Coworking Scheduling with Network Flows

Mariia Anapolska[*]
Research Group Combinatorial Optimization
RWTH Aachen University, Germany
anapolska@combi.rwth-aachen.de

Christina Büsing[†]
Research Group Combinatorial Optimization
RWTH Aachen University, Germany
buesing@combi.rwth-aachen.de

Tabea Krabs[‡]
Research Group Combinatorial Optimization
RWTH Aachen University, Germany
krabs@combi.rwth-aachen.de

Tobias Mömke[§]
Department of Computer Science
University of Augsburg, Germany
moemke@informatik.uni-augsburg.de

ABSTRACT

Collaborative usage of resources is becoming increasingly popular in various fields. One common example are coworking spaces — office rooms with work places that can be rented by individuals on hourly basis. We consider the problem of assigning all booking requests for a day to equivalent office rooms with different but fixed opening times and fixed interchangeable closing times. The closing times are flexible due to daily maintenance, e.g. cleaning, which must be done in all rooms in an arbitrary order.

This problem is related to the known Interval Scheduling Problem with Machine Availabilities (ISMA), where each machine has a contiguous availability interval, and each job presents a specific time interval which has to be scheduled. According to our coworking scheduling application, we extend ISMA to Flexible Multithread ISMA (FlexMISMA) by introducing machine capacities that model the number of work places per room and by allowing to permute the end times of machines' availability periods.

In this paper, we determine a tight classification of necessary conditions for the existence of a polynomial time algorithm for FlexMISMA, assuming P ≠ NP. More specifically, we develop a network flow model and present polynomial time algorithms for instances (i) with two machines, and (ii) with arbitrarily many machines of capacity one each. In the same time, we prove that increasing the machine capacity to two renders FlexMISMA NP-hard for arbitrarily many machines. Furthermore, we complement result (i) by showing that the problem is NP-hard already for instances with three machines as a special case of the Vertex-Disjoint Paths problem.

[*]Supported by the Freigeist-Fellowship of the Volkswagen Stiftung and by the German research council (DFG) Research Training Group 2236 UnRAVeL.
[†]Supported by the German Federal Ministry of Education and Research (grant no. 05M16PAA) within the project "HealthFaCT - Health: Facility Location, Covering and Transport', by the Freigeist-Fellowship of the Volkswagen Stiftung and by the German research council (DFG) Research Training Group 2236 UnRAVeL.
[‡]Supported by the Freigeist-Fellowship of the Volkswagen Stiftung and by the German research council (DFG) Research Training Group 2236 UnRAVeL.
[§]Partially supported by the DFG Grant 439522729 (Heisenberg-Grant).

1 INTRODUCTION

Collaborative usage of resources is becoming increasingly popular in various fields, presenting new challenges for planning and scheduling. For example, small businesses rent parts of storage facilities or computer clusters for fixed time intervals when owning a whole facility is not economical. One further example of such collaboration are coworking spaces – office spaces typically consisting of several office rooms, with work places that can be rented by individuals on an hourly basis. Consider a coworking space with several identical office rooms. The customers can book single desks for an arbitrary continuous time period during the day, and they are guaranteed not to be required to change the room during their stay. Every day at the beginning and at the end of the official opening hours, a cleaning team must visit all the rooms one by one. Cleaning every room takes the same amount of time, thus the times at which the team switches to another room are fixed (these times are equal to the start time of the cleaning phase plus multiples of the cleaning duration). The order in which the rooms are cleaned is flexible. Since the rooms must be empty during the cleaning, the order of room cleaning defines the effective availability periods of rooms to the customers. When a new booking comes in, the planner needs to decide whether it can be accepted, i.e. whether there is a cleaning sequence and an assignment of all received bookings to rooms which respects the rooms' capacity and does not interfere with the cleaning.

It is easy to recognize that this problem presents a variant of Interval Scheduling with restricted machine availabilities, where machines represent the rooms, intervals represent single bookings, and restricted availability periods of machines are caused by room cleaning in the morning and in the evening. Note that the rooms accommodate several desks, which corresponds to machines being able to process several jobs at a time.

1.1 Related work

There have been many approaches to extend Interval Scheduling by restricted machine availabilities, cf. [8]. Brucker and Nordman [3] 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 the case of identical machines and a generalization where machines can process only given subsets of jobs. Later, Kolen et al. [8] 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. [9] 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 [7].

UFP has a geometric version called the Storage Allocation Problem (SAP) [2, 10] where all scheduled jobs have to be drawn as non-overlapping axis-parallel rectangles. SAP with uniform job demands corresponds to a multithread version of ISMA, 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.

1.2 Our Contribution

To model the coworking scheduling problem, we propose a twofold generalization of the Interval Scheduling problem with Machine Availabilities (ISMA) [8].

For one thing, we switch to *multithread* machines, i.e., we allow machines to process several jobs at a time. In order to keep the machines equivalent, we require all machines to have the same capacity. Note that in this case, we may assume that the machines' end times are interchangeable.

In the setting of coworking scheduling, we assume that all rooms are identical, and that the transition time between the rooms for the cleaning team can be neglected, so the rooms can be cleaned in any order. However, every time the cleaning crew enters a new room in the evening, the room becomes unavailable for the customers. This implies that the periods during which the rooms are available for bookings are not fixed, since the cleaning start times can be permuted; the number of available rooms, in contrast, is fixed for any point in time.

Therefore, the second adjustment to ISMA is the possibility to permute the machine end times. In other words, the start and end times mainly depict information about when and how the number of available machines changes. 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 Flexible Multithread ISMA (FlexMISMA). Interestingly, despite the added flexibility, FlexMISMA turns out to be at least as hard as ISMA.

In this paper, we determine a tight classification of conditions that are required to obtain a polynomial time algorithm for FlexMISMA assuming $\mathsf{P} \neq \mathsf{NP}$.

We start with analyzing the hardness depending on the number of machines. In Section 3.1 we provide a network flow interpretation of the problem. With its help we show that FlexMISMA can be solved in polynomial time if the number of machines is at most two. The general idea consists in computing a schedule for only one machine at a time 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 3.2 that such instances are NP-hard for FlexMISMA.

We continue by considering the case of machines with fixed capacity and show in Theorem 2 that FlexMISMA is NP-hard for machine capacity equal to two. To this end, we show that FlexMISMA is at least as hard as ISMA, which is known to be NP-hard [8]. For FlexMISMA with unit machine capacities, however, we show in Theorem 3 that the problem essentially boils down to solving an interval coloring instance and is thus solvable in polynomial time.

2 PROBLEM FORMULATION

We formulate the problem of coworking scheduling in terms of classical machine scheduling.

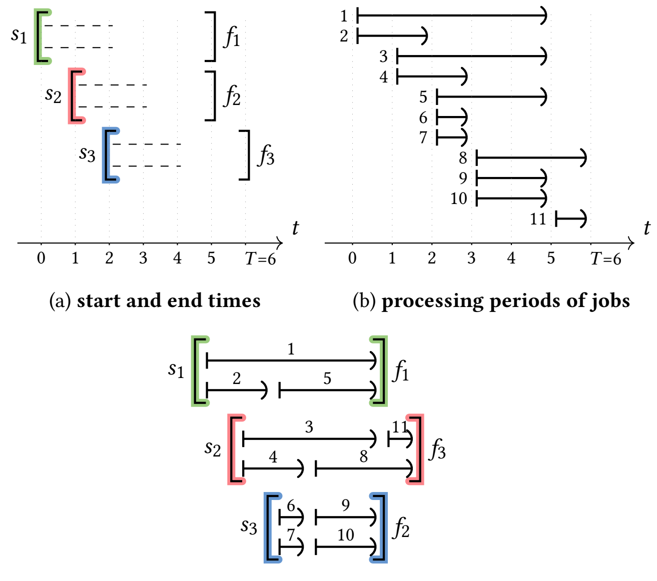
Interval Scheduling with Machine Availabilities (ISMA) is a known scheduling problem that incorporates a machine availability constraint into the Interval Scheduling problem. Specifically, ISMA assumes that every machine has a fixed availability period. Kolen et al. define ISMA as follows [8].

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.

Since ISMA assumes that every machine processes at most one job at a time, it is insufficient for modeling the entire coworking scheduling problem. We extend the formulation of ISMA to allow for *multithread* machines that can process several jobs simultaneously, which models the coworking offices hosting several desks.

Further, we integrate the interchangeability of the machines' end times (which represent the start times of the evening room cleaning). We formulate the problem in a general way by allowing arbitrary, not necessarily equidistant, start and end times. This leads to the following variant of Interval Scheduling where each machine has an assigned start time, and the end times are given but not preassigned to the machines.

Definition 2 (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



(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 capacity** $C = 2$**.**

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

All input parameters are assumed to be integer. Without loss of generality, we assume that the earliest start time is 0, i.e., $0 = \min_{i \in [m]} s_i$, 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 on set $[m]$. Therefore, we use in the following the permutation representation. For example, in the solution presented in Figure 1c, the end time assignment is given by the permutation $\tau = (2, 3)$.

The time scale of an instance of both ISMA and FlexMISMA can be compressed so as to contain only the *significant* time units – the time units which are start or end times of machines or jobs. The number of such time units is at most $2n + 2m$. Hence, we may assume that $T \leqslant 2n + 2m$. Rescaling an instance in this manner takes $O((n + m) \log(n + m))$ time.

Hence, we can check in polynomial time whether, at some point in time, more jobs need to be processed than there are machine threads available, as the number of available machines at each point in time can be derived 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 threads 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 this is not the case, we transform the instance by adding auxiliary jobs with processing intervals of length one for the respective underutilized time periods. This transformation has no influence on the feasibility of a solution and needs only a polynomial number of auxiliary jobs. Thus, we assume without loss of generality that every machine is utilized to full capacity in the available time period.

Finally, note that if there exist $i, k \in [m]$ so that $s_i = f_k$, we can simply remove them from the instance and reduce the number of machines by one. Otherwise, depending on the assignment of end times τ, we obtain either one machine with an empty availability period or 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 $i, k \in [m]$.

The size of an instance of FlexMISMA is defined by three parameters: the number of jobs n, the number of machines m and the machine capacity C. Remarks above imply that the number of jobs is bounded from below by the total machine capacity, i.e., $n \geqslant m \cdot C$. 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 COMPLEXITY FOR A CONSTANT NUMBER OF MACHINES

In this section, we start studying the complexity of FlexMISMA. We consider the case of a fixed number of machines, which corresponds to scheduling a coworking space with a constant number of rooms. The case of one machine is trivial, as it reduces to Interval Scheduling; therefore, we proceed with the case of two machines, which represents a coworking space with two equivalent rooms.

3.1 Polynomial-time algorithm for 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 for every time unit $s_i \leqslant t < f$, the set $\mathcal{J}$ contains exactly C jobs that must be processed at time t, i.e., if for all $t \in \mathbb{N}$

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

Note that all constraints of FlexMISMA are satisfied for a machine with some assigned end time if the jobs of a corresponding 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 1. *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 $f_{i'}$.*

We omit the straightforward but technical proof of this Lemma.

As a consequence of Lemma 1, 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 becomes available, 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,

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

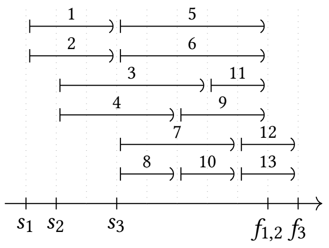
An exemplary FlexMISMA instance and its corresponding successor graph are shown in Figure 2. Remark that the successor graph is acyclic and has $|V| = 2 \cdot m + n$ vertices and $|A| = O(n^2 + m \cdot n)$ arcs. Therefore, its construction requires time polynomial in the size of the underlying FlexMISMA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

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

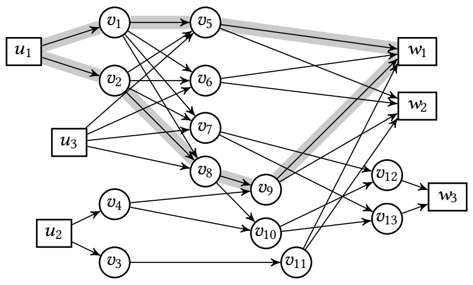
Lemma 2. *Let C vertex-disjoint u_i-$w_{i'}$-paths in the successor graph of a* FlexMISMA *instance be given, where u_i is a source node and $w_{i'}$ is a target node. Then the set of jobs represented by the nodes that are traversed by these paths is a feasible set for machine $i \in [m]$ and end time $f_{i'}$. 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.*

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 these job sets:

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



(a) **job set, start and end times**



(b) **successor graph**

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

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,

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

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 intersecting jobs 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 each 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 1 and Lemma 2 imply that to solve FlexMISMA with two machines, it suffices to find a family of C disjoint paths with common source and target vertices in the successor graph, or to show that no such family exists. We suggest using a MaxFlow subroutine to search for such disjoint paths. A flow in a graph with unit edge capacities corresponds in general to 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 make all incoming arcs of the original vertex incident to v_j^- whereas the outgoing arcs of the original vertex become 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 $i \in \{1, 2\}$, assign to machine 1 the end time f_i and all jobs $j \in [n]$ for which the corresponding node v_j was traversed by the flow. Next, delete the sub-graph induced by the flow ϕ_1 from the network. The remaining graph, called *reduced*, contains exactly one source node, u_2, and one target node, say w'. The remaining end time, as well as all remaining jobs, are assigned to machine 2. 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 is determined by the runtime of the MaxFlow subroutine, which, for $m = 2$, is called at most two times. Hence, the procedure runs in polynomial time, and its correctness follows from Lemmata 1 and 2.

Naturally, the algorithm can be applied to instances with any constant number of machines while preserving the polynomial runtime. However, since Lemma 1 does not hold for three or more machines, the algorithm is not guaranteed to find a feasible solution. An example for which the algorithm fails is shown in Figure 2: If the highlighted flow is selected in the first iteration, then the reduced graph contains no further u-w-flow of value C. Nevertheless, due to its polynomial runtime and partial correctness, our algorithm is a promising heuristic for FlexMISMA.

3.2 FlexMISMA is NP-complete for three or more machines

In the previous section, we saw a polynomial time algorithm for FlexMISMA with two machines which loses its complete correctness for three or more machines. In fact, FlexMISMA is NP-complete for more than two machines.

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

We prove the NP-completeness by a three-staged reduction. Due to the page limit, we present only a sketch of the proof. While the main ideas of the proof were already used by Garey et al. [6], we have to take care of some small but important differences.

First, we show that FlexMISMA is at least as hard as Multithread ISMA (MISMA) — an extension of ISMA which allows machines to have different capacities greater than one (but preserves fixed availability periods). The reduction is similar to the one presented further in Theorem 2. Next, we show that MISMA is at least as hard as the Permutation Partition problem (PPP), 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. [6].

PPP receives as input 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 $[k]$ 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 embedded permutation groups $G_u := \mathrm{Stab}([k] \setminus P_u) \subseteq S_k$ for $u \in [t]$. Here $\mathrm{Stab}(U)$ denotes the pointwise stabilizer of a set $U \subseteq [k]$, and S_k the symmetric group on k elements. 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 $i \in [m]$.

Finally, we prove the NP-completeness of PPP by a reduction from the Directed Vertex-Disjoint Paths problem.

4 COMPLEXITY FOR A CONSTANT MACHINE CAPACITY

In this section, we study 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 $C \geqslant 2$.

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

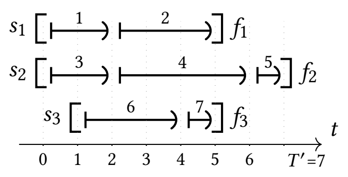
Proof. We present a reduction from ISMA, which was proven to be NP-hard [8]. Suppose that an instance of ISMA with m machines available in periods $[s_i, f_i)$ for $i \in [m]$ and n jobs with processing intervals $[a_j, b_j)$ for $j \in [n]$ is given. We denote the time horizon of this ISMA instance 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 with end times $f'_i := T+m+i$ for $i \in [m]$, and with capacity $C' := 2$, see Figure 3. Note that, by construction, all start (end) times are pairwise different, and every machine has one thread more than its counterpart in ISMA. We further construct $n' := n + 3m$ jobs with the following processing intervals. We first shift the periods of jobs of the ISMA instance by m; then, we add m auxiliary jobs to occupy the additional threads; last, we add $2m$ jobs to pad the increased availability periods. We obtain

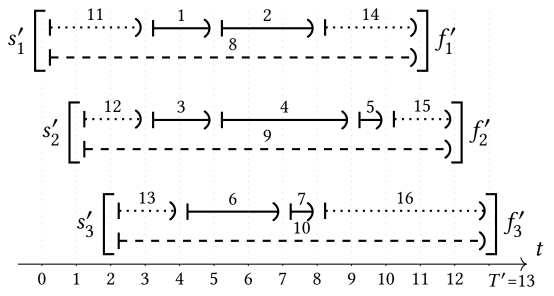
$$[a'_j, b'_j) := \begin{cases} [a_j + m, b_j + m), & j \in [n], \\ [s'_i, f'_i), & i \in [m],\ j = n + i, \\ [s'_i, s_i + m), & i \in [m],\ j = n + m + i, \\ [f_i + m, f'_i), & i \in [m],\ j = n + 2m + i. \end{cases}$$

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

Let $\alpha : [n] \to [m]$ represent a feasible solution to the ISMA instance. We extend this solution to a solution for FlexMISMA as follows: choose the end time assignment $\tau := \mathrm{id}_{[m]}$ and extend assignment α to $\alpha' : [n'] \to [m]$ by filling the additional threads



(a) **ISMA instance**



s'_1 11 1 2 14 8 f'_1
s'_2 12 3 4 5 15 9 f'_2
s'_3 13 6 7 16 10 f'_3
t
0 1 2 3 4 5 6 7 8 9 10 11 12 T'=13

(b) **resulting FlexMISMA instance with capacity** $C' = 2$

Figure 3: Transformation of ISMA to FlexMISMA for an instance with three machines.

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 $\tau = \mathrm{id}_{[m]}$ yields a feasible solution for the FlexMISMA instance.

Conversely, let $\alpha' : [n'] \to [m]$ and $\tau : [m] \to [m]$ represent a solution for the FlexMISMA instance. We still assume that all machines are utilized to full capacity during their entire 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 remaining capacity $C' - 1 = 1$. This corresponds one to one to the availability periods and machine capacities of the original ISMA instance. Thus, $\alpha'|_{[n]}$ is a feasible solution for the ISMA instance. □

It remains to consider FlexMISMA with machine capacity equal to one. We show an efficient algorithm for this case.

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 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 [4]. All interval endpoints in the instance of Interval Scheduling are non-negative integers not greater than $T+1$. Therefore, the counting sort can be applied, which has runtime in $O(2N + T + 1) = O(n)$ [5]. □

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

This observation completes the study of complexity of FlexMISMA with fixed machine capacity. We conclude that the coworking scheduling problem is polynomial time solvable for single-desk offices, but becomes NP-hard if offices have two or more desks each.

5 CONCLUSION

In this paper, we presented the interval scheduling extension FlexMISMA and provided a tight classification of its hardness with respect to the number of machines and their capacity. Furthermore, we provided constructive algorithms for the polynomial-time solvable cases.

There are natural optimization versions of the considered problem, which aim to find a maximum cardinality or maximum weight subset of jobs that can be scheduled. 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 1.1.

REFERENCES

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

[2] Reuven Bar-Yehuda, Michael Beder, and Dror Rawitz. 2017. A Constant Factor Approximation Algorithm for the Storage Allocation Problem. *Algorithmica* 77, 4 (2017), 1105–1127.

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

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

[5] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2001. *Introduction To Algorithms* (2 ed.). MIT Press, Cambridge, Chapter 8.2.

[6] M. R. Garey, D. S. Johnson, G. L. Miller, and C. H. Papadimitriou. 1980. The Complexity of Coloring Circular Arcs and Chords. *SIAM Journal on Algebraic Discrete Methods* (1980). https://doi.org/10.1137/0601025

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

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

[9] George B. Mertzios, Mordechai Shalom, Ariella Voloshin, Prudence W.H. Wong, and Shmuel Zaks. 2015. Optimizing busy time on parallel machines. *Theoretical Computer Science* 562 (2015), 524–541. https://doi.org/10.1016/j.tcs.2014.10.033

[10] Tobias Mömke and Andreas Wiese. 2020. Breaking the Barrier of 2 for the Storage Allocation Problem. In *ICALP (LIPIcs, Vol. 168)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 86:1–86:19.