Online Scheduling with Interval Conflicts

Magnús M. Halldórsson* mmh@ru.is
Boaz Patt-Shamir†‡ boaz@eng.tau.ac.il
Dror Rawitz† rawitz@eng.tau.ac.il

June 27, 2011

Abstract

In the problem of Scheduling with Interval Conflicts, there is a ground set of *items* indexed by integers, and the input is a collection of *conflicts*, each containing all the items whose index lies within some interval on the real line. Conflicts arrive in an online fashion. A scheduling algorithm must select, from each conflict, at most one survivor item, and the goal is to maximize the number (or weight) of items that survive all the conflicts they are involved in. We present a centralized deterministic online algorithm whose competitive ratio is $O(\lg \sigma)$, where σ is the size of the largest conflict. For the distributed setting, we present another deterministic algorithm whose competitive ratio is $2 \lg \sigma$, in the special *contiguous* case, in which the item indices constitute a contiguous interval of integers. Our upper bounds are complemented by two lower bounds: one that shows that even in the contiguous case, all deterministic algorithms (centralized or distributed) have competitive ratio $\Omega(\lg \sigma)$, and that in the non-contiguous case, no deterministic oblivious algorithm (i.e., a distributed algorithm that does not use communication) can have a bounded competitive ratio.

Keywords: online scheduling, online set packing, interval conflicts, competitive analysis, compound tasks, distributed algorithms.

*School of Computer Science, Reykjavik University, Menntavegur 1, 101 Reykjavik, Iceland. Supported in part by the Iceland Research Fund (grant 90032021).

†School of Electrical Engineering, Tel Aviv University, Tel Aviv 69978, Israel. Research supported in part by the Next Generation Video (NeGeV) Consortium, Israel.

‡Supported in part by the Israel Science Foundation (grant 1372/09) and by a grant from Israel Ministry of Science and Technology.

1 Introduction

We study the following abstract problem, which we call *Scheduling with Interval Conflicts*. There is a universe U of n *items*, each with an integer identifier. The input is a collection $\mathcal{C}$ of *conflicts*, where each conflict $C \in \mathcal{C}$ is a set containing all the items of U within some interval on the real line. A conflict represents an event where specified items compete for a resource that can be granted to only one item. Conflict resolution is carried out by a scheduling algorithm that decides which item survives: all other items in the conflict set are eliminated. The goal of the scheduling algorithm is to maximize the number (or weight) of items that survive *all* their conflicts.

Scheduling with Interval Conflicts arises naturally in some scenarios. One interpretation of the model is when we have a set of permanently-running stations that may interfere only with other neighboring stations, where the underlying metric space is a line, and the interference range in each direction may change in every step. In each step we need to choose a station that will win the current conflict, if any. The goal is to maximize the number of stations that never fail.

Another example for our model are tasks that must be processed by a few bounded-capacity servers located at different sites on the Internet. The tasks are sent to these servers in the same order, and due to varying congestion conditions in the network, they arrive at the servers with varying burstiness: for example, the input to server A may be such that at step t task i arrives, at step $t+1$ tasks $i+1, i+2, i+3$ arrive together, at step $t+2$ no task arrives etc. The input to another server B may exhibit a different burst structure, e.g., tasks i and $i+1$ arrive together, and tasks $i+2$ and $i+3$ arrive together. Assume that the servers can process only one task at a step, and tasks cannot be stored for later processing. Then a time step in which more than a single task arrives can be represented as an interval conflict. The main question in our model is which tasks to process and which to drop, so as to maximize the total number of tasks that receive all the processing they require.

Finally, consider multiple streams of data-frames (e.g., video frames) that need to be transmitted across the Internet. Since data frames are typically too large to fit in a single packet, the frames are broken into a number of packets, and reconstructed at the receiver. However, if a packet is lost in transit, its whole constituent frame (i.e., *item*) becomes useless. Interval conflicts arise if the streams pass through a congested router which can forward only one packet from each burst of packets that arrive together (all other packets are dropped).

Problem variants. One desirable feature of a solution method is when conflicts are resolved without knowledge of other conflicts. This would be necessary if conflicts arrive in different locations and communication would not be possible or feasible, or if the conflict resolution protocol must be stateless. We call this variant *oblivious* (or *distributed*) scheduling. The variant when all previous conflicts and their outcomes are known to the algorithm when a new conflict arrives will be called *sequential* scheduling. Note that both oblivious and sequential scheduling algorithms are *online*, i.e., no information about future conflicts is available to the algorithm (the offline variant of the problem is when all conflicts are given ahead of time).

An interesting special case of interval conflicts is when the universe of items contains no gaps, i.e., the items have identifiers $i_0, i_0+1, \ldots, i_0+n-1$ (in general, item identifiers are only required to be totally ordered). We refer to this as the *contiguous* case.

1.1 Our Contribution

In this paper we introduce and formalize the problem of Scheduling with Interval Conflict (abbreviated SIC below), and give deterministic online algorithms and lower bounds on the competitive ratio of deterministic algorithms. We start off with considering oblivious algorithms. It turns out that contiguous conflicts allow for an oblivious (and hence distributed) algorithm, guaranteeing competitive ratio of $\Theta(\lg \sigma)$, where σ is the maximal number of items in a conflict. This algorithm can be used to design an oblivious $O(\lg \sigma/b)$-competitive algorithm for the generalized problem when b items may survive each conflict. However, no competitive oblivious algorithm exists if item identifiers are not contiguous, as we show. We then give a sequential $O(\lg \sigma)$-competitive algorithm that works also in the case of weighted items and non-contiguous item identifiers. Both upper bounds are matched by a $\Omega(\lg \sigma)$-lower bound on the competitive ratio of any deterministic online algorithm, even sequential algorithms for unweighted contiguous SIC.

Finally, we provide two additional sequential algorithms. In the first the competitiveness is expressed in terms of the *depth* of the interval structure, where depth is defined to be the maximal number of conflicts that any single item is involved in. We also provide a matching lower bound. The second is a simple online algorithm that is allowed to deliver up to two items from each conflict. We show that this algorithm is 1-competitive compared to an optimal offline algorithm that may deliver only one item per conflict.

1.2 Related Work

The offline version of our problem, finding a maximum subset of points with no two in a common interval, is easily solvable in polynomial time (see Section 2). A related minimization problem is finding the minimum number of points intersecting all intervals, or alternatively minimum clique partition. A 2-competitive online algorithm for the latter problem is given and shown to be the best possible in [6].

A different dual problem is the interval selection problem, where we seek a maximum cardinality subset of disjoint intervals. In the online version, the intervals that arrive over time must be irrevocably accepted or rejected. Randomized algorithms for different cases are known [9, 1, 2]; the result closest in spirit to ours is an $O(\log m)$-competitive algorithm (originally for call control on the line) [1], where m is the number of possible interval endpoints. In general, however, a $\Omega(n)$ lower bound holds for the competitive ratio of randomized algorithms [2], where n is the number of intervals. Interval selection can be seen as an instance of scheduling with conflicts, which has been studied extensively (see, e.g., the surveys of [7, 10]), but to the best of our knowledge, we are the first to consider online conflicts in the form of groups of consecutive items.

The problem of multi-packet frames (sketched above) was introduced in [8], where it is shown that if packet ordering is arbitrary (namely conflicts are not necessarily intervals), then the competitive ratio is $\Omega(\sigma)$ even for two-packet frames. A general framework that deals with transmission of multi-packet frames is described in [3]. The problem is modeled as an online version of Set Packing, nearly tight bounds of $\tilde{\Theta}(k\sqrt{\sigma})$ are proven on the competitive ratio of randomized algorithms for Online Set Packing, and a $\Omega(\sigma^{k-1})$ deterministic lower bound is shown, where k is the maximum size of a set and σ is the maximum number of sets that contain the same element. In our terms, it is assumed there that each item is involved in up to k conflicts, and conflicts need not be intervals.

Note the unusual characteristic of our problem is that the solution only decreases as more of the input arrives. Little is known about online maximization problems of this sort; the only

related result we are aware of is [3].

1.3 Paper Organization

The remainder of this paper is organized as follows. In Section 2 we formalize the problem and present the basic arguments we use in analyzing our algorithms. We study oblivious algorithms in Section 3, and sequential algorithms are considered in Section 4. In Section 5 we prove a lower bound on the competitive ratio of online algorithms. The two additional sequential algorithm are given in Sections 6 and 7, with concluding remarks are given in Section 8.

2 Preliminaries and Basic Argument

In this section we formalize the problem, define the concepts and notation we use, and present the basic argument we employ in the analysis of our algorithms.

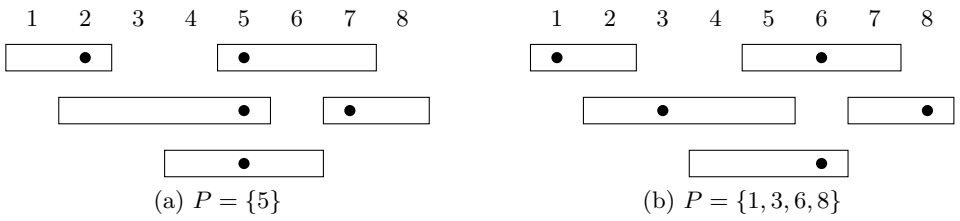
2.1 Problem Statement and Notation

Scheduling with Interval Conflicts (abbreviated SIC) is defined as follows. There is a set U of n integer *items*. The input is a collection $\mathcal{C}$ of *conflicts*, where each conflict $C \in \mathcal{C}$ contains all items of U within some interval on the real line. Namely, $C = U \cap [\min(C), \max(C)]$. In this paper we also consider the *Contiguous Model*, where U is a set of consecutive integers. The size of the largest conflict is denoted by $\sigma_{\mathcal{C}}$, namely $\sigma_{\mathcal{C}} \stackrel{\text{def}}{=} \max\{|C| : C \in \mathcal{C}\}$ (the subscript is omitted when the instance is clear from the context). A feasible schedule is a set of items $P \subseteq U$ containing at most one item from any given conflict, i.e. $|P \cap C| \leq 1$ for every $C \in \mathcal{C}$. An item in P is said to be a *survivor* of its conflicts, while the other items were *eliminated*. If item i survives conflict $C \ni i$ by algorithm A, we say that A *delivers* i from C. The goal is to find a maximum cardinality feasible schedule, i.e. maximize the number of items surviving all their conflicts (see example in Figure 1). We also consider a generalized version of SIC, where we are given an integer $b \geq 1$, and a feasible schedule is a set of points $P \subseteq U$ such that no conflict contains more than b items from P, i.e. $|P \cap C| \leq b$ for every $C \in \mathcal{C}$. We shall assume, without loss of generality, that $|C| > b$ for all conflicts $C \in \mathcal{C}$ ($|C| > 1$ in the basic version of SIC). Note that this implies that $b < \sigma$.

In the *weighted* case, each item i has a real-valued weight $w(i) > 0$ and the objective is to find a maximum weight subset of weights satisfying the conflict constraints. For a set S of items, $w(S) \stackrel{\text{def}}{=} \sum_{i \in S} w(i)$.

We consider two models of algorithms. In the *oblivious* model, the selection of a survivor from a conflict is a function of that conflict only, which allows for distributed conflict resolution. In the *sequential* model, conflicts arrive over time, i.e., they are ordered as a sequence $C_1, C_2, \ldots$, and the resolution of conflict C_t may be a function of the full history $C_1, \ldots, C_t$.

We note that simple heuristics for SIC may perform poorly. For example, selecting the leftmost item in each given conflict is $\Omega(n)$-competitive as demonstrated by the instance in Figure 2. The same goes for the sequential strategy of picking the leftmost item among the items that were not eliminated in previous conflicts. (We assume that the top conflict is the first to arrive.)



(a) $P = \{5\}$ (b) $P = \{1, 3, 6, 8\}$

Figure 1: *An instance of* SIC *with two possible solutions. Rectangles represent conflicts, and dots represent items that were selected as survivors in conflicts. The instance contains eight items and five conflicts whose size is at most* 4 *(i.e.,* $\sigma = 4$*). In Figure 1a (top), only one item survives all conflicts, while an optimal solution can have four such items (Figure 1b).*

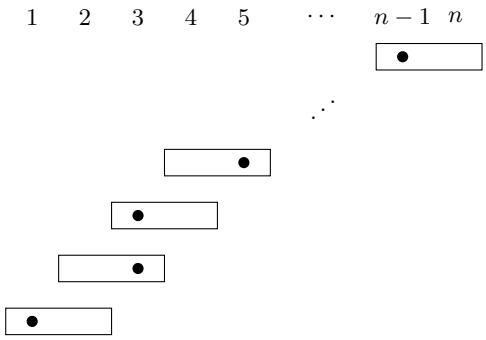


Figure 2: *An instance of* SIC *with* n *items and* $n - 1$ *conflicts (all of size* 2*, i.e.,* $\sigma = 2$*). If one selects the leftmost item in each given conflict, only one item (number* 1*) survives all conflicts, while by always picking the odd-numbered items (represented by dots in the figure), one gets an optimal solution of size* $\lceil n/2 \rceil$*.*

2.2 Characterizing Optimal Solutions

The offline version of SIC can be formulated as finding a maximum number of disjoint cliques in an interval graph, which can reduced to maximum independent set in proper intervals graphs, which is solvable in polynomial time [5]. The reduction is as follows. First, remove all conflicts that are properly contained in other conflicts. It follows that there is a total order on the remaining conflicts, given equally by their left and right endpoints. We may then view each item as an interval representing a contiguous sequence of conflicts. Finding an optimal schedule now corresponds to finding a maximum interval selection (subset of mutually disjoint intervals) in this interval system. We can similarly reduce the interval selection problem to the offline SIC problem.

We give a more direct description below. First, we provide an upper bound on the optimal solution due to duality. Let $\textsc{opt}(\mathcal{C})$ denote an optimal solution of SIC to instance $\mathcal{C}$, and let $U(\mathcal{C}) = \bigcup_{C \in \mathcal{C}} C$ denote the set of items involved in conflicts in $\mathcal{C}$.

Observation 1. *For all* $\mathcal{C}' \subseteq \mathcal{C}$*: If* $U(\mathcal{C}') = U(\mathcal{C})$*, then* $|\textsc{opt}(\mathcal{C}) \cap U(\mathcal{C})| \leq |\mathcal{C}'|$*.*

Observation 1 motivates a simple polynomial offline algorithm for SIC. Briefly, the idea is to scan the item set from left to right (the examples in Figure 1 may help the reader), initially selecting the leftmost item. The next element selected, following a selected element i_j, is then inductively the leftmost element among those that are not in conflicts with i_j, i.e.,

$i_{j+1} = \min\{i' : i' > i_j \text{ and } \forall C \in \mathcal{C}, |\{i_j, i'\} \cap C| \leq 1\}$. This forms a feasible solution, since for any consecutively chosen items i_j and i_{j+1}, there is no conflict containing both i_j and i_{j+1}. To prove that the selected elements constitute an optimal solution, let C'_j be the conflict that contains i_j and $i_{j+1} - 1$. Since $\bigcup_j C'_j = U(\mathcal{C})$, optimality follows from Observation 1.

3 Oblivious Algorithms

In this section we consider oblivious algorithms. Oblivious algorithms are attractive because they can be implemented in a distributed system. The main result of this section is a $2 \lg \sigma$-competitive oblivious algorithm for unweighted contiguous SIC. We also show that this algorithm generalizes to the non-unit capacity case. We start the section by showing that if the instance is not contiguous, then no oblivious algorithm can be competitive.

3.1 A Lower Bound for the Non-Contiguous Case

We argue that no deterministic oblivious algorithm is competitive in the general (non-contiguous) case even if $\sigma = 2$.

Theorem 1. *The competitive ratio of any deterministic oblivious algorithm for* SIC *is* $\Omega(n)$*, even for the unweighted case and for* $\sigma = 2$*.*

Proof. Fix a deterministic oblivious algorithm ALG. By definition of obliviousness, the decision of ALG for a given conflict C depends only on its items. Let n be a number and let $N = 2^n$. We 2-color the edges of an N-vertex clique K_N as follows. Edge (v_i, v_j), for $i < j$, is colored blue if ALG prefers v_i over v_j, and otherwise red. By Ramsey's theorem [4], K_N contains a monochromatic subgraph of $\log N = n$ vertices. That means that there is either an increasing or a decreasing sequence of n items $i_1, \ldots, i_n$ such that ALG prefers i_ℓ over $i_{\ell-1}$, for any $\ell \in \{2, \ldots, n\}$. We introduce the conflicts $\{i_{\ell-1}, i_\ell\}$, for $\ell \in \{2, \ldots, n\}$. Then, only i_n will survive the execution of ALG, whereas $\{i_\ell : \ell \text{ is odd}\}$ is a feasible solution of size $n/2$. □

3.2 Oblivious Algorithm for Contiguous SIC

In this section we present a simple $2\lceil \lg \sigma \rceil$-competitive algorithm for unweighted contiguous SIC. We note that the algorithm needs not know σ in advance.

The basic idea of the algorithm is to assign to each item a fixed priority, and the conflict resolution rule is to always prefer the item with the highest priority. Specifically, our algorithm, **Priority**, defines the priority of item i by

$$p(i) \stackrel{\text{def}}{=} \max\left\{\ell \in \mathbb{Z} \mid i \text{ is divisible by } 2^\ell\right\} . \tag{1}$$

For example, if i is odd, then $p(i) = 0$, and if $i = 2^\ell$ then $p(i) = \ell$.[1]

One nice consequence of this definition is the following observation:

Observation 2. *If* $i_\ell < i_r$ *and* $p(i_\ell) = p(i_r) = p$ *for some* p*, then there exists* $i_\ell < i < i_r$ *such that* $p(i) > p$*.*

[1] For an efficient implementation (in AC0), e.g. to use in routers, it suffices to extract the smallest bit set, using the bit-wise operations (i XOR $(i-1)$) AND i.

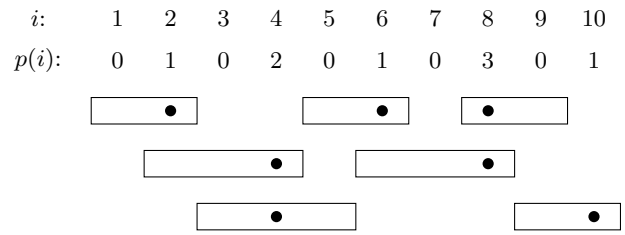


Figure 3: *Execution of Algorithm* ***Priority*** *on an instance with ten items and seven conflicts. The dots represent the items that were chosen by the algorithm. The computed solution is* $P = \{4, 8, 10\}$*, while the optimum is* $\{1, 3, 6, 9\}$*.*

Observation 2 implies that any conflict contains exactly one item with maximum priority, and hence Algorithm **Priority** is well-defined: Upon arrival of conflict C, the algorithm delivers the unique item with highest priority (as defined by (1)) among the items in C. See Figure 3 for an example. Note that the algorithm makes decisions without knowing or even estimating σ, and that it is completely distributed: the identity of the winner of a conflict is independent of other conflicts.

Next, observe that even though there is *a priori* no bound on the maximum priority (since the item sequence can have an arbitrarily high starting point), we need only to concern ourselves with $\lceil \lg \sigma \rceil$ priorities.

Observation 3. *Each conflict contains at most one item* i *with* $p(i) \geq \lceil \lg \sigma \rceil$*.*

Observation 3 implies that given conflicts whose length is bounded by σ, all priorities greater than or equal to $\lg \sigma$ are indistinguishable from the viewpoint of Algorithm **Priority**.

We now turn to prove that the competitive ratio of Algorithm **Priority** is at most $2 \lceil \lg \sigma \rceil$. We use the following concept.

Definition 1. *Let* $\mathcal{C}$ *be an instance and let* A *be an algorithm for* SIC*. A sequence of items* $i_0, i_1, \ldots, i_m$ *is an* elimination chain of length m *if for all* $0 < j \leq m$ *we have that item* i_j *eliminates item* i_{j-1} *when* A *runs on* $\mathcal{C}$*.*

Note that an elimination chain of length m contains $m + 1$ items, but implies the existence of m conflict intervals. Elimination chains have the following property.

Lemma 4. *Let* $\mathcal{C}$ *be an instance and let* A *be an algorithm for* SIC*. Suppose that* $i_0, \ldots, i_m$ *is an elimination chain for* $\mathcal{C}$ *under* A*. Then, the interval* $[i', i'']$ *is covered by* m *conflicts in* $\mathcal{C}$*, where* $i' = \min\{i_j \mid 0 \leq j \leq m\}$ *and* $i'' = \max\{i_j \mid 0 \leq j \leq m\}$*.*

Proof. By Definition 1, for any $0 < j \leq m$ there exists a conflict $I_j \in \mathcal{C}$ such that $i_{j-1} \in I_j$ and $i_j \in I_j$. It follows, by induction on m, that $\bigcup_{j=1}^{m} I_j \supseteq [i', i'']$. □

Lemma 4 implies the following consequence. Say that an algorithm is *reasonable* if it delivers an item from each conflict, i.e. it does not unnecessarily eliminate items.

Proposition 1. *The competitive ratio of any reasonable algorithm for* SIC *is at most* $2m$*, where* m *is the length of the longest elimination chain of the algorithm.*

Proof. Fix an instance $\mathcal{C}$. Say that an item i' is *dominated* by an item i if there is an elimination chain that starts with i' and ends with i. For each item i, let $D(i)$ be the set of items dominated

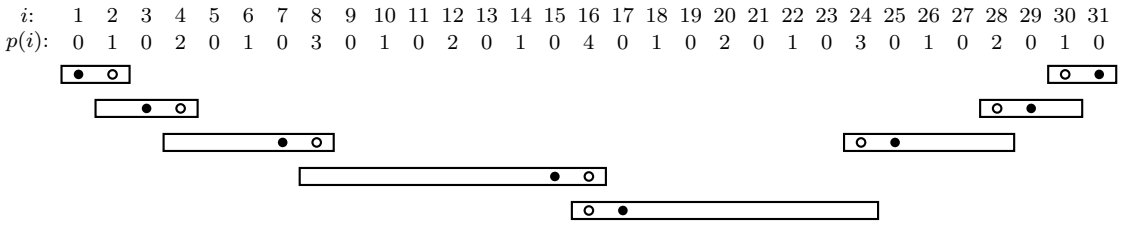


Figure 4: *The input sequence for $\ell = 5$. Algorithm **Priority** delivers only item 16; the items corresponding to all other unfilled dots are eliminated. On the other hand, the solution $P = \{1, 3, 7, 15, 17, 25, 29, 31\}$ (solid dots) is optimal.*

by i. Now, consider a reasonable algorithm A, and let P be the set of items delivered by A running on $\mathcal{C}$. Clearly, each item i was either delivered by A, i.e., $i \in P$, or eliminated, in which case $i \in D(i')$ for some $i' \in P$ (because A is reasonable). In addition, we have by Lemma 4 that $D(i')$ can be covered by the conflicts of two elimination chains: one starting from $\min(D(i'))$ and one that starts from $\max(D(i'))$. We can therefore conclude that the set of all items can be covered by $2m|P|$ intervals. The result now follows from Observation 1. □

Using Proposition 1 and Observation 3 we can easily bound the competitive ratio of Algorithm **Priority**.

Theorem 2. *The competitive ratio of Algorithm **Priority** is at most $2\lceil \lg \sigma \rceil$.*

Proof. By Observation 3 and the fact that under **Priority** an item i is eliminated by an item i' if and only $p(i) < p(i')$, we have that the length of any elimination chain under **Priority** is at most $\lg \sigma$. The theorem therefore follows directly from Proposition 1. □

We show that the analysis of Algorithm **Priority** is tight.

Proposition 2. *The competitive ratio of Algorithm **Priority** is at least $2\lceil \lg \sigma \rceil$.*

Proof. Let $\ell \geq 2$ be a number and let $n = 2^\ell - 1$. Consider the following conflict set that contains 2ℓ conflicts:

$$\mathcal{I}_\ell = \bigcup_{j=1}^{\ell-1} \left\{ [2^{j-1}, 2^j], [2^\ell - 2^j, 2^\ell - 2^{j-1}] \right\} .$$

See example with $\ell = 5$ in Figure 4. Observe that the maximum conflict size is $\sigma_\ell = 2^{\ell-2} + 1$.

Algorithm **Priority** delivers only item $2^{\ell-1}$, since any other item is eliminated by a higher priority item. Any item $i \in [2^{j-1}, 2^j - 1]$, for some $j \in [1, \ell - 1]$, is eliminated by 2^j, while any item $i \in [2^\ell - 2^j - 1, 2^\ell - 2^{j-1}]$, for some $j \in [1, \ell - 1]$, is eliminated by $2^\ell - 2^j$.

On the other hand, observe that the items in the set $P = \bigcup_{j=1}^{\ell-1} \left\{ 2^j - 1, 2^\ell - 2^j + 1 \right\}$ are each contained in only one conflict. Thus they form a feasible solution with $2\ell - 2 = 2(\lg(\sigma_\ell - 1) + 1) = 2\lceil \lg \sigma_\ell \rceil$ items. □

3.3 Oblivious Algorithms: The b-Capacity Case

In the capacitated version of SIC, we are given an integer $b \geq 1$, and a feasible schedule is a set of points $P \subseteq U$ such that no conflict contains more than b items from P, i.e. $|P \cap C| \leq b$ for every $C \in \mathcal{C}$. We shall assume, without loss of generality, that $|C| > b$ for all conflicts $C \in \mathcal{C}$. Note that this implies that $b < \sigma$.

We first provide a capacitated version of Observation 1.

Observation 5. *For all $\mathcal{C}' \subseteq \mathcal{C}$: If $U(\mathcal{C}') = U(\mathcal{C})$, then $|\text{OPT}(\mathcal{C}) \cap U(\mathcal{C})| \leq |\mathcal{C}'| \cdot b$.*

Next, we extend Algorithm **Priority** to the case where b items can be delivered from each conflict. For this case we obtain a competitive ratio of $O(\lg(\sigma/b))$. Our algorithm, called Algorithm **SuperItem**, is based on Algorithm **Priority** as follows. We statically partition the items into intervals of size b called *super-items*. Formally, the jth super-item is defined by $S_j \stackrel{\text{def}}{=} [(j-1)b+1, jb]$. The idea is to run Algorithm **Priority** on the super-items: given a conflict I, the algorithm delivers all items in I that are part of the super-item S_j with the highest priority $p(j)$. Note that in general, not all items of S_j must be included in conflict I.

We analyze **SuperItem** using the following unit capacity SIC instance. The new instance contains an item for every super-item in the original instance, and the new conflict set $\mathcal{C}'$ contains a conflict I'_t, for every $I_t \in \mathcal{C}$. I'_t is obtained by taking the set of super-items that intersect I_t. That is, $I'_t = \{j \mid S_j \cap I_t \neq \emptyset\}$. Let σ' denote the maximum size of a conflict interval in the new instance.

Observation 6. $\sigma' < \frac{\sigma}{b} + 1$.

Proof. Let I'_t be a conflict in $\mathcal{C}'$ of size σ'. Then, I_t contains $b-1$ items between each of the items of I'_t, for a total of at least $\sigma' + (\sigma' - 1)(b-1) > \sigma' b - b$. □

Let P be the set of surviving items obtained by executing Algorithm **SuperItem** with the original instance as input, and let P' be the set of surviving items obtained by Algorithm **Priority** with the new instance as input.

Lemma 7. $|P| \geq b \cdot |P'| - 2(b-1)$.

Proof. Observe that if j survives according to Algorithm **Priority**, then there is no conflict in $\mathcal{C}'$ that drops j. Hence, there is no conflict in $\mathcal{C}$ that drops any of the items in S_j. The additive factor of $2(b-1)$ is because the item range may not end at block boundaries. □

To compare P to an optimal solution, we use the following observation.

Observation 8. *Let $S \subseteq U(\mathcal{C})$ be an interval such that $|S| \leq b$. Then there exist $I', I'' \in \mathcal{C}$ (possibly $I' = I''$) such that $S \subseteq I' \cup I''$.*

Proof. If $S \subseteq I$ for some $I \in \mathcal{C}$, then we are done. Otherwise, let I' be the conflict with the rightmost right endpoint among conflicts whose right endpoint is in S. Similarly, let I'' be the conflict with the leftmost left endpoint among conflicts whose left endpoint is in S. Since $|I'|, |I''| > b$ we have that $\min(I') < \min(S)$ and $\max(I'') > \max(S)$. Hence, $S \subseteq I' \cup I''$, since otherwise $S \not\subseteq U(\mathcal{C})$. □

Theorem 3. *The competitive ratio of Algorithm **SuperItem** is $O(\lg(\sigma/b))$.*

Proof. Let $I_t \in \mathcal{C}$. Consider the interval I''_t obtained by extending I_t in both directions until we reach super-items boundaries or the boundaries of the contiguous component. Clearly $I_t \subseteq I''_t$. We claim that I''_t can be covered by at most five conflicts from $\mathcal{C}$. Since I''_t is covered by conflicts from $\mathcal{C}$, each of the two super-items at the edges of I''_t must be contained in at most two conflicts from $\mathcal{C}$ by Observation 8. It follows that I''_t can be covered by at most five conflicts from $\mathcal{C}$.

Now, due to Lemma 4 and Observation 3 we know that the items in $U(\mathcal{C}')$ can be covered by $2 \lg \sigma' \cdot |P'|$ conflicts from $\mathcal{C}'$. Hence, $U(\mathcal{C})$ can be covered by

$$10 \lg \sigma' \cdot |P'| < 10 \lg(\sigma/b + 1) \cdot (|P| + 2b)/b$$

conflicts. Finally, due to Observation 5 we have that

$$|\textsc{opt} \cap U(\mathcal{C})| \leq 10 \lg(\sigma/b + 1) \cdot (|P| + 2b) \ ,$$

as required. □

4 Sequential Algorithms

In this section we present an $O(\lg \sigma)$-competitive sequential algorithm for SIC, extending algorithm **Priority** to the weighted and non-contiguous case.

In describing our algorithm, we say that item i *fired* item i' if i was the first item to eliminate i'. In the remainder of this section we say that an item i_0 is *dominated* by item i_m if there is an elimination chain $i_0, i_1, \ldots, i_m$ such that i_j fires item i_{j-1}, for every j. Let $D(i)$ be the set of items that are dominated by i. Note that if i survives then $D(i) \neq \emptyset$, and in particular $i \in D(i)$. Observe that each item is dominated by exactly one surviving item, hence $D(i) \cap D(i') = \emptyset$ for each pair of surviving items i and i'.

We now describe the algorithm in the weighted case. Define the *weight class* of item i to be $c(i) = \lfloor \lg w(i) \rfloor$, the base-2 logarithm of the item weight rounded down to an integer. Our algorithm is called **Seq**, and it proceeds as follows.

With each item i, we associate two values $\text{left}(i)$ and $\text{right}(i)$ (initially both zero), referred to as the *left* and *right levels* of i, respectively. When an interval I arrives, the algorithm determines the highest weight class of active items in I. If there is only one active item of the highest weight class, it simply survives. Otherwise, let l and r be the leftmost and rightmost active items of the highest weight class. The algorithm compares $\text{left}(l)$, the left level of l, and $\text{right}(r)$, the right level of r. If $\text{left}(l) > \text{right}(r)$, then l survives and $\text{right}(l)$ is set to $\text{right}(r)+1$; otherwise, r survives and $\text{left}(r)$ is set to $\text{left}(l) + 1$. (The algorithm arbitrarily favors r, in case of a tie.) Notice that $\text{left}(i)$ and $\text{right}(i)$ may increase and decrease during execution.

Fix some optimal solution OPT. The following upper bound is what motivates the numbering of the levels. Let m_i be the larger of the levels of i, namely $m_i = \max\{\text{left}(i), \text{right}(i)\}$. Also, let $n_i = \max_{i' \in D(i)} m_{i'}$ be the largest level of an item in $D(i)$.

Lemma 9. $w(\textsc{opt} \cap D(i)) = O(n_i \cdot w(i))$.

Proof. Let $l_1, \ldots, l_t$ be the sequence of items in $D(i)$ such that l_1 is the leftmost item in $D(i)$ and, inductively, l_{j+1} is the item that fired item l_j. Observe that the sequence extends monotonically from left to i, with $l_t = i$. According to the survival rule of the algorithm, the weight classes of the items are monotonically non-decreasing, and for a pair of items l_j and l_{j+1} in the same weight class, the left levels are strictly increasing, namely $\text{left}(l_{j+1}) > \text{left}(l_j)$. It follows that there are at most n_i items from the item set $\{l_1, \ldots, l_t\}$ in each weight class. Hence, the sum of the weights of the items $l_1, \ldots, l_t$ is bounded by

$$\sum_{j=1}^{t} w(l_j) < 2 \sum_{j=1}^{t} 2^{c(l_j)} \leq 2n_i \sum_{c \leq c(i)} 2^c < 2n_i \cdot 2^{c(i)+1} \leq 4n_i \cdot w(i) \ . \tag{2}$$

Let $D^-(i)$ ($D^+(i)$) be the subset of items in $D(i)$ to the left (right) of i, up to and including i. That is, $D^-(i) \cup D^+(i) = D(i)$ and $D^-(i) \cap D^+(i) = \{i\}$. Partition $D^-(i)$ into ranges $[l_{j+1}, l_j]$, for $j = 1, \ldots, t-1$. Observe that l_{j+1} must be in the largest weight class among the items in the range $[l_j, l_{j+1}]$, for all $j = 1, \ldots, t-1$. (Namely, if there was a item in $[l_j + 1, l_{j+1} - 1]$ belonging to a larger weight class, the largest such item could not have been eliminated without either l_j or l_{j+1} being also eliminated.) Since OPT can contain at most one item from each range $[l_j, l_{j+1}]$, it follows from (2) that

$$w(\text{OPT} \cap D^-(i)) < 2 \sum_{j=1}^{t} w(l_j) \leq 8n_i \cdot w(i) \ .$$

Applying the same arguments to $D^+(i)$ yields that $w(\text{OPT} \cap D(i)) \leq 16n_i \cdot w(i)$, implying the lemma. □

We now show that high levels imply very large intervals.

Lemma 10. $n_i = O(\lg \sigma)$.

Proof. Let $\hat{D}(i')$ be the set of items dominated by i' that are from the same weight class, $c(i')$, as i'.

Claim 1. $|\hat{D}(i')| \geq 2^{m_{i'}}$, *for any item* i' *in* $D(i)$.

Proof. The proof is by induction on $m_{i'}$. The base case $m_{i'} = 0$ is trivially true, since $i' \in \hat{D}(i')$. For the inductive step, suppose that $m_{i'} \geq 1$. Consider the most recent conflict I that i' survived and in which an item from class $c(i')$ was fired. Let l and r be the leftmost and rightmost active items in I, respectively, such that $c(l) = c(r) = c(i')$, when I was presented. Observe that $i' \in \{l, r\}$. By the inductive hypothesis, $|\hat{D}(l)| \geq 2^{m_l}$ and $|\hat{D}(r)| \geq 2^{m_r}$. If $m_{i'} = \max\{m_l, m_r\}$, then we are done. Suppose then that $m_{i'} = \max\{m_l, m_r\} + 1$, which happens only when $m_l = m_r$. Since $\hat{D}(l)$ and $\hat{D}(r)$ are disjoint, we have that

$$|\hat{D}(i')| \geq |\hat{D}(l)| + |\hat{D}(r)| \geq 2^{m_l} + 2^{m_r} = 2^{m_{i'}} \ .$$

and the claim follows. □

Claim 2. $\hat{D}(i')$ *is covered by at most* $2m_{i'}$ *intervals.*

Proof. Let $l_1, l_2, \ldots, l_t$ be the sequence of items defined such that l_1 is the leftmost item in $\hat{D}(i')$ and, inductively, l_{j+1} is the item that fired l_j, for $j = 1, \ldots, t-1$. Also, let I_j be the interval presented upon which l_{j+1} fired l_j, for $j = 1, \ldots, t-1$. Clearly, $I_1, \ldots, I_{t-1}$ cover the items to the left of $l_t = i'$, up to and including i'. According to the survival rule of the algorithm, the left levels of the items are strictly increasing. It follows that $t \leq m_{i'} + 1$. By symmetry, $m_{i'}$ intervals also cover the items in $\hat{D}(i')$ to the right of i'. □

We resume with the proof of Lemma 10. Let $i' \in D(i)$ such that $n_i = m_{i'}$. By the two claims above, some interval covers at least $|\hat{D}(i')|/(2m_{i'}) \geq 2^{m_{i'}-1}/m_{i'}$ items. Hence, $\sigma \geq 2^{m_{i'}-1}/m_{i'}$, or $n_i = m_{i'} \leq \lg \sigma(1 + o(1))$. □

The following theorem is now immediate from Lemmas 9 and 10 when observing that the sets $\{D(i) : i \text{ survived}\}$ partition the set U of items.

Theorem 4. *The competitive ratio of the oblivious algorithm* ***Seq*** *for the weighted and non-contiguous case is* $O(\lg \sigma)$.

5 A Lower Bound on the Competitive Ratio

In this section we show that the competitive ratio of any deterministic online algorithm for contiguous SIC is $\Omega(\lg \sigma)$. Our lower bound construction is sequential, namely the conflicts arrive one by one, and the algorithm knows the complete history when a new conflict arrives. Since any algorithm for oblivious SIC can be used in the sequential model, the lower bound holds for oblivious SIC as well.

Fix a deterministic online algorithm A. Based on the way A picks items to survive conflicts, we construct in an online fashion a sequence of conflicts along with an optimal scheduling denoted by OPT. To facilitate the description, define a conflict I to be *active* with respect to algorithm A if upon arrival, I contains a item that was not already eliminated by A in previous conflicts. W.l.o.g., we consider only algorithms that always deliver an item from an active interval.

In general, there can be conflicts that are active with respect to A, OPT, or both. We call a conflict interval *neutral* if it is active with respect to both A and OPT, and *positive* if it is active with respect to OPT only (there will be no "negative" intervals in our construction).

The conflict sequence consists of a sequence of *epochs* satisfying the following epoch invariant:

- In each epoch, all conflicts are disjoint and their union is $\{1, \ldots, n\}$.
- The set of items delivered by A and by OPT from epoch $q \geq 1$ are disjoint.
- In epoch q there are $q-1$ positive intervals between any two consecutive neutral intervals.

Note that the last property means that after epoch q, the optimal number of surviving items is q times larger than the number of items delivered by A.

We now describe the construction of epochs inductively. Assume that n is an even integer. The first epoch consists of $n/2$ intervals of size 2: for every $t \in [1, n/2]$, the tth interval is $[2t-1, 2t]$. Let A_1 be the set of items that are delivered by the algorithm after the first epoch. Clearly $|A_1| = n/2$. The optimal solution is the complement of A_1, namely $\text{OPT}_1 = \{1, \ldots, n\} \setminus A_1$. It is straightforward to verify that the epoch invariant holds for $q = 1$ (the last property follows from the fact that all intervals in epoch 1 are neutral).

The more interesting part is the inductive step. Let A_q and OPT_q be the set of active items with respect to A and OPT, respectively, immediately after epoch q. Assume that the invariant holds for epoch q. We construct epoch $q+1$ and OPT_{q+1} as follows. Number the neutral intervals of epoch q sequentially $I_1, I_2, \ldots$, starting from the leftmost neutral interval. This numbering skips the positive intervals between neutral intervals. Let $I_t = [\ell_t, r_t]$ be the tth neutral interval, where t is even. We break I_t into two parts as follows. Let a and o be the indices of items that are delivered from I_t by A_q and OPT_q, respectively in epoch q. We proceed by two cases. If $a < o$, then we introduce the conflict interval $[o, r_t]$ and extend I_{t-1} to the right up to $o-1$ (see Figure 5). Otherwise, if $a > o$, then we introduce the conflict interval $[\ell_t, o]$ and extend I_{t+1} to the left up to $o+1$ (see Figure 6). Notice that an odd neutral interval from epoch q can be either extended to the left, or to the right, or in both directions, or not extended at all. Finally, positive intervals from epoch q that are not covered by the above intervals are added to epoch $q+1$. Figure 7 illustrates a complete example.

It remains to determine OPT_{q+1}. Let I'_t be the extended version of an odd interval I_t from epoch q. OPT_{q+1} will deliver the active item from I_t. Since I_t does not intersect any even neutral interval, OPT_{q+1} may deliver an item in any part of an even neutral interval that was added to epoch $q+1$. Also, since I_t does not intersect any positive interval from epoch q, OPT_{q+1} may

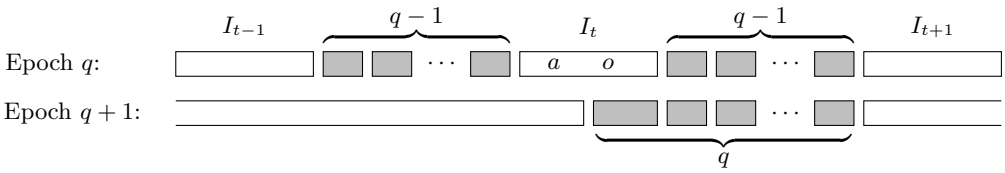


Figure 5: *Construction of epoch $q+1$: The case where $a < o$. Gray boxes represent positive intervals. I_t is split into two parts: the left part is combined with I_{t-1}, while the right part becomes a positive interval.*

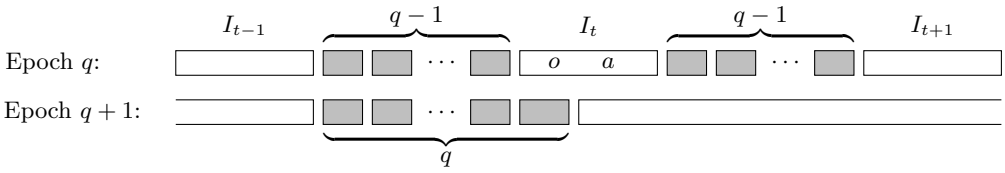


Figure 6: *Construction of epoch $q+1$: The case where $a > o$. Gray boxes represent positive intervals. I_t is split into two parts: the right part is combined with I_{t+1}, while the left part becomes a positive interval.*

deliver an item in any positive interval that was not merged with an odd neutral interval from epoch q.

The first and second properties of the invariant are clearly satisfied by the construction. To see that the last property of the invariant holds, observe that any extended odd neutral interval remains neutral. We claim that there are q positive intervals between any consecutive neutral intervals. Let I_t be an even neutral interval that was split in the construction of epoch $q+1$. If $a < o$ the interval $[o, r_t]$ is positive because $[o, r_t] \cap A_q = \emptyset$. Moreover, all positive intervals between I_t and I_{t+1} remain as they were. Similarly, if $a > o$ the interval $[\ell_t, o]$ is positive, because $[\ell_t, o] \cap A_q = \emptyset$, and all positive intervals between I_t and I_{t-1} are left unchanged. Hence, there are q positive intervals between any two consecutive neutral intervals in epoch $q+1$.

The following lemma bounds the size of intervals.

Lemma 11. *Let σ_q be the maximum interval size in epoch q. Then $\sigma_q \leq 2 \cdot 5^{q-1}$.*

Proof. By induction on the number of epochs. In the base case (epoch 1), $\sigma_1 = 2$. For the inductive step, observe that an interval in the epoch $q+1$ may consist of (i) an odd neutral interval, (ii) parts of two even neutral intervals, (iii) $2(q-1)$ positive intervals. Since positive intervals that are created in epoch q' are of size smaller than $\sigma_{q'-1}$ and due to the inductive hypothesis, we have that

$$\sigma_{q+1} < \sigma_q + 2\sigma_q + 2\sum_{q'=1}^{q-1} \sigma_{q'} \leq 3\sigma_q + 2\sum_{q'=1}^{q-1} \sigma_{q'} < 5\sigma_q \ ,$$

and the lemma follows. □

We can now prove the lower bound.

Theorem 5. *The competitive ratio of any deterministic online algorithm for sequential* SIC *is $\Omega(\lg \sigma)$, even in the contiguous case.*

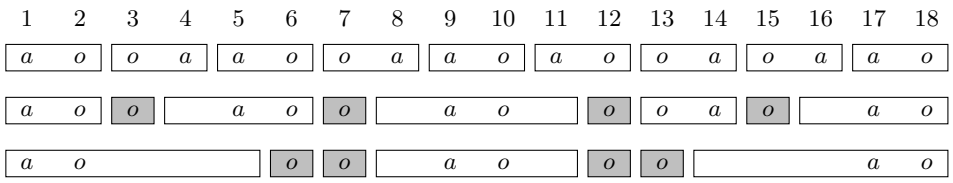


Figure 7: *The lower bound construction with $n = 18$ and three epochs. The gray boxes represent positive intervals.*

Proof. Let A be a deterministic algorithm. Construct instance $\mathcal{C}$ as described above. The epoch invariant implies that after q epochs, $|\textsc{opt}(\mathcal{C})| \geq q(|A(\mathcal{C})| - 1)$. Hence, the competitive ratio of A is $\Omega(q)$. Let σ be a given parameter. By Lemma 11 we have that $\sigma_q \leq 2 \cdot 5^{q-1}$, and therefore, setting $q = \lfloor \log_5(\sigma/2) \rfloor = \Omega(\lg \sigma)$ we have $\sigma_q \leq \sigma$, and the result follows. □

6 Online Algorithm in Terms of Depth of Intervals

We consider here an online algorithm that is competitive in terms of another parameter, called *depth*: Given an instance $\mathcal{I}$, we define its depth, denoted $\omega_{\mathcal{I}}$, to be the maximal number of conflicts any single item is involved in. We note that $\omega_{\mathcal{I}}$ is the clique number and the chromatic number of the underlying interval graph. It turns out that if the depth of the instance is small, then good performance can be attained. We also provide a matching lower bound.

Consider the following algorithm, which we call **Experience**. The algorithm maintains, for each active item i the count $c(i)$ of the number of conflicts in which it was delivered. (Note that this is possible only in the sequential model.) The algorithm operates by the following simple rule: deliver an active item i of a maximum weight class, and among those choose one of maximum count $c(i)$, breaking ties arbitrarily. We now prove that Algorithm **Experience** is ω-competitive. Once again, we invoke the concept of elimination chains.

Lemma 12. *Any elimination chain for Algorithm **Experience** with instance $\mathcal{I}$ contains at most $\omega_{\mathcal{I}}$ items of any given weight class.*

Proof. Follows from the fact that if item i is fired by item i' of same weight class, then the final counter values of i and i' satisfy $c(i) > c(i')$ (even if their counts were equal at the time i fired i'). □

Theorem 6. *The competitive ratio of Algorithm **Experience** is $O(\omega_{\mathcal{I}})$.*

Proof. First observe that if $i' \in D(i)$, then it must be that $w(i') < 2w(i)$. Lemma 12 implies that all items in $D(i)$ from weight class j can be covered by at most $\omega_{\mathcal{I}}$ conflicts, for every j. It follows that

$$w(\textsc{opt} \cap D(i)) < \omega_{\mathcal{I}} \cdot 2w(i) \cdot \sum_{j=0}^{\infty} 2^{-j} = \omega_{\mathcal{I}} \cdot 4w(i) \ .$$

The theorem follows. □

Sometimes, Algorithm **Experience** performs better than Algorithm **sPriority**. Such an instance is given in Figure 4.

The construction of Section 5 shows that the competitive ratio of Algorithm **Experience** is tight:

Theorem 7. *The competitive ratio of any deterministic online algorithm for sequential* SIC *is* $\Omega(\omega_{\mathcal{I}})$*, even in the contiguous case.*

7 Online Problem with Resource Augmentation

We explore now "resource augmentation": allowing the online algorithm to deliver up to two items from each conflict, while comparing the result to the number of items delivered by the optimal offline algorithm that may deliver only one item per conflict. Much stronger bounds hold here: we present a sequential algorithm that guarantees a competitive ratio 1 when given such resource augmentation.

Specifically, we propose the following algorithm, which we call **L&R**: at each step t, deliver the leftmost and rightmost (i.e., minimal and maximal) active items from the conflict I_t.

To analyze the performance of **L&R**, fix an instance $\mathcal{C}$, and let $\text{OPT} = \{o_1, \ldots, o_\ell\}$, where $o_j \leq o_{j+1}$, for every $1 \leq j < \ell$, be any optimal solution to $\mathcal{C}$. Let O_j denote the interval $[o_j, o_{j+1} - 1]$ for $1 \leq j < \ell$. The following lemma contains our main argument.

Lemma 13. *Let P be the set of items delivered by Algorithm **L&R**. Then, for any j, we have $|O_j \cap P| \geq 1$.*

Proof. Let $1 \leq j < \ell$. Assume, for contradiction, that no item from O_j survives by Algorithm **L&R**. Since O_j is not empty and all items are initially active, there exists t such that I_t is the conflict that eliminates the last remaining active items in O_j. Let $i \in O_j$ be one of these eliminated items. By the specification of **L&R**, i can only be eliminated if there are two active items $i_1, i_2 \in I_t$ such that $i_1 < i < i_2$. Moreover, they are active after step t, while O_j then contains only inactive items, implying that $i_1 < o_j$ and $i_2 \geq o_{j+1}$. But this is impossible, because both o_j and o_{j+1} belong to a feasible solution, and hence they cannot both be members in the same conflict I_t. □

Theorem 8. *Algorithm **L&R** is* 1*-competitive.*

Proof. By Lemma 13, **L&R** delivers at least $|\text{OPT}| - 1$ items, one for each item in OPT, with the exception of the last item in OPT. Observe that Algorithm **L&R** always delivers the last item, and therefore it delivers at least OPT items. □

8 Conclusion

In this paper we have introduced the problem of scheduling with interval conflicts and proved tight bounds on the competitive ratio of online algorithms to solve them. It would be interesting to consider other conflict topologies, and to understand to which degree randomness can help.

References

[1] B. Awerbuch, Y. Bartal, A. Fiat, and A. Rosén. Competitive non-preemptive call control. In *Proc. 5th SODA*, pages 312–320, 1994.

[2] U. T. Bachmann, M. M. Halldórsson, and H. Shachnai. Online selection of intervals and t-intervals. In *Proc. 11th SWAT*, volume 6139 of *Lecture Notes in Computer Science*, pages 383–394, 2010.

[3] Y. Emek, M. M. Halldórsson, Y. Mansour, B. Patt-Shamir, J. Radhakrishnan, and D. Rawitz. Online set packing and competitive scheduling of multi-part tasks. In *Proc. 29th PODC*, pages 440–449, 2010.

[4] P. Erdős and G. Szekeres. A combinatorial problem in geometry. *Compositio Math.*, 2:463–470, 1935.

[5] M. Golumbic. *Algorithmic graph theory and perfect graphs*. Academic Press, New York, 1980.

[6] J. W. Jaromczyk, A. Pezarski, and M. Slusarek. An optimal competitive algorithm for the minimal clique covering in circular arc graphs. In *Proc. 19th EWCG*, 2003.

[7] D. Karger, C. Stein, and J. Wein. Scheduling algorithms. In M. J. Atallah, editor, *Algorithms and Theory of Computation Handbook*. CRC Press, 1998.

[8] A. Kesselman, B. Patt-Shamir, and G. Scalosub. Competitive buffer management with packet dependencies. In *Proc. 23rd IPDPS*, pages 1–12, 2009.

[9] R. Lipton and A. Tomkins. Online interval scheduling. In *Proc. 5th SODA*, pages 302–311, 1994.

[10] J. Sgall. On-line scheduling. In *Online Algorithms*, volume 1442 of *Lecture Notes in Computer Science*, pages 196–231, 1996.