Interval Selection with Machine-Dependent Intervals

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

[1] Institute of Theoretical Computer Science, ETH Zürich, Zürich, Switzerland
{katerina.boehmova,matus.mihalak,widmayer}@inf.ethz.ch
[2] Department of Mathematics, TU Berlin, 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). 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: Intervals, Scheduling, Complexity, 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., [2,5,6]. 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

F. Dehne, R. Solis-Oba, and J.-R. Sack (Eds.): WADS 2013, LNCS 8037, pp. 170–181, 2013.

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 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 [16]). 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 [14,16]. 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 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 JISP_m. 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) [15], and a randomized $\approx \frac{e-1}{e}$-approximation algorithm [5] that gives a 3/4-approximation for 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 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 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" [14,16], and it is an interesting special case of JISP_k [5,6,15]. 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.

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 †
≥ 3 machines	**NP-complete** †	NP-hard †	NP-hard †
JISP_k (single machine)			
2 intervals per job	$O(n^2)$	NP-hard†	NP-hard†
≥ 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

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 [4]. 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 [16]. 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 [16]; this open problem has also been stated in a recent survey on just-in-time job scheduling [14].

As outlined beforehand, our problem is a special class of JISP_k (job interval scheduling problem on a single machine with k intervals per job). Nakajima and Hakimi [11] showed that the decision version of JISP_3 is NP-complete. Keil [8] showed that this is the case even if the intervals have the same length, while the general decision version of JISP_2 can be solved in polynomial time. The maximization version has been studied as outlined earlier by Spieksma [15] and Chuzhoy [5]. Erlebach and Spieksma [6] consider the weighted 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.

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. [9] 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 [13] 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 [15], 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.

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.

In Figure 1, the interval for job β_1 on M_A reduces the selection on M_B by one, but the interval for job α_1 on M_A reduces the selection on M_B by one

Fig. 1. Instance where A^3 returns exactly $2/3 \cdot |O|$ jobs: O contains all jobs α_i and β_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 $\alpha_1, \alpha_2, \beta_1, \beta_2$.

only with the help of β_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. 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 1. *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. Some of the proofs, as well as the parts discussing the correctness of the reductions, are omitted due to space constraints. They can be found in full version in technical report [3].

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., [7] 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) [7], and even, if each clause

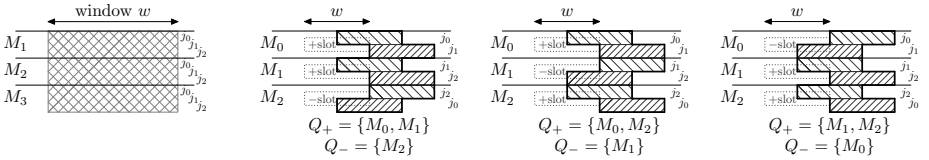

Fig. 2. The first drawing illustrates how we depict a blocking gadget for a window w. The last three drawings illustrate decision gadgets on three machines M_0, M_1, and M_2. Each of the decision gadgets has two positive slots on machines in Q_+ and one negative slot on the machine in Q_-. The crucial 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.

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) [7]. 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) [12].

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.

Lemma 2. *In any complete schedule for* IntervalSelection *that contains the blocking gadget B for window w, no selected interval outside B intersects w.*

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 $M_0, M_1, \ldots, M_{k-1}$ the machines in Q. Without loss of generality, we assume that w has unit length. We use k jobs $j_0, j_1, \ldots, j_{k-1}$, 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 3. *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.*

Corollary 1. *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 [14,16]. 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 [16].

Theorem 2. *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.

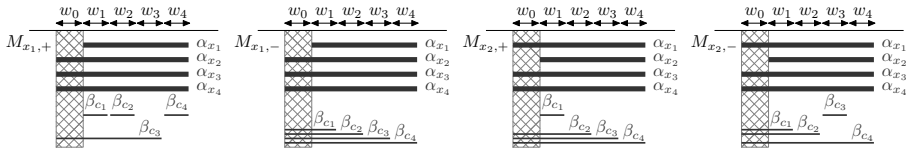

Fig. 3. Example of the construction of IntervalSelection with cores at the end for an instance Φ of the 3-Sat problem (each figure shows the intervals on a single machine, the figures of the machines $M_{x_3,+}$, $M_{x_3,-}$, $M_{x_4,+}$, $M_{x_4,-}$ are not displayed), where $\Phi = (x_1 \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_2} \vee \overline{x_3} \vee x_4) \wedge (x_1 \vee \overline{x_3} \vee x_4)$

Let us consider an arbitrary instance Φ 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 α_{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 α_{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 α_{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 α_{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 Φ are satisfied. For each clause c_j we have one *clause job* denoted by β_{c_j}. We place the intervals for the job β_{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 β_{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 β_{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 β_{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. □

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. Moreover, using the gadgets described before, we can construct a reduction from ($\leq$3,3)-SAT to prove the following theorem concerning INTERVALSELECTION with arbitrary cores and unit length intervals.

Theorem 3. *The problem of deciding whether there exists a complete schedule in* INTERVALSELECTION *with cores is NP-complete even if all intervals have unit length.*

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 [8]. 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. We will use the following lemma in the subsequent hardness result.

Lemma 4. *Let Φ 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 Δ-edge-colorable in polynomial time, where Δ is the maximum degree [10]. 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 4. *The problem of deciding whether there exists a complete schedule in* INTERVALSELECTION *is NP-complete even for three machines and unit length intervals.*

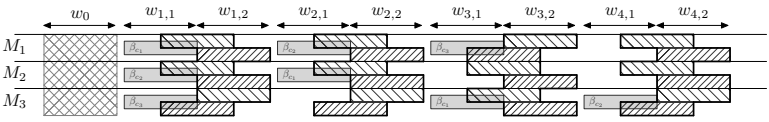

Fig. 4. Instance $\Phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_3})$ of ($\leq$3,3)-SAT and the corresponding instance of INTERVALSELECTION with three machines. Intervals intersecting the blocking gadget are not shown in the figure.

Proof. The problem is obviously in NP. To show the hardness, we reduce ($\leq$3,3)-SAT to it. Let Φ be an instance of ($\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 Φ the following instance $\mathcal{S}$ of the INTERVALSELECTION problem (cf. Figure 4 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 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* β_{c_j} for each clause c_j. To place the intervals for β_{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 β_{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 β_{c_j} on the negative slot of $D_{x_{i'}}$. If c_j contains only two literals, we place one interval for β_{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 β_{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 4. □

Unit Interval Selection with Two Machines. The maximization variant of INTERVALSELECTION turns to be NP-hard already for two machines. The proof is similar to that of Theorem 4, but uses a reduction from (2,3)-MAXSAT.

Theorem 5. *Maximizing the number of scheduled intervals in* IntervalSelection *is NP-hard, even for two machines and unit length intervals.*

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. Arkin, E.M., Silverberg, E.B.: Scheduling jobs with fixed start and end times. Discrete Applied Mathematics 18(1), 1–8 (1987)
2. Bar-Noy, A., Guha, S., Naor, J., Schieber, B.: Approximating the throughput of multiple machines in real-time scheduling. SIAM J. Comput. 31(2), 331–352 (2001)
3. Böhmová, K., Disser, Y., Mihalák, M., Widmayer, P.: Interval selection with machine-dependent intervals. Tech. Rep. 786, Institute of Theoretical Computer Science, ETH Zurich (2013)
4. Bouzina, K.I., Emmons, H.: Interval scheduling on identical machines. Journal of Global Optimization 9, 379–393 (1996)
5. Chuzhoy, J., Ostrovsky, R., Rabani, Y.: 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), pp. 348–356 (2001)
6. Erlebach, T., Spieksma, F.C.R.: Interval selection: applications, algorithms, and lower bounds. Journal of Algorithms 46(1), 27–53 (2003)
7. Garey, M.R., Johnson, D.S.: Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman & Co., New York (1979)
8. Keil, J.M.: On the complexity of scheduling tasks with discrete starting times. Operations Research Letters 12(5), 293–295 (1992)
9. Kolen, A.W.J., Lenstra, J.K., Papadimitriou, C.H., Spieksma, F.C.R.: Interval scheduling: a survey. Naval Research Logistics (NRL) 54(5), 530–543 (2007)
10. König, D.: Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre. Mathematische Annalen 77, 453–465 (1916)
11. Nakajima, K., Hakimi, S.L.: Complexity results for scheduling tasks with discrete starting times. Journal of Algorithms 3(4), 344–361 (1982)
12. Raman, V., Ravikumar, B., Rao, S.S.: A simplified NP-complete MAXSAT problem. Information Processing Letters 65(1), 1–6 (1998)
13. Sgall, J.: Open problems in throughput scheduling. In: Epstein, L., Ferragina, P. (eds.) ESA 2012. LNCS, vol. 7501, pp. 2–11. Springer, Heidelberg (2012)
14. Shabtay, D., Steiner, G.: Scheduling to maximize the number of just-in-time jobs: a survey. In: Just-in-Time Systems. Springer Optimization and Its Applications, vol. 60, pp. 3–20. Springer, New York (2012)
15. Spieksma, F.C.R.: On the approximability of an interval scheduling problem. Journal of Scheduling 2(5), 215–227 (1999)
16. Sung, S.C., Vlach, M.: Maximizing weighted number of just-in-time jobs on unrelated parallel machines. Journal of Scheduling 8, 453–460 (2005)