# Interval Selection with Machine-Dependent Intervals

**Report**

**Author(s):**
Böhmová, Kateřina; Disser, Yann; Mihalák, Matúš; Widmayer, Peter

**Publication date:**
2013

**Permanent link:**
https://doi.org/10.3929/ethz-a-009911712



**Originally published in:**
Technical Report 786

# Interval Selection with Machine-Dependent Intervals*

**Technical Report 786, ETH Zurich, Department of Computer Science**


Kateřina Böhmová[1], Yann Disser[2], Matúš Mihalák[1] and Peter Widmayer[1]

[1]Institute of Theoretical Computer Science, ETH Zurich, Switzerland,
{katerina.boehmova, matus.mihalak, widmayer}@inf.ethz.ch
[2]Department of Mathematics, TU Berlin, Germany,
disser@math.tu-berlin.de



**Abstract**

We study an offline interval scheduling problem where every job has exactly one associated interval on every machine. To schedule a set of jobs, exactly one of the intervals associated with each job must be selected, and the intervals selected on the same machine must not intersect. We show that deciding whether all jobs can be scheduled is NP-complete already in various simple cases. In particular, by showing the NP-completeness for the case when all the intervals associated with the same job end at the same point in time (also known as just-in-time jobs), we solve an open problem posed by Sung and Vlach (J. Sched., 2005). Furthermore, we show NP-completeness for the variant with unit-length intervals where all intervals associated with the same job have a common point, and for the variant with unit-length intervals and three machines. We also study the related problem of maximizing the number of scheduled jobs. We prove that the problem is NP-hard even for two machines and unit-length intervals. We present a 2/3-approximation algorithm for two machines (and intervals of arbitrary lengths).



**Keywords:** Scheduling, Intervals, Complexity, Algorithms, Approximation


## 1 Introduction

We consider an interval scheduling problem with $m$ machines and $n$ jobs. A job consists of $m$ open intervals—each associated with exactly one machine. In other words, each job has exactly one interval on each machine. To *schedule* a job, exactly one of its intervals must be selected. To schedule several jobs, no two selected intervals on the same machine may intersect. The goal is to schedule the maximum number of jobs. We will refer to this problem as IntervalSelection.

The presented problem (much like general interval scheduling problems) is motivated by several applications, see, e.g., [2, 4, 5]. Our motivation comes from the area of car-sharing where a set of users (jobs) wish to reserve a car (machine) for a certain amount of time (interval), sufficiently large to drive to an appointment location (specific to each user) and back. The distance of the parking place of each car to the destination may vary, and this results, for each user, in various time intervals for the cars.

In the special case of a single machine, our problem becomes the classical interval scheduling problem which is solvable in polynomial time by a simple greedy algorithm that considers the intervals in increasing order of their right end-points. For the case of two machines, it can be decided in polynomial time whether all jobs can be scheduled (by a reduction to 2-Sat). In contrast to this, in the present paper we show that the same question is NP-complete for the case of three machines. Moreover, we show that the problem of maximizing the number of scheduled


*A conference version appeared in the *Proceedings of the Algorithms and Data Structures Symposium (WADS) 2013* and is available at link.springer.com

jobs is NP-hard already for two machines. Both results hold even if all the intervals have unit length.

We also consider variants of IntervalSelection where all intervals of the same job, when seen on the real line, have a non-empty intersection (e.g., this would be the time around the user's appointment in the mentioned car-sharing application). We call such a non-empty intersection a *core* of a job. We refer to IntervalSelection where each job has a core as IntervalSelection *with cores*. A special case of such a variant is when all intervals of a job have the same end-point (so called just-in-time jobs [15]). We show that, in this setting, the problem of deciding whether all jobs can be scheduled is NP-complete. This solves an open problem posed by Sung and Vlach [13, 15]. If the cores do not have to be at the right-end of the intervals, we show that deciding whether all jobs can be scheduled is NP-complete already when all intervals have unit length.

Our problem can be seen as a special case of the *job interval selection problem*, denoted as $\text{JISP}_k$, where each job has $k$ associated intervals on the real line. To see the relation, consider the machines of an instance of IntervalSelection in any order, and just concatenate the intervals for the machines along the real line, thus creating an instance of $\text{JISP}_m$. $\text{JISP}_k$ is APX-hard for any $k \geq 2$, and only a deterministic 1/2-approximation algorithm is known (in fact, a simple greedy algorithm) [14], and a randomized $\approx \frac{e-1}{e}$-approximation algorithm [4] that gives a 3/4-approximation for $\text{JISP}_2$. We present a simple deterministic 2/3-approximation algorithm for IntervalSelection with two machines. Thus, our algorithm is the first deterministic algorithm for a non-trivial special case of $\text{JISP}_2$ that beats the barrier of 2.

Table 1 provides an overview of the known (white background) and new (grey background) complexity results for IntervalSelection and related problems. The columns distinguish three basic computational goals: scheduling *all* jobs, the *maximum number* of jobs, or jobs of *maximum weight*. Each row, from top to bottom, is a generalization of the problem in the previous row, starting with IntervalSelection on a single machine, and ending with $\text{JISP}_k$. As can be seen from the table, the (general) IntervalSelection, denoted as "no core required" in the table, is closely related to well-known and studied problems: it offers a natural generalization of the setting "with cores" [13, 15], and it is an interesting special case of $\text{JISP}_k$ [4, 5, 14]. Previous work left a gap in the understanding of the complexity of the problems (the grey areas in the table), which we address and completely close in this paper. To achieve tight hardness results for the boundary cases of 2 and 3 machines (for the decision variant), or 1 and 2 machines (for the maximization objective), we devise gadgets that we plug together using known results on a specific graph coloring problem (solvable in polynomial time), which might be of independent interest. Notably, where meaningful, the hardness results hold even if all intervals are of unit length.

**Related Work.**

The general interest in interval scheduling problems dates back to the 1950s. The classical variant, in which each job has associated an interval and can be scheduled on any of the machines (i.e., in our setting, each job has exactly the same interval on every machine) and the goal is to decide whether all the jobs can be scheduled, is polynomially solvable [1]. The maximization version is polynomially solvable as well, even if the jobs are weighted [3]. However, Arkin and Silverberg [1] showed that if each job can only be scheduled on a subset of the $m$ machines, the problem becomes NP-hard (even in the unweighted case). They also gave a $O(n^{m+1})$-time algorithm (i.e., polynomial for a constant $m$).

The special case of our problem with just-in-time jobs (i.e., where all intervals of a job have the same right end point) has been studied by Sung and Vlach [15]. They showed that the weighted version is NP-hard and presented a dynamic programming algorithm that solves the problem in time $O(m \cdot n^{m+1})$. Settling the complexity of the problem with unit-weight jobs was posed as an open problem [15]; this open problem has also been stated in a recent survey on just-in-time job scheduling [13].

As outlined beforehand, our problem is a special class of $\text{JISP}_k$ (job interval scheduling problem on a single machine with $k$ intervals per job). Nakajima and Hakimi [10] showed that the decision

Table 1: Summary of the complexity of IntervalSelection problems with $n$ jobs, and $m$ machines. The cells in gray indicate our contribution.

| | Schedule all jobs | Max # jobs | Max $\sum$ weights |
|---|---|---|---|
| single machine | $O(n \log n)$ | $O(n \log n)$ | $O(n \log n)$ |
| identical intervals per job | $O(n \log n)$ | $O(n \log n)$ | $O(n^2 \log n)$ |
| with cores, any $m$ | **NP-complete** † § | NP-hard † § | NP-hard † § |
| | $O(mn^{m+1})$ | $O(mn^{m+1})$ | $O(mn^{m+1})$ |
| no core required | NP-complete † | NP-hard † | NP-hard † |
| 2 machines | $O(n^2)$ | **NP-hard** † | NP-hard † |
| $\geq 3$ machines | **NP-complete** † | NP-hard † | NP-hard † |
| $\text{JISP}_k$ (single machine) | | | |
| 2 intervals per job | $O(n^2)$ | NP-hard† | NP-hard† |
| $\geq 3$ intervals per job | NP-complete† | NP-hard† | NP-hard† |

§ even if
– all cores at the end, or
– all cores in the middle

† even if all intervals have unit length

version of $\text{JISP}_3$ is NP-complete. Keil [7] showed that this is the case even if the intervals have the same length, while the general decision version of $\text{JISP}_2$ can be solved in polynomial time. The maximization version has been studied as outlined earlier by Spieksma [14] and Chuzhoy [4]. Erlebach and Spieksma [5] consider the weighted $\text{JISP}_k$ with more than one machine (every job has the same set of $k$ intervals on every machine) and they study myopic (single-pass) greedy algorithms.

$\text{JISP}_k$ is, in some sense, a discrete variant of the throughput-maximization problem (also known as the time-constrained scheduling problem, or the real-time scheduling problem), in which each job has a length, a release time, and a deadline, and a job is associated with the (infinite) set of intervals of given length lying between the job's release time and the deadline. Bar-Noy et al. [2] study this problem and give the currently best approximation algorithms for most of the existing variants of the problem.

There are many other, for the scope of the paper less relevant variants of scheduling where intervals "come into play". We refer to the survey by Kolen et al. [8] for more information on the topic. We also stress that online variants of the presented problems have been studied as well, see e.g., the recent paper of Sgall [12] on online throughput maximization.

## 2 Approximation of Interval Selection on Two Machines

In this section we present a 2/3-approximation algorithm for IntervalSelection with two machines. We stress that by *interval* we understand a time interval associated with both a job, and a machine. Recall that IntervalSelection on one machine is solvable by a simple greedy algorithm that considers all intervals on the machine sorted by the right end-points in the ascending order and selects each considered interval if it does not intersect any of the previously selected intervals. We denote this algorithm by $A^1$. We can also apply the greedy algorithm in the setting with two machines $M_A$ and $M_B$. More formally, let $A^2(M_A, M_B)$ be the algorithm that first runs $A^1$ on machine $M_A$, removes from $M_B$ the intervals for jobs whose intervals were selected on machine $M_A$, and runs $A^1$ on $M_B$. This algorithm gives a 1/2-approximation [14], which is tight for the algorithm.

Obviously, we can run the greedy algorithm in the other direction, i.e., first on $M_B$ and then on $M_A$ (denoted by $A^2(M_B, M_A)$), which again gives a 1/2-approximation. Perhaps surprisingly, the algorithm that chooses the better solution of the two provided by $A^2(M_A, M_B)$ and $A^2(M_B, M_A)$ is a 2/3-approximation. Even though the algorithm, let us call it $A^3$, is extremely simple, the analysis thereof is more interesting.

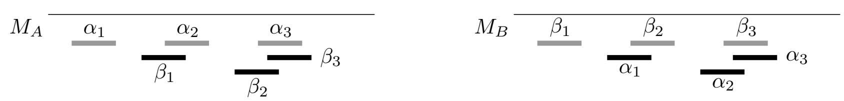


Figure 1: Instance where $A^3$ returns exactly $2/3 \cdot |O|$ jobs: $O$ contains all jobs $\alpha_i$ and $\beta_i$ for $i = 1, 2, 3$ (in grey), but both $A^2(M_A, M_B)$ and $A^2(M_B, M_A)$ schedule only the jobs $\alpha_1, \alpha_2, \beta_1, \beta_2$.

Consider an optimum solution $O$ where $O_A$ denotes the intervals selected on $M_A$ and $O_B$ the intervals selected on $M_B$. Consider $A^2(M_A, M_B)$ and let $S_A$ be the intervals selected by $A^2(M_A, M_B)$ on $M_A$. Obviously, $A^2(M_A, M_B)$ selects on $M_A$ at least $|O_A|$ intervals (which follows from the fact that $A^1$ finds an optimum on a single machine). The only reason that $A^2$ selects less than $|O_B|$ intervals on $M_B$ is that it cannot select intervals that correspond to jobs already scheduled on $M_A$ (see Figure 1 for illustration). In fact, every job scheduled on machine $M_A$ prevents selecting one interval on $M_B$ (the one that corresponds to the same job) and each such selected interval on $M_A$ can cause that we can select one interval less on $M_B$. We introduce the following definition to measure how a selection $S_A$ on $M_A$ reduces the size of the solution on $M_B$ with respect to $O$. We say that a set $I$ of intervals *reduces the selection on* $M_B$ *by* $k$ if after selecting the intervals $I$ on $M_A$ the algorithm $A^1$ selects $|O_B| - k$ intervals on $M_B$. Note that a set $I$ can never reduce the selection by more than $|I|$ intervals; in particular, a single interval can reduce the selection by at most one.

Observe in Figure 1 that the interval for the job $\beta_1$ on $M_A$ reduces the selection on $M_B$ by one, but the interval for the job $\alpha_1$ on $M_A$ reduces the selection on $M_B$ by one only with the help of $\beta_2$ on $M_A$. That is, sometimes we need more than one interval to reduce the selection by one. Accordingly, we will further distinguish the intervals in $S_A$ as follows. $S_A^O$ are the intervals that are both in $S_A$ and in $O_A$. Observe that every interval $i_O \in O_A \setminus S_A$ has an interval $i_A \in S_A$ such that its right end-point intersects $i_O$. For each such $i_O$ we place the leftmost such interval $i_A$ in the set $S_A^\cap$. We define $S_A^\emptyset$ to be the remaining intervals of $S_A$. Note that, by definition, $|S_A^O \cup S_A^\cap| = |O_A|$. Similarly, we define $S_B$ to be the intervals scheduled by the "reverse" algorithm $A^2(M_B, M_A)$ on $M_B$, and we analogically define the sets $S_B^O$, $S_B^\cap$, $S_B^\emptyset$.

Intuitively, if $S_A^\cap$ or $S_B^\cap$ is small, then the choice of $A^2(M_A, M_B)$ or $A^2(M_B, M_A)$ on the first machine reduces the selection on the second machine only a little (and thus it schedules many jobs). On the other hand, if both $S_A^\cap$ and $S_B^\cap$ are large, we need to select twice as many jobs to reduce the selection (recall the example from Figure 1). We will show that the trade-off between these constraints lies at $|S_A^\cap| = 1/3 \cdot |O|$. To make this formal, we analyze how much the selection $S_A$ reduces the selection on $M_B$.

**Lemma 1.** *Assume that $A^2(M_A, M_B)$ selects $r$ intervals on $M_A$ corresponding to jobs from $S_B^O$, $s$ intervals corresponding to jobs from $S_B^\cap$, and $t$ intervals corresponding to jobs from $O_B \setminus S_B^O$. Then the selection on $M_B$ is reduced by at most $r + \min\{s, t\}$.*

*Proof.* Observe that $O_B$ and $S_B^O \cup S_B^\cap$ are two selections of size $|O_B|$ having exactly $S_B^O$ in common. Now, it is enough to realize that, after removing the intervals corresponding to jobs in $S_A$, we can select $|O_B| - r - t$ intervals from $O_B$, and we can select $|O_B| - r - s$ intervals from $S_B^O \cup S_B^\cap$. □

**Theorem 2.** *$A^3$ is a 2/3-approximation algorithm. This bound is tight for the algorithm.*

*Proof.* Without loss of generality, we assume that $|S_A^\cap| \leq |S_B^\cap|$. We distinguish two cases. First, assume that $|S_A^\cap| \leq 1/3 \cdot |O|$. Since $S_A^O$ are the intervals from $O$, they correspond to different jobs than the jobs to which the intervals in $O_B$ correspond. Thus, on $M_A$, at most $|S_A^\cap| + |S_A^\emptyset|$ intervals corresponding to jobs in $O_B$ are selected, and the selection on $M_B$ is reduced by at most this amount. Therefore, among the intervals in $O_B$, algorithm $A^2(M_A, M_B)$ selects at least $|O_B| - |S_A^\cap| - |S_A^\emptyset|$ intervals. In total, algorithm $A^2(M_A, M_B)$ selects at least $|S_A^O| + |S_A^\cap| + |S_A^\emptyset| + |O_B| - |S_A^\emptyset| - 1/3 \cdot |O| = 2/3 \cdot |O|$ jobs.

Now, assume that $|S_B^\cap| \geq |S_A^\cap| > 1/3 \cdot |O|$. We analyze how much the intervals $S_A$ can reduce the selection on $M_B$. At most $|S_B^O|$ intervals corresponding to jobs in $S_B^O$ can be selected on $M_A$.

By Lemma 1, the selection on $M_B$ is reduced at the maximum possible way if in $S_A$ there is the same number of intervals corresponding to jobs in $S_B^{\cap}$ as the number of intervals corresponding to jobs in $O_B \setminus S_B^O$. Thus, the selection on $M_B$ will be reduced the most, if $|S_B^O|$ intervals in $S_A$ correspond to jobs in $S_B^O$, and the rest of $S_A$ is split evenly between $S_B^{\cap}$ and $O_B \setminus S_B^O$. The selection on $M_B$ can thus be reduced by at most

$$|S_B^O| + \frac{|S_A| - |S_B^O|}{2} = \frac{|S_A^{\emptyset}| + |O_A| + |S_B^O|}{2} = \frac{|S_A^{\emptyset}| + |O| - |S_B^{\cap}|}{2} \leq \frac{|S_A^{\emptyset}| + 2/3 \cdot |O|}{2}.$$

Thus, also in this case, algorithm $A^2(M_A, M_B)$ schedules at least $|S_A^O| + |S_A^{\cap}| + |S_A^{\emptyset}| + |O_B| - \frac{|O|}{3} - \frac{|S_A^{\emptyset}|}{2} \geq |O_A| + |O_B| - \frac{|O|}{3} = \frac{2}{3}|O|$ jobs.

Therefore, in every case, the algorithm $A^3$ schedules at least $2/3 \cdot |O|$ jobs. The analysis is tight, as the example from Figure 1 shows. □

As an obvious future work, we want to analyze the natural generalization of the algorithm to $m \geq 3$ machines.

## 3 Hardness Results

In this section we study the complexity of IntervalSelection and show that most of the natural variants are NP-complete or NP-hard. We first describe generic gadgets that we will use as building blocks in our hardness proofs. In the subsequent sections we give the actual hardness proofs.

Recall that by an *interval* we understand a time interval associated with both a job and a machine. In the following, we will also use time intervals not associated with a job or a machine. To avoid confusion, we use the following terminology. When we consider a time interval with respect to a single machine, but independently of the jobs, we call it a *slot*. And when considering a time interval independently of machines and jobs, we call it a *window*.

We will also use the notion of *blocking*. We say that an interval $i$ *blocks a slot* $s$ if $i$ intersects $s$ and both are associated with the same machine. We say that a set of intervals $I$ *blocks window* $w$ *on a set of machines* $\mathcal{M}$ if for each machine $M$ in $\mathcal{M}$ there is an interval in $I$ that blocks the slot corresponding to $w$ on $M$. We say that a set of intervals $I$ *completely blocks a window* $w$ if each slot that intersects the window $w$ is blocked by some interval in $I$.

We call a schedule in which all jobs are scheduled a *complete schedule*.

Our hardness results are shown by a reduction from variants of the NP-complete *satisfiability problem* (Sat). Sat is the problem of finding, for a given a set of $r$ clauses $C = \{c_1, c_2, \ldots, c_r\}$ over a set of Boolean variables $X = \{x_1, x_2, \ldots, x_s\}$, a truth assignment such that every clause is satisfied, i.e., at least one literal in every clause evaluates to `TRUE` (see, e.g., [6] for an exact definition of the problem). Sat is NP-complete, even if every clause is restricted to have at most three literals (denoted as 3-Sat) [6], and even, if each clause contains at most three literals and each variable appears in the formula at most three times, once as a negative literal and at most twice as a positive literal (denoted as ($\leq$3,3)-Sat) [6]. The problem of finding a truth assignment that maximizes the number of satisfied clauses is NP-hard, even if each clause contains two literals and each variable appears at most three times in the formula (denoted as (2,3)-MaxSat) [11].

**Building Blocks for Hardness Proofs.**

In order to simplify the explanation of the hardness proofs, we define the following two gadgets (specific sets of jobs) and use them as building blocks in our reductions.

The purpose of the *blocking gadget* is to completely block a certain window $w$, i.e., to make sure that in any complete schedule no interval that intersects $w$ is ever scheduled, with the exception of the intervals of the jobs that constitute the gadget itself. Let $w$ be a window (that we want to completely block). The gadget consists of $m$ jobs, each having $w$ as their interval on every machine. We visually depict a blocking gadget as in Figure 2.

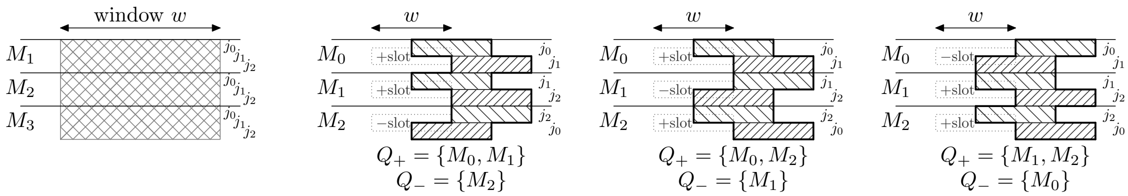


Figure 2: The first drawing illustrates how we depict a blocking gadget for a window $w$. The last three drawings illustrate decision gadgets on three machines $M_0$, $M_1$, and $M_2$. Each of the decision gadgets has two positive slots on machines in $Q_+$ and one negative slot on the machine in $Q_-$. The crucial intervals constituting the gadget are depicted by the shaded boxes (always one interval spans the respective box). The associated jobs of the intervals are indicated on the sides. The remaining intervals of the jobs are blocked by a blocking gadget, and thus never selected. These intervals and the blocking gadget are for simplicity not displayed. The two different shades in the boxes depict the only two possibilities how to select the intervals in the decision gadget.

**Lemma 3.** *In any complete schedule for an instance of* IntervalSelection *that contains the blocking gadget $B$ for window $w$, no selected interval outside $B$ intersects $w$.*

*Proof.* In any complete schedule all the jobs have to be scheduled, including all the jobs of the blocking gadget. Each of these jobs can only be scheduled onto $w$ and there are as many jobs as machines, so there is no other way than to schedule exactly one of the jobs onto the window $w$ on each machine. Thus, the selected intervals of these jobs completely block window $w$ and no other interval intersecting $w$ can be selected in the schedule. □

The purpose of the *decision gadget* is to mimic a truth assignment to a variable in a boolean formula of 3-Sat. This is done by blocking a certain window either on one set of machines or on another disjoint set. Given a window $w$ and two disjoint subsets $Q_-, Q_+$ of machines, we will call the window $w$ on the machines in $Q_+$ the *positive slots* and $w$ on $Q_-$ the *negative slots* of the gadget (cf. Figure 2). With our gadget we want to achieve that in any complete schedule either all the positive slots of the gadget are free and all the negative slots are blocked by the schedule, or vice versa. Let us refer to the former situation as the *positive decision* of the gadget and to the latter as the *negative decision*. Intuitively, we achieve this effect by using jobs with intervals placed so that we have exactly two ways how to schedule all jobs. To ensure that there is no other way to schedule the jobs of the gadget, we may need to block some intervals of these jobs. For this purpose we use a blocking gadget.

Formally, we construct the decision gadget as follows. We denote by $Q$ the union of $Q_-, Q_+$, by $k$ the size of $Q$, and by $M_0, M_1, \dots, M_{k-1}$ the machines in $Q$. Without loss of generality, we assume that $w$ has unit length. We use $k$ jobs $j_0, j_1, \dots, j_{k-1}$, one job per machine in $Q$. The intervals for all these jobs have unit length $|w|$. There is a blocking gadget $B$ such that all intervals of the decision gadget except for intervals of $j_i$ on $M_i, M_{i-1}$ intersect $B$ (we write $M_{i-1}$ instead of $M_{i-1 \bmod k}$ for simplicity). The exact placement of $j_i$ and $j_{i+1}$ on $M_i$ depends on whether the window $w$ is supposed to be a positive or a negative slot on $M_i$. In particular, if $M_i$ is in $Q_-$ ($w$ is a negative slot on $M_i$), the interval for $j_i$ is placed directly to the right of $w$ and the interval for $j_{i+1}$ is placed so that its left end is at the center of $w$. Otherwise, if $M_i$ is in $Q_+$, the left end of the interval for $j_i$ is at the center of $w$ and the interval for $j_{i+1}$ is directly to the right of $w$. Note that the intervals constituting the gadget occupy a window of length 2 (excluding the intervals that are blocked by the blocking gadget).

**Lemma 4.** *In any complete schedule for an instance of* IntervalSelection *that contains the decision gadget $D$ for window $w$ and subsets $Q_-, Q_+$ of machines, either $D$ blocks $w$ on all machines in $Q_-$ and leaves it free on all machines in $Q_+$, or vice versa.*

*Proof.* We observe that the interval for job $j_i$ intersects with the interval for job $j_{i+1}$ on machine $M_i$ and with the interval for job $j_{i-1}$ on machine $M_{i-1}$. Furthermore, because of the blocking

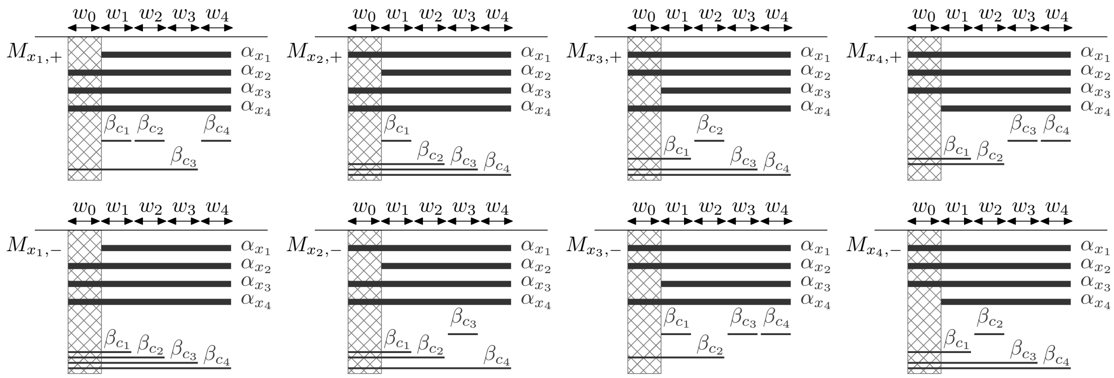

Figure 3: Example of the construction of IntervalSelection with cores at the end for an instance $\Phi$ of the 3-Sat problem (each figure shows the intervals on a single machine), where $\Phi = (x_1 \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_2} \vee \overline{x_3} \vee x_4) \wedge (x_1 \vee \overline{x_3} \vee x_4)$.

gadget, $j_i$ cannot be scheduled on any other than these two machines. First, let us assume that in a complete schedule $S$ job $j_i$ is scheduled on machine $M_i$. The schedule $S$ needs to schedule job $j_{i+1}$, which can only be scheduled on machine $M_{i+1}$. A similar situation occurs for job $j_{i+2}$. In fact, since $S$ is a complete schedule, the initial decision is propagated over all the jobs of the gadget. Conversely, if we assume that a complete schedule $S$ schedules a job $j_i$ on machine $M_{i-1}$, then each job $j_{i'}$ must be scheduled on machine $M_{i'-1}$. Finally, note that the case where each $j_i$ is scheduled on $M_i$ corresponds to the negative decision, i.e., the situation where $w$ is blocked on machines in $Q_+$. Whereas, each $j_i$ being scheduled on $M_{i-1}$ corresponds to the positive decision, i.e., window $w$ is blocked on machines in $Q_-$. □

**Corollary 5.** *Given a window $w$ and subsets $Q_-, Q_+$ of machines, in any complete schedule, the intervals of the decision gadget as constructed above enforce the following. Either on all the positive slots of the gadget intervals can be scheduled and all the negative slots are blocked, or vice versa.*

## 3.1 Interval Selection with Shared Cores

In this section we analyze the complexity of IntervalSelection with cores. We study two variants. First, we consider the case when every job has a core at the end, i.e., all intervals of a job end at the same point in time. We show that deciding whether there is a complete schedule for this variant is NP-complete. By this we resolve an open problem posed by Sung and Vlach [13, 15]. Afterwards, we consider the case where every job has a core at an arbitrary position and show that this variant is NP-complete even if all intervals have unit length. We note that both variants are solvable in time $O(m \cdot n^{m+1})$, and thus in polynomial time if $m$ is constant [15].

**Theorem 6.** *The problem of deciding whether there exists a complete schedule in* IntervalSelection *with cores at the end is NP-complete.*

*Proof.* The problem is in NP, since the completeness of a given schedule can be checked in linear time. To show the hardness, we present a reduction from 3-Sat.

*Construction.* Let us consider an arbitrary instance $\Phi$ of 3-Sat given by a set of clauses $C = \{c_1, c_2, \ldots, c_r\}$ over a set of Boolean variables $X = \{x_1, x_2, \ldots, x_s\}$. We construct the following instance $\mathcal{S}$ of the IntervalSelection problem (cf. Figure 3 along with the construction). We use two machines for each variable $x_i$, denoted by $M_{x_i,+}$ and $M_{x_i,-}$. The machine $M_{x_i,+}$ corresponds to the positive literal of $x_i$, whereas $M_{x_i,-}$ corresponds to the negative literal of $x_i$. On the machines we consider a window of $r+1$ units and we denote the unit windows constituting it by $w_0, w_1, w_2, \ldots, w_r$. We place a blocking gadget over all machines on the window $w_0$. Next, for each variable $x_i$ we add a job $\alpha_{x_i}$ with two possible ways of scheduling it (in any complete schedule). This mimics a truth assignment to the variable $x_i$. We call these jobs the *variable jobs*. We place the intervals of a variable job $\alpha_{x_i}$ as follows. On $M_{x_i,+}$ and $M_{x_i,-}$ we place an interval such that it covers $w_1, w_2, \ldots, w_r$, and on every other machine we place an interval such that it covers $w_0, w_1, w_2, \ldots, w_r$. Note that the blocking gadget ensures that in any complete schedule each job $\alpha_{x_i}$ is scheduled on one of the machines $M_{x_i,+}$, $M_{x_i,-}$, and no other job is scheduled on that machine on any window $w_1, w_2, \ldots, w_r$. Intuitively, by scheduling $\alpha_{x_i}$, one of the two literals of $x_i$ is selected and thus set to `FALSE`, implicitly setting a truth assignment for variable $x_i$. Lastly, we add $r$ jobs linked to the clauses so that the actual scheduling of these jobs is related to the way how the clauses of $\Phi$ are satisfied. For each clause $c_j$ we have one *clause job* denoted by $\beta_{c_j}$. We place the intervals for the job $\beta_{c_j}$ on window $w_j$ on those machines that correspond to literals that appear in the clause $c_j$, and on the windows $w_0, w_1, \ldots, w_j$ on the other machines. In other words, in any complete schedule, a job $\beta_{c_j}$ can only be scheduled on a machine that corresponds to a literal that appears in clause $c_j$, since on all other machines the intervals for $\beta_{c_j}$ intersect the blocking gadget. Moreover, if the same literal appears in clauses $c_j$ and $c_{j'}$, $j \neq j'$, then the intervals for jobs $\beta_{c_j}$ and $\beta_{c_{j'}}$ do not intersect on the machine that corresponds to this literal.

Note that the constructed instance of IntervalSelection has the property that all the intervals corresponding to one job have at their end a unit window in common. Obviously, the above construction can be done in polynomial time.

*Correctness.* We now show that $\Phi$ is satisfiable if and only if there exists a complete schedule for $\mathcal{S}$. First, given a complete schedule $S$ for $\mathcal{S}$, we construct a satisfying truth assignment $A$ for $\Phi$ as follows. The blocking gadget ensures that no selected interval for a variable or clause job intersects window $w_0$. Since $S$ is a complete schedule, all the variable jobs are scheduled. Due to the blocking gadget, each variable job $\alpha_{x_i}$ can only be scheduled on $M_{x_i,+}$, or $M_{x_i,-}$; and we set the variable $x_i$ in $A$ to `FALSE` or `TRUE`, respectively. We argue that all clauses are satisfied by $A$. Let $c_j$ be any clause of $\Phi$. As $S$ is a complete schedule, the clause job $\beta_{c_j}$ that corresponds to $c_j$ is scheduled on $M_{x_i,+}$, or $M_{x_i,-}$ for some variable $x_i$ appearing in $c_j$. Assume $x_i$ appears as a positive literal in $c_j$. Then $\beta_{c_j}$ must be scheduled on $M_{x_i,+}$. Also, the variable job $\alpha_{x_i}$ has to be scheduled on $M_{x_i,-}$, since the intervals for both jobs on $M_{x_i,+}$ overlap. But this means that $A$ sets $x_i$ to `TRUE` and hence satisfies $c_j$. Analogously, $A$ satisfies $c_j$ also if $x_i$ appears as a negative literal in $c_j$.

Conversely, we can construct a complete schedule from a truth assignment $A$ that satisfies $\Phi$ as follows. We schedule all the jobs that form the blocking gadget on $w_0$ in any order. Then, we schedule all variable jobs according to $A$: If a variable $x_i$ is set to `TRUE`, we schedule $\alpha_{x_i}$ on machine $M_{x_i,-}$, otherwise on machine $M_{x_i,+}$. Lastly, as all the clauses are satisfied by assignment $A$, each clause $c_j$ is satisfied by some variable $x_i$. Either $x_i$ appears in $c_j$ as a positive literal, or as a negative literal. In the former case, $x_i$ is set to `TRUE` in $A$ and machine $M_{x_i,+}$ is not occupied by $\alpha_{x_i}$, so the clause job $\beta_{c_j}$ can be scheduled on machine $M_{x_i,+}$. In the latter case, the clause job $\beta_{c_j}$ can be scheduled on $M_{x_i,-}$. We can construct a complete schedule since no two scheduled clause jobs overlap. □

The presented hardness implies the hardness of other variants of IntervalSelection, such as that of cores at arbitrary positions, or with no required core at all. Similarly, the presented hardness implies the hardness of the maximization versions of these variants.

### Unit Interval Selection with Shared Cores

We show that this variant of the IntervalSelection problem with cores is NP-complete even if we require all the intervals to have unit length.

**Theorem 7.** *The problem of deciding whether there exists a complete schedule in* IntervalSelection *with cores is NP-complete even if all intervals have unit length.*

*Proof.* The problem is obviously in NP. To show NP-hardness, we present a reduction from (≤3,3)-Sat.

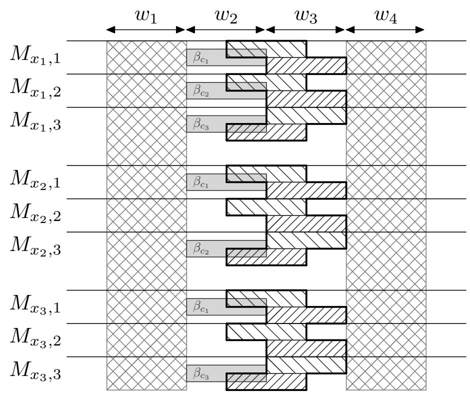


Figure 4: Instance $\mathcal{S}$ of our problem corresponding to an instance $\Phi$ of ($\leq$3,3)-Sat where $\Phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3})$. For simplification, the intervals that intersect the blocking gadgets are not shown.

*Construction.* Let $\Phi$ be an arbitrary instance of ($\leq$3,3)-Sat given by a set of clauses $C = \{c_1, c_2, \ldots, c_r\}$ over a set of Boolean variables $X = \{x_1, x_2, \ldots, x_s\}$, where each variable appears in $\Phi$ at most three times, once as a negative literal and once or twice as a positive one.

We construct an instance $\mathcal{S}$ of the scheduling problem as follows (cf. Figure 4 along with the construction). We introduce three machines for each variable $x_i$, denoted by $M_{x_i,1}, M_{x_i,2}$, and $M_{x_i,3}$. On the machines we consider a window of four units and denote the unit windows constituting it by $w_1, w_2, w_3, w_4$. We introduce jobs as follows, using unit length intervals only. We place two blocking gadgets spanning all machines, one on the window $w_1$ and the other on $w_4$. For each variable $x_i$, we place a decision gadget $D_{x_i}$ on machines $M_{x_i,1}, M_{x_i,2}, M_{x_i,3}$, such that $D_{x_i}$ has positive slots on $w_2$ on $M_{x_i,1}, M_{x_i,2}$, and a negative slot on $w_2$ on $M_{x_i,3}$ (i.e., $Q_+ = \{M_{x_i,1}, M_{x_i,2}\}$, $Q_- = \{M_{x_i,3}\}$) and occupies the windows $w_2$ and $w_3$. Recall that each decision gadget requires a blocking gadget—we use the blocking gadget on $w_4$ for this purpose. We place the intervals of $D_{x_i}$ that need to be blocked in such a way that they cover three quarters of the window $w_3$ and one quarter of $w_4$. By this, we achieve that they are never selected in any complete schedule and, at the same time, all the intervals for a single job of $D_{x_i}$ have a window in common (the second quarter of $w_3$). The decision of $D_{x_i}$ in a complete schedule will correspond to a truth assignment to variable $x_i$: A positive decision will correspond to a `TRUE` assignment and a negative decision will correspond to a `FALSE` assignment. Note that the positive/negative decision of $D_{x_i}$ is independent of the decisions of the other gadgets constructed the same way. Finally, we introduce a *clause job* $\beta_{c_j}$ for each clause $c_j$. Recall that each variable appears in $\Phi$ at most three times, once as a negative literal and once or twice as a positive literal. For each appearance of a variable $x_i$ as a positive literal in $c_j$ we place an interval $\beta_{c_j}$ on an unoccupied positive slot of the gadget $D_{x_i}$. Similarly, if $x_i$ appears in $c_{j'}$ as a negative literal, we place an interval for $\beta_{c_{j'}}$ on the negative slot of $D_{x_i}$. Note that we can place the clause jobs on the slots so that no slot is used twice, since each $x_i$ appears at most twice as a positive literal and once as a negative literal and $D_{x_i}$ has two positive slots and one negative. We place all the remaining intervals for clause jobs in a way that they cover half of the window $w_1$ and half of $w_2$. Because of the blocking gadget at $w_1$, none of these intervals can be selected in any complete schedule. Observe also that each clause job has a core (the first half of $w_2$), all intervals have the same length, and that the construction can be done in polynomial time.

*Correctness.* First, suppose that there is a complete schedule $S$ for $\mathcal{S}$. We construct a satisfying truth assignment $A$ for $\Phi$ as follows. For each variable $x_i$ we look at the schedule for the corresponding decision gadget $D_{x_i}$: If it decides positively, we set the value of $x_i$ in $A$ to `TRUE`, otherwise to `FALSE`. We show that every clause $c_j$ of $\Phi$ is satisfied by the resulting assignment $A$. The completeness of $S$ ensures that $\beta_{c_j}$ is scheduled on some machine, say $M_{x_i,k}$. Either it is scheduled on one of the two positive slots of $D_{x_i}$ and $x_i$ appears in $c_j$ as a positive literal, or it is scheduled on the negative slot of $D_{x_i}$ and $x_i$ appears in $c_j$ as a negative literal. The former case implies a positive decision of the gadget $D_{x_i}$, and hence $x_i$ is `TRUE` in $A$ by construction. The latter case implies a negative decision of $D_{x_i}$ and $x_i$ being set to `FALSE` in $A$. In both cases the clause $c_j$ is satisfied by $x_i$ in $A$.

Conversely, given a truth assignment $A$ that satisfies $\Phi$, we construct a complete schedule for $\mathcal{S}$ as follows. We schedule all the jobs of the blocking gadgets in any order. We schedule all the jobs constituting a decision gadget $D_{x_i}$ so that if the variable $x_i$ is `TRUE` in $A$, we make a positive decision for $D_{x_i}$, and if $x_i$ is `FALSE` in $A$, we make a negative decision for $D_{x_i}$. We schedule the clause job for each clause $c_j$ as follows. Since $A$ satisfies $\Phi$, $c_j$ is satisfied by some variable $x_i$. Either $x_i$ appears in $c_j$ as a positive literal, in which case we schedule $\beta_{c_j}$ on its positive slot of $D_{x_i}$, or it appears as a negative literal, in which case we schedule it on the negative slot of $D_{x_i}$. In the former case $x_i$ is set to `TRUE` in $A$ which implies the positive decision of $D_{x_i}$, in the latter case $x_i$ is set to `FALSE` implying the negative decision of $D_{x_i}$. In both cases, $\beta_{c_j}$ can be scheduled on the specified slot of $D_{x_i}$ without overlapping with the scheduled jobs comprising $D_{x_i}$. Since no two scheduled clause jobs overlap, we can schedule them all in this way and obtain a complete schedule for $\mathcal{S}$. ☐

## 3.2 Interval Selection with Restricted Number of Machines

In this section we consider the complexity of non-restricted IntervalSelection. We show that, in contrast to IntervalSelection with cores, the problem is NP-hard even if the number of machines is constant. In particular, we prove that deciding whether there is a complete schedule is NP-complete already for three machines. In contrast, the problem is polynomially solvable for two machines [7]. We show that the problem of maximizing the number of scheduled intervals, on the other hand, is NP-hard already for two machines (while polynomially solvable for one machine). Moreover, all these hardness results hold even when all intervals have the same length.

We believe that the techniques used in the proofs may be of independent interest. The decision gadgets capture the relation between a schedule and an assignment. However, we also use properties of edge coloring that provide us with a mapping that lets us put the pieces together and finalize the construction of a scheduling problem under the required, rather restrictive conditions.

**Unit Interval Selection with Three Machines.**

We consider IntervalSelection with three machines and unit length intervals, with the objective of deciding whether there is a complete schedule. We will present a reduction from ($\leq$3,3)-Sat.

**Lemma 8.** *Let $\Phi$ be an instance of ($\leq$3,3)-Sat, given by a set of clauses $C$ over a set of Boolean variables $X$. Then, there exists a mapping $p$ from $E = \{(x,c) \in X \times C \mid x \in c\}$ to the set $\{M_1, M_2, M_3\}$, such that $p(x,c) \neq p(x,c')$ for $c \neq c'$ and $p(x,c) \neq p(x',c)$ for $x \neq x'$. Moreover, such a mapping $p$ can be found in polynomial time.*

*Proof.* We prove the statement by edge-coloring the bipartite graph $G = (X \cup C, E)$. The structure of ($\leq$3,3)-Sat implies that all vertices of the constructed graph $G$ have a degree at most 3. A bipartite graph is $\Delta$-edge-colorable in polynomial time, where $\Delta$ is the maximum degree [9]. Therefore, the graph $G$ is 3-edge-colorable, with colors from $\{M_1, M_2, M_3\}$. This coloring gives us the desired mapping from $E$ to $\{M_1, M_2, M_3\}$. ☐

**Theorem 9.** *The problem of deciding whether there exists a complete schedule in* IntervalSelection *is NP-complete even for three machines and unit length intervals.*

*Proof.* The problem is obviously in NP. To show the hardness, we reduce ($\leq$3,3)-Sat to it.

*Construction.* Let $\Phi$ be an instance of ($\leq$3,3)-Sat, given by a set of clauses $C = \{c_1, c_2, \ldots, c_r\}$ over a set of Boolean variables $X = \{x_1, x_2, \ldots, x_s\}$. We construct from $\Phi$ the following instance $\mathcal{S}$ of the IntervalSelection problem (cf. Figure 5 along with the construction), using three machines $M_1, M_2, M_3$. We use a window of $2s+1$ units, and denote the unit windows constituting

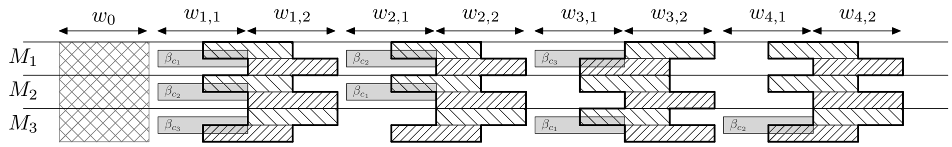


Figure 5: Instance $\Phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_3})$ of ($\leq$3,3)-Sat and the corresponding instance of IntervalSelection with three machines. Intervals intersecting the blocking gadget are not shown in the figure.

it by $w_0, w_{1,1}, w_{1,2}, w_{2,1}, w_{2,2}, \ldots, w_{s,1}, w_{s,2}$. We introduce jobs with unit length intervals as follows. We place a blocking gadget on window $w_0$ over all machines. For each variable $x_i$ we place a decision gadget $D_{x_i}$ on the machines such that it has two positive and one negative slot on window $w_{i,1}$, in an arrangement that we will specify later. The gadget $D_{x_i}$ occupies windows $w_{i,1}$ and $w_{i,2}$ and uses internally the blocking gadget on $w_0$. The positive/negative decision of gadget $D_{x_i}$ corresponds to the truth assignment of the variable $x_i$ and the decision of $D_{x_i}$ is independent of the other decision gadgets. We introduce a *clause job* $\beta_{c_j}$ for each clause $c_j$. To place the intervals for $\beta_{c_j}$, we look at the literals that appear in $c_j$. For each appearance of a positive literal of some variable $x_i$ in $c_j$ we place an interval for $\beta_{c_j}$ on a positive slot of $D_{x_i}$, and for each appearance of a negative literal of $x_{i'}$ in $c_j$ we place an interval for $\beta_{c_j}$ on the negative slot of $D_{x_{i'}}$. If $c_j$ contains only two literals, we place one interval for $\beta_{c_j}$ on the window $w_0$ so that it intersects the blocking gadget and cannot be selected in any complete schedule.

To obtain a valid construction, we need to ensure that all the intervals for each clause job $\beta_{c_j}$ are placed on different machines, and at the same time, we require that each positive/negative slot of the decision gadgets is occupied by at most one interval. We now explain the exact placement of the positive/negative slots, as well as the distribution of the clause jobs over the slots that achieve this. We have three machines and we need to place each decision gadget so that it has its negative slot on some machine and its positive slots on the other two machines. Finding a way to arrange the decision gadgets and distribute their slots is equivalent to finding a mapping from a set of pairs (variable $x$, clause $c$ containing $x$) to the set $\{M_1, M_2, M_3\}$ that assigns different machines to the variables in each clause and different machines to the clauses containing a fixed variable. Such a mapping can be efficiently constructed due to Lemma 8.

*Correctness.* The correctness of the construction can be showed by a similar argument as in the proof of Theorem 7. In short, first we suppose that there is a complete schedule $S$ for $\mathcal{S}$ and we derive the truth assignment for each variable from the decisions of the corresponding decision gadget. Our construction ensures that this assignment satisfies every clause, since the corresponding clause job needs to be scheduled in $S$. Conversely, we explain how to construct a complete schedule $S$ from a satisfying truth assignment $A$: First we schedule the jobs of the blocking gadget in any order. Then we schedule the jobs constituting each decision gadget according to the truth assignment of the corresponding variable in $A$. Finally, for each clause, we pick one literal that satisfies it in $A$ and schedule its clause job on the corresponding slot of a decision gadget. By construction, no two intervals of clause jobs placed on slots of decision gadgets overlap (Lemma 8), hence all clause jobs can be scheduled in this way. □

**Unit Interval Selection with Two Machines.**

We consider IntervalSelection with two machines and unit intervals and show that the problem of maximizing the number of scheduled intervals is NP-hard. The proof is similar to that of Theorem 9, but uses a reduction from (2,3)-MaxSat. We state the following lemma, similar to Lemma 8.

**Lemma 10.** *Let $\Phi$ be an instance of (2,3)-*MaxSat*, given by a set of clauses $C$ over a set of Boolean variables $X$. Then, there exists a mapping $p$ from $E = \{(x, c) \in X \times C \mid x$ appears in $c$ as a positive literal$\}$ to the set of machines $\{M_1, M_2\}$, such that $p(x, c) \neq p(x, c')$ for*

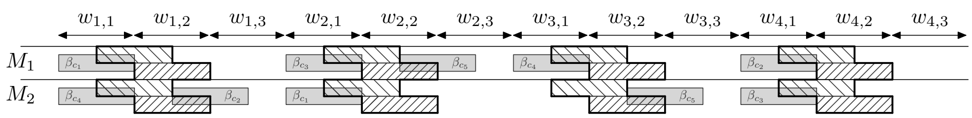

Figure 6: Example instance $\Phi = (x_1 \vee x_2) \wedge (\overline{x_1} \vee x_4) \wedge (x_2 \vee x_4) \wedge (x_1 \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3})$ of (≤3,3)-Sat and the corresponding instance of IntervalSelection with two machines.

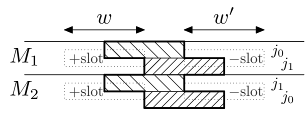

Figure 7: The decision gadget used in the construction.

*$c \neq c'$ and $p(x,c) \neq p(x',c)$ for $x \neq x'$. Moreover, such a mapping $p$ can be found in polynomial time.*

*Proof.* As in the proof of Lemma 8, it is enough to realize that bipartite graph $G = (X \cup C, E)$ is 2-edge-colorable. □

**Theorem 11.** *Maximizing the number of scheduled intervals in* IntervalSelection *is NP-hard, even for two machines and unit length intervals.*

*Proof.* To show the hardness of this IntervalSelection problem, we provide a reduction from (2,3)-MaxSat. The construction is similar to the construction for Theorem 9, but we use refined decision gadgets.

*Construction.* Let $\Phi$ be an instance of (2,3)-MaxSat, given by a set of clauses $C = \{c_1, c_2, \ldots, c_r\}$ over a set of Boolean variables $X = \{x_1, x_2, \ldots, x_s\}$. We construct from $\Phi$ the following instance $\mathcal{S}$ of the IntervalSelection with two machines $M_1$ and $M_2$ (consider Figure 6 along with the construction). On the machines we consider a window of $3s$ units and we denote the unit windows constituting it by $w_{1,1}, w_{1,2}, w_{1,3}$, $w_{2,1}, w_{2,2}, w_{2,3}, \ldots,$ $w_{s,1}, w_{s,2}, w_{s,3}$.

Before we introduce the jobs, we refine the decision gadgets that we use in the construction. Each decision gadget is for the two machines $M_1, M_2$, has two positive slots on some window *and*, additionally, two negative slots on another window, in such a way that if the jobs of the gadget are scheduled, either both positive slots are blocked and the negative slots are free, or vice versa. More precisely, we use a decision gadget $D$ for machines $M_1, M_2$ with two positive slots on some window $w$ and introduce two *negative slots* on both machines on the unit length window $w'$ that begins half a unit after the end of $w$ (see Figure 7). Recall that a decision gadget is made up of two jobs $j_0$ and $j_1$ with an interval for $j_0$ placed on $M_1$ so that its left end is at the center of $w$ and an interval for $j_1$ on $M_1$ placed so that its left end touches the right end of $w$. The intervals for $j_0$ and $j_1$ on $M_2$ are in the opposite arrangement. Therefore, if both $j_0$ and $j_1$ are scheduled, either $w$ is blocked on both machines and $w'$ is free (negative decision) or $w$ is free on both machines and $w'$ is blocked (positive decision).

We proceed with the actual construction. For each variable $x_i$ we place a refined decision gadget $D_{x_i}$ on the machines $M_1, M_2$ with positive slots on window $w_{i,1}$ and negative slots on the unit window consisting of the second half of $w_{i,2}$ and the first half of $w_{i,3}$. Again, a positive/negative decision of the gadget $D_{x_i}$ is in correspondence with a truth assignment of the variable $x_i$, and the decisions of different decision gadgets do not interfere.

For each clause $c_j$ we introduce a *clause job* $\beta_{c_j}$. In order to place intervals for clause jobs, we first look at all positive literals that appear in $\Phi$. For each $x_i$ that appears in $c_j$ as a positive literal, we place an interval for $\beta_{c_j}$ on a positive slot of $D_{x_i}$. For now, we assume that we can place clause jobs on positive slots in a way that no two intervals for a clause job are placed on the same machine and that at most one interval is placed on on each positive slot. If $x_i$ appears in $c_j$ as a negative literal, we place an interval for $\beta_{c_j}$ on a negative slot of $D_{x_i}$ in such a way that the two intervals of a job $\beta_{c_j}$ are placed on different machines. This can be achieved since only exactly one negative slot is used in each gadget, and we are thus free to choose which machine to place the interval on.

We now show that the intervals for clause jobs can be placed on positive slots so that no clause job has two intervals on the same machine and no positive slot is occupied twice. This is equivalent to requiring that there is a mapping from $X \times C$ to the set $\{M_1, M_2\}$ that assigns different machines to two positive literals appearing in the same clause and assigns different machines to the same positive literals in different clauses. Lemma 10 ensures the existence of such a mapping. Thus, we can obtain a valid placement for interval of clause jobs such that no two of them overlap.

The above construction for an instance $\mathcal{S}$ of IntervalSelection has two machines and $2s+r$ jobs. All intervals have unit length and we can construct the presented reduction from (2,3)-MaxSat in linear time. We conclude the proof by showing that there is a schedule for $\mathcal{S}$ that schedules at least $2s+k$ jobs if an only if there is a truth assignment for $\Phi$ that satisfies at least $k$ clauses.

*Correctness.* We can prove the correctness by a similar argument as in the proof of Theorem 9. First we suppose there is a maximum schedule $S$ for $\mathcal{S}$ that schedules at least $2s+k$ jobs and we derive a truth assignment for $\Phi$ that satisfies at least $k$ clauses. As the first step, we show that there exists a schedule $S'$ that schedules also at least $2s+k$ jobs but in which all jobs of the decision gadgets are scheduled. Each interval of a job of a decision gadget intersects with exactly one positive/negative slot and therefore it intersects with at most one scheduled interval of a clause job in $S$. Therefore, we can modify $S$ by scheduling each not yet scheduled job of a decision gadget instead of a scheduled clause job. We obtain $S'$ that schedules all $2s$ jobs of decision gadgets and at least $k$ clause jobs. We construct a truth assignment $A$ of the variables in $\Phi$ according to the decisions of decision gadgets in $S'$. At least $k$ clauses are satisfied by $A$ in $\Phi$, due to the fact that $k$ clause jobs are scheduled in $S'$, with a similar argument as in the proof of Theorem 9.

Conversely, we suppose there is a truth assignment $A$ that satisfies at least $k$ clauses of $\Phi$ and construct a schedule $S$ that schedules at least $2s+k$ jobs. We schedule the $2s$ decision gadgets' jobs according to the truth assignment of the variables in $A$. For each clause $c_j$ that is satisfied by $A$ we schedule the clause job $\beta_{c_j}$ on a slot of decision gadget that corresponds to a literal that satisfies the $c_j$ in $A$. The construction (together with Lemma 10) ensures that no two intervals of clause jobs overlap and thus at least $k$ clause jobs can be scheduled in this way. Therefore, we can construct a schedule $S$ that contains at least $2s+k$ jobs. □

**Acknowledgements.**

This work was partially supported by the EU FP7/2007-2013 (DG CONNECT.H5-Smart Cities and Sustainability), under grant agreement no. 288094 (project eCOMPASS). Kateřina Böhmová is a recipient of the Google Europe Fellowship in Optimization Algorithms, and this research is supported in part by this Google Fellowship. We would like to thank Thomas Graffagnino from Swiss Federal Railways (SBB) and Rastislav Šrámek for pointing out optimization problems with applications in car sharing. The first author would like to thank Petr Škovroň for feedback on preliminary versions of the paper.

## References

[1] E. M. Arkin and E. B. Silverberg. Scheduling jobs with fixed start and end times. *Discrete Applied Mathematics*, 18(1):1–8, 1987.

[2] A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Approximating the throughput of multiple machines in real-time scheduling. *SIAM J. Comput.*, 31(2):331–352, 2001.

[3] K. I. Bouzina and H. Emmons. Interval scheduling on identical machines. *Journal of Global Optimization*, 9:379–393, 1996.

[4] J. Chuzhoy, R. Ostrovsky, and Y. Rabani. Approximation algorithms for the job interval selection problem and related scheduling problems. In *Proc. of the 42nd IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 348–356, 2001.

[5] T. Erlebach and F. C. R. Spieksma. Interval selection: applications, algorithms, and lower bounds. *Journal of Algorithms*, 46(1):27–53, 2003.

[6] M. R. Garey and D. S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[7] J. M. Keil. On the complexity of scheduling tasks with discrete starting times. *Operations Research Letters*, 12(5):293–295, 1992.

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

[9] D. König. Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre. *Mathematische Annalen*, 77:453–465, 1916.

[10] K. Nakajima and S. L. Hakimi. Complexity results for scheduling tasks with discrete starting times. *Journal of Algorithms*, 3(4):344–361, 1982.

[11] V. Raman, B. Ravikumar, and S. S. Rao. A simplified NP-complete MAXSAT problem. *Information Processing Letters*, 65(1):1–6, 1998.

[12] J. Sgall. Open problems in throughput scheduling. In L. Epstein and P. Ferragina, editors, *ESA 2012*, volume 7501 of *LNCS*, pages 2–11. Springer, 2012.

[13] D. Shabtay and G. Steiner. Scheduling to maximize the number of just-in-time jobs: a survey. In *Just-in-Time Systems*, volume 60 of *Springer Optimization and Its Applications*, pages 3–20. Springer New York, 2012.

[14] F. C. R. Spieksma. On the approximability of an interval scheduling problem. *Journal of Scheduling*, 2(5):215–227, 1999.

[15] S. C. Sung and M. Vlach. Maximizing weighted number of just-in-time jobs on unrelated parallel machines. *Journal of Scheduling*, 8:453–460, 2005.