# Interval Scheduling: A Survey

Antoon W.J. Kolen[*], Jan Karel Lenstra[†], Christos H. Papadimitriou[‡]
Frits C.R. Spieksma [§]

April 21, 2006

**Abstract**

In interval scheduling, not only the processing times of the jobs but also their starting times are given. This paper surveys the area of interval scheduling and presents proofs of results that have been known within the community for some time. We first review the complexity and approximability of different variants of interval scheduling problems. Next, we motivate the relevance of interval scheduling problems by providing an overview of applications that have appeared in literature. Finally, we focus on algorithmic results for two important variants of interval scheduling problems. In one variant we deal with nonidentical machines: instead of each machine being continuously available, there is a given interval for each machine in which it is available. In another variant, the machines are continuously available but they are ordered, and each job has a given 'maximal' machine on which it can be processed. We investigate the complexity of these problems and describe algorithms for their solution.

*Analysis of algorithms, computational complexity: exact algorithms. Production/scheduling, sequencing, deterministic: interval scheduling*

[*]Antoon Kolen sadly passed away on October 3, 2004. His coauthors dedicate their contribution to this paper to his memory.

[†]CWI, P.O. Box 94079, NL-1090 GB Amsterdam, The Netherlands. E-mail: `Jan.Karel.Lenstra@cwi.nl`

[‡]University of California, Berkeley, CA 94720, USA. E-mail: `christos@cs.berkeley.edu`

[§]Katholieke Universiteit Leuven, Naamsestraat 69, B-3000, Leuven, Belgium. E-mail: `frits.spieksma@econ.kuleuven.be`

## 1 Introduction

Scheduling problems are formulated in terms of machines and jobs. The machines represent resources and the jobs represent tasks that have to be carried out using these resources. In a traditional scheduling problem there is freedom in determining the starting times of the jobs. The scheduler uses this freedom by attempting to construct a schedule that satisfies certain constraints or even one that optimizes a certain criterion. Performance measures for this type of scheduling problem reflect how well the resources are used. For instance, minimizing the makespan measures the time it takes the machines to process all given jobs. Thus, the quality of a schedule measures the performance of a set of machines processing all given jobs. Research on these classical scheduling problems is abundantly present in the operations research literature.

This survey deals with *interval scheduling* problems, also known as *fixed job scheduling* or *k-track assignment* problems. In this type of scheduling problem there is no freedom in determining the starting time of the job; instead this is input for the problem. Decisions that the scheduler now faces involve whether or not to accept the job and what resource to assign to it. Performance measures here can focus on the individual jobs; for instance, one may wish to maximize the total weight of the accepted jobs. Thus, performance measures in interval scheduling allow to take into account the cost of rejecting (or the profit of accepting) an individual job. Although there is a sizable amount of literature on interval scheduling problems, research on these problems has often been tailored to the application at hand, and results are scattered through literature. For instance, the well known crew scheduling problem is an interval scheduling problem (see Section 3).

Although the relevance of interval scheduling problems stems to a large extent from the variety of applications that have an interval scheduling decision as a core, recent trends in operations management also motivate their study. In the following we elaborate on this subject. Recent literature in operations management (see e.g. Graves et al. [42]) describes very clearly how a transition took place in the last decennia from *resource oriented* logistics (where the availability of resources dictated the planning and completion of jobs) to *demand oriented* logistics (where the jobs and their completion are more or less fixed and the appropriate resources must be found). This transition has mainly been caused by increased competition and the continuous drive to improve service to clients. In an environment where clients are becoming more important they will have more influence on the delivery date, and hence on the production date of their product. Thus, dates for production tend to become more exogeneously determined, as opposed to a more traditional setting where the plant itself could decide when to manufacture a job (see e.g. Hall et al. [45]). This phenomenon is enhanced by current logistic developments in which production is organized by supply chains. To summarize: when phrasing the transition sketched above in scheduling terminology, it is akin to going from traditional scheduling to interval scheduling. We therefore believe that interval scheduling can provide a useful way to model some of the operational planning problems in current logistics.

The history of interval scheduling can be traced back to the 1950's when Dantzig and Fulkerson [23] described a tanker scheduling problem. Ford and Fulkerson [35] solve a basic interval scheduling problem (see Section 2.1), using Dilworth's theorem.

Resource allocation problems (see Ibaraki and Katoh [49]) are problems where a given, fixed amount of a single resource is allocated to a number of competing activities while optimizing some objective function. In an interval scheduling problem the set of machines (the resource) is, at any given moment, used to process (some of) the jobs (the competing activities). Hence, in this way, interval scheduling can be seen as a *dynamic* resource allocation problem, since the resource allocation changes over time.

The purpose of this survey is twofold. First, we intend to give an overview of work on interval scheduling; second, we present algorithmic results on two fundamental problem variants. Except for Section 2.2.4, we restrict ourselves to off-line settings, i.e., we assume that we know all data at the outset.

The paper is organized as follows. In the next section we give a problem description, and describe related literature. Section 3 discusses applications of interval scheduling problems. These applications are not meant to form an exhaustive list, but rather intend to give a flavor of the variety of practical applications of our model. Section 4 presents our results on an interval scheduling problem with nonidentical machines. Section 5 gives our results on the so-called hierarchical interval scheduling problem. Finally, Section 6 states some conclusions.

# 2 Problem description

We introduce the basic interval scheduling problem, our notation and terminology, and results from literature in Section 2.1. We then consider different variants of the basic interval scheduling problem, and describe known results focussing on the complexity and approximability of these variants in Section 2.2.

## 2.1 The basic interval scheduling problem

The basic interval scheduling problem is stated as follows. Given are $n$ intervals of the form $[s_j, f_j)$ with $s_j < f_j$, for $j = 1, \ldots, n$. These intervals are the *jobs* that require uninterrupted processing during that interval. We will assume (without loss of generality) that the $s_j$'s and the $f_j$'s are nonnegative integers. We say that two intervals (or jobs) *overlap* if their intersection is non-empty, otherwise they are called *disjoint*. Further, there are machines. In the basic interval scheduling problem each machine can process at most one job at a time and is always available, i.e., each machine is continuously available in $[0, \infty)$. We assume that, when processed, each job is assigned to a single machine, thus, we do not allow interrupting a job and resuming it on another machine, unless explicitly stated otherwise. The basic interval scheduling problem is now to process all jobs using a minimum number of machines. In other words, find an assignment of jobs to machines such that no two jobs assigned to the same machine overlap while using a minimum number of machines. We call an assignment of (a subset of) the jobs to the machines a *schedule*.

It is well known that interval scheduling problems and interval graphs are related. Indeed, the graph that arises when there is a node for each interval and when there is an edge between two nodes if and only if the corresponding intervals overlap, is a so-called interval graph. Hence, the basic interval scheduling problem is in fact nothing else but finding a coloring of an interval graph (see e.g. Golumbic [41]); the chromatic number of an interval graph corresponds to the minimum number of machines used in the interval scheduling context.

Another observation concerning the basic interval scheduling problem is as follows. Consider a subset of jobs that pairwise overlap. Obviously, the size of this set is a lower bound for the number of machines needed to process all jobs. Given an instance of the basic interval scheduling problem, let us call the maximum size of such a set the *job overlap* of the instance. Obviously, the job overlap is a lower bound for the number of machines needed. However, using Dilworth's chain decomposition theorem, one can show that this number of machines actually suffices to process all jobs. An algorithm to construct an optimal schedule (that is, one with a minimum number of machines) is described by Ford and Fulkerson [35], who specified the so-called *staircase rule*, which is based on Dilworth's theorem. The rule involves $O(n^2)$ operations. Alternative methods for the basic interval scheduling problem are proposed by Hashimoto and Stevens [46] and by Gertsbakh and Stern [40]. Gupta et al. [43] propose yet another procedure that runs in $O(n \log n)$ time, which they show to be best possible.

A succinct description of this procedure ([43], see also Kolen and Lenstra [56]) is as follows. Assign the jobs to the machines in order of nondecreasing starting times, using a machine used before whenever possible. We refer to this algorithm as the *left-edge* algorithm, see also Section 4.1.

Dekel and Sahni [24] describe a parallel implementation of this algorithm, which solves the problem in $O(\log n)$ time using $O(n^2/\log n)$ processors.

## 2.2 Variants of the basic interval scheduling problem

In this section we consider different (optimization) variants of the basic interval scheduling problem. In particular, we discuss here the known results for the following settings:

- Cost minimization problems (see Subsection 2.2.1). Given the jobs, given nonidentical (types of) machines, and given a specific cost function that assigns to each schedule a certain cost, find a minimum-cost schedule in which all jobs are scheduled.

- Profit maximization problems (see Subsection 2.2.2). Given the jobs, given (types of) machines, and given a profit when job $j$ is carried out by machine (type) $i$, find a maximum-profit schedule (in which not all jobs need to be assigned).

- Job interval scheduling problems (see Subsection 2.2.3). In this setting, instead of a single start time $s_j$ for each job $j$, a discrete set of possible starting points $S_j = \{s_{j1}, s_{j2}, \ldots, s_{jk}\}$ is given for each job $j$.

- Online interval scheduling problems and interval scheduling problems with preemption (see Subsection 2.2.4).

For an overview of complexity theory we refer to Garey and Johnson [37]. For an overview of approximation algorithms we refer to Vazirani [71]. For an overview of online algorithms in general, and the associated terminology, we refer to Borodin and El-Yaniv [27]. For a survey on online scheduling problems we refer to Sgall [69].

### 2.2.1 Cost minimization problems

In this section we consider the variant where the jobs are given, and we need to minimize the total costs induced by carrying out all jobs on the nonidentical machines. (Notice that if the machines were identical, an instance of the basic interval scheduling problem arises). There are different ways in which the machines can be nonidentical. For instance, machines may differ with respect to their *availability.* Indeed, consider the variant where to each machine $i$, $1 \leq i \leq m$, an availability interval $[a_i, b_i)$ is associated, where there is a given cost for using machine $i$, and where the goal is to minimize total costs while scheduling all jobs. We denote this problem by Interval Scheduling with Machine Availabilities, and address this variant extensively in Section 4. The following statement applies to this variant: unless P=NP, there is no approximation algorithm that achieves a finite worst-case ratio. This statement follows essentially from Theorem 1 which says that deciding whether a feasible schedule exists is NP-complete.

Bhatia et al. [9] consider a related setting where a machine *type* $i$ corresponds to an an availability interval $[a_i, b_i)$, $1 \leq i \leq m$. Thus, for each availability interval there is an unbounded number of machines available instead of a single one, and there is a given cost $k_i$ for using a machine of type $i$. Assuming that each job can go to each machine, Bhatia et al. [9] give a 3-approximation algorithm (and a 2-approximation algorithm in case one wants to minimize the number of machines used, i.e., in case $k_i = 1$ for all $i$).

Another way in which the machines can be nonidentical is by assuming that there is a *linear order* given for the machines, and, in addition, for each job $j$ a 'maximal' machine $m(j)$, $1 \leq m(j) \leq m$ is given on which it can be processed. Thus machines $1, \ldots, m(j)$ are capable of processing job $j$ while machines $m(j) + 1, \ldots, m$ cannot process job $j$, $j = 1, \ldots, m$. Given a cost $k_i$ for using machine $i$, the goal is to minimize costs while scheduling all jobs (recall that each machine is continuously available). We call the resulting problem the hierarchical interval scheduling problem, and address this variant in Section 5. Again, unless P=NP, there is no approximation algorithm that achieves a finite worst-case ratio. This statement follows essentially from Theorem 4 which says that deciding whether a feasible schedule exists is NP-complete. However, in case there are *types* of machines, and there is a linear order for these machine types, Bhatia et al. [9] give a 2-approximation algorithm. When there are two machine-types, Dondeti and Emmons [25] and Huang and Lloyd [48] show that (a generalization of) the resulting problem is solvable in polynomial time. In Section 5 we prove that the problem becomes NP-complete for three machine types. In case there is an arbitrary distance given between each pair of jobs, and the cost of a machine depends on the amount of distance travelled by the machine in order to process its jobs, Faneyte et al. [31] show that the problem with two machine types is already NP-complete.

Machines can also be nonidentical by associating to each job an arbitrary subset of machines capable of processing that job. Hence, the machines differ with respect to the set of jobs they can process in a more general way than in the hierarchical interval scheduling problem. For this general setting, Jansen [50] gives an $O(\log n)$ approximation algorithm. It is observed in Bhatia et al. [9] that this is essentially best possible, since this problem is as hard as set cover. The problem is further discussed in Kroon et al. [58]), and - with additional side-constraints - is also considered by Fischetti et al. [32, 33, 34]. Another variant arises when for each job that machine $i$ processes, a cost of $k_i$ is incurred, $1 \leq i \leq m$. The resulting cost minimization problem is investigated by Kroon et al. [59] and by Jansen [51]. Huang and Lloyd [48] consider a similar setting where the cost of of processing job $j$ on machine $i$ equals $k_i(f_j - s_j)$.

### 2.2.2 Profit Maximization problems

An important optimization variant is the case where one treats the machines as given and the objective is to maximize the number of (weighted) jobs that can be feasibly scheduled. This can be solved using a min-cost flow formulation (see Arkin and Silverberg [3] and Bouzina and Emmons [12]). In case each job has unit weight, a greedy algorithm finds a maximum number of jobs (see Faigle and Nawijn [30], and Carlisle and Lloyd [16]). If each job can only be carried out by an arbitrary given subset of the machines, the problem becomes NP-hard ([3]). Heuristics and exact algorithms are proposed by Kroon et al. [57]. In case an availability interval is associated to each machine, Brucker and Nordmann [14] describe an $O(n^{m-1})$ algorithm that maximizes the number of jobs scheduled. Bhatia et al. [9] describe a randomized approximation algorithm achieving a ratio of $1 - \frac{1}{e}$ when there is a given profit $w_j$ for each job $j$ scheduled.

### 2.2.3 Job interval scheduling problems

Job interval scheduling problems constitute an interesting generalization of interval scheduling problems. Instead of a single starting time $s_j$ for each job $j$, a discrete set of possible starting points $S_j = \{s_{j1}, s_{j2}, \ldots, s_{jk}\}$ is given. Of course, at most one starting time from each set $S_j$ can be chosen.

Job interval scheduling problems are related to so-called *time-constrained* scheduling problems. In a time-constrained scheduling problem, a release date $r_j$, a deadline $d_j$, and a processing time $p_j$ is given for each job $j$. Obviously, if $d_j = r_j + p_j$ for each job $j$, an interval scheduling problem arises, but also, when assuming that there is an interval for each possible realization of job $j$, we can model a time-constrained scheduling problem as a job interval scheduling problem. Of course, one faces here the difficulty of having to deal with, possibly, a continuum of intervals reflecting all possible starting times of a job (notice that the word 'job' refers here to a set of intervals), see [8, 19] for ways of dealing with this difficulty.

Results for job interval scheduling problems are given by Nakajima and Hakimi [65], Nakajima et al. [66] (for a setting with *fast* and *slow* machines), Keil [52], and Spieksma [70]. More in particular, in [70], a single machine is considered, and the goal is to maximize the number of jobs selected. It is shown that this problem is APX-hard (even if $|S_j| = 2$ for each job $j$), that the integrality gap of a straightforward integer programming formulation equals 2, and that a greedy algorithm yields a 2-approximation. Chuzhoy et al. [21] improve this ratio by exhibiting a randomized $\frac{e}{e-1} + \epsilon$-approximation algorithm for any $\epsilon > 0$. Berman and Dasgupta [8] describe a combinatorial approximation algorithm to deal with the weighted case of the job interval scheduling problem, and show how this can be used to derive approximation factors for time-constrained scheduling problems involving identical machines and unrelated machines, improving upon the LP-based approach described in Bar-Noy et al. [6]. Similar bounds in a very general framework are explained in Bar-Noy et al. [4].

Notice that these results apply to maximizing the (weighted) number of jobs. There has also been work dealing with minimizing the number of machines for job interval scheduling problems; we refer to Cieliebak et al. [22], Chuzhoy et al. [19], and Chuzhoy and Naor [20].

### 2.2.4 Online interval scheduling problems

Lipton and Tomkins [61] introduce the online interval scheduling problem, where intervals with a given length $f_j - s_j$ are presented to the scheduler in the order of their start time $s_j$. The scheduler must decide whether to accept each interval before a new interval is presented. The objective is to maximize total length of the accepted intervals while ensuring that no pair of accepted intervals overlap. They show that no (randomized) algorithm can achieve a competitive ratio better than $O(\log \Delta)$ (where $\Delta$ denotes the ratio between the longest and the shortest interval), and gave an $O((\log \Delta)^{1+\epsilon})$-competitive algorithm. They also present a 2-competitive algorithm for the case of two lengths. These latter results have been generalized by Goldman et al. [44], to the case of so-called *delays* where a delay $\delta_j$ of an interval $j$ means that if an interval is accepted it must start between $s_j$ and $s_j + \delta_j$.

A related setting where there is a given weight for each interval (not necessarily equal to its length) is considered by Woeginger [72]. Here it is allowed for the scheduler to interrupt a previously accepted interval in order to accept a new interval; the weight of the interrupted interval is then lost. The problem is then to maximize the total weight of the accepted (and not interrupted) intervals. Woeginger [72] shows that no deterministic algorithm can achieve a finite competitive ratio. In fact, even a randomized algorithm does not achieve a finite competitve ratio (Canetti and Irani [15]). Further, Woeginger [72] shows that in case the weight of an interval is a function of its length, there are classes of functions for which matching upper and lower bounds on the competitive ratio for deterministic algorithms can be shown. Seiden [68] exhibits a randomized algorithm for a specific class of functions that improves the competitive ratio achieved by a deterministic algorithm. Further, Miyazawa and Erlebach [64] investigate the case of equal lengths for the intervals and non-decreasing weights of arriving intervals. They show that randomized algorithms can give a better competitive ratio than the best possible deterministic algorithm.

In case each interval has unit weight, and there are $m$ machines, Faigle and Nawijn [30], and independently Carlisle and Lloyd [16], observed that a greedy algorithm is in fact an online algorithm that always outputs an optimal solution. This algorithm is extended to deal with the case of time-windows in Faigle et al. [29].

Erlebach and Spieksma [28] study the online version of the job interval scheduling problem (see Section 2.2.3) for multiple machines, and they consider the intervals in the order of right endpoints. For this setting they give best possible algorithms for various classes of weight functions.

Finally, preemptive versions of interval scheduling problems are investigated by Bouzina [11], Dondeti and Emmons [26], Kroon et al. [58], and Fischetti et al. [33]. Notice that with preemption we refer here to interrupting a job, and resuming it immediately on another machine.

## 3 Applications of interval scheduling problems

Interval scheduling problems have a broad diversity of applications, as is witnessed by the following selection.

**Crew/vehicle scheduling.** The basic crew scheduling problem can be formulated as follows (see e.g. Beasley and Cao [7] or Mingozzi et al. [63]). Given is a set of crews, a set of locations, and a set of tasks. For each task a starting time, an ending time and a location is given; for each pair of locations a distance is given. The problem of assigning tasks to crews using a minimum number of crews can be phrased in interval scheduling terminology by viewing the tasks as the intervals, the crews as machines, and allowing for a distance between any pair of intervals. Crew scheduling problems are among the most celebrated problems in operations research. They also appear in bus driver applications (see Martello and Toth [62]). Fischetti et al. [32, 33, 34] treat two cases. In a first case each machine is only available for a given amount of time $w$, i.e., the latest ending time of an interval assigned to machine $i$ minus the earliest starting time of an interval assigned to that machine $i$ should not exceed $w$. A second case assumes that the sum of the lengths of the intervals assigned to a same machine should not exceed a given number $w$. Heuristics, lower bounds, and exact approaches are described. A specific problem involving hierarchies is described by Faneyte et al. [31].

**Telecommunication.** Consider a setting where users communicate with each other using a network. A user communicates by requesting capacity of a link (called bandwidth) in the network during a given interval. Of course, requests of different users may not be compatible due to the amount of capacity requested or the intervals requested. How to utilize the network optimally by allocating the available bandwidth to the users is the main question in this application. Notice that this application is especially relevant in an online context. Using interval scheduling terminology, we can view requests as intervals, and links as machines. We refer to Bar-Noy et al. citebarcankutmansch, Kumar [60], and Bhatia et al. [9] and the references contained therein for more details.

**Other applications.** Gabrel [36] considers the following problem in *satellite photography.* A satellite makes orbits around the earth. Its task is to photograph pieces of the earth's surface. Fulfilling a request for photographing a specific piece of the earth implies that the camera needs to start and end filming at given moments in time. When formulated in terms of interval scheduling, a request becomes a job and the camera is the machine.

Anthonisse and Lenstra [2] describe a *cottage rental* problem. Given a set of identical cottages, can a client's request for a reservation for a given time-period immediately be answered? And what if some periods have already been preassigned? Obviously, in an interval scheduling context, a cottage is a machine and a request is an interval. The results described in this paper answer the two questions above.

The interest of Gupta et al. [43] and Hashimoto and Stevens [46] in the basic interval scheduling problem was sparked by a problem in VLSI-layout, called the *channel assignment* problem. Given are pairs of components. Each component must be placed on a printed circuit board at a given $y$-coordinate. Moreover, components of a pair must be placed on the same $x$-coordinate. Then, the two components are interconnected using a so-called channel. Channels are not allowed to overlap. Observe now that minimizing the number of channels boils down to solving the basic interval scheduling problem.

In a series of papers, Kolen and Kroon [53, 54, 55] discuss a *maintenance* problem in the aviation industry. More specifically, engineers need to carry out maintenance jobs on aircraft. There are different types of aircrafts, and an engineer is only allowed to perform a job on a specific type of aircraft if (s)he has a license for that type. The jobs have fixed starting and ending times. With the licensed engineers in the role of machines, the problem is an interval scheduling problem where each machine can process a given set of jobs.

Carter and Tovey [17] describe a problem in *class room assignment.* The problem is to assign $n$ classes that come together during given periods to rooms $R_j$, $j = 1, \ldots, m$, under various constraints and objectives. One of the variants they consider is the setting where the rooms are ordered such that if a class will accept room $R_j$, then it will also accept any room $R_k$ with $k > j$. With classes in the role of intervals and rooms in the role of machines we have a hierarchical interval scheduling problem.

Brehob et al. [13] investigate *replacement policies* for non-standard caches (a cache is a piece of memory in a computer; items are stored in cache-locations). They draw a parallel with interval scheduling by letting each access to a certain item correspond to a start time of an interval, and the next access to that item serves as the finishing time of that interval. A machine corresponds to a cache-location.

Another problem comes from computational biology (see Chen et al. [18]). Given a sequence of amino-acids (the single machine) and so-called segments (these correspond to the jobs), and a profit for each possible allocation of segment to a position in the sequence of amino-acids, find an allocation such that no two segments overlap, and total weight is maximized.

## 4 Interval scheduling with machine availabilities

In the basic interval scheduling problem each machine is continuously available. When we associate to each machine an interval $[a_i, b_i)$ during which it is available, the interval scheduling problem with machine availabilities arises (observe that a machine with non-contiguous availability can be viewed as multiple machines each having a single contiguous availability interval). Although the left-edge algorithm can be easily modified to deal with this setting, it is easy to see that there are instances for which it fails to produce a feasible solution while one exists. A formal description of the problem is as follows.

Problem: Interval Scheduling with Machine Availabilities (ISMA).

> INSTANCE: $m$ machines, continuously available in $[a_i, b_i)$, $i = 1, \ldots, m$; $n$ jobs, requiring processing from $s_j$ to $f_j$, $j = 1, \ldots, n$.
>
> QUESTION: Does there exist a feasible schedule? That is, can each job be processed by a machine such that no two jobs processed by a same machine overlap while respecting the availability of each machine?

Consider an instance of ISMA. With each instance of ISMA we can associate a parameter $l$, referred to as *machine-overlap*, that is the maximum number of machines that contain some point $p$ (a machine $i$ is said to contain point $p$ if $a_i \leq p < b_i$). Formally, we define

$$l = \max_p |\{i : a_i \leq p < b_i, 1 \leq i \leq m\}|. \tag{1}$$

Papadimitriou [67] proved that ISMA is NP-complete, by a reduction from the circular arc coloring problem; see also [2, 32, 10, 26, 14]. In Section 4.1 we show that ISMA and CAC are, in fact, polynomially equivalent. We further sketch two algorithms for ISMA, a dynamic programming algorithm (DP) and a breadth-first algorithm (BF). The latter algorithm runs in time that is linear in the number of jobs. Finally, we end Section 4 by discussing various optimization problems corresponding to ISMA.

## 4.1 Complexity

As argued in Section 2.1, the basic interval scheduling problem is equivalent to coloring an interval graph. This equivalence can be extended to show that ISMA is polynomially equivalent to circular arc coloring:

**Circular arc coloring**

Consider the following problem. Given are $n^{CAC}$ *circular arcs*, each defined by a pair of distinct, positive integers $\{s_i^{CAC}, f_i^{CAC}\}$ $(i = 1, \ldots, n)$, and an integer $K$. (Observe that it may happen that $s_i^{CAC} > f_i^{CAC}$ for some $i$, see Figure 1 for an example.)

As before, two arcs are said to *overlap* if their intersection is nonempty, otherwise they are called *disjoint*.

The problem is to decide whether there exists a coloring of the arcs using at most $K$ colors such that arcs that overlap receive different colors. This problem is known as the circular arc coloring problem (CAC).

INSTANCE: $n^{CAC}$ circular arcs $\{s_i^{CAC}, f_i^{CAC}\}$ and an integer $K$.

QUESTION: Does there exist a partition of the circular arcs into at most $K$ sets such that arcs in a same set are disjoint?

Although an efficient algorithm for deciding the existence of a $K$-coloring of an interval graph is known (see Golumbic [41]), such an algorithm is not likely to exist for circular arc graphs. Indeed, Garey et al. [38] prove that CAC is NP-complete. Notice that we can restrict ourselves to instances where

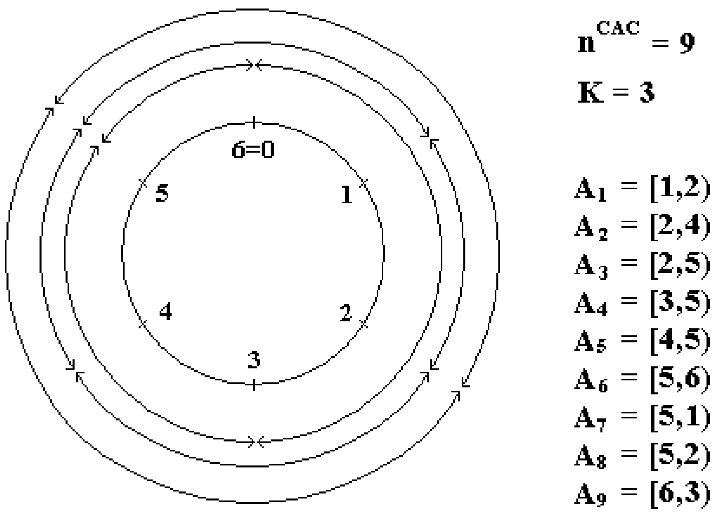

Figure 1: An instance of CAC where $A_i$ represents arc $i$, $i = 1, \ldots, n^{CAC}$.

(i) $K \leq n^{CAC}$, (ii) $\max_i(s_i^{CAC}, f_i^{CAC}) \leq 2n^{CAC}$, and (iii) where each point $p$ on the circle is contained in exactly $K$ arcs. To explain (ii), notice that it is not the size of an integer $s_i^{CAC}$ or $f_i^{CAC}$ that matters, but rather their ordering. We can therefore reduce any instance of CAC to an instance where the integers defining the CAC instance are bounded by $2n^{CAC}$. Concerning the latter restriction, if a point $p$ is contained in more than $K$ arcs, the instance is trivially seen to be a no-instance; if a point $p$ is contained in less than $K$ arcs, we can add arcs until point $p$ is contained in exactly $K$ arcs. More specifically, consider all different $s_i^{CAC}$ and $f_i^{CAC}$ values, and rename them $u_1, u_2, \ldots, u_k$ such that $u_1 < u_2 < \ldots < u_k$ ($k \leq 2n^{CAC}$). Now, if point $p$ is contained in less than $K$ arcs, there exists precisely one arc from $\cup_{i=1}^{k-1}\{u_i, u_{i+1}\} \cup \{u_k, u_1\}$ that contains $p$. Next, we add copies of this arc to the instance until $p$ is contained in exactly $K$ arcs. This does not affect the $K$-colorability and the size of the extended instance is polynomial in the size of the original instance (see Garey et al. [38]). In our reductions we assume that the CAC-instances have these properties.

There is an intimate connection between CAC and ISMA. Intuitively, this can be seen as follows. Let $L$ be the largest integer among the endpoints $\{s_i^{CAC}, f_i^{CAC}\}$ of the CAC instance. Then we partition the circle using $L$ equally spaced points numbered clockwise $1, 2, \ldots, L$ (recall that we can assume that $L \leq 2n^{CAC}$). By disconnecting (or "cutting") the circle at point $L$, associating a machine to each *cut* arc (i.e., an arc containing $L$), and associating a job to each *uncut* arc (i.e., an arc not containing $L$), an essential part of the reduction of CAC to ISMA is described (see Figures 1 and 2). This idea is used in Papadimitriou's proof ([67]); it also appears in Fischetti et al. [32], Biró et al. [10], Dondeti and Emmons [26], and Brucker and Nordmann [14]. The reverse operation, that is, viewing the ISMA instance on the time axis, and next connecting the leftmost point and the rightmost point to each other, is used to turn an ISMA instance in a CAC instance.

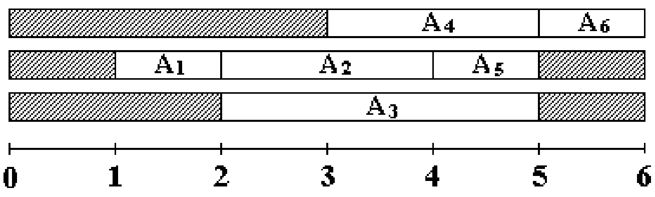

Figure 2: A solution to the ISMA-instance corresponding to the CAC instance from Figure 1.

**Theorem 1** *ISMA is polynomially equivalent to CAC.*

**Proof:** We exhibit a reduction from ISMA to CAC. Together with the reduction of (Papadimitriou [67]) the result follows.

Given an instance of ISMA, we build an instance of CAC as follows. We set $K = m$, and $n^{CAC} = m + n$. There is an arc for each job with characteristics $s_j^{CAC} = s_j, f_j^{CAC} = f_j$ for $j = 1, \dots, n$ (the job-arcs), and there is an arc for each machine with characteristics $s_{n+i}^{CAC} = b_i, f_{n+i}^{CAC} = a_i$ for $i = 1, \dots, m$ (the machine-arcs). This completes the description of the CAC instance. If a feasible schedule exists, we turn it into a feasible coloring by giving the job-arcs corresponding to jobs on a same machine as well as the corresponding machine-arc a same color. Finally, observe that in a feasible coloring all $K$ colors are used to color the machine-arcs. Therefore, each job-arc is colored using one of these $K$ colors and a feasible schedule follows. $\square$

An implication of Theorem 1 is that an algorithm for CAC can be used to solve ISMA and vice versa. We close this section with the following two remarks.

**Remark:** One-sided ISMA, i.e., ISMA with $a_i = 0$ for all $i$, (or $b_i = b$ for all $i$), is solvable in polynomial time. One easily verifies that by adding appropriate dummy jobs the basic interval scheduling problem discussed in Section 2.1 arises.

**Remark:** Allowing preemption drastically changes the solvability of ISMA. Observe that for any ISMA input, the existence of a preemptive schedule is equivalent to whether at each moment $t$, the number of machines containing $t$ is not smaller than the number of jobs containing $t$. Indeed, a simple generalization of the left-edge algorithm (see Section 2.1) decides whether a preemptive schedule exists. Informally, this can be described as follows.

We order the $s_j$ and $f_j$ ($j = 1, \ldots, n$), and the $a_i$ and $b_i$ ($i = 1, \ldots, m$) in nondecreasing order. Ties are handled in such a way that an $f_j$ precedes an $s_j$, and an $a_i$ as well as a $b_i$ are inserted after an $f_j$ and before an $s_j$. Also, there is a stack where machines can be placed; at the outset of the algorithm, the stack is empty. We refer to the first machine in this stack as the *top-machine*. The algorithm now scans the list of ordered numbers. If it meets an $a_i$, it puts the corresponding machine on top of the stack; if it meets a $b_i$, it removes the corresponding machine from the stack; if the algorithm meets an $s_j$, it assigns job $j$ to the top-machine, and removes this machine from the stack; if it meets an $f_j$, then the machine to which job $j$ was assigned becomes the top-machine. Finally, if a job has not finished processing when its machine is removed from the stack, the remainder of this job is placed on the top-machine. If, during the course of this algorithm, a job needs to be assigned while there is no top-machine (i.e., the stack of machines is empty), no preemptive schedule exists, otherwise the algorithm produces a preemptive schedule.

## 4.2 Algorithms

Let us now proceed by describing two exact algorithms for ISMA. Admittedly, the running times of each of these algorithms is - in the worst case - exponential in the size of the input, which should come as no surprise given the hardness of ISMA. However, these algorithms may be efficient in practice, in particular for instances with bounded machine overlap. In Section 4.2.1

we describe a dynamic programming algorithm, called DP; in Section 4.2.2 we describe a breadth-first algorithm, called BF.

### 4.2.1 Algorithm DP

Consider an instance of ISMA, and its overlap $l$ (see (1)). Observe that we can partition the set of $m$ machines into $l$ subsets $S_h$ ($h = 1, \ldots, l$), such that each machine belongs to exactly one subset, and any pair of machines in a same subset is disjoint. In fact, by viewing the machine intervals as an instance of the basic interval scheduling problem, it follows from the discussion in Section 2.1 that we can use the algorithms described there to compute the sets $S_h$. We define:

$$L_h = \bigcup_{i \in S_h} [a_i, b_i) \qquad \text{for } h = 1, \ldots, l.$$

We refer to $L_h$ as *layer* $h$, and we say that job $j$ *can be processed on layer* $h$ if the job-interval $[s_j, f_j)$ is contained in a machine-interval $[a_i, b_i)$ that belongs to a machine in $S_h$. To proceed, let us index the jobs so that $0 \equiv f_0 < f_1 \leq f_2 \leq \ldots \leq f_n$.

The idea behind the dynamic programming algorithm is to decide whether a given set of largest ending times of jobs on the layers can correspond to a partial feasible schedule of jobs $1, 2, \ldots, j$. Indeed, observe that in order to decide whether we can add job $j$ to a partial feasible schedule consisting of jobs $1, 2, \ldots, j-1$, it is sufficient to know the largest ending time of a job on each layer. The dynamic programming recursion is based on the fact that if a partial feasible schedule exists for jobs $1, 2, \ldots, j-1$ (which is characterized by the largest ending times of the jobs on layers $1, 2, \ldots, l$), the existence of a feasible schedule for jobs $1, 2, \ldots, j$ depends on verifying whether we can feasibly assign job $j$ to some layer $h$ given the partial feasible schedule for jobs $1, 2, \ldots, j-1$. Verifying this amounts to checking whether $s_j$ exceeds the largest ending time on some layer $h$, and whether job $j$ can be processed on layer $h$.

To state the algorithm formally, we define:

- $F = \{f_0, f_1, \ldots, f_n\}$, and $F^l$ is the cartesian product of $l$ sets $F$, i.e., $F^l = F \times F \times \ldots \times F$. We use a vector $u = (u_1, u_2, \ldots, u_l)$ to denote an element of $F^l$.

- $f(s_j) = \max_{0 \le r \le n}\{f_r : \ f_r \le s_j\}$ for $j = 1, \dots, n$. Thus, $f(s_j)$ is the largest ending time not exceeding $s_j$, $j = 1, \dots, n$.
- the boolean parameters $B_{hj} =$ TRUE iff job $j$ can be processed on layer $h$ for $j = 1, \dots, n$ and $h = 1, \dots, l$.
- the boolean variables $g_j(u) =$ TRUE iff jobs $1, 2, \dots, j$ can be processed in $L_1 \cap [0, u_1), L_2 \cap [0, u_2), \dots, L_l \cap [0, u_l)$ for $j = 1, \dots, n$ and $u \in F^l$.
- $g_0(u) =$ TRUE for all $u \in F^l$.

**Algorithm** DP:

**Step 1 (initialization):** $g_0(u) =$ TRUE for all $u \in F^l$;
$g_j(u) =$ FALSE for all $u \in F^l$, $j = 1, \dots, n$.

**Step 2 (recursion):** compute $g_j(u)$ using

$$g_j(u) = \bigvee_{h: u_h \ge f_j} (g_{j-1}(u_1, \dots, u_{h-1}, f(s_j), u_{h+1}, \dots, u_l) \wedge B_{hj})$$

for $j = 1, \dots, n$ and for all $u \in F^l$.

**Step 3:** Evaluate $g_n(f_n, \dots, f_n)$.

Figure 3: **Algorithm** DP.

**Theorem 2** DP *solves ISMA in* $O(ln^{l+1})$ *time.*

**Proof:** First, we argue that DP is correct, then we prove its complexity.

Correctness: We show by induction on $j$ that the values of $g_j(u)$ computed in Step 2 satisfy their definition. The case $j = 0$ is trivial. Next, observe that, given some $j$ and $u \in F^l$, the left-hand side of the equality in Step 2 of DP is only true when, using the induction hypothesis that the $g_{j-1}(u)$ values satisfy their definition, jobs $1, 2, \dots, j-1$ can be processed such that there exists a layer $h$ that is occupied up to $f(s_j)$ and this layer $h$ allows job $j$ to be processed on (i.e., $B_{hj}$ is TRUE). Since $s_j \ge f(s_j)$ (by definition) and since $B_{hj}$ is apparently TRUE, we can assign job $j$ to layer $h$. It follows that DP is correct.

Complexity: The time bound follows from the observation that Step 2 is the dominating step in DP. To compute a single $g_j(u)$ variable at most $l$ values of $g_{j-1}(\cdot)$ variables are needed. It follows that Step 2 can be carried out in $ln^{l+1}$ operations, and the time bound follows. $\square$

Algorithm DP does in fact more than announced in Theorem 2: not only does it output a correct answer to the instance of ISMA (yes or no), in case of a no-answer it also computes the largest $j$ for which there exists a feasible schedule of the jobs $1, 2, \ldots, j$.

We can achieve a slight improvement over the complexity of DP stated in Theorem 2 by not explicitly considering all vectors $u \in F^l$. We refrain from sketching the details.

### 4.2.2 Algorithm BF

Our breadth-first enumerative algorithm requires that at each moment $t$, the number of active machines equals the number of jobs containing $t$. To accomplish this we add in a first step a number of dummy jobs, as follows. First, we order the values of all starting and ending times $s_j$ and $f_j$ of the $n$ jobs and rename them $u_1, u_2, \ldots, u_k$ such that $u_1 < u_2 < \ldots < u_k$ ($k \leq 2n$). If, in some interval $[u_i, u_{i+1})$ the number of jobs is strictly less than the number of available machines, we add dummy jobs $j$ with $s_j = u_i$ and $f_j = u_{i+1}$ as needed. This procedure is akin to the procedure described in Section 4.1 for CAC that achieves that each point $p$ is contained in exactly $K$ arcs. Let us redefine $n$ as the number of jobs in the resulting instance.

The algorithm makes use of the idea of *partial* schedules. A partial schedule up to $u_i$ is a schedule that has assigned all jobs that start before $u_i$ to the machines. We say that the partial schedule is *feasible* if no two jobs overlap and each job can be carried out by the machine it is assigned to. The algorithm makes a single pass through the time intervals $[u_i, u_{i+1})$. The idea is to enumerate all partial feasible schedules up to $u_i$. Thus, given a partial schedule that has assigned all jobs starting before $u_i$, the algorithm considers the jobs that start at $u_i$ and explores all possible assignments of these jobs to the machines. By detecting duplication of partial feasible schedules, it can be shown that the number of partial feasible schedules that need to be maintained in each iteration remains bounded. More precisely, consider two partial schedules up to $u_i$, say A and B. Observe that if each job $j$ for which $s_j < u_i < f_j$ is assigned to the same machine in both partial schedules A and B, then these two partial schedules can be considered to be equivalent. Indeed, each way that we can proceed from partial schedule A is also a way to proceed from partial schedule B, and vice versa. It follows that, in order to decide feasibility, we only need to maintain partial schedules that are not equivalent.

This algorithm has been inspired by the algorithm for CAC due to Garey et al. [38], and can be seen as an adaptation of that algorithm to the ISMA-setting. We refrain from giving the exact details, and simply state the result.

**Theorem 3** *BF solves ISMA in $O((n+m)m!m\log m)$ time.*

**Proof:** This result follows from Garey et al. [38] and Theorem 1. □

**Remark:** Consider the case of a fixed number of machines in ISMA. Then algorithm **BF** is linear in the number of jobs.

## 5 Hierarchical interval scheduling

A formal description of this problem is as follows.

Problem: Hierarchical Interval Scheduling with $T$ machine types (HIS($T$)).

> INSTANCE: $m_t$ machines of type $t$, $t = 1, \ldots, T$, continuously available in $[0, \infty)$; $n_t$ jobs of type $t$, $t = 1, \ldots, T$. A job of type $t$ can only be processed by a machine of type $r, r \leq t$ ($t = 1, \ldots, T$). Each job $j$ requires processing from $s_j$ to $f_j$, $j = 1, \ldots, n \equiv \sum_t n_t$.
>
> QUESTION: Does there exist a feasible schedule? That is, can each job be processed by an appropriate machine such that no two jobs processed by a same machine overlap?

When the number $T$ of machine types is part of the input, a reduction from CAC, very similar to the one proposed by Papadimitriou [67] for ISMA, shows that HIS($T$) is NP-complete, even when there is exactly one machine of each type. Dondeti and Emmons [26] establish NP-completeness of HIS($T$) without the latter restriction.

As stated in Section 2.2.1, Dondeti and Emmons [25] and Huang and Lloyd [48] show that a generalization of HIS(2) is solvable in polynomial time. HIS(3), however, turns out to be hard again, as we show by a reduction from Numerical 3-Dimensional Matching (N3DM).

First we consider the case where $T$ is part of the input. We use a similar construction as in the proof of Theorem 1 to turn an instance of CAC into an instance of HIS($T$) (see Figure 4).

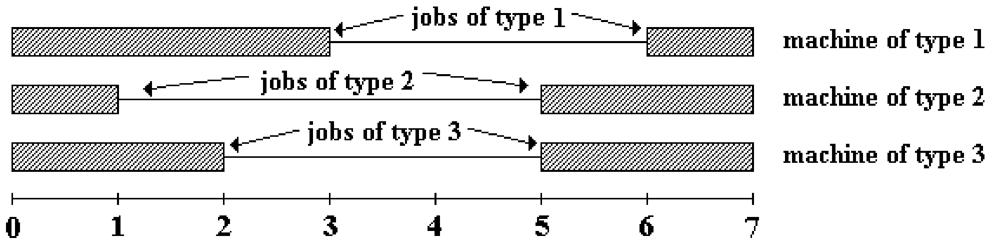

Figure 4: An instance of HIS($T$) corresponding to the CAC instance from Figure 1.

**Theorem 4** HIS($T$) *is NP-complete, even if* $m_t = 1$ *for each* $t$*.*

**Proof:** Clearly, HIS($T$) is in NP. We use CAC to prove the result. Consider an instance of CAC. Select point $L$ on the circle and index the arcs containing $L$ by $1, 2, \ldots, K$. Arbitrarily index the remaining arcs $K + 1, \ldots, n^{CAC}$. Set $T = K$, and "cut" the circle open at $L$. Each arc $j$ containing $L$ corresponds to two jobs of type $j$, one with $s_{2j-1} = 0$, $f_{2j-1} = f_j^{CAC}$, and the other with $s_{2j} = s_j^{CAC}$, $f_{2j} = L+1$, $j = 1, \ldots, T$. Each arc not containing $L$ corresponds to a job of type $T$ with $s_{2K+j} = s_{K+j}^{CAC}$, $f_{2K+j} = f_{K+j}^{CAC}$ for $j = K+1, \ldots, n^{CAC}$. There is one machine of each type. This completes the description of an instance of HIS($T$).

Now, suppose the answer to the CAC instance is yes, that is, a feasible coloring exists. We place the two jobs corresponding to an arc containing $L$ on the same machine, together with those jobs of type $T$ that correspond to arcs that received the same color. Thus, a feasible schedule exists. Conversely, if there exists a feasible schedule, we know that the two jobs of type 1 must be present on the machine of type 1 (since they cannot be processed by any other machine). In fact, from this we deduce the more general observation that the two jobs of type $t$ must be present on the machine of type $t$, $t = 1, \ldots, T$. Thus, a feasible coloring exists. $\square$

Let us now consider the case where $T$ is fixed. Obviously, the case $T = 1$ is simply the basic interval scheduling problem discussed in Section 2.1.

Consider now the case $T = 2$. Notice that the input of an instance of HIS(2) consists of $2n+2$ numbers. Thus, in order for an algorithm to be polynomial in the size of the input, the number of machines ($m \equiv m_1 + m_2$) should not appear linearly in the running time. However, it is easy to first verify whether $m_1 \geq n$ or whether $m_2 \geq n$. If yes, it is trivial to determine whether a feasible schedule exists: in case $m_1 \geq n$ a feasible schedule exists, and in case $m_2 \geq n$, the instance reduces to the basic interval scheduling problem involving only jobs of type 1. Thus, we can assume that $m_1$ and $m_2$ are bounded by $n$. In the sequel we will assume that this test is performed, i.e., we assume that $\max(m_1, m_2) \leq n$.

In their work on interval scheduling problems, Dondeti and Emmons [25] consider a more general problem than HIS(2). They consider a variant which, in addition to jobs of type 1 and type 2, also contains jobs of type 3. Jobs of type 3 can only be processed by machines of type 2. An important concept to decide feasibility of this problem is the so-called *job schedule network*; we give, in this survey, a description of this network (see [25]).

We order the $2n$ numbers $s_j, f_j$, $j = 1, \ldots, n$. We get a sequence of values that we denote by $u_1 < u_2 < \ldots < u_k$, where $k \leq 2n$. We define $l_i^1$ ($l_i^2$) as the overlap of jobs of type 1 (2) in the time-interval $[u_i, u_{i+1})$, $i = 1, \ldots, k-1$. Further, we define $l^1 = \max_i l_i^1$; similarly we define $l^2 = \max_i l_i^2$. The network has a vertex $i$ for each $u_i$ value, $i = 1, \ldots, k$. For each job $j$ of type 1, let $p$ and $q$ be such that $u_p = s_j$ and $u_q = f_j$. We draw an arc from vertex $p$ to vertex $q$. This arc has a lower bound on the flow of value 1 and a capacity of value 1. We will refer to such an arc as a job arc of type 1. Similarly, for each job $j$ of type 2, let $p$ and $q$ be such that $u_p = s_j$ and $u_q = f_j$. We draw an arc from vertex $p$ to vertex $q$. This arc has a lower bound on the flow of value 0 and a capacity of value 1. We will refer to such an arc as a job arc of type 2. Finally, we draw an arc from vertex $i$ to vertex $i+1$; this arc has a lower bound of 0, and a capacity of value $m_1 - l_i^1 - \max(0, l_i^2 - m_2)$ for $i = 1, \ldots, k-1$. We will refer to such an arc as a dummy arc. This completes the construction of the network. Notice that this network has $O(n)$ nodes and $O(n)$ arcs. As mentioned above, in [25] a problem more general than HIS(2) is solved; they use a job schedule network involving $O(n)$ nodes and $O(n^2)$ arcs.

Below we will argue that the existence of a flow of value $m_1$ between vertex 1 and vertex $k$ determines whether a feasible schedule exists. If a feasible schedule exists, the flow in the network will determine which jobs of type 2 go to machines of type 1 (the arcs with flow value 1), and which will go to machines of type 2 (the arcs with flow value 0). Thus, by solving a maximum flow problem on this network we determine the answer. For the time-bounds that can be achieved when solving a maximum flow problem we refer to Ahuja et al. [1].

Of course, necessary conditions for the existence of a feasible schedule are $l^1 \leq m_1$ and $\max_i(l_i^1 + l_i^2) \leq m_1 + m_2$. In the sequel we assume these conditions to hold.

**Theorem 5** HIS(2) *can be solved as a max-flow problem on a network with* $O(n)$ *nodes and* $O(n)$ *arcs.*

**Proof:** We argue that the existence of a flow of value $m_1$ between vertices 1 and $k$ implies the existence of a feasible schedule and vice versa. Obviously, we can use a max-flow algorithm to determine whether a flow of that value exists.

Suppose that a flow of value $m_1$ exists between vertices 1 and $k$. Consider this flow in the network. The amount of flow that crosses a cut of the form $(\{1,\ldots,i\},\{i+1,\ldots,k\})$ equals $m_1$ for each $i = 1,\ldots,k-1$. In each of these cuts, this flow consists of three parts: flow that goes through job arcs of type 1 (this must equal $l_i^1$), flow that goes through the dummy arc (say $D_i$), and finally flow that goes through job arcs of type 2 (say $F2_i$). We have, for each $i$,

$$l_i^1 + D_i + F2_i = m_1. \tag{2}$$

It follows that the number of job arcs with flow 1 is less than or equal to $m_1$ in each time interval $[u_i, u_{i+1})$ $(1 \leq i \leq k-1)$. Thus, we can view the set of jobs whose corresponding arcs have a flow of value 1 as an instance of the basic interval scheduling problem. Since the overlap in this instance is bounded by $m_1$, the results in Section 2.1 imply that the jobs whose job arcs have a flow of 1 can be feasibly scheduled on the type 1 machines.

Let us now argue that the remaining jobs (whose arcs have a flow of 0) can be feasibly scheduled on the machines of type 2. Rewriting (2) gives:

$$F2_i = m_1 - l_i^1 - D_i. \tag{3}$$

The capacity restriction on the dummy arc implies that

$$D_i \leq m_1 - l_i^1 - \max(0, l_i^2 - m_2). \tag{4}$$

(3) and (4) imply that for each $i = 1, \ldots, k-1$

$$F2_i \geq \max(0, l_i^2 - m_2),$$

which implies that out of the $l_i^2$ jobs of type 2 that are active in $[u_i, u_{i+1})$ at least $\max(0, l_i^2 - m_2)$ jobs of type 2 are scheduled on the machines of type 1. There remain at most

$$l_i^2 - \max(0, l_i^2 - m_2) = \min(l_i^2, m_2) \leq m_2$$

jobs of type 2 to be processed on machines of type 2 in $[u_i, u_{i+1})$, $i = 1, \ldots, k-1$. Again, since apparently the number of remaining type 2 jobs in each time-interval $[u_i, u_{i+1})$ is bounded by the number of available machines we can use the results in Section 2.1 to find a feasible schedule for these jobs.

To complete the proof, notice that if there exists a feasible schedule to the HIS(2) instance, we can send, for each machine of type 1, a flow of value 1 through the network. Each arc corresponding to a job processed by a machine of type 1 has a flow of value 1. Notice that the capacity of each dummy arc suffices to accommodate the resulting flow. This gives us a flow of value $m_1$. $\square$

Finally, we show that the hierarchical interval scheduling problem with three machine types is NP-complete.

**Theorem 6** HIS(3) *is NP-complete.*

**Proof:** Observe that HIS(3) is in the class NP. We use N3DM.

**Numerical 3-dimensional matching**.
The numerical 3-dimensional matching (N3DM) can be stated as follows.

> INSTANCE: $3t$ positive integers $c_i, d_i, e_i$, $i = 1, \ldots, t$, and a positive integer $B$ such that $c_i, d_i, e_i \leq B$ $(i = 1, \ldots, t)$ and $\sum_i (c_i + d_i + e_i) = tB$.
>
> QUESTION: Do there exist two permutations $\pi$ and $\sigma$ of $\{1, \ldots, t\}$ such that $c_i + d_{\pi(i)} + e_{\sigma(i)} = B$, for $i = 1, \ldots, t$ ?

N3DM is well known to be NP-complete (Garey and Johnson [37]).

Let $m_1 = t$, $m_2 = t(t-1)$, and $m_3 = t^2$, i.e., there are $t$ machines of type 1, $t(t-1)$ machines of type 2, and $t^2$ machines of type 3. Each machine is available from time 0 to $t^2 + 2t + 2B$. For some moments in time we define specific parameters: for $i, j = 1, \ldots, t$ let $A_i = i$, $B_i = t + i$ and $X_{ij} = 2t + (i-1)t + j$. Finally, let $S = t^2 + 2t$.

Let us now consider the jobs. We distinguish three types of jobs: we set $n_1 = 2t$, $n_2 = 2t^2 - t$, and $n_3 = 4t^2$. To specify each individual job $j$ we need to specify its starting time $s_j$ and its finishing time $f_j$. We do this by simply stating an interval of the form $[s_j, f_j)$. For the $2t$ jobs of type 1 we have:

- $[0, A_i)$ for $i = 1, \ldots, t$, and
- $[S + B - e_i, S + 2B)$ for $i = 1, \ldots, t$.

For the $2t^2 - t$ jobs of type 2 we have:

- $t - 1$ jobs of the form $[0, B_i)$ for $i = 1, \ldots, t$, and
- $[X_{ij}, S + c_i + d_j)$ for $i, j = 1, \ldots, t$.

For the $4t^2$ jobs of type 3 we have:

- $t - 1$ jobs of the form $[0, A_i)$ for $i = 1, \ldots, t$,
- $[0, B_i)$ for $i = 1, \ldots, t$,
- $[A_i, X_{ij})$ for $i, j = 1, \ldots, t$,
- $[B_i, X_{ij})$ for $i, j = 1, \ldots, t$,
- $[X_{ij}, S + 2B)$ for $i, j = 1, \ldots, t$,

This completes the description of an instance of HIS(3). Let us now argue that the existence of a feasible schedule implies a yes-answer to the N3DM instance and vice versa.

Assume that the answer to the N3DM instance is yes, i.e., there exist two permutations $\pi$ and $\sigma$ such that $c_i + d_{\pi(i)} + e_{\sigma(i)} = B$ for $i = 1, \ldots, t$. Then we construct the following schedule. There are $t^2$ type 2 jobs of the form $[X_{ij}, S + c_i + d_j)$, $i, j, = 1, \ldots, t$. Out of these we select $t$ jobs such that $j = \pi(i)$, $i = 1, \ldots, t$. These $t$ jobs go to the $t$ machines of type 1. Notice that since each of these $t$ jobs contains $S$ they must go to a unique machine. Moreover, each job of this form, i.e., $[X_{i,\pi(i)}, S + c_i + d_{\pi(i)})$, is followed by the job $[S+B-e_{\sigma(i)}, S+2B)$ of type 1 for $i=1,\dots,t$. This is feasible since $S+c_i+d_{\pi(i)} = S+B-e_{\sigma(i)}$ by the solution to N3DM. Next we schedule each type 1 job of the form $[0,A_i)$ on a unique machine of type 1, and select the type 3 jobs of the form $[A_i, X_{i,\pi(i)})$ to complete the assignment of jobs to machines of type 1. Each of the type 2 jobs of the form $[0,B_i)$ goes to a unique machine of type 2, and is followed by the appropriate type 3 jobs of the form $[B_i, X_{ij})$, which are in turn followed by the remaining type 2 jobs of the form $[X_{ij}, S+c_i+d_j)$. All remaining type 3 jobs fit on the machines of type 3.

Conversely, assume that a feasible schedule exists. Consider the total processing time required in the interval $[0,S)$ for each of the job types: jobs of type 1 require time $\frac{1}{2}t(t+1)$, jobs of type 2 require time $\frac{1}{2}t(t-1)(3t+1)+t^2S-\sum_{i,j}X_{ij}$, and jobs of type 3 require time $\frac{1}{2}t(t-1)(t+1)+\frac{1}{2}t(3t+1)+(\sum_{i,j}X_{ij}-t\sum_i A_i)+(\sum_{i,j}X_{ij}-t\sum_i B_i)+(t^2S-\sum_{i,j}X_{ij})$. Summing this up yields $2t^2S$, which equals the total available machine time. Thus, each job that ends before $S$ is followed immediately by another job. In particular, in any feasible schedule each job of the form $[0,A_i)$ is followed by a job of the form $[A_i, X_{ij})$ on the same machine. Also, each job of the form $[0,B_i)$ is followed by a job of the form $[B_i, X_{ij})$ on the same machine.

Recall that a job of type 1 can only be processed by a machine of type 1, a job of type 2 can only be processed by a machine of type 1 or 2, and a job of type 3 can be processed by each machine.

Consider the $t^2$ type 2 jobs of the form $[X_{ij}, S+c_i+d_j)$, $i,j=1,\dots,t$. Since each of them contains point $S$, and since there are $t^2$ machines of types 1 and 2, each of these jobs is assigned to a unique machine of type 1 or 2. Moreover, it follows that each of the $t^2$ type 3 jobs of the form $[X_{ij}, S+c_i+d_j)$ is assigned to a unique machine of type 3. Observe further, that each machine of type 2 must start with a type 2 job of the form $[0,B_i)$, since the machines of type 1 must start with the type 1 jobs of the form $[0,A_i)$. Therefore, each machine of type 2 must look as follows: $[0,B_j)$, $[B_j, X_{ij})$, $[X_{ij}, S+c_i+d_j)$. Notice that each index $j$ occurs exactly $t-1$ times. The $t$ machines of type 1 each have the following jobs: $[0,A_i)$, $[A_i,X_{ij})$, $[X_{ij}, S+c_i+d_j)$, where each index $i$ and each index $j$ occur exactly once.

Consider now the interval $[S, S+2B)$ for machines of type 1. The total required processing time equals $\sum_i c_i + \sum_j d_j$ (by the arguments sketched above) $+\sum_k(S+2B-(S+B-e_k))$. This sum amounts to $2tB$, which is equal to the total amount of processing time that is available. It follows that a job of the form $[X_{ij}, S+c_i+d_j)$ must be directly followed by a job of the form $[S + B - e_k, S + 2B)$. Thus $S + c_i + d_j = S + B - e_k$. Setting $\pi(i) = j$ and $\sigma(i) = k$ when a job of the form $[X_{ij}, S + c_i + d_j)$ is followed by a job of the form $[S + B - e_k, S + 2B)$ gives us the required permutations for the instance of N3DM. □

## 6 Conclusions

Interval scheduling problems appear in diverse fields ranging from crew scheduling to telecommunication. They form an important class of scheduling problems and have been studied under various names and with application-specific constraints. We have surveyed the complexity and approximability of different variants of the basic interval scheduling problem. We have shown that interval scheduling with machine availabilities is related to the problem of coloring a circular arc graph. We provided algorithms for this variant of interval scheduling. Further, we have shown that in the case of ordered machine types, the case of two machine types can be efficiently solved whereas the problem for three machine types is NP-complete.

## Acknowledgment

We thank Esther Arkin for her comments on an earlier version of this paper. We also thank the referees for their remarks which led to an improved presentation of this work.

## References

[1] Ahuja, R.K., T.L. Magnanti, and J.B. Orlin (1993), *Network Flows*, Prentice Hall, New Jersey.

[2] Anthonisse, J.M. and J.K. Lenstra (1984), *Operational operations research at the Mathematical Centre*, European Journal of Operational Research **15**, 293–296.

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

[4] Bar-Noy, A., R. Bar-Yehuda, A. Freund, J.S. Naor, and B. Schieber. (2005), *A unified approach to approximating resource allocation and scheduling*, Journal of the ACM **48**, 1069–1090.

[5] Bar-Noy, A., R. Canetti, S. Kutten, Y. Mansour, and B. Schieber (2005), *Bandwith allocation with preemption*, SIAM Journal on Comnputing **28**, 1806–1828.

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

[7] Beasley, J.E. and B. Cao (1998), *A dynamic programming based algorithm for the crew scheduling problem*, Computers and Operations Research **25**, 567–582.

[8] check Berman, P. and B. DasGupta (2000), *Multi-phase algorithms for throughput maximization for real-time scheduling*, Journal of Combinatorial Optimization **4**, 307–323.

[9] Bhatia, R., J. Chuzhoy, A. Freund, and J. Naor (2003), *Algorithmic aspects of bandwidth trading*, Proceedings of the 30-th International Conference on Automata, Languages, and Programming, Lecture Notes in Computer Science **2719**, 751–766, Springer, Heidelberg.

[10] Biró, M., M. Hujter, and Zs. Tuza (1992), *Precoloring extensions. I. Interval graphs*, Discrete Mathematics **100**, 267–279.

[11] Bouzina, K.I. (1994), *On interval scheduling problems: a contribution*, Ph.D. Thesis of Case Western Reserve University.

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

[13] Brehob, M., S. Wagner, E. Torng, and R. Enbody (2004), *Optimal replacement is NP-hard for nonstandard caches*, IEEE Transactions on Computers **53**, 73–76.

[14] Brucker, P. and L. Nordmann (1994), *The k-track assignment problem*, Computing **54**, 97–122.

[15] Canetti, R. and S. Irani (1998), *Bounding the power of preemption in randomized scheduling*, SIAM Journal on Computing **27**, 993–1015.

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

[17] Carter, M.W. and C.A. Tovey (1992), *When is the classroom assignment problem hard?*, Operations Research **40**, S28–S39.

[18] Chen, Z., G. Lin, R. Rizzi, J. Wen, D. Xu, Y. Xu, and T. Jiang (2005), *More reliable protein NMR peak assignment via improved 2-interval scheduling*, Journal of Computational Biology **12**, 129–146.

[19] Chuzhoy, J., S. Guha, S. Khanna, and J.S. Naor (2004), *Machine minimization for scheduling jobs with interval constraints*, Proceedings of the 45th Annual Symposium on Foundations of Computer Science, 81–90.

[20] Chuzhoy, J., J.S. Naor (2004), *New hardness results for congestion minimization and machine scheduling*, Proceedings of the 36th ACM Symposium on the Theory of Computing, 28–34.

[21] Chuzhoy, J., R. Ostrovsky, and Y. Rabani (2001), *Approximation algorithms for the job interval selection problem and related scheduling problems*, Proceedings of the 42nd Annual Symposium on Foundations of Computer Science, 348–356.

[22] Cieliebak, M., T. Erlebach, F. Hennecke, B. Weber, and P. Widmayer (2004), *Scheduling with Release Times and Deadlines on a Minimum Number of Machines*, Proceedings of the 3rd IFIP International Conference on Theoretical Computer Science, 209–222.

[23] Dantzig, G.B. and D.R. Fulkerson (1954), *Minimizing the number of tankers to meet a fixed schedule*, Naval Research Logistics Quarterly **1**, 217–222.

[24] Dekel, E. and S. Sahni (1983), *Parallel scheduling algorithms*, Operations Research **31**, 24–49.

[25] Dondeti, V.R. and H. Emmons (1992), *Fixed job scheduling with two types of processors*, Operations Research **40**, S76–S85.

[26] Dondeti, V.R. and H. Emmons (1993), *Algorithms for preemptive scheduling of different classes of processors to do jobs with fixed times*, European Journal of Operational Research **70**, 316–326.

[27] Borodin, A. and R. El-Yaniv (1998), *On-Line Computation and Competitive Analysis*, Cambridge University Press, Cambridge.

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

[29] Faigle, U., W. Kern, and W.M. Nawijn (1999), *A greedy on-line algorithm for the k-track assignment problem*, Journal of Algorithms **31**, 196–210.

[30] Faigle, U., and W.M. Nawijn (1995), *Note on scheduling intervals on-line*, Discrete Applied Mathematics **58**, 13–17.

[31] Faneyte, D.B.C., F.C.R. Spieksma, and G.J. Woeginger (2001), *A branch-and-price algorithm for a hierarchical crew scheduling problem*, Naval Research Logistics **49**, 743–759.

[32] Fischetti, M., S. Martello, and P. Toth (1987), *The fixed job schedule problem with spread-time constraints*, Operations Research **35**, 849–858.

[33] Fischetti, M., S. Martello, and P. Toth (1989), *The fixed job schedule problem with working-time constraints*, Operations Research **37**, 395–403.

[34] Fischetti, M., S. Martello, and P. Toth (1992), *Approximation algorithms for fixed job schedule problems*, Operations Research **40**, S96–S108.

[35] Ford, L.R., Jr., and D.R. Fulkerson (1962), *Flows in networks*, Princeton University Press, Princeton, New Jersey.

[36] Gabrel, V. (1995), *Scheduling jobs within time windows on identical parallel machines: New models and algorithms*, European Journal of Operational Research **83**, 320–329.

[37] Garey, M.R. and D.S. Johnson (1979), *Computers and intractability, a guide to the theory of NP-completeness*, W.H. Freeman and Company, New York.

[38] Garey, M.R., D.S. Johnson, G.L. Miller, and C.H. Papadimitriou (1980), *The complexity of coloring circular arcs and chords*, SIAM Journal on Algebraic and Discrete Mathematics **1**, 216–227.

[39] Gertsbach, I. and Y. Gurevich (1977), *Constructing an optimal fleet for a transportation schedule*, Transportation Science **11**, 20–36.

[40] Gertsbakh, I. and H.I. Stern (1978), *Minimal resources for fixed and variable job schedules*, Operations Research **26**, 68–85.

[41] Golumbic, M.C. (1980), *Algorithmic graph theory and perfect graphs*, Academic Press, San Diego, California.

[42] Graves, S.C., A.H.G. Rinnooy Kan, and P.H. Zipkin (editors) (1993), *Logistics of production and inventory*, Handbooks in Operations Research and Management Science **4**, North-Holland, Amsterdam.

[43] Gupta, U.I., D.T. Lee, and J.Y.-T. Leung (1979), *An optimal solution for the channel-assignment problem*, IEEE Transactions on Computers **C-28**, 807–810.

[44] Goldman, S.A., J. Parwatikar, and S. Suri (2000), *Online scheduling with hard deadlines*, Journal of Algorithms **34**, 370–389.

[45] Hall, N.G., M. Lesaoana, and C.N. Potts (2001), *Scheduling with fixed delivery dates*, Operations Research **49**, 134–144.

[46] Hashimoto, A. and J. Stevens (1971), *Wire routing by optimizing channel assignment within large apertures*, Proceedings of the 8th Design Automation Workshop, 155–169.

[47] Hochbaum, D. (editor) (1997), *Approximation algorithms for NP-hard problems*, PWS Publishing Company, Boston.

[48] Huang, Q. and E. Lloyd (2003), *Cost constrained fixed job scheduling*, Italian Conference on Theoretical Computer Science, Lecture Notes in Computer Science **2841**, 111–124.

[49] Ibaraki, T. and N. Katoh (1988), *Resource allocation problems: algorithmic approaches*, The MIT Press, Cambridge.

[50] Jansen, K. (1994), *An approximation algorithm for the license and shift class design problem*, European Journal of Operational Research **73**, 127–131.

[51] Jansen, K. (2000), *Approximation results for the optimal cost chromatic partition problem*, Journal of Algorithms **34**, 54–89.

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

[53] Kolen, A.W.J. and L.G. Kroon (1991), *On the computational complexity of (maximum) class scheduling*, European Journal of Operational Research **54**, 23–38.

[54] Kolen, A.W.J. and L.G. Kroon (1993), *On the computational complexity of (maximum) shift scheduling*, European Journal of Operational Research **64**, 138–151.

[55] Kolen, A.W.J. and L.G. Kroon (1994), *An analysis of shift class design problems*, European Journal of Operational Research **79**, 471–430.

[56] Kolen, A.W.J. and J.K. Lenstra (1995), *Combinatorics in operations research*, Handbook of Combinatorics, edited by R.L. Graham, M. Grötschel, and L. Lovász, North-Holland, Amsterdam, pp. 1875–1910.

[57] Kroon, L.G., M. Salomon, and L. van Wassenhove (1995), *Exact and approximation algorithms for the operational fixed interval scheduling problem*, European Journal of Operational Research **82**, 190–205.

[58] Kroon, L.G., M. Salomon, and L. van Wassenhove (1997), *Exact and approximation algorithms for the tactical fixed interval scheduling problem*, Operations Research **45**, 624–638.

[59] Kroon, L.G., A. Sen, H. Deng, and A. Roy (1996), *The optimal cost chromatic partition problem for trees and interval graphs*, Proceedings of the 22nd International Workshop on Graph-Theoretic Concepts in Computer Science, Lecture Notes in Computer Science **1197**, 279–292.

[60] Kumar, V. (1998), *Approximating circular arc colouring and bandwidth allocation in all-optical ring networks*, Proceedings of the first APPROX Conference, Lecture Notes in Computer Science **1444**, 147–158, Springer, Heidelberg.

[61] Lipton, R.J. and A. Tomkins (1994), *Online interval scheduling*, Proceedings of the fifth annual ACM-SIAM Symposium On Discrete Algorithms, 302–311.

[62] Martello, S. and P. Toth (1986), *A heuristic approach to the bus driver scheduling problem*, European Journal of Operational Research **24**, 106–117.

[63] Mingozzi, A., M.A. Boschetti, S. Ricciardelli, and L. Bianco (1999), *A set partitioning approach to the crew scheduling problem*, Operations Research **47**, 873–888.

[64] Miyazawa, H. and T. Erlebach (2004), *An improved randomized on-line algorithm for a weighted interval selection problem*, Journal of Scheduling **7**, 293–311.

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

[66] Nakajima, K., S.L. Hakimi, and J.K. Lenstra (1982), *Complexity results for scheduling tasks in fixed intervals on two types of machines*, SIAM Journal on Computing **11**, 512–520.

[67] Papadimitriou, C.H. (1982), Private communication to J.K. Lenstra, April 25, 1982.

[68] Seiden, S.S. (1998), *Randomized online interval scheduling*, Operations Research Letters **22**, 171-177.

[69] Sgall, J. (1998), *On-line scheduling - a survey*, in: Online Algorithms: The State of the Art, (editors: A. Fiat and G. J. Woeginger), Lecture Notes in Computer Science **1442**, 196-231, Springer, Heidelberg.

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

[71] Vazirani, V.V. (2002), *Approximation Algorithms*, Springer, Heidelberg.

[72] Woeginger, G.J. (1994), *On-line scheduling of jobs with fixed start and end times*, Theoretical Computer Science **130**, 5–16.