ we have $\alpha^{-1}(\mu) \cap \mathcal{J}(\mu) = \emptyset$. Therefore, for $t \geqslant f_i$

$$\begin{aligned}
\left|\beta^{-1}(i+1) \cap \mathcal{J}(t)\right| &= \begin{cases} \left|\alpha^{-1}(i+1) \cap \mathcal{J}(t)\right|, & \text{if } \tau(\ell) = i, \\ \left|\alpha^{-1}(\ell) \cap \mathcal{J}(t)\right|, & \text{if } \tau(i+1) = i \end{cases} \\
&= \begin{cases} C, & \text{if } \tau(\ell) = i \text{ and } t < f_{\tau(i+1)}, \\ 0, & \text{if } \tau(\ell) = i \text{ and } t \geqslant f_{\tau(i+1)}, \\ C, & \text{if } \tau(i+1) = i \text{ and } t < f_{\tau(\ell)}, \\ 0, & \text{if } \tau(i+1) = i \text{ and } t \geqslant f_{\tau(\ell)} \end{cases} \\
&= \begin{cases} C, & \text{if } t < f_{\varphi(i+1)}, \\ 0, & \text{if } t \geqslant f_{\varphi(i+1)}. \end{cases}
\end{aligned}$$

For $t \in [s_{i+1}, f_i)$ we make the following observation for the set $\mathcal{J}'' \cap \mathcal{J}(t)$: any job $j \in \mathcal{J}(t)$ starts before time point s_{i+2}, hence $\alpha(t) \leqslant i+1$; simultaneously, for any job $j \in \mathcal{J}''$ we have $\alpha(j) \in \{\ell\} \cup \{i+1, \ldots, m\}$. Overall, we conclude that $\alpha(j) \in \{i+1, \ell\}$ holds for any $j \in \mathcal{J}'' \cap \mathcal{J}(t)$. Hence

$$\begin{aligned}
\mathcal{J}'' \cap \mathcal{J}(t) &= \{j \in \mathcal{J}'' \cap \mathcal{J}(t) \mid \alpha(j) = i+1\} \,\dot\cup\, \{j \in \mathcal{J}'' \cap \mathcal{J}(t) \mid \alpha(j) = \ell\} \\
&= \left(\alpha^{-1}(i+1) \cap \mathcal{J}'' \cap \mathcal{J}(t)\right) \,\dot\cup\, \left(\alpha^{-1}(\ell) \cap \mathcal{J}'' \cap \mathcal{J}(t)\right) \\
&= \beta^{-1}(i+1) \cap \mathcal{J}(t).
\end{aligned}$$

Analogously to Case 1, we conclude that $\left|\beta^{-1}(i+1) \cap \mathcal{J}(t)\right|$ for $t \in [s_{i+1}, f_i)$. Furthermore, we again note that $\beta^{-1}(i+1) \cap \mathcal{J}(t) = \emptyset$ for $t < s_{i+1}$. This shows that $(\star)$ holds for assignments φ and β in Case 3, which finalizes the proof of Lemma 5. □

Appendix B. Algorithm for 2-FlexMISMA

In the listing below, for better readability, we use an operator $get(S, k)$ that receives a set S and a number $k \in \mathbb{N}$ as input and returns a list of k distinct elements of S, and simultaneously removes the elements of the list from S. If the set S contains less than k elements, the algorithm aborts.

Algorithm 2: Greedy algorithm for 2-FlexMISMA

Input: number of machines m, capacity C, start times $(s_i)_{i\in[m]}$, end times $(f_i)_{i\in[m]}$, jobs $[n]$ with $b_j = a_j + 2$ for all $j \in [n]$
Output: assignments $\alpha\colon [n] \to [m]$, $\tau\colon [m] \to [m]$
$E \coloneqq \emptyset$, $F \coloneqq \emptyset$, $h \coloneqq \perp$;
foreach $t = 1, \dots, T$ **do**
  $E \coloneqq E \cup \{i \in [m] \mid s_i = t\}$;
  // close machines:
  $X \coloneqq \{i \in [m] \mid f_i = t\}$;
  if $X \neq \emptyset$ **then**
    $[i_1, \dots, i_{|X|}] \coloneqq get(E, |X|)$;
    foreach $l = 1, \dots, |X|$ **do**
      $\tau(i_l) \coloneqq X_l$;
  // assign starting jobs:
  $\bar{J}(t) \coloneqq \{j \in [n] \mid a_j = t\}$;
  $n_t \coloneqq \left|\bar{J}(t)\right|$;
  $k_t,\ r_t \coloneqq \lfloor n_t / C \rfloor,\ n_t \bmod C$; // $n_t = k_t \cdot C + r_t$
  let $\bar{J}(t) = \dot{\bigcup}_{l=1}^{k_t} J_t^l \,\dot{\cup}\, R_t$, s.t. $\left|J_t^l\right| = C$ for all l ; // partition the set of starting jobs
  $N \coloneqq [e_1, \dots, e_{k_t}] \coloneqq get(E, k_t)$;
  foreach $l = 1, \dots, k_t$ **do**
    $\alpha(j) \coloneqq e_l$ for all $j \in J_t^l$;
  if $R_t \neq \emptyset$ **then**
    if $h \neq \perp$ **then**
      if $\left|\alpha^{-1}(h) \cap J(t)\right| \leqslant C - r_t$; // half-full machine has enough space for r_t jobs
       then
        $\alpha(j) \coloneqq h$ for all $j \in R_t$;
      else
        $[e] \coloneqq get(E, 1)$;
        $\alpha(j) \coloneqq e$ for all $j \in R_t$;
        $E \coloneqq E \cup \{h\}$;
        $h \coloneqq e$;
    else
      $[e] \coloneqq get(E, 1)$;
      $\alpha(j) \coloneqq e$ for all $j \in R_t$;
      $h \coloneqq e$;
  else
    if $h \neq \perp$ **then**
      $E \coloneqq E \cup \{h\}$;
      $h \coloneqq \perp$;
  // prepare for next iteration:
  $E \coloneqq E \cup F$; // machines that were full will become empty
  $F \coloneqq N$; // machines that were assigned new jobs