Efficient Algorithms for Multi-Threaded Interval Scheduling with Machine Availabilities

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

[a] *Lehrstuhl II für Mathematik, RWTH Aachen University, Pontdriesch 10-12, 52062 Aachen, Germany, {anapolska,buesing,krabs}@math2.rwth-aachen.de*

[b] *Department of Computer Science, University of 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 to model parallel processing of multiple jobs per machine: the Multi-thread Interval Scheduling with Machine Availabilities (MISMA).

In machine scheduling, maintenance plays a crucial role in guaranteeing an efficient operation. The 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 where 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 two machines, and (ii) with constantly many parallel jobs at each point in time which both also solve MISMA; and (iii) with arbitrarily many machines of capacity one each in which case MISMA is known to be NP-hard. However, we prove that increasing the capacity to two renders FlexMISMA also NP-hard for arbitrarily many machines. 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.

Keywords: Interval Scheduling, Algorithms, Complexity

1. Introduction

Interval scheduling problems frequently appear both in theory and in practice [11]. However, the classical interval scheduling problem has only limited power to model real world applications, and already slight generalizations are NP-hard [8, 3]. One example of a real world application is the inclusion of maintenance intervals for machines which is known to be NP-hard [3].

We propose two generalizations of the Interval Scheduling with Machine Availabilities problem (ISMA) [11], which in turn is a generalization of the well-studied Interval Coloring[4] problem [9, 14]. The first generalization is the Multi-thread Interval Scheduling problem with Machine Availabilities (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

*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.

demand of one, and a processing interval $[a_j, b_j)$ which has to be scheduled. The task is to schedule all jobs or to decide that no such schedule exists.

If the machines are equivalent, in particular if they have the same capacity, we may assume that the machines' end times are interchangeable. In other words, the start and end times mainly depict information about when and how the number of available machines changes. In the setting of maintenance schedules, this would mean that the maintenance team fixes the time slots for serving the machines, but does not decide which machine they maintain in which slot. Therefore, the second generalization is a version of MISMA in which the algorithm may permute the finishing times. 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 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. Interestingly, despite the added flexibility, FlexMISMA turns out to be at least as hard as MISMA.

1.1. Our Contribution

In this paper, we determine a tight classification of conditions that are required for obtaining a polynomial time algorithm. 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 instance is feasible. Subsequently, it is generally sufficient to show all algorithmic results for FlexMISMA and all hardness results for MISMA, which implies that all provided results hold for both FlexMISMA and MISMA. There is, however, the following caveat. The transformation of a MISMA instance to the equivalent FlexMISMA instance increases the machine capacities by one. This increase cannot be avoided as it is known that already the special version of MISMA with unit capacities is NP-hard [11]. For FlexMISMA, however, we show in Theorem 2 that the problem essentially boils down to solving an interval coloring instance and thus it is solvable in polynomial time. We complement the result by showing in Theorem 1 that FlexMISMA is NP-hard for machine capacity two.

We continue with an analysis of the hardness depending on the number of machines. We show in Section 3.3.2 that both MISMA and FlexMISMA can be solved in polynomial time if the number of machines is limited to at most two. The general idea is that we 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, we then transform the solution to a feasible solution for both machines.

However, this technique does not work if the number of machines is at least three. We show in Section 3.3.1 that such instances are NP-hard for both MISMA and FlexMISMA. The hardness proof has two steps. We first show in Lemma 2 that MISMA, and therefore FlexMISMA, is at least as hard as a specific permutation partition problem PPP. The problem PPP is related to similar permutation partition problem which was introduced by Garey et al. [8] 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. [8], we have to take care of some small but important differences.

1.2. Other related work

There have been many approaches to extend Interval Scheduling by restricted machine availabilities, cf. [11]. Brucker and Nordman [4] 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. [11] studied this problem using the name Interval Scheduling with Machine Availabilities (ISMA). They showed 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.

There are several approaches to allow for multitasking machines in ISMA. Mertzios et al. [12] 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 active time of the machines needed to process all jobs, and maximizing the number of processed jobs given a number of machines. Another approach to allow for multitasking machines was presented by Angelelli and Filippi [1]. They introduce Interval Scheduling with a

Resource Constraint (FISRC), where every machine and every job is additionally characterized by a resource supply or demand.

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 [10].

UFP has a geometric version called the Storage Allocation Problem (SAP) [2, 13] 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, where 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.

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. Kolen et al. define ISMA as follows [11].

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 of each machine and schedules at all times no more than C_i jobs simultaneously onto machine $i \in [m]$.

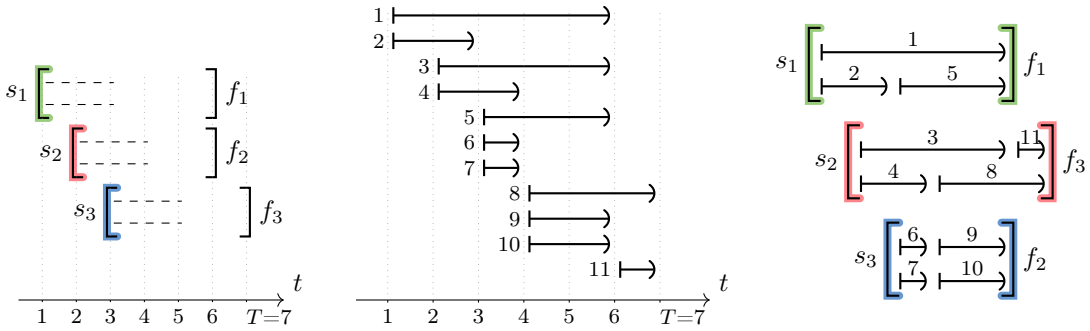
Interchangeability of the machines' end times leads to the following variant of Interval Scheduling where each machine has an assigned start time, and the end times are fixed but not preassigned to the machines.

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 $j \in \alpha^{-1}(i)$.

Without loss of generality, we assume that the earliest start time is 1, i.e., $1 = \min_{i\in[m]} s_i$, and we define the latest end time as $T := \max_{i\in[m]} f_i$. Observe that we can consider every multi-thread 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 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.

The bijective function τ can also be interpreted as a permutation $\tau \in S_m$. In the following, we therefore use the permutation representation. For example, in the solution presented in Figure 1c, the end time assignment is given by the permutation $\tau = \big(2, 3\big)$.



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

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

Remark that we can check in polynomial time whether, at some point in time, more jobs need to be processed than there are machines available, as the number of available machines at each point in time can be computed from the start and end times. If that is the case, the instance is automatically infeasible. We thus assume in the following that at no point in time more jobs need to be processed than there are machines available. Note that this does not automatically imply feasibility. In the same way, we can verify in polynomial time whether all machines are utilized to full capacity. If not all machines are utilized to full capacity, 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, we assume without loss of generality that every machine is utilized to full capacity in the available time period.

Note further that if there exist $i, k \in [m]$ so that $s_i = f_k$, we can simply remove them and reduce the number of machines by one. Otherwise, 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 not greater than $2n$, i.e., $T \leqslant 2n$.

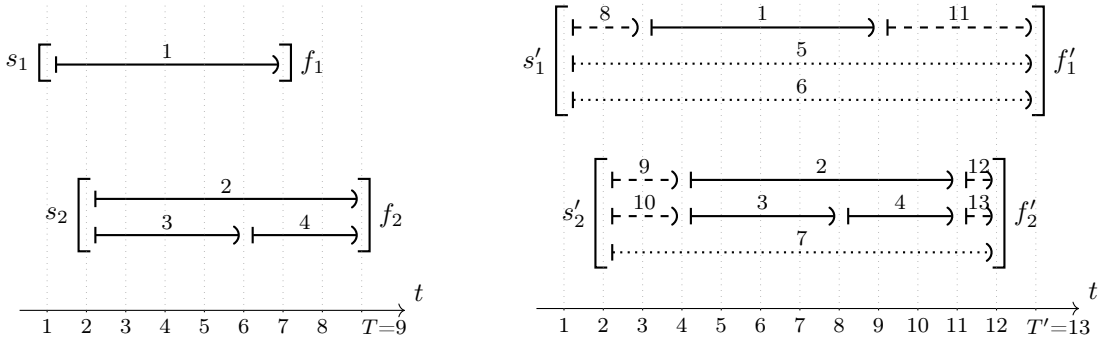
3. Complexity of the Flexible Multithread ISMA problem

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. Remarks at the end of Section 2 imply that the number of jobs is bounded from below by the total machine capacity: $n \geqslant m \cdot C$. 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.

3.1. Relation between the problems

In this section, we 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 $C' := 1 + \max_{i\in[m]} C_i$ and $n' := n + mC' + \sum_{i\in[m]} C_i$ jobs.*



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

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

Figure 2: Transformation of MISMA to FlexMISMA for an instance with two machines.

Proof. Suppose that an instance of MISMA with m machines available in periods $[s_i, f_i)$ and with capacities C_i for $i \in [m]$, and 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 $k := \sum_{i\in[m]} C_i$ and the time horizon by $T := \max_{i\in[m]} f_i$.

Then, we construct an instance of FlexMISMA with m machines, with start times $s'_i := i$ and capacity $C' := 1 + \max_{i\in[m]} C_i$ for $i \in [m]$, and with end times $f'_i := T + 2m + 1 - i$, $i \in [m]$. 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 $k' := \sum_{i\in[m]}(C' - C_i)$ denote the number of additional threads with respect to the MISMA instance. We further construct $n' := n + k' + 2k$ jobs with the following processing intervals: We first shift the periods of jobs of the ISMA instance by m. Then, we add k' jobs to occupy the additional threads. Last, we add $2k$ jobs to pad the increased availability periods, i.e.,

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

It remains to prove that the constructed FlexMISMA instance is feasible if and only if the original MISMA instance is feasible.

Let $\alpha : [n] \to [m]$ represent a feasible solution to the MISMA instance. We extend this solution to a solution for FlexMISMA as follows: choose the end time assignment $\tau = \mathrm{id}$ 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. We still assume that all machines are utilized to full capacity during their complete availability period. Moreover, all start and end times of machines are distinct by construction. 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 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 the complete availability period. Therefore, every job $j \in [n]$ is assigned by α to a machine $i \in [m]$ during the remaining availability period $[s_i + m, f_i + m)$ with capacity restriction $C' - (C' - C_i) = C_i$. This corresponds one to one to the availability periods and machine capacities 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.

3.2. Constant machine capacity

In this section, we study MISMA and FlexMISMA in the case of a fixed machine capacity C. We show that FlexMISMA can be solved in linear time for $C = 1$ using Interval Coloring, but is NP-complete for fixed capacities $C \geqslant 2$. 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 [11]. 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.

It remains to consider FlexMISMA with machine capacity equal to one. We show that in this case, the problem can be solved efficiently.

Theorem 2. 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 well-known Interval Scheduling problem. Given an instance of FlexMISMA with n jobs and m machines, we transform it into an instance of Interval Coloring with $N := n+2m$ 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 $i \in [m]$.

The Interval Coloring problem is solved by a greedy algorithm in time linear in the number of intervals, provided that the endpoints of the intervals are sorted [5]. All interval endpoints in the instance of Interval Scheduling are non-negative integers not greater than $T+1$. Therefore, counting sort can be applied, which has runtime in $\mathcal{O}(2N + T + 1) = \mathcal{O}(n)$ [6]. □

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-hard ISMA into a polynomially solvable problem.

This remark completes the study of complexity of FlexMISMA with fixed machine capacity. In the next subsection, we consider the case where the number of machines is constant.

3.3. Constant number of machines

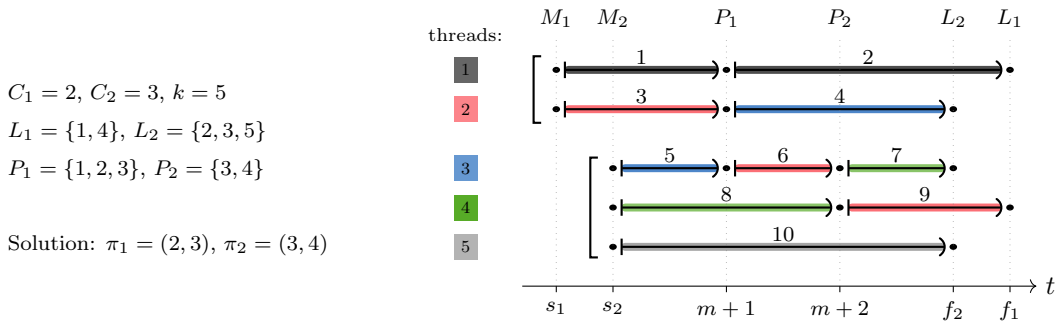
In the previous section, we have seen that FlexMISMA is NP-complete even for a constant machine capacity of two. This section is devoted to FlexMISMA's complexity in the case that the number of machines is constant.

3.3.1. FlexMISMA *is* NP*-complete for three or more machines*

We prove in this section that MISMA and thus also FlexMISMA are NP-complete for more than two machines. We prove the NP-completeness by a two-staged reduction. We already proved in Lemma 1 that FlexMISMA is at least as hard as MISMA. Next, we show that MISMA is at least as hard as the Permutation Partition problem (PPP). Last, we prove the NP-completeness of PPP by a reduction from the Directed Vertex-Disjoint Paths problem.

Beforehand, we present the *Permutation Partition 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. [8]. 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 (Permutation Partition Problem (PPP)). Given are m numbers $C_i \in \mathbb{N}$, $i \in [m]$, a partition $\mathcal{L} := \{L_1, \ldots, L_m\}$ of $[k]$, where $k = \sum_{i\in[m]} C_i$ and $|L_i| = C_i$, as well as t arbitrary subsets $P_u \subseteq [k]$ for $u \in [t]$. We further define the canonical partition $\mathcal{M} := \{M_1, \ldots, M_m\}$ of $[k]$



(a) PPP instance with $m = 2$

(b) resulting MISMA instance with color-coded job assignment

Figure 3: Transformation of PPP to MISMA.

via $M_i := \{r \in \mathbb{N} \mid \sum_{l=1}^{i-1} C_l < r \leqslant \sum_{l=1}^{i} C_l\} \subset [k]$ for $i \in [m]$ and the corresponding embedded permutation groups $G_u := \mathrm{Stab}([k] \setminus P_u) \subseteq S_k$ for $u \in [t]$. PPP then asks whether there exists a permutation $\pi \in G_t \circ \ldots \circ G_1$ such that $\pi(M_i) = L_i$ for all $i \in [m]$.

Remark that a permutation from G_u operates only on the elements of P_u and keeps all other elements fixed.

Lemma 2. MISMA *is at least as hard as* PPP*, i.e.,* PPP $\leqslant_p$MISMA.

Proof. We prove that we can solve PPP if we can solve MISMA. Given a PPP instance in the above notation, we construct a MISMA instance with m machines of capacity C_i for $i \in [m]$. The availability period of machine $i \in [m]$ is set to $[s_i := i, f_i := 2m + t + 1 - i)$. We construct k job sequences with a total of $n := k + \sum_{u \in [t]} |P_u|$ jobs. The start and end times of the first and last job in the sequences represent the two partitions $\mathcal{M}$ and $\mathcal{L}$: The first job of sequence number $l \in [k]$ has start time s_i with $l \in M_i$ and $i \in [m]$. Respectively, the last job of sequence number $l \in [k]$ has end time f_i with $l \in L_i$ and $i \in [m]$. The intermediate jobs' start and end times represent the P-sets: for every $u \in [t]$ and every $l \in P_u$, we construct a job in sequence l that ends at time unit $m + u$. For easier enumeration of the jobs, let $A_l := (u)_{u \in [t],\ l \in P_u}$ be the ascending sequence of indices of P-sets that contain $l \in [k]$. Remark that $\sum_{l \in [k]} |A_l| = \sum_{u \in [t]} |P_u|$. Then we define $f(l, z) := \sum_{r=1}^{l-1} (1 + |A_r|) + z \in [n]$ and construct for all $l \in [k]$ and $z \in [|A_l| + 1]$ a job with processing interval

$$[a_{f(l,z)}, b_{f(l,z)}) := \begin{cases} [s_i, f_{i'}), & i, i' \in [m],\ l \in M_i,\ l \in L_{i'},\ \text{and}\ |A_l| = 0, \\ [s_i, m + (A_l)_z), & i \in [m],\ l \in M_i,\ |A_l| > 0 \text{ and } z = 1, \\ [m + (A_l)_{z-1}, m + (A_l)_z), & 2 \leqslant z \leqslant |A_l|\,, \\ [m + (A_l)_{z-1}, f_{i'}), & i' \in [m],\ l \in L_{i'},\ |A_l| > 0 \text{ and } z = |A_l| + 1. \end{cases}$$

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 3. Such a more detailed assignment can be derived easily from an assignment to machines using a greedy, first fit interval scheduling algorithm.

First, assume that a solution $\pi = \pi_t \circ \ldots \circ \pi_1 \in G_t \circ \ldots \circ G_1$ for the original PPP instance is given. Then $\alpha \colon [n] \to [k]$, defined for all $l \in [k]$ as

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

is a feasible assignment of jobs to machines: 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 $\mathcal{M}$. Each permutation π_u, for $u \in [t]$, operates only on those sequences where a job ends at the corresponding time unit $m + u$, and represents that the later jobs of those

sequences are moved to threads according to π_u. Thus, the machines' capacities are respected at all times, and $L_i = \pi(M_i)$ ensures that all threads of machine $i \in [m]$ finish at the same time.

Remark that $(\pi_u \circ \ldots \circ \pi_1)(l)$ gives us the job sequence that contains the job processed on thread l at time u. Accordingly, $(\pi_u \circ \ldots \circ \pi_1)^{-1}(l)$ is the thread to which the job of sequence l at time u is assigned.

Second, assume that a solution $\alpha\colon [n] \to [k]$ to the constructed MISMA instance is given. Without loss of generality, we further assume that $\alpha(f(l,1)) = l$ for all $l \in [k]$ as otherwise we simply renumber the sequences. We then define $\tau_0 := \mathrm{id} \in S_k$ and $\tau_u(l) \in [k]$ as the thread to which the job of sequence $l \in [k]$ at time unit $m + u \in [t]$ is assigned, i.e., $\tau_u \in S_k$ with

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

We define $\pi_u := \tau_u^{-1} \circ \tau_{u-1}$ for $u \in [t]$. 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 $\pi(M_i) = L_i$: By construction, exactly the jobs $f(l, |A_l| + 1)$ with $l \in L_i$ have the same end time, and are assigned to threads in M_i, i.e., $\tau_t(L_i) = M_i$. Thus,

$$\pi(M_i) = (\pi_t \circ \ldots \circ \pi_1)(M_i) = \left(\tau_t^{-1} \circ \tau_{t-1} \circ \tau_{t-1}^{-1} \circ \ldots \circ \tau_1^{-1} \circ \tau_1\right)(M_i) = \tau_t^{-1}(M_i) = L_i.$$

This concludes the proof as the reduction can be performed in polynomial time. □

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. [8].

Lemma 3. PPP *is* NP*-complete for* $m = 3$.

Proof. An instance of the directed Vertex-Disjoint Paths problem is given by a tuple 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 H contains only arcs between two different pairs of vertices [7]. 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).

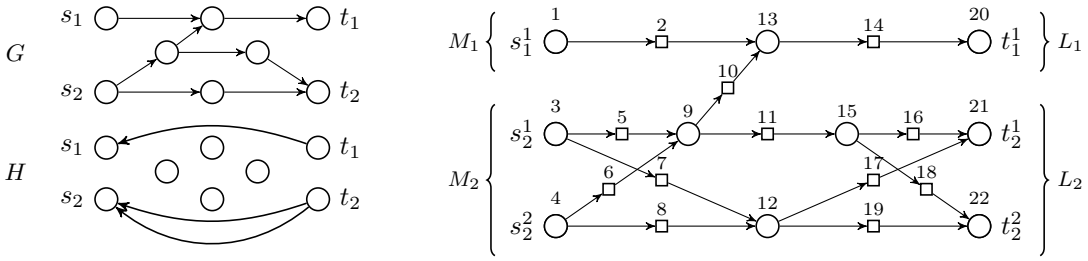
We begin by constructing an auxiliary graph G' in two steps. First, we copy the vertices s_1 and t_1 exactly k_1 times including all in- and outgoing arcs, and s_2, t_2 respectively k_2 times. We set $M_i := \{s_i^1, \ldots, s_i^{k_i}\}$ and $L_i := \{t_i^1, \ldots, t_i^{k_i}\}$ for $i \in \{1,2\}$.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

$$\begin{aligned} V' :=& (V \setminus \{s_1, s_2, t_1, t_2\}) \cup M_1 \cup M_2 \cup L_1 \cup L_2 \\ & \cup \{v_a \mid a \in A(G - \{s_1, s_2, t_1, t_2\})\} \\ & \cup \{v_a \mid a \in (M_i \times N^+(s_i)) \cup (N^-(t_i) \times L_i),\ i \in \{1,2\}\}, \text{ and} \\ A' :=& \{(v, v_a), (v_a, w) \mid a = (v,w) \in A(G)\} \\ & \cup \{(s, v_a), (v_a, w) \mid a = (s,w) \in M_i \times N^+(s_i),\ i \in \{1,2\}\} \\ & \cup \left\{(w, v_a), (v_a, t) \mid a = (w,t) \in N^-(t_i) \times L_i,\ i \in \{1,2\}\right\}, \end{aligned}$$

where $N^+(s_i) := \{v \in V(G) \mid (s_i, v) \in A(G)\}$ denotes the outgoing neighborhood of s_i and, respectively, $N^-(t_i) := \{v \in V(G) \mid (v, t_i) \in A(G)\}$ the ingoing neighborhood. Remark that any set of $k_1 + k_2$ pairwise vertex disjoint paths that start in M_i and end in L_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., such that $v < u$ for all $(v,u) \in A'$, see Figure 4.

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 M_1 and M_2, as well as L_1 and L_2. We set $M_3 := [k] \setminus (M_1 \cup M_2)$ and $L_3 := [k] \setminus (L_1 \cup L_2)$. Further, we set $t = k$ and define $P_u := \{u\} \cup \{v \in [k] \mid (v,u) \in A'\}$ for $u \in [k]$.

We claim that the constructed PPP instance is feasible if and only if the vertex-disjoint paths problem is feasible. On the one hand, let a set $\{\mathcal{P}_i^j \mid i \in \{1,2\},\ j \in [k_i],\ \mathcal{P}_i^j \text{ is } s_i^j\text{-}t_i^j\text{-path}\}$ of pairwise



(a) original graphs G and H

(b) resulting graph with subdivided arcs

Figure 4: Transformation of a Disjoint Paths instance.

vertex-disjoint paths be given. For any such path $\mathcal{P}_i^j = (p_1, \ldots, p_r)$, we define permutations $\pi_{p_l} \in G_{p_l}$ via $\pi_{p_l} = (p_{l-1},\ p_l)$ for $2 \leqslant l \leqslant r$ and $\pi_{p_1} = \mathrm{id}$. For all $u \in [k]$ that are not part of any vertex-disjoint path, we also set $\pi_u = (1)$. Then, $(\pi_{p_l} \circ \ldots \circ \pi_{p_1})(p_1) = p_l$ gives us the l-th vertex on the path from p_1 to p_r; in particular, $(\pi_{p_r} \circ \ldots \circ \pi_{p_1})(p_1) = p_r$. Moreover, an additional composition with any other 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, also $\pi(s_i^j) = t_i^j$ for $\pi := \pi_k \circ \pi_{k-1} \circ \ldots \circ \pi_2 \circ \pi_1$ and all $i \in \{1, 2\}$ and $j \in [k_i]$. Hence $\pi(M_i) = L_i$ for $i \in \{1, 2\}$, and, since π is bijective, also $\pi(M_3) = L_3$ follows. Therefore, the constructed 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 modify the resulting paths 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 $(s_i^j, \pi_1(s_i^j), (\pi_2 \circ \pi_1)(s_i^j), \ldots, \pi(s_i^j) = t_i^j)$ represents an s_i^j-t_i^j-path. To this end, we show by induction over u that

$$(\pi_u \circ \ldots \circ \pi_1)(s_i^j) = \begin{cases} v \in [k] & \text{with } ((\pi_{u-1} \circ \ldots \circ \pi_1)(s_i^j), v) \in A',\ \text{or} \\ (\pi_{u-1} \circ \ldots \circ \pi_1)(s_i^j) & \end{cases} \tag{E1}$$

for all $i \in \{1, 2\}$ and $j \in [k_i]$. We start with the case $u = k$: We already know that $(\pi_k \circ \ldots \circ \pi_1)(s_i^j) = t_i^j$ and $\pi_k \in G_k$ with $P_k = \{k\} \cup \{j \mid (j, k) \in A'\}$. As $k \notin P_u$ for $u \neq k$ by construction, we have $\pi_u(k) = k$ for all $u \neq k$. If $t_i^j \neq k$, then $t_i^j \notin P_k$ and thus $\pi_k(t_i^j) = t_i^j$ and therefore $(\pi_{k-1} \circ \ldots \circ \pi_1)(s_i^j) = t_i^j$. If $t_i^j = k$, then $\pi(s_i^j) = k$, and hence $\pi(k) \neq k$. Therefore, there exists a $v \in P_k \setminus \{k\}$ 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 (E1) be true for all $u \leqslant l \leqslant k$ where $u > 1$, and let $v = (\pi_{u-2} \circ \ldots \circ \pi_1)(s_i^j)$, i.e., the sequence $(\pi_{u-1}(v), \pi_u(\pi_{u-1}(v)), \ldots, (\pi_k \circ \ldots \circ \pi_u)(\pi_{u-1}(v)) = t_i^j)$ represents a path from $\pi_{u-1}(v)$ to t_i^j in G'. We prove that (E1) holds also true for $u - 1$, i.e., 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'. It thus remains to prove that either $\pi_{u-1}(v) = v$ or $\pi_{u-1}(v) = w \in [k]$ with $(v, w) \in A'$. Recall that $\pi_{u-1} \in G_{u-1}$ and $P_{u-1} = \{u-1\} \cup \{w \in [k] \mid w \leqslant u-1,\ (w, u-1) \in A'\}$. Further, by definition of v, we have either $v \in P_w$ for a $w \leqslant u - 2$ or $u = s_i^j$.

If $v \notin P_{u-1}$, then $\pi_{u-1}(v) = v$ by definition and (E1) holds true for $u - 1$. So, let $v \in P_{u-1}$, and thus $v = u - 1$ or $(v, u-1) \in A'$. If $v = u - 1$, then $v \notin P_w$ for a $w \leqslant u - 2$. Therefore, $v = s_i^j$ and thus $P_{u-1} = \{u - 1\}$ and $\pi_{u-1} = (1)$ and (E1) holds true for $u - 1$.

So suppose that $v \in P_{u-1} \setminus \{u - 1\}$, i.e., $(v, u - 1) \in A'$ and $v < u - 1$. Assume that $\pi_{u-1}(v) \notin \{v, u - 1\}$, i.e., especially $\pi_{u-1}(v) < u - 1$. Then $(v, u - 1)$ and $(\pi_{u-1}(v), u - 1)$ are two distinct arcs in A'. Moreover, v and $\pi_{u-1}(v)$ are both subdivision vertices by construction of G'. Hence $\pi_{u-1}(v) \notin P_w$ for any $w \geqslant u$, and thus $(\pi_k \circ \ldots \circ \pi_{u-1})(v) = v$. This is a contradiction to the assumption $\pi(s_i^j) = t_i^j$. Therefore, either $\pi_{u-1}(v) = v$ or $\pi_{u-1}(v) = u - 1$ and (E1) holds true $u - 1$.

In conclusion, (E1) 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 follows directly from Lemma 2 and Lemma 3. Thus, FlexMISMA is also NP-complete by Lemma 1.

Theorem 1. *Both* MISMA *and* FlexMISMA *are* NP*-complete if the number of machines is fixed and greater than two.*

3.3.2. Polynomial-time algorithm for two machines

Having seen that FlexMISMA is NP-complete for three machines, we next present a polynomial algorithm for solving the problem with two machines.

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

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

Note that all constraints of FlexMISMA are satisfied for a machine with assigned end time if the jobs of a feasible set are assigned to it. In general, finding a feasible job set for one machine and some end time is not sufficient to solve FlexMISMA. However, this is sufficient if an instance of FlexMISMA has only two machines.

Lemma 4. *For an 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 feasible 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 feasible for machine 2 with end time* $f_{i'}$*.*

Proof. By construction, assignment $\tau = (1\ i)$ 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 execution at an arbitrary time unit $t \in \mathbb{N}$. 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 Sec. 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 must be being processed at time $t \in \mathbb{N}$. By definition, the job set $\mathcal{J}_1$ fully utilizes the availability period of machine 1, i.e.

$$J_1(t) = \begin{cases} C, & \text{if } s_1 \leqslant t < f_i, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

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

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

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

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 (1) from Equation (2a) 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 (1) and (2b), 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. □

As a consequence of Lemma 4, it suffices to find a feasible job set for one machine and end time in order to solve FlexMISMA for two machines. Therefore, we continue by proposing a method based on network flow for finding such a feasible job set.

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

We use this connection 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 v vertex for every job, i.e., $V := \{u_i, w_i \mid i \in [m]\} \cup \{v_j \mid j \in [n]\}$.

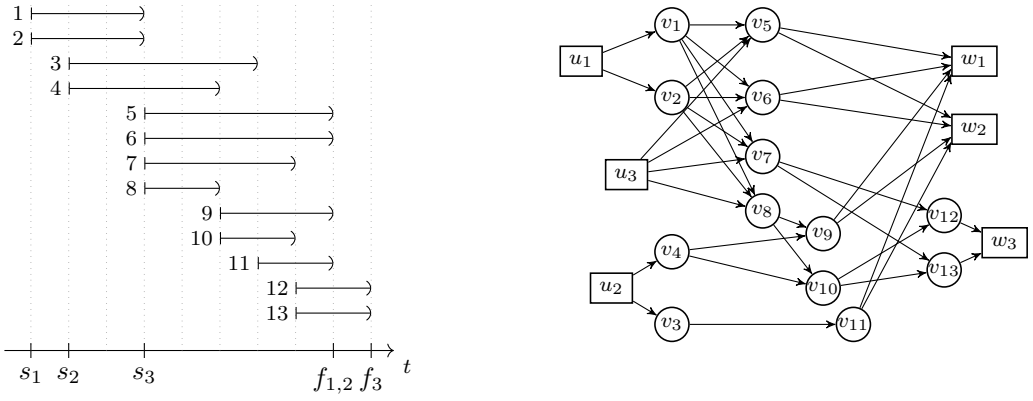
The arcs of the network G reflect the succession relationship: For machine $i \in [m]$, we construct arcs between the source vertex u_i and all vertices v_j whose corresponding jobs start at the same time as i, i.e., $s_i = a_j$. For end time number $i \in [m]$, we construct arcs between the target vertex w_i and every vertex v_j whose corresponding job ends at f_i, i.e., $b_j = f_i$. 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 is shown in Figure 5. Remark that the successor graph is acyclic and has $|V| = 2 \cdot m + n$ vertices and $|A| = \mathcal{O}\big(n^2 + m \cdot n\big)$ arcs. Therefore, its construction requires time polynomial in the size of the underlying FlexMISMA instance.

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

Lemma 5. *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 represented by the nodes that are traversed by these paths is a feasible set for machine $i \in [m]$ and end time $f_{i'}$. Conversely, if there is a feasible job set for machine i and end time $f_{i'}$, then the successor graph contains C disjoint u_i-$w_{i'}$-paths.*



(a) job set, start and end times

(b) successor graph

Figure 5: Successor graph for the FlexMISMA instance with $m = 3$ and $C = 2$.

Proof. Let C u_i-$w_{i'}$-paths $\mathcal{P}_1, \ldots, \mathcal{P}_C$ be given in the successor graph, where $i, i' \in [m]$. For each path $\mathcal{P}_l$, with $l = 1, \ldots, C$, let $J_l \subseteq [n]$ be the set of corresponding jobs, i.e., $J_l = \{j \in [n] \mid v_j \in \mathcal{P}_l\}$. 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 feasible 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,

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

holds true and $\mathcal{J}$ is a feasible set for machine i.

Conversely, let $\mathcal{J} \subseteq [n]$ be a feasible 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 interval have different colors. Such a coloring exists by definition of a feasible job set. Next, we color the corresponding transit vertices in the successor graph accordingly. In addition, for easier notation, we assign all C colors to vertices u_i and $w_{i'}$.

Next, 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 feasible set for machine i, for every time unit t the set $\mathcal{J}$ contains C jobs spanning t. Therefore, at any time unit of the machine's availability period and for each color $l \in [C]$, there is a job 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 $\mathcal{J}$. □

Lemma 4 and Lemma 5 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. We suggest using a MaxFlow algorithm to search for such disjoint paths. Computing a MaxFlow in a graph with edge-capacities equal to 1 results in general in a family of 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 are incident to v_j^- whereas the outgoing arcs of the original vertex are incident to v_j^+.

We use 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. The procedure solving FlexMISMA with two machines works as follows after the successor graph is constructed. First, ask for a u_1-w_1 flow of value C. If there is no such flow, repeat the request for the target w_2 instead of w_1. If again no flow was found, abort — the instance is infeasible. Once a u_1-w_i flow ϕ_1 of value C was found for some $i \in \{1, 2\}$, delete the sub-graph induced by the flow from the network. Next, assign the end time f_i and all jobs $j \in [n]$ for which the corresponding node v_j was traversed by the flow to machine 1. The remaining graph, called *reduced*, contains exactly one source node, u_2, and one target node, say w'. Next, ask if the reduced graph has a u_2-w' flow of value C. If yes, the remaining end time, as well as all remaining jobs, are assigned to machine 2. Otherwise, the instance is infeasible. In this manner, the procedure not only finds two families of disjoint paths in the successor graph, but also directly constructs a solution for FlexMISMA.

The runtime of the procedure for $m = 2$ is determined by the runtime of the MaxFlow subroutine, which is called at most three times. Hence, the procedure runs in polynomial time and its correctness follows from Lemmata 5 and 4. Note that, due to Lemma 4, the flow request for source u_2 serves merely to verify the feasibility of the input instance.

3.4. Constant number of threads

To complete the complexity overview, 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 [11].

The algorithm for ISMA suggested by Kolen et al. [11] 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 required fixed end times.

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 also 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. 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. Furthermore, we provided constructive algorithms for the polynomial-time solvable cases.

For both problems, there are natural optimization versions where we aim to find a maximum cardinality or maximum weight subset of jobs that can be scheduled. We think that an interesting direction for future work is to find approximation algorithms for these problems. 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, see Section 1.2.

References

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

[2] Reuven Bar-Yehuda, Michael Beder, and Dror Rawitz. A constant factor approximation algorithm for the storage allocation problem. *Algorithmica*, 77(4):1105–1127, 2017.

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

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

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

[6] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction To Algorithms*, chapter 8.2. MIT Press, Cambridge, 2 edition, 2001.

[7] 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.

[8] 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.

[9] Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs*, chapter 8 – Interval Graphs, pages 171 – 202. Academic Press, 1980.

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

[11] Antoon W.J. Kolen, Jan Karel Lenstra, Christos H. Papadimitriou, and Frits C.R. Spieksma. Interval scheduling: A survey. *Naval Research Logistics (NRL)*, 54(5):530–543, 2007.

[12] George B. Mertzios, Mordechai Shalom, Ariella Voloshin, Prudence W.H. Wong, and Shmuel Zaks. Optimizing busy time on parallel machines. *Theoretical Computer Science*, 562:524–541, 2015.

[13] Tobias Mömke and Andreas 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.

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