Online t-Interval Scheduling

Unnar Þór Bachmann

December 19, 2009

REYKJAVÍK UNIVERSITY
HÁSKÓLINN Í REYKJAVÍK

Online t-Interval Scheduling

Unnar Þór Bachmann

Master of Science
December 19, 2009

Reykjavík University - Department of Computer Science

M.Sc. Thesis

REYKJAVÍK UNIVERSITY
HÁSKÓLINN Í REYKJAVÍK

Online t-Interval Scheduling

by

Unnar Þór Bachmann

Thesis submitted to the School of Computer Science
at Reykjavík University in partial fulfillment
of the requirements for the degree of
Master of Science

December 19, 2009

Thesis Committee:

Dr. Magnús Már Halldórsson, supervisor
Professor, Reykjavik University

Dr. Bjarni V. Halldórsson
Associate Professor, Reykjavik University

Dr. Hadas Shachnai

Online t-Interval Scheduling

by

Unnar Þór Bachmann

December 19, 2009

Abstract

This paper deals with non-preemptive online t-interval scheduling. A t-interval is a union of t half-open intervals (segments). In online scheduling the t-intervals are presented incrementally and each presented interval must be accepted or lost forever. A presented t-interval which overlaps a previously accepted t-interval cannot be accepted. The decision of whether or not to accept an interval is made without knowledge of the future.

Scheduling t-intervals has an application in bandwidth allocation, transmission of continuous-media data, linear resource allocation and genomic sequence similarity. Online scheduling is of increasing importance in a high-speed world with an uncertain future.

The most famous version of the problem is the interval scheduling problem ($t = 1$). This version has been analyzed both when it is online and offline. It has also been analyzed with weighted intervals. Scheduling t-intervals for $t > 1$ is on the other hand area covered to lesser extent.

The performance of the algorithm is the ratio between the number of t-intervals in its output vs. the optimal offline schedule. If the intervals are weighted, the performance is the ratio between the total weight of these sets. The maximum ratio, taken over all input instances, is the competitive ratio. The competitive ratio is measured with respect to different factors. One factor is the input size n. Another one is Δ, the ratio between the length of the longest and the shortest intervals.

Raðbundið t-bilaval

eftir

Unnar Þór Bachmann

September 2009

Útdráttur

Þessi ritgerð fjallar um raðbundin reiknirit fyrir t-bilaval. Sammengi hálf opinna bila, t að tölu, kallast t-bil. Í raðbundnu t-bilavali er eitt t-bil birt í senn. Á þeim tímapunkti þarf að velja bilið, ella glata því fyrir fullt og allt.

t-bilaval hefur margvíslegan hagnýtan tilgang: Úthlutun á bandvídd, gagnaflutningi, auðlindastjórnun og mynsturgreiningu genamengjum. Mikilvægi raðbundinna reiknirita er sívaxandi í heimi hraða og óvissu um framtíð.

Þekktasta útgáfan af t-bilavali er bilaval ($t = 1$). Þetta vandamál hefur verið kannað að nokkru leiti. Bæði fyrir raðbundin reiknirit og þegar bilin eru gefin fyrirfram. Einnig hefur bilaval verið kannað með vegnum bilum. Almennt t-bilaval er hins vegar mun minna kannað.

Frammistaða reiknirits er mæld með hlutfallinu milli fjölda t-bila í úttaki og stærsta mögulega bilasafnsins, þar sem engin tvö bil skarast. Nálgunarhlutfall er hámark þessa hlutfalls, reiknað yfir öll möguleg inntök. Nálgunarhlutfallið er oft metið m.t.t. tiltekinna stuðla. Dæmi um slíka stuðla er n, fjöldi bila í inntaki. Annað dæmi er Δ, hámarks hlutfall milli lengda bila í inntaki.

To myself.

Acknowledgements

First and foremost I would like to thank Dr. Magnús Már Halldórsson for helping me through the work needed to finish this thesis. Secondly, I would like to thank the staff of the Computer Science School of Reykjavik University for warm and supportive attitude. At last I would like to thank RANNIS for financial assistance during this work .

Contents

List of Figures

Chapter 1

Introduction and Principles

1.1 Introduction

Assume that you are running a *resource online*. Customers arrive and request to use it from time to time, for up to t time periods, not necessarily of same length. These requests must either be accepted or declined. If a request is accepted, then it *occupies* the resource for these periods of time. A request cannot be accepted if one or more of its periods intersect a period of a previously accepted request. Your aim is to accept as many requests as possible. How will you accomplish that? Consider the requests in Figure 1.1 arriving in the order a, b, c, d, e, f and g. The *optimal schedule* contains the requests a, f and g. Many interesting questions arise when this problem is being solved. Is it wise to accept the first request a upon arrival? If accepted, it would use the resource for two periods. The first one is short but the second one is long. The short period is not likely to hinder many future requests from being scheduled, but the long period may.

This uncertainty of future events is often characterized in an *adversary*. The adversary designs an input with the sole purpose of making your algorithm look bad in comparison to the optimal schedule. The comparison is made by calculating the ratio between the number of requests your algorithm accepts to the number of requests in the optimal schedule, which the adversary *outputs*. If an algorithm schedules only c and d, in the example in Figure 1.1, vs. 3 in the optimal schedule, the ratio is $3/2$. We call it the *performance ratio*. It is ideal for the adversary to keep the ratio as high as possible. The power of the adversary is to know your algorithm in advance. Then the adversary designs an input instance which leaves the performance ratio as high as possible. We take some other attributes of the adversary for granted, such as unlimited power of calculation. This means that it can determine the optimal schedule and determine the performance ratio in advance.

Since the adversary knows the structure of your algorithm before it designs the input instance, it is easy to see that a *deterministic algorithm* is not good. The adversary will first consider a request which uses the resource for a “very long” time period. If it is not accepted by your algorithm, then no other interval will be in the input instance. In this case, the performance ratio is as bad as possible since the optimal schedule contains one request but your algorithm contains none. If on the other hand, your algorithm accepts the first interval, a sequence of requests would be added to the input, requesting to use a fraction of the “very long” time period. These requests could not be accepted by your algorithm.

An *oblivious* adversary knows the structure of the algorithm but not the outcome of its randomized decisions. For such an adversary, a *randomized algorithm* can hide the outcome of its choices. Randomized algorithms do therefore make the input instance design harder for an oblivious adversary. In this paper, we will always assume that the adversary is oblivious. Other common types of adversaries are the *adaptive offline adversary* and *adaptive online adversary*. The adaptive offline adversary is so powerful that it knows the outcome of random decisions beforehand. Therfore, the input design is no harder for adaptive offline if the algorithm is randomized. The adaptive online adversary has the ability to foresee the outcome of random decisions. On the other hand, the adversary is not allowed to output the optimal schedule. Instead, it has to output a schedule made online, without knowing future events.

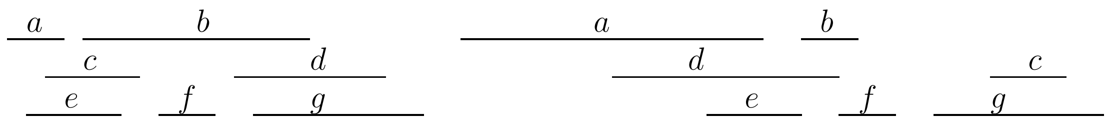


Figure 1.1: The resource is requested by customer a, b, c, d, e, f and g for two periods each. These requests arrive online in the same order. The request a arrives first and before b arrives we must decide whether to accept a or not and so on. If we decide to accept a the requests c, d and e cannot be accepted. The optimal schedule is therefore a, f and g.

1.2 Online t-Interval Scheduling

The problem described in the last section is the *online t-Interval Scheduling Problem* ($tISP$), which is the subject of this paper. The requests are jobs or *t-intervals*. No *preemption is allowed*. This means that once a t-interval is scheduled, it cannot be ejected to free the resource. The version of the problem with $t = 1$, *Interval Scheduling Problem* (ISP), is probably the most analyzed. If the t-intervals are allowed to have *weight*, then these problems become Weighted t-Interval Scheduling Problem ($tWISP$) and $WISP$ when $t = 1$. The aim in the non-weighted version, is to schedule as many disjoint t-intervals as possible. In the weighted version the aim is to output the set of disjoint t-intervals of maximum weight. All the t-intervals are jobs carried out on a single *resource* or *machine*. This means that no two overlapping t-intervals can be scheduled. A related problem is the *Interval Partitioning Problem*. In this problem, multiple (usually identical) resources can carry out each job. The aim in this problem is to use as few machines as possible.

Scheduling of t-intervals can be viewed by graph theory as an independent set problem. Each t-interval is a vertex and two intervals do intersect if and only if there is an edge between them. Such an intersection graph is called a *t-interval graph* or *interval graph* if $t = 1$. Interval graphs are a subclass of *chordal graphs*. Chordal graphs are the largest class of intersection graph which can be colored in a greedy fashion. The reason for this is that chordal graphs have a *simplicial elimination ordering*. The greedy algorithm *Simplical Coloring* colors an interval graph optimally using this order (Agnarsson & Greenlaw, 2007). A greedy algorithm (Kleinberg & Tardos, 2005) finds a maximum independent set in an interval graph optimally by using it. It starts by scheduling the interval with the smallest finishing time and rejects every interval that intersects it. Then it continues until no interval is left. *Dynamic programming* can be used to solve the $WISP$ (Kleinberg & Tardos, 2005).

Finding a maximum independent set in a t-interval graph is on the other hand more challenging. It has been shown that finding maximum independent set in a t-interval graph is NP-hard, even when $t = 2$ (Bafna, Narayanan, & Ravi, 1995). Bar-Yehuda et al. (Bar-Yehuda, Halldórsson, Naor, Shachnai, & Shapira, 2002) showed that the problem is APX-hard, even when $t = 2$. Subversions of the problem are hard as well. An example is *Strip graphs* (Crama & Spieksma, 2008)(Keil, 1992)(Karlsson, 2005). Each vertex in a strip graph is represented by an interval and a unit interval representing an equivalence class. The intervals representing the equivalence classes are all disjoint from the intervals representing the requests.

Online $tISP$ is a harder problem because the t-interval graph is introduced in steps.

1.3 Applications and Previous Results

The scheduling of t-intervals and intervals have numerous applications in the real world.

Crew scheduling is the problem of assigning flight crews to flights where each flight has a start time, end time and duration. The aim of the calculation is to find the minimum number of flight crews needed for fixed amount of flights. Each flight is described by an interval and each crew by a machine. The problem is to minimize the number of crews/machines needed.

Bandwith allocation (Bar-Noy, Canetti, Kutten, Mansour, & Schieber, 1995) in a network is another application. Number of users communicate via network with a limited bandwidth. Each communication request can be thought of as an interval requiring a certain amount of bandwidth (demand). These communications can have different priorities. The competitiveness of the system is with respect to *throughput* or the number of successful requests. Here, the online version is particularly importan. Preemption usually improves the competitive ratio.

Scheduling applications of split intervals are considered in (Bar-Yehuda et al., 2002). Scheduling of continuous-media data occurs where multimedia servers broadcast streams of data to clients upon request. Requests from the clients can be modeled as t-intervals, since requests can be split into views and breaks. Another application considered in that paper is linear resource allocation. An example of a linear resource is disk drive with contiguous blocks.

In (Goldman, Parwatikar, & Suri, 2000) the importance of online interval scheduling for high-speed networks is highlighted. An example of this is a packet-switched network such as the Internet. Streams are passed to nodes/routers which forward them onward. In order to do so, they need a scheduling algorithm. A suitable algorithm can be an interval scheduling algorithm, since each stream can be thought of as an interval.

Other applications are pattern matching over a set of 2-intervals discussed by Vialette (Vialette, 2004). The problems are of two kinds. The first problem is to find a given 2-interval pattern and the second is to find the longest 2-interval from a given graph. This problem arises in molecular biology where a given RNA secondary structure has to be found in a database.

Bar-Yehuda et al. (Bar-Yehuda et al., 2002) came up with a $2t$-approximation for offline $tWISP$ using the *Local Ratio Technique*, a method similar to dynamic programming and introduced in (Bar-Yehuda & Even, 1985) to solve the *Weighted Vertex Cover Problem*. Similarities between Local Ratio Technique and *Primal-Dual Schema* are the issue of (Bar-Yehuda & Rawitz, 2001). The paper also gives a 2-approximation algorithm using the Primal-Dual Method for $WISP$ with intervals of even length.

The Primal-Dual Schema is a linear programming method. Most combinatorial optimization problems can be formulated as an integer programming problem. Solving a linear programming relaxation of such a problem does not lead to a feasible solution. In the Primal-Dual Schema the dual complementary slackness conditions are relaxed. The solution becomes a feasible approximation solution to the original problem. Using Primal-Dual Schema, Bar-Yehuda et al. (Bar-Yehuda & Rawitz, 2006) gave a $6t$-approximation algorithm for offline weighted t-interval scheduling. Each interval was allowed to have a demand $d_i \in [0, 1]$. This means that intervals are allowed to overlap if the sum of their demands is not greater than one.

1.4 Online Scheduling

The traditional assumption that inputs of algorithms are known in advance is not always realistic. An example of this is *scheduling*. In a global economy, the demand is ever changing and it is vital to remain adaptive to changes in demand. An area where this is relevant is in networks, The most important being the Internet. *Caching* of web documents is an important problem. Web clients store actively accessed web data in caches. This is practical in order to have less stress on the network. Since the network traffic is not known in advance, the problem is online.

A related problem to this is the *paging problem*. The paging problem is a problem of two storage units. One is fast but limited, the other is slow but vast. The goal here is to keep as many actively requested pages in the fast memory. An algorithm must decide online which page to evict from the fast memory. A common deterministic algorithm used to solve this problem is LRU. This algorithm evicts the page whose most recent access was earliest. This algorithm is k-competitive where k is the size of the fast memory. A more complicated randomized algorithm is $\mathcal{H}(k)$-competitive. Here $\mathcal{H}(k) = 1 + \frac{1}{2} + \cdots + \frac{1}{k}$. A good survey of this and other online problems is in the book *Approximation Algorithms for NP-hard Problems* (Hocbaum, 1997).

A closely related online problem to interval scheduling and interval partitioning is *scheduling on parallel machines* and *load balancing*. In scheduling on parallel machines the online scheduling is as follows: there are m machines, given and jobs arrive online. These jobs are scheduled in order to optimize certain objective function. A common objective function is the *makespan*. This is the time it takes to finish all jobs. For the most basic version of the problem no preemption of intervals is allowed and all the machines are identical. A famous greedy algorithm (Graham, 1969) is $\left(2 - \frac{1}{m}\right)$-competitive. The algorithm is very simple. It assigns each job to the least loaded machine. A lot of work has since on both this version and others. In load balancing, the machines are m as well but each job is weighted. The goal becomes to minimize the load on single machine from each set of job requests.

Buchbinder and Naor (Buchbinder & Naor, 2009) use the Primal-Dual Schema to solve online *covering* and *packing* problems. Given a set S to be covered with n known subsets of S, where each set has a weight. The set cover problem is the problem of finding a union of minimum weight where each element of S is covered. The set packing problem is the problem of finding the maximum weighted union of sets which do not intersect. For the covering problem there is a variable x_i for each set. The weight of each set, c_i is known in advance, and the objective function is therefore $\sum_{i=1}^{n} c_i \cdot x_i$. The constraints $\sum_{i=1}^{n} a(i,j) \cdot x_i \geq 1$, represent the elements of S which need to be covered. They are presented one by one. For the covering problem, the competitive ratio of B can by obtained by violating the constraints by a factor no more than $1/B$. They also showed that $O(\log n)$-competitive algorithm exists for the fractional version of the problem, which does not violate any constraints. A good overview of the influence of the Primal-Dual method for online scheduling is in the paper (Chrobak, 2007).

In the paper (Goldman et al., 2000) nonpreemptive online interval scheduling is considered. Each interval is allowed to have a delay and the weight of each interval is its length. In this problem each start point can be delayed within certain certain limits. The paper considers both unit lengths intervals, intervals with two possible lengths and intervals of various length. In the case of unit length, the paper shows that the lower bound of any deterministic algorithm is at least 2 and $4/3$ for a randomized one if the delay is arbitrary. With intervals of two lengths they give a 4-competitive algorithm for the same delay restrictions.

Another problem is *Job Scheduling*, where each job has a starting time, deadline and a fixed processing time p. Each job can be viewed as an interval of length p which has to be scheduled within a given frame. This problem is considered in (Chrobak, Jawor, Sgall, & Tichy´, 2007). They give a randomized $5/3$-competitive algorithm. This algorithm is *barely random*, meaning that that it uses two deterministic algorithms and chooses between them with probability factor of $1/2$. They do also give a lower bound for barely random algorithms of $3/2$.

One version of online $WISP$ is where the intervals are of equal length and preemption is allowed. The algorithm $Greedy_2$, which schedules each interval greedily (first fit) and preempts a job if another job with double its weight arrives, was proved to be 4-competitive by Woeginger (Woeginger, 1994). This proof was based on the so-called *doubling method*. Furthermore, he proved that this constant was optimal for a deterministic algorithm.

When intervals in $WISP$ are allowed to have variable lengths, the ratio Δ, between the shortest and the longest interval becomes significant. Allowing the algorithm to preempt, Canetti and Irani (Canetti & Irani, 1995) found a lower bound of $\Omega(\sqrt{\log\Delta/\log\log\Delta})$ and upper bound of $O(\log\Delta)$ for randomized algorithms.

Lipton and Tomkins (Lipton & Tomkins, 1994) studied an algorithm where the weight of each interval is its length. Their problem was non-preemptive and the intervals are introduced at their start time. They gave a strongly 2-competitive algorithm when the intervals are of same length. They gave a randomized algorithm which is strongly $O((\log\Delta)^{1+\epsilon})$-competitive when the intervals are allowed to be of various lengths. Their algorithm did not know Δ in advance. They introduced a method which we call the *Bucket Method*.

1.5 Our Work

In Chapter 2 we will deal with ISP. We start with introducing a so-called t-interval stacking construction used to derive lower bounds for $tISP$. Next we demonstrate that every deterministic or randomized algorithm has a competitive ratio of at least 2, when the intervals are of unit length. Furthermore, we relate the lower bound to the maximum clique size of the intersection graph, τ. We show that every algorithm has a competitive ratio of at least $2-\dfrac{1}{\tau}$. In the case when $\tau = 1$ we give a strongly 1.5-competitive algorithm. We give a strongly 4-competitive algorithm for ISP when intervals are allowed to be of two lengths, 1 and d. In terms of input size n, we show that every algorithm has a competitive ratio at least $\Omega(n)$. Another important factor is, Δ, the ratio between the length of the longest and the shortest interval. We show that every deterministic algorithm has a competitive ratio at least $\Omega(\Delta)$. In the randomized version we show that every algorithm has a competitive ratio at least $\Omega(\log \Delta)$. We analyze a related problem to online ISP with unit intervals, the online *Prefix Interval Scheduling Problem*. For this problem we give a strongly 1.5-competitive algorithm.

In Chapter 3 we analyze the case of $t = 2$, for unit segments, we show that every randomized algorithm has a competitive ratio of at least 3. When the segments are allowed to be of two lengths, 1 and d, we demonstrate a randomized lower bound of 6 and a randomized upper bound of 8. In terms of the factor Δ, we give a $O(\log^2 \Delta)$-competitive algorithm when Δ is known in advance. For the case when Δ is not known, we give a $O(\log^{2+\epsilon} \Delta)$-competitive algorithm

In Chapter 4 we show that for $tISP$ with unit intervals every randomized algorithm has a competitive ratio of $\Omega(t)$.

1.6 Definitions

If $s, f \in \mathbb{R}$ and $s < f$, then a *half-open interval* is the set $[s, f) = \{x \in \mathbb{R} \mid s \leq x < f\}$. For $I = [s, f)$, s is the *start time* of I and f the *finish time*. In this paper, we do not distinguish between half-open intervals and *intervals*. Two intervals, I and J, are *disjoint* if $I \cap J = \emptyset$ and they *intersect* or *overlap* if $I \cap J \neq \emptyset$.

A *t-interval* is a union of t half-open intervals. Each interval is a *segment*. Two t-intervals, I and J, are disjoint if none of their segments intersect. They do intersect if at least one segment of I intersects one or more segments of J or vice versa. A subset of a t-interval is called a *t-sub-interval*. If $t = 1$ the subset is called a *sub-interval*. If the segments of a t-interval I are i_1, ..., i_t then $I = (i_1, \ldots, i_t)$.

A t-interval is *scheduled* on a *machine* or a *resource* if each of its segments occupies the resource from its start time to its finish time. Once a t-interval is scheduled, no preemption is allowed. A t-interval is *blocked* if it cannot be scheduled. If one or more segments of a t-interval overlap a segment of a scheduled t-interval, then it is blocked. If a t-interval is not scheduled, then it is lost forever. A t-interval is *virtually scheduled* if it does not occupy the resource, but blocks other intersecting t-intervals from being scheduled. A scheduled t-interval that occupies the resource is scheduled *directly*. In online scheduling *the resource is free* if a presented t-interval does overlap a scheduled t-interval. Otherwise, *the resource is in use*.

A *t-Interval Scheduling Problem* ($tISP$) is the problem of scheduling as many disjoint t-intervals from a given set of t-intervals on a single resource. The *Weighted t-Interval*

Scheduling Problem ($tWISP$) is the problem of scheduling a set of t-intervals of maximum weight from a given set of t-intervals with weight. Note that the $tISP$ is a special case of $tWISP$, were the weight of all t-intervals is equal. We assume that $tISP$ is $tWISP$ where each t-interval has the weight 1. If $t = 1$, we call the former problem *Interval Scheduling Problem* (ISP) and the latter one *Weighted Interval Scheduling Problem* ($WISP$). We also distinguish between *offline scheduling* and *online scheduling*. In offline scheduling, the set of t-intervals are given in advance. In online scheduling, the t-intervals are presented one by one. The ratio between the longest and the shortest segment of any t-interval in a given problem is denoted by Δ.

A *t-interval graph* is an intersection graph where each vertex is represented by one and only one t-interval. Two vertices are adjacent if and only if the t-intervals they represent intersect. A *t-union graph* is an edgewise union of t interval graphs over the same vertex set, whereby they form a subclass of t-interval graphs. $tISP$ and $tWISP$ are equal to maximum independent set and weighted maximum independent set problems in a t-interval graph. We say that an instance of $tISP$ *induces* the intersection graph G. If the max clique size of the intersection graph is $\tau = \chi(G)$, then we say that the problem has the *depth* of τ.

The weight of a t-interval I is denoted by $w(I)$, while a problem instance is denoted by S. The set of all problem instances is denoted by $\mathcal{S}$. We say that $\sigma \subseteq S$ is a *feasible schedule* or an *output* if no t-intervals of σ intersect. A feasible schedule, σ, has a weight $w(\sigma) = \sum_{I \in \sigma} w(I)$. A feasible schedule, which is an output of algorithm *A*, is denoted by σ_A or $A(S)$. If *R* is a randomized algorithm, then σ_R is a random variable of $\mathcal{F}(S)$, the set of all feasible schedules. The expected weight of $I \in S$ is $E_R[w(I)] = w(I) \cdot \Pr[I \in \sigma_R]$. The expected weight of the output of *R* is $E[\sigma_R] = \sum_{I \in S} w(I) \cdot \Pr[I \in \sigma_R]$. The feasible set of maximum weight is called the *optimal schedule*, denoted by $\sigma_\star$ or *OPT(S)* . If the t-intervals are nonweighted we can assume that each t-interval has the weight 1. The weight of a feasible schedule, σ, is in this case the number of intervals in it. Instead of using $w(\sigma)$ we use $|\sigma|$.

If p is a distribution over $\mathcal{S}$, then S_p is a *random input instance*. In this case, the expected weight of an output is calculated with respect to p and the algorithm itself, both if the algorithm is randomized or deterministic. The expected weight of the output or the expected number of t-intervals in the output is calculated both with respect to p and *A* , $E_p[A(S)]$. In the case of a random input instance S_p, the weight of $\sigma_\star$ becomes the expected weight calculated with respect to the probability distribution p. If there is a positive probability that I is an instance from S_p, then $I \in S_p$. Furthermore, if there is a positive probability that σ is a feasible schedule of S, then $\sigma \subseteq S_p$.

If $I \in \sigma$ and $J \in \sigma_\star$, it can be useful to *assign* a part of $w(I)$ to J if I intersects J. Let $w(I, J)$ denote the weight of I that is assigned to J. The expected weight of the fraction of I distributed to J is then $w(I, J)$ times the probability that I is scheduled. For a given input instance S, the assignment of weights is *proper* if $\sum_{J \in \sigma_\star} w(I, J) \leq w(I)$ for all $I \in S$.

The performance ratio of an algorithm in this paper is measured against an *oblivious adversary*. Such an adversary recognizes the structure of algorithm but not the outcome of its random decisions. The purpose of the adversary is to design as difficult input for the algorithm as possible, difficult in the sense that the ratio between the optimal solution and the expected weight of the algorithm is maximized. We say that the adversary *outputs* the optimal schedule or that the optimal schedule is the adversary's schedule.

If p is a probability distribution over all input instances of $tISP$ and A is an online algorithm, then A is ρ-competitive against if for all input instances S, $E[A(S)] \geq \frac{1}{c} \cdot |\sigma_\star|$. The competitive ratio of A is ρ if for input instances S, $\rho \leq \sup\{c \,|E\,[A(S)] \geq \frac{1}{c} \cdot |\sigma_\star|\}$. An algorithm is *strongly ρ-competitive* if it receives the best possible competitive ratio for the problem. The expected weight is calculated with respect to both (random) decisions of A and the distribution of the input instance.

Prefix Online Interval Scheduling Problem ($PISP$) is an online interval scheduling problem where the input is known in advance by the online algorithm A. The input consists of intervals I_1, I_2, ... , I_n presented in this order. After presenting I_j, where $j \in \{1, 2, ..., n\}$, the adversary can decide to stop the presentation at a *terminal point* j. After stopping the presentation, it outputs the largest set of disjoint intervals from $C = \{I_1, ..., I_j\}$ vs. the intervals scheduled online by A from C. The competitive ratio for this problem is not only calculated over all input instances, but all subinstances induced by different terminal points.

1.7 The Bucket Method

In this section we introduce a method for proving upper bounds for $tISP$. Recall that $tISP$ is a special version of $tWISP$ where each t-interval I has the weight of $w(I) = 1$. We call it the *Bucket Method* (Lipton & Tomkins, 1994). By using this method, an algorithm is proven to be ρ-competitive. This is done by proving that for each $J \in \sigma_\star$, the algorithm receives an expected weight of $1/\rho \cdot w(J)$.

The method involves two steps:

1. The weight of each t-interval in the problem instance is assigned "properly" onto the t-intervals in $\sigma_\star$.
2. Show that for each $J \in \sigma_\star$ the expected weight of J and the weights assigned to J, is at least $1/c \cdot w(J)$.

For a given online algorithm A , the weight of the optimal schedule is compared, piece by piece, to the expected weight of σ_A. These results hold equally for randomized input instances. This is because of linearity of expectation and the independence of the random decisions of A from the distribution of the input instances.

The fundamental elements of the Bucket Method are the buckets and a proper distribution of weights into each bucket. For each $J \in \sigma_\star$, there is one and only one bucket. Into that bucket we put the weight of J along with a fraction of weights assigned to it from other intervals in S. Recall that the weight of $I \in S$ assigned to $J \in \sigma_\star$ is $w(I, J)$.

Definition 1.1: If S is an instance of an online $tISP$, then for each $J \in \sigma_\star$ we define *bucket(J)* of J as the set:

$$\{w(J)\} \cup \{w(I, J)\,|I \in S\backslash\sigma_\star \wedge I \cap J \neq \emptyset\,\}.$$

The weight of an interval I is assigned properly if $\sum_{J\in\sigma_\star} w(I, J) \leq w(I)$.

Definition 1.2: Let S be an instance of an online $tISP$ and $I \in S$. The weight of $I \in S$ is assigned *properly* if $\sum_{J\in\sigma_\star} w(I, J) \leq w(I)$.

For the rest of this section, we demonstrate how the buckets are used to find lower bound of the expected weight of σ_A. As a consequence of linearity of expectation, the expected weight of $\sigma_A \subseteq S$ is split into the expected weight of each $I \in S$. The expected weight is calculated from possible random decisions of A and the distribution of the input instance.

Theorem 1.1: If A is an online algorithm and S an instance of $tISP$, then $E[\sigma_A] = \sum_{I\in S} E_A[w(I)]$.

If $I \in S$ and $J \in \sigma_\star$ then the expected weight of $w(I, J)$ is $E_A[w(I, J)] = w(I, J) \cdot Pr[I \in \sigma_A]$. Given that the weights of all $I \in S$ are assigned properly, the expected weight of I can be lower bounded by the expected weight of the assigned weights.

$$
\begin{aligned}
E_A\left[w(I)\right] &= w(I)\cdot Pr[I\in\sigma_A] \\
&\geq \sum_{J\in\sigma_\star} w(I,J)\cdot Pr[I\in\sigma_A] = \sum_{J\in\sigma_\star} E_A\left[w(I,J)\right]. \qquad (1.1)
\end{aligned}
$$

If we denote the sum of the expected weight of all the elements of $bucket(J)$ by $E_A(w(bucket(J)))$, we get the following theorem:

Theorem 1.2: Assume the weights of all $I \in S$ are assigned properly. If A is a randomized algorithm then:

$$E[\sigma_A] \geq \sum_{J\in\sigma_\star} E_A[w(bucket(J)]. \qquad (1.2)$$

Proof. Assume that S is an instance of online $tISP$ and the weights of all $I \in S$ are assigned properly. For simplicity let $w(J,J) = w(J)$ for all $J \in \sigma_\star$. If A is a randomized algorithm, then Equation 1.1 gives us:

$$
\begin{aligned}
E[\sigma_A] &= \sum_{I\in S} E_A(w(I)) \\
&\geq \sum_{I\in S}\sum_{J\in\sigma_\star} E_A(w(I,J)) \\
&= \sum_{J\in\sigma_\star}\sum_{I\in S} E_A(w(I,J)) \\
&= \sum_{J\in\sigma_\star} E_A[w(bucket(J)].
\end{aligned}
$$

■

Instead of discussing $E_A(w(bucket(J)))$ as the expected weight of all the elements of $bucket(J)$, we simply state that it is *the expected weight of the* $bucket(J)$. Last theorem shows us that the expected weight of an algorithm for a given problem instance is lower bounded by the sum of the expected weight of each bucket. If we show that an algorithm gets a fraction of $1/c$ from each bucket for any problem instance, then the algorithm is ρ-competitive. We state this in the following corollary:

Corollary 1.1: Let A is an online algorithm and S be an instance of online $tISP$. If the weights of all $I \in S$ are assigned properly and

$$E_A[w(bucket(J))] \geq \frac{1}{\rho}\cdot w(J),$$

then A is ρ-competitive.

An online algorithm A is proven to be ρ-competitive for $tISP$ if for any instance S:

a) the weights of all $I \in S$ are assigned properly into buckets.

b) and for all $J \in \sigma_\star$

$$E_A[w(bucket(J)] \geq \frac{1}{c} \cdot w(J).$$

1.8 Yao's Minimax Theorem

Yao's Minimax Theorem (Yao, 1977) is an important theorem for proving lower bounds for randomized online algorithms. This theorem states that the expected weight of the optimal online deterministic algorithm for an arbitrary input distribution p of finite size is a lower bound on the expected weight of the optimal randomized algorithm for the ISP.

Theorem 1.3 (Yao's Minimax Theorem)**:** Let $\mathcal{D}$ be a set of deterministic algorithms and $\mathcal{S}' \subseteq \mathcal{S}$ be a finite set of input instances of $tISP$. If p is a finite probability distribution over $\mathcal{S}'$ and q a probability distribution over $\mathcal{D}$, then S_p and D_q are the random variables representing a random input instance and a randomized algorithm. For all distributions p over a finite set $\mathcal{S}' \subseteq \mathcal{S}$ and q over $\mathcal{D}$ then

$$\max_{D \in \mathcal{D}} E[\sigma_D] \leq \min_{S \in \mathcal{S}'} E[\sigma_{D_q}].$$

This theorem was derived by Andrew Yao from a more general theorem, von Neumann's Minimax Theorem. A practical version of the rule is stated in the following corollary.

Corollary 1.2: If R is a randomized algorithm with the competitive ratio of ρ_R, then ρ_R is no less than the competitive ratio of the best deterministic algorithm, performing against the worst probability distribution over finite set of input instances. More precisely, if $\mathcal{D}$ is the set of all deterministic algorithms and p a probability distribution over a finite set $\mathcal{S}' \subseteq \mathcal{S}$ then $\sup_R c_R = \inf_p \sup_{D \in \mathcal{D}} c_D$.

Proof. Assume that S_p is a random input instance, where p is a probability distribution over a finite set $\mathcal{S}' \subseteq \mathcal{S}$. Assume furthermore that $D \in \mathcal{D}$ has the competitive ratio c_D. If R is a randomized algorithm with the competitive ratio ρ_R then Theorem 1.3 gives that

$$\frac{E[\sigma_\star]}{E[\sigma_R]} \leq \min_{D \in \mathcal{D}} \frac{E[\sigma_\star]}{E[\sigma_D]}.$$

since $\max_{D \in \mathcal{D}} E[\sigma_D] \leq E_R[\sigma_R]$ by Yao's Minimax Principle. This implies that $\sup\limits_{R} c_R \leq \sup\limits_{D \in \mathcal{D}} c_D$ where *R* is from the set of all randomized algorithms. This gives us $\sup\limits_{R} c_R = \inf\limits_{p} \sup\limits_{D \in \mathcal{D}} c_D$ where p is from the set of all probability distributions over a finite set of input instances.

■

This corollary is very practical, as it implies that the competitive ratio of any randomized algorithm is lower bounded by the performance of the best deterministic algorithm on any distribution of a finite number of problem instances. Therefore, it is possible to take any distribution of finite instances and measure the best performance possible for any deterministic algorithm and use the results as a lower bound for any randomized algorithm.

Chapter 2

The Interval Scheduling Problem

In this chapter we analyze online ISP. In Section 2.1 we introduce t-interval stacking construction, which is a technique to prove randomized lower bound. It is used in Section 2.2 to demonstrate that any randomized algorithm has a competitive ratio at least two for ISP with unit lengths. We will also use this technique later when analyzing $2ISP$. We use Yao's Minimax Theorem to derive a lower bound related to the depth of the online ISP. In Section 2.3 we demonstrate that First Fit is d-competitive, if the intervals are of two lengths, 1 and d. We demonstrate as well that no deterministic algorithm does better. In the same section we come up with a strongly randomized 4-competitive algorithm for ISP with intervals of two lengths. In Section 2.4 we demonstrate that any algorithm has a competitive ratio at least $\Omega(n)$. In Section 2.5 we demonstrate a strongly $O(\log \Delta)$-competitive algorithm for ISP with intervals of various lengths. We close the chapter with Section 2.6 by giving a strongly 1.5-competitive algorithm for the Online Prefix Interval Scheduling Problem.

2.1 t-Interval Stacking Construction

When deriving lower bound for deterministic algorithms, the adversary takes advantage of the fact that it can foresee the outcome of a deterministic algorithm. An oblivous adversary can make the outcomes of a randomized algorithm predictable by "stacking" them. We will spend the rest of this section demonstrating this technique.

The idea is simple. Given a randomized algorithm *R* , either *R* must schedule a single interval from I_1, ..., $I_{\sqrt{d}}$ (Figure 2.1 a)), as they mutually overlap, with probability one, or it must schedule at least one interval with probability less than $1/\sqrt{d}$. Depending on the structure of *R* , the adversary will either design the input according to Figure 2.1 a) or b).

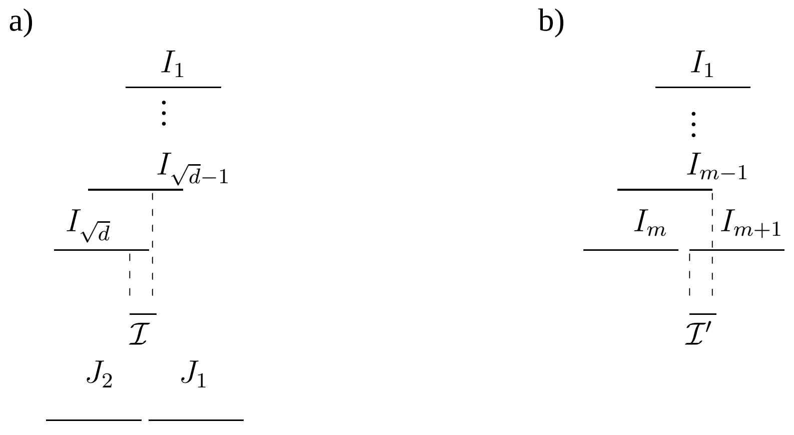


Figure 2.1: a) Input instance designed by the adversary in the proof Theorem 2.1 if *R* schedules intervals from $I_1, ..., I_{\sqrt{d}}$ "too greedily". b) How the adversary alters the input instance in the proof of Theorem 2.1 if some interval $I_1, ..., I_{\sqrt{d}}$ is scheduled with "diminishing" probability.

Let *R* be a randomized algorithm and I_1, I_2, ..., I_m be set of mutually overlapping intervals, presented in this order as shown in Figure 2.1. b). The *conditional probability* of scheduling I_i is the probability of scheduling I_i given that no overlapping interval has been scheduled. We denote this probability with p_i. On the other hand, the *unconditional probability* of scheduling I_i is $q_i = \prod_{j=1}^{i-1}(1 - p_j) \cdot p_i$.

A *interval d-stacking construction* is an input, such that the expected number of intervals in σ_R is $1 + 1/\sqrt{d}$ vs. two in the optimal schedule.

If instead of intervals we have t-intervals with segments of same length, then this input is called a *t-interval d-stacking construction*.

Let $I_1, I_2, ..., I_m$ ($m \in \mathbb{N}$), as in Figure 2.1 a), be a set of mutually overlapping intervals of length $l \in \mathbb{R}$, presented in order of decreasing left endpoints. The interval $\mathcal{I} = \bigcap_{i=1}^{m} I_i$ is a *left-oriented intersection interval*. If $I_m = [s_m, f_m)$ and $I_{m+1} = [f_m, f_m + l)$ then $\mathcal{I}' = \bigcap_{i=1}^{m-1} I_i \cap I_{m+1}$ (Figure 2.1 b)) is a *right-oriented intersection interval*. When referring to either of the two, we talk about *intersection interval*.

We can broaden the definition of the last paragraph, if $I_1, ..., I_m$ are t-intervals, with segments of length s, and the intersection graph is a t-union graph. If the segments of I_i are

$\alpha_1^i, ..., \alpha_t^i$ then $\mathcal{I} = \bigcap_{i=1}^{m} I_i$ is a *left-oriented intersection t-interval* if $\forall j \in \{1, ..., t\} : \bigcap_{i=1}^{m} \alpha_j^i$ is a left-oriented intersection interval. Furthermore, if $I_m = \bigcup_{i=1}^{t} [s_i^m, f_i^m)$, $I_{m+1} = \bigcup_{i=1}^{t} [f_i^m, f_i^m + l)$, and $\forall i \in \{1, ..., t\} : \bigcap_{i=1}^{m-1} \alpha_i \cap [f_i^m, f_i^m + l)$ is a right-oriented intersection interval, then $\mathcal{I}' = \bigcap_{i=1}^{m-1} I_i \cap I_{m+1}$ is a *right-oriented t-interval.*

Theorem 2.1: Let d be a square number. For any randomized algorithm *R* , there exists an interval d-stacking construction. Furthermore, the left-oriented intersection interval $\mathcal{I}$ and the right-oriented intersection interval $\mathcal{I}'$ as defined in the proof of this theorem are of length $\sqrt{d}$.

Proof. First let $I_1, I_2, I_3, \cdots, I_{\sqrt{d}}$ be intervals as in Figure 2.1 a), namely, $I_i = [s_i, f_i)$ for $1 \leq i \leq \sqrt{d}$, where $s_i = d - (i-1) \cdot \sqrt{d}$ and $f_i = 2d - (i-1) \cdot \sqrt{d}$ for $1 \leq i \leq \sqrt{d}$. Furthermore, let p_i denote the conditional probability of *R* scheduling I_i and q_i the unconditional probability, assuming that the intervals are presented in the order $1, 2, ..., \sqrt{d}$.

The adversary alters the input design, depending on the structure of *R* . Either each interval from $I_1, I_2, I_3, \cdots, I_{\sqrt{d}}$ is scheduled “too greedily” or at least one of them is scheduled with “diminishing probability” . By scheduling “too greedily” we mean that each interval is scheduled with unconditional probability of at least $1/\sqrt{d}$. By “diminishing” probability we mean that at least one interval is scheduled with unconditional probability less than $1/\sqrt{d}$.

“Too greedily”: If *R* schedules each interval from $I_1, \ldots, I_{\sqrt{d}}$ with unconditional probability of at least $1/\sqrt{d}$, then the probability of scheduling some interval I_i ,$1 \leq i \leq \sqrt{d}$, is one, since $\sum_{i=1}^{\sqrt{d}} q_i = 1$.

Setting $m = \sqrt{d}$, the left-oriented intersection interval

$$\mathcal{I} := \bigcap_{i=1}^{\sqrt{d}} I_i = [s_1, f_{\sqrt{d}}),$$

is of length $f_{\sqrt{d}} - s_1 = 2d - (\sqrt{d} - 1) \cdot \sqrt{d} - d = \sqrt{d}$.

In this case, the adversary designs the input instance such that two disjoint intervals, J_1 and J_2, are presented after $I_1, \ldots, I_{\sqrt{d}}$ (Figure 2.1 a)), both overlapping $[s_1, f_{\sqrt{d}})$. Since *R* schedules intervals $I_1, \ldots, I_{\sqrt{d}}$ with probability 1, there is zero probability of scheduling J_1 and J_2. The expected number of intervals in σ_R is in this case, 1 vs. 2 in the optimal schedule.

"Diminishing Probability": In the second case, there exists $j \leq \sqrt{d}$, such that I_j is scheduled with unconditional probability less than $1/\sqrt{d}$. Let m be the first such interval in the order $1, ..., \sqrt{d}$. In this case, the adversary takes the input design into different direction. He presents $I_{m+1} = [f_m, f_m + d)$ after I_m as in Figure 2.1 b) and no more intervals. The expected number intervals in σ_R becomes a sum of three terms: The probability of scheduling some interval $I_1, ..., I_{m-1}$, the probability of scheduling I_m and the probability of scheduling I_{m+1}. If p' is the probability of R scheduling some interval I_1, ..., I_m, the expected number of intervals in σ_R is $p' + (1-p') \cdot p_{m+1} + 1/\sqrt{d} \leq 1 + 1/\sqrt{d}$, where p_{m+1} is the conditional probability of scheduling I_{m+1}. The number of intervals in the optimal schedule is, on the other hand, 2, given by I_m and I_{m+1}. The right-oriented intersection interval

$$\mathcal{I}' := \bigcap_{i=1}^{m-1} I_i \cap I_{m+1} = [f_m, f_{m-1})$$

is of length $f_{m-1} - f_m = 2d - ((m-1)-1) \cdot \sqrt{d} - (2d - (m-1) \cdot \sqrt{d}) = \sqrt{d}$.

■

It is interesting to look at how Theorem 2.1 changes if the interval $I_1 = [d, d + \sqrt{d})$ is of length $\sqrt{d}$ but every other interval is of length d in the proof. If we assume that d is a fourth power of an integer, then the intervals $I_1, I_2, ..., I_{\sqrt[4]{d}}$ are altered, such that $I_i = [s_i, f_i) = [\sqrt{d} - (i-1) \cdot \sqrt[4]{d}, d + \sqrt{d} - (i-1) \cdot \sqrt[4]{d})$ for all $2 \leq i \leq \sqrt{d}$. The left-oriented intersection interval

$$\mathcal{I} := \bigcap_{i=1}^{\sqrt[4]{d}} I_i = [s_1, f_{\sqrt[4]{d}}) \tag{2.1}$$

becomes an interval of length $f_{\sqrt[4]{d}} - s_1 = d + \sqrt{d} - (\sqrt[4]{d} - 1) \cdot \sqrt[4]{d} - d = \sqrt[4]{d}$. If m in Theorem 2.1 is refined such that if I_m is the first interval in $I_1, I_2, ..., I_{\sqrt[4]{d}}$, scheduled with unconditional probability less than $1 + 1/\sqrt[4]{d}$ (if the structure of R is such) then $I_{m+1} = [f_m, f_m + \sqrt{d})$ and the right-oriented intersection interval

$$\mathcal{I}' := \bigcap_{i=1}^{m-1} I_i \cap I_{m+1} = [f_m, f_{m-1}) \tag{2.2}$$

is an interval of length

$$f_{m-1} - f_m = d + \sqrt{d} - (m-1-1) \cdot \sqrt[4]{d} - (d + \sqrt{d} - (m-1) \cdot \sqrt[4]{d}) = \sqrt[4]{d}.$$

We state this as a corollary.

Corollary 2.1: Let d be a fourth power of an integer. For a given ISP with intervals of length d, except for the first interval presented of length $\sqrt{d}$, there exists an interval $\sqrt{d}$-stacking construction. Furthermore, the intervals $\mathcal{I}$ and $\mathcal{I}'$ as defined in Equations 2.1 and 2.2 are of length $\sqrt[4]{d}$.

Theorem 2.1 and Corollary 2.1 can be extended for $tISP$.

Theorem 2.2: Assume that online $tISP$ with segments of square length d is given as well as a randomized algorithm *R* . Then there exists a t-interval d-stacking construction. Furthermore, the left-oriented intersection t-interval $\mathcal{I}$ and the right-oriented intersection t-interval $\mathcal{I}'$ have segments of length $\sqrt{d}$.

If we view d in Theorem 2.1 of this chapter as a relative length, then for all input instances, where the intervals are of same length, and any randomized online algorithm *R* , there exists an input instance, such that the expected number of intervals in σ_R is $1 + 1/\sqrt{d}$ vs. 2 in the optimal schedule. The adversary needs no more than an area of length three to construct this input, since it uses an area of length no more than $3d$ in the proof of Theorem 2.1. In other words, for a given number $0 < \delta$, there exists an input instance, such that the performance ratio is at least $\frac{2}{1+\delta}$.

2.2 ISP with Unit Intervals

This section deals with ISP where the intervals are of unit length. We prove that no randomized algorithm has competitive ratio less than 2. In terms of depth, these lower bound are $2 - \frac{1}{\tau}$, where τ is the depth.

2.2.1 Randomized Lower Bounds

By viewing the length d, from last section, as a relative length we get the following theorem (Yao, 1977).

Theorem 2.3: If *R* is a randomized algorithm for ISP with intervals of same length, then $\rho_R \geq 2$.

Proof. Assume that *R* is a randomized algorithm for online ISP with intervals of same length and d a square number. By Theorem 2.1 there exists an input instance, such that the expected number of intervals in *R* is $1 + 1/\sqrt{d}$ vs. 2 from the optimal schedule. Then

$$\rho_R \geq \sup_d \frac{2}{1 + \frac{1}{\sqrt{d}}} = 2$$

■

Similar result can be obtained with Yao's Minimax Theorem. The results are somewhat stronger because they relate the lower bound directly to the clique size (depth) of the interval graph induced by the ISP instance.

Theorem 2.4: If τ is the depth of ISP with unit intervals and *R* is an online randomized algorithm, then $\rho_R \geq 2 - \frac{1}{\tau}$.

Proof. Consider m instances of ISP with depth of m as in Figure 2.1 b). For the first instance $m = 1$, then $m = 2$ and so on. The first instance consists only of I_1, the next one of I_1, I_2 and I_3 (I_2 and I_3 being disjoint), then I_1, I_2, I_3 and I_4 (I_3 and I_4 being disjoint), ... and the last one of I_1, I_2, ..., I_m, I_{m+1}. Assume that each instance is presented with probability $1/m$. The expected number of intervals in the schedule which the adversary outputs is $\dfrac{1}{m} + \displaystyle\sum_{j=2}^{m} \frac{1}{m} \cdot 2 = 2 - \frac{1}{m}$.

A deterministic algorithm, D_1 , which schedules I_1, outputs a schedule with $E[\sigma_{D_1}] = m \cdot 1/m = 1$.

A deterministic algorithm, D_2 , which schedules one of the interval I_k where $2 \leq k \leq m - 1$, outputs a schedule with expected number of intervals at most

$$\sum_{i=2}^{k-1} \frac{1}{m} + \frac{2}{m} + \sum_{i=k+1}^{m} \frac{1}{m} = 1\ .$$

This is because D_2 schedules at most two intervals when $m = k$ but otherwise at most one.

A deterministic algorithm, D_3 , which schedules I_m and rejects all other intervals in the clique $I_1, \ldots, I_{m-1}$, outputs a schedule with expected number of intervals at most $\dfrac{1}{m} \cdot (m - 2) + \dfrac{2}{m} = 1$.

The competitive ratio of any randomized algorithm is therefore at least $2 - \dfrac{1}{m}$. ■

The theorem above gives us in particular a lower bound for an ISP is 1.5 if the depth is two and $1\frac{2}{3}$ if the depth is three.

2.2.2 Randomized Upper Bound when Depth is Two.

Algorithm 2.1: Assume an online ISP with maximum depth of two is given. A randomized algorithm is defined as follows.

1. If the interval does not overlap any previously presented interval, schedule it with probability $2/3$ and with probability $1/3$ not.
2. Schedule interval greedily if it overlaps a previously presented interval.

Algorithm 2.1 splits intervals into four groups, 1, 2, 3 and 4. We denote the number of interval in group i by n_i. Because the depth is two and each interval is a unit interval, then no interval can overlap more than two intervals. A presented interval which does not overlap a previously presented interval is scheduled with probability $2/3$. Let these intervals be in group 1 as well as every other interval scheduled with probability $2/3$, by Algorithm 2.1.

If a presented interval intersects a previously presented interval, then it can overlap one or two previously presented intervals. Presented intervals which intersect only one previously presented interval are either scheduled with probability $2/3$ or $1/3$, as in Figure 2.2. Some of them are therefore in group 1. Let the ones that are scheduled with probability $1/3$ be in group 2.

When other intervals are presented, they intersect two previously presented intervals. Therefore, they must overlap intervals from either group 1 or group 2. Intervals which overlap two previously presented intervals from group 2 are in group 3. They are scheduled with probability $2/3 \cdot 2/3 = 4/9$.

Intervals which intersect two previously presented intervals but are not in group 3 are in group 4. Since the depth is two, no interval in group 4 can intersect interval from group 3 or 4. By definition of group 4, an interval in group 4 must either intersect two intervals from group 1 or one from each of groups 1 and 2. This means that they are either scheduled with probability, $1/3 \cdot 1/3$ or $1/3 \cdot 2/3$, less than $1/3$. If we order them by decreasing left endpoints, then two adjacent intervals with respect to that order, are seperated by chains of intervals, such as in Figure 2.3. Each of these chains are seperated from the rest of the input by two intervals from group 3, presented after each interval in the chains. Because of this, the first interval presented in the chain, does not intersect any other interval, presented prior to its arrival, and is therefore in group 1. Therefore, $n_1 \geq n_4 + 1$.

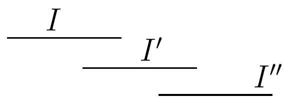


Figure 2.2: The intervals I, I' and I'' are presented in this order. This picture demonstrates how Algorithm 2.1 groups these intervals. I and I'' fall into group 1 but I' into group 2.

Theorem 2.5: Algorithm 2.1 is strongly $3/2$-competitive for online ISP with unit intervals and depth two.

Proof. Without a loss of generality, we can assume that the intersection graph is connected. This means that the input instance is a chain of unit intervals as in Figure 2.3. If n is the number of intervals, the number of intervals in the optimal schedule is at most $\left\lceil \frac{n}{2} \right\rceil \leq \frac{n+1}{2}$. We group the intervals in the input instance according to the probability by which Algorithm 2.1 schedules them, according to the remarks above. Let the number of intervals in group $i \in \{1, 2, 3, 4\}$ be n_i.

The expected number of intervals in the scheduele which Algorithm 2.1 outputs is therefore at least

$$\frac{2}{3} \cdot n_1 + \frac{1}{3} \cdot n_2 + \frac{4}{9} \cdot n_3 \geq \frac{1}{3} \cdot (n_4 + 1) + \frac{1}{3} \cdot n_1 + \frac{1}{3} \cdot n_2 + \frac{4}{9} \cdot n_3 \geq \frac{n+1}{3} .$$

This proves that the competitive ratio of Algorithm 2.1 is $3/2$.

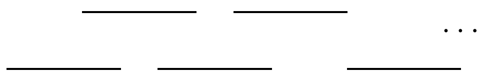


Figure 2.3: Without a loss of generality in the proof of Theorem 2.5, the intersection graph of the instance is connected.

■

2.3 ISP with Intervals of Two Lengths

2.3.1 Deterministic Upper Bounds

A deterministic algorithm is bound to have bad competitive ratio for online ISP. This is because the adversary can determine which intervals are scheduled and which not when he designs the input. Therefore we do not expect a deterministic algorithm to do better than *First-Fit*, which schedules intervals whenever the resource is free. In Theorem 2.7 we will prove this.

Theorem 2.6: First Fit is $d+1$-competitive for online ISP with intervals of length 1 and d.

Proof. Each interval scheduled by First-Fit can overlap at most $d + 1$ intervals in the optimal solution. ■

Allowing d to become 1 in Theorem 2.6 we get

Corollary 2.2: First Fit is 2-competitive for online ISP where intervals are of unit length.

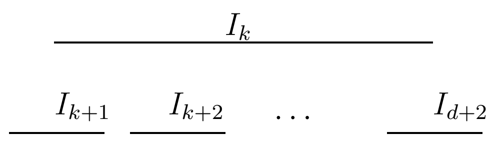
Next theorem shows that no deterministic algorithm gains better bound than First Fit.

Theorem 2.7: Any deterministic algorithm for online ISP with intervals of length d and 1 has a competitive ratio at least $d + 1$.

Proof. Let a deterministic algorithm D be given. For simplicity we assume that $d \in \mathbb{Z}_+$. The adversary presents $d+2$ disjoint intervals in the order of increasing left endpoints, I_1, ..., I_d, I_{d+1}, I_{d+2}, each of length d, separated by unit space, in this order. The adversary knows exactly which intervals are scheduled by D and which not. If D schedules no interval, then there is nothing left to prove.

If on the other hand D schedules some interval I_k, $k \leq d + 1$, then the adversary alters the input instance. The coordinate system is then shifted so that $I_k = [0, d)$. The intervals I_{k+1}, I_{k+2}, ..., I_{d+}, I_{d+2} are altered as follows: $I_{k+1} = [-0.5, 0.5)$, $I_{k+2} = [0.5, 1.5)$, ..., $I_{d+1} = [-0.5 + d - k, 0.5 + d - k)$ and $I_{d+2} = [-0.5 + d - k + 1, 0.5 + d - k + 1)$.

Since I_{k+1}, I_{k+2}, ..., I_{d+1}, I_{d+2} overlap I_k, D cannot accept any interval after accepting I_k. Therefore, D accepts only a single interval. The optimal schedule consists of all intervals except I_k. This proves that any deterministic algorithm is $d + 1$-competitive.



■

Notice that in Theorem 2.7, the intervals are presented in order of increasing endpoints.

The adversary in Theorem 2.7 has the property of being *transparent* (Halldórsson & Szegedy, 1992).

2.3.2 Randomized Lower Bound

Theorem 2.8: Any randomized online algorithm for ISP has a competitive ratio at least 4, where the intervals are of two possible lengths, 1 and $d \gg 1$.

Proof. Assume that a randomized algorithm *R* is given. Let d be a square number. By Theorem 2.1, there exists an interval d-stacking construction.

Depending on the structure of *R* the adversary can design an input instance such that, there exists a left-oriented intersection interval $\mathcal{I}$ or a right-oriented intersection interval $\mathcal{I}'$, as defined in the proof of Theorem 2.1, both of length $\sqrt{d}$.

Left-oriented intersection interval. The probability that *R* schedules some interval I_i, $1 \leq i \leq \sqrt{d}$, is one (see Figure 3.1 a)). The intersection interval $\mathcal{I} = \bigcap_{i=1}^{\sqrt{d}} I_i$ is an interval of length $\sqrt{d}$. The adversary adds $\sqrt{d}+1$ disjoint intervals to the input, all intersecting $\mathcal{I}$. By selecting $d \geq 9$, then $\rho_R \geq 4/1 = 4$.

Right-oriented intersection interval. The unconditional probability of scheduling $I_m = [s_m, f_m)$ is less than $1/\sqrt{d}$. After presenting I_m, the adversary presents $I_m = [f_m, f_m + d)$.

Let p' be the probability that *R* schedules some interval I_k, $1 \leq k \leq m-1$, and p_{m+1} be the conditional probability that *R* schedules I_{m+1}, (see Figure 1.1 b)). The expected number of intervals in σ_R is at most $p' + (1-p') \cdot p_{m+1} + 1/\sqrt{d}$ as in proof of Theorem 2.1. The expected number of intervals in the optimal schedule is on the other hand 2 since I_j and I_{j+1} are disjoint.

Therefore,

$$\rho_R \geq \sup_d \frac{2}{p' + (1-p') \cdot p_{m+1} + 1/\sqrt{d}} = \frac{2}{p' + (1-p') \cdot p_{m+1}} . \tag{2.3}$$

By the final remarks in Section 2.1, the adversary can use an area of length three to design d-interval stacking construction, resulting in an optimal schedule of weight 2 versus $1 + \delta$ of σ_R, where $0 < \delta$.

If $m^\star = \max\{i \in \mathbb{N} \mid \frac{1}{3} \cdot \sqrt{d} \geq i\}$ then $m^\star$ represents the number of such designs, which can be constructed by the adversary, on the right-oriented interval $\mathcal{I}'$ of length $\sqrt{d}$. Assuming that the adversary decides to play $m^\star$ of these games, the number of intervals in σ_R on the area covered by $\mathcal{I}'$ is $(1-p')(1-p_{m+1})m^\star(1+\delta)$. Here $(1-p')(1-p_{m+1})$ is the probability of scheduling none of the intervals $I_1, ..., I_{m-1}, I_{m+1}$.

The competitive ratio must therefore be at least

$$\begin{aligned}\rho_R &\geq \sup_{d, m^\star, \delta} \frac{2m^\star}{p' + (1-p')p_{m+1} + 1/\sqrt{d} + (1-p')(1-p_{m+1})m^\star(1+\delta)} \\ &= \frac{2}{(1-p')(1-p_{m+1})} . \end{aligned} \tag{2.4}$$

When comparing the inequalities 2.3 and 2.4, it is convenient to use $p = p' + (1-p')p_{m+1}$. Then 2.3 becomes $\rho_R \geq 2/p$ and 2.4 becomes $\rho_R \geq 2/(1-p)$. The competitive ratio must be at least $\min_p\{\max\{2/p, 2/(1-p)\}\}$. Notice that $2/p$ is a strictly decreasing function of p and $2/(1-p)$ a strictly increasing function of p. These functions intersect in $p = 1/2$ and because of that $\rho_R \geq 2/(1/2) = 4$.

■

2.3.3 Randomized Upper Bound

Algorithm 2.2: Let ISP be given, where the intervals of length 1 and d. A randomized algorithm for this problem is defined as follows.

1. Schedule unit intervals greedily.
2. Schedule intervals of length d directly with probability $1/2$ and virtually with probability $1/2$, if the resource is in free.
3. Do not schedule intervals of length d if they overlap virtually scheduled intervals.

In order to prove that Algorithm 2.2 is 4-competitive we use the Bucket Method. The weights are assigned in the following fashion.

1. If $I \in S$ is a unit interval then $w(I)$ is assigned evenly onto each $J \in \sigma_\star$. This means that if a unit interval overlaps two intervals in the optimal schedule, then weight of $1/2$ is assigned to each. If it overlaps only one $J \in \sigma_\star$ then the weight of 1 is assigned to J.
2. If $I \in S$ is an interval of length d then $w(I)$ is assigned onto each $J \in \sigma_\star$, of length d.
3. If I_1 is a unit interval, and I_2 is of length d, then I_1 is a *terminal interval* of I_2 if I_2 overlaps one endpoint of I_1. Assign weights evenly from segments of length d onto its terminal intervals in $\sigma_\star$. Since one interval can have only two terminal intervals then either the weight of $1/2$ or 1 is distributed to a terminal interval.

Theorem 2.9: Algorithm 2.2 is 4-competitive for ISP where the intervals of length 1 and d.

Proof. If $J \in \sigma_\star$, the algorithm can treat J in four different ways, depending on the order of which the intervals are presented.

1. **The resource is free**. In this case the expected weight of $bucket(J)$ is at least $1/2$.
2. **Interval of length 1 is presented prior to J and overlaps it**. In this case the expected weight of $bucket(J)$ is at least $1/2 \cdot 1/2 = 1/4$.
3. **J is of length 1 and an intersecting interval I of length d is presented prior to J.** If J is a terminal interval fo J then the expected weight of $bucket(J)$ is $1/2 \cdot 1/2$. Assume J is not a terminal interval of I. With probability $1/2$, I is scheduled virtually. Either J is scheduled greedily or another overlapping unit interval arrives before it, scheduled greedily. In this case the expected weight of $bucket(J)$ is at least $1/2 \cdot 1/2 = 1/4$.
4. **J is of length d and an overlapping interval of length d is presented prior to J.** In this case the expected weight is $1/2 \cdot 1/2 = 1/4$.

Since 1-4 occurs with probability 1 and in each case the expected weight of $bucket(J)$ is at least $1/4$, the algorithm is 4-competitive. ■

2.4 ISP with Parameter n

The parameter n, the number of intervals in the input instance, is an interesting parameter. It is interesting to see that the lower bound for deterministic algorithms and randomized algorithms are the same.

2.4.1 Deterministic Lower Bound

Notice that in the proof of Theorem 2.7, there exists an input instance for each deterministic algorithm, such it schedules no more than a single interval vs. $d + 1$ intervals in the optimal schedule. In terms of n, this ratio is $n - 1$. Therefore, we have the following corollary.

Corollary 2.3: Any deterministic algorithm has a competitive ratio at least $n - 1$, both if n is known by the algorithm in advance or not.

As in Theorem 2.7, this result does hold when the intervals are presented in order of increasing endpoints.

It is surprising to see that any randomized algorithm has a competitive ratio at least $\Omega(n)$. In order to do so we apply the Yao's Minimax Theorem.

2.4.2 Randomized Lower Bound

Theorem 2.10: Any online randomized algorithm for ISP has a competitive ratio at least $\frac{n}{4}$, when n is not known in advance.

Proof. Let the length of the intervals be 2^{n-1}, ... ,2^1, 2^0, presented in that order. The nature of the intervals are such that each interval is longer than the total length of the intervals that follow.

An interval can either intersect none of the intervals that follow or all of them. This is the reason why the adversary can make each interval „good“ or „bad“. An interval is *good* if it does not overlap any other interval that follow. On the other hand an interval is *bad* if it overlaps all the intervals that follow. The adversary makes each interval bad with probability $1/2$ and good with probability $1/2$. This defines a probability distribution on input instances with n intervals.

Assume that *R* is a randomized algorithm and the adversary designs a random input instance as above. If *R* is to have a lower competitive ratio than n it must accept at least one interval.

The number of good intervals presented before a bad interval is presented is a geometric random variable, since each interval is good with probability $1/2$ and bad with probability $1/2$. The expected number of good intervals presented before a bad one is presented is therefore $\frac{1}{1/2} = 2$. This is also true for each subsequence of intervals in the input. This is because each interval is good or bad, independent of other intervals. In particular, this holds for the intervals that *R* attempts to schedule. Therefore, the number of good intervals accepted by *R* , before it accepts a bad one, is also a geometric random variable with expected value 2. This means that $E[\sigma_R] \leq 2$.

On the other hand, the expected number of intervals in the optimal schedule is $n/2$, since the expected number of good intervals is $n/2$. The competitive ratio of *R* is therefore at least $n/4$.

This shows that any randomized algorithm has a competitive ratio at least $n/4$. ■

2.5 ISP with Intervals of Various Lengths

We assume that Δ is known in advance by the algorithms in this section. We demonstrate a randomized algorithm which is $O(\log \Delta)$-competitive. This bound is tight for randomized algorithms by Corollary 2.4. On the other hand every deterministic algorithm is $\Omega(\Delta)$-competitive.

2.5.1 Deterministic Lower Bound

The following theorem gives us lower bound in terms of Δ for deterministic algorithms.

If we let $d = \lceil \Delta \rceil$ in Theorem 2.7, we get the following results.

Theorem 2.11: Any deterministic online algorithm has a competitive ratio at least $\Omega(\Delta)$ for ISP, when Δ is known in advance.

Notice that the intervals in the input instance in Theorem 2.7 are ordered by increasing left endpoints. The results do therefore hold for the version of the problem where the intervals are presented in increasing left endpoint order.

Notice furthermore that the adversary uses only intervals of length 1 and Δ to design the input instance.

2.5.2 Randomized Lower Bound

Let Δ be the ratio between the shortest and the longest segment. We can simply observe that $\lceil \log \Delta \rceil = \Theta(n)$ in the proof of Theorem 2.10.

Corollary 2.4: Any online randomized algorithm for ISP has a competitive ratio at least $\Omega(\log \Delta)$.

2.5.3 Randomized Upper Bound

Theorem 2.12: There exists a randomized algorithm which is $O(\log \Delta)$-competitive for ISP where Δ is known in advance.

Proof. Assume that the shortest intervals are of length 1 and the longest one of length Δ. We design the randomized online algorithm *R* as follows. First we split the possible intervals in the input S into groups, G_i, where $i = 1, \ldots, \lceil \log \Delta \rceil$. In the first group there are intervals of length between 1 and 2. In group i the intervals are of length 2^{i-1} to 2^i. Let $g = \frac{1}{\lceil \log \Delta \rceil}$. Each group G_i is selected with probability $1/g$.

Intervals from this group are then scheduled greedily but intervals from other groups rejected automatically. First-Fit (*FF*) is 3-competitive on the selected group by Theorem 2.6. If $|\sigma_\star|$ is the size of the optimal schedule and $|OPT(G_i)|$ the size of the optimal schedule on group G_i, then $|OPT(G_i)| \leq \frac{1}{3} \cdot |FF(G_i)|$ and

$$\begin{aligned}
|\sigma_\star| &\leq \sum_{i=1}^{\lceil \log \Delta \rceil} |OPT(G_i)| \leq \sum_{i=1}^{\lceil \log \Delta \rceil} \frac{1}{3} \cdot |FF(G_i)| = \frac{\lceil \log \Delta \rceil}{3} \cdot \sum_{i=1}^{\lceil \log \Delta \rceil} \frac{1}{\lceil \log \Delta \rceil} \cdot |FF(G_i)| \\
&= \frac{\lceil \log \Delta \rceil}{3} \cdot \sum_{i=1}^{\lceil \log \Delta \rceil} E[R(G_i)] = \frac{\lceil \log \Delta \rceil}{3} \cdot E[\sigma_R].
\end{aligned}$$

This means that *R* is $O(\log \Delta)$-competitive. ∎

2.6 Online $PISP$

2.6.1 Different Online Scheduling Models

Online computation has been criticized to for being pessimistic and unrealistic. This is mainly because algorithms are compared to adversaries with unlimited computational capacity on a worst case input. The competitive ratio is therefore perhaps not as informative as it should be. Two algorithms can have the same competitive ratio. Both do very badly for a particular instance but one of them performs better in practice.

In paging (Chrobak & Noga, 1998), there are two known deterministic algorithms, LRU and FIFO that are strongly k-competitive. In practice, however, LRU performs much better. We can consider our randomized algorithm in Theorem 2.12. Each solution outputs a solution with very little variance, with respect to lengths. The algorithm is not likely to take advantage of easy input instances. This is because it always restricts it schedule to intervals from a single group.

This has been a motivation for different online scheduling models. Either the algorithm is allowed to know more or the adversary is restricted in some way. Shortcomings of competitive analysis and possible improvements of its definitions can be found in the paper (Koutsoupias & Papadimitriou, 2000). This is a motivation to consider other scheduling models for ISP. An online $PISP$ is the problem where the algorithm is allowed to look ahead into the future. To be more precise we assume that the algorithm knows the intervals in the problem as well as the order which they are presented. What the algorithm does not know is the terminal point, the point where the presentation stops and the performance ratio is calculated.

2.6.2 Randomized Lower Bound

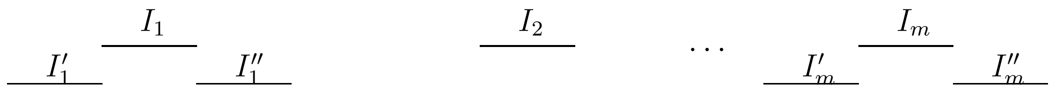


Figure 2.4: Online $PISP$. The intervals are presented in the order I_1, I_2, ..., I_m. In this figure, I_2 is scheduled with probability less than $2/3$ but I_1 and I_m with probability at least $2/3$.

Theorem 2.13: Any online algorithm has a competitive ratio of at least 1.5 for online $PISP$, when the intervals are of unit length.

Proof. Let *R* be a randomized algorithm. Consider online $PISP$ where the intervals, in Figure 2.4 are presented in the order: I_1, I_2, ... ,I_m, where $I_i = [2i, 2i + 1)$. The intervals are divided into two groups depending on the probability that *R* schedules them. In the first group are those who are scheduled with less probability than $2/3$ and in the other are those scheduled with probability at least $2/3$. No more intervals are presented, which overlap the ones in the first group. On the other hand, for each $I_j = [2j, 2j + 1)$, in the second group, two intervals $I'_j = [2j - 0.5, 2j + 0.5)$ and $I''_j = [2j + 0.5, 2j + 1.5)$ are presented, as in Figure 2.4. Two intervals, which overlap I_j, and no other interval in the instance.

Let the number of intervals in the first group be n_1. The expected number of intervals in σ_R is at most $2/3 \cdot n_1$+$2/3 \cdot (m-n_1)$+$1/3 \cdot 2(m-n_1) = 4/3 \cdot m - 2/3 \cdot n_1 = 2/3(2m-n_1)$. On the other hand the optimal schedule contains $n_1 + 2(m - n_1) = 2m - n_1$. This proves that the competitive ratio of *R* is at least 1.5.

■

Chapter 3

2-Interval Scheduling Problems

In this chapter we analyze $2ISP$. For simplicity we assume that $2ISP$ contains intervals as well. An interval in an input instance of $2ISP$ is a 2-interval, with one segment disjoint from every other 2-interval, in the input instance. In Section 3.1. we show that every algorithm has a competitive ratio at least 3, when segments are of unit length. Secondly, we show that for the version of the problem with depth 2, every algorithm has a competitive ratio at least $11/6$. Finding an algorithm with lower competitive ratio than First Fit remains an unsolved problem. We give an algorithm with a competitive ratio at least 3.375. In Section 3.2. we analyze the version of the problem with segments of length 1 and $d \gg 1$. We give an 8-competitive algorithm for this problem. Furthermore, we prove that any algorithm has a competitive ratio at least 6. In Section 3.3 we analyze the version of the problem where the segments are of various lengths with Δ being the maximum ratio between the shortest and the longest segment. We demonstrate an upper bound of $O(\log^2 \Delta)$ for this problem, when Δ is known in advance and $O(\log^{2+\epsilon} \Delta)$ when Δ is not known.

3.1 $2ISP$ with Intervals of Unit Length

3.1.1 Randomized Lower Bounds

Theorem 3.1: Any online randomized algorithm for the $2ISP$ with unit lengths has a competitive ratio at least 3

Proof. Assume that a randomized algorithm *R* is given. Let d be a square number. By Theorem 2.2 there exists a 2-interval d-stacking construction.

Depending on the structure of R , the adversary can design an input instance such that, there exists a left-oriented intersection interval $\mathcal{I}$ or a right-oriented intersection interval $\mathcal{I}'$, as defined in the Section 2.1, with segments of length $\sqrt{d}$.

Left-oriented intersection 2-interval. The probability that R schedules some 2-interval $I_i = [d-(i-1)\cdot\sqrt{d}, 2d-(i-1)\cdot\sqrt{d}) \cup [12d-(i-1)\cdot\sqrt{d}, 14d-(i-1)\cdot\sqrt{d})$, $1 \leq i \leq \sqrt{d}$, is one (see Figure 3.1 a)). The intersection 2-interval $\mathcal{I} = \bigcap_{i=1}^{\sqrt{d}} I_i$ is a 2-interval with both segments of length $\sqrt{d}$. The adversary adds four disjoint intervals of length d to the input, two which intersect the left segment of $\mathcal{I}$ (J_1 and J_2) and two which intersect the right one (J_3 and J_4). Then $c_R \geq 4/1 = 4 > 3$.

Right-oriented intersection 2-interval. Assume m is the lowest number in, $1, ..., \sqrt{d}$, such that the unconditional probability of scheduling $I_m = [s_m, f_m) \cup [s_m, f_m)$ is less than $1/\sqrt{d}$. After presenting I_m, the adversary presents $I_{m+1} = [f_m, f_m + d) \cup [f_m, f_m + d)$.

Let p' be the probability that R schedules some 2-intervals I_k, $1 \leq k \leq m-1$, and p_{m+1} the conditional probability that R schedules I_{m+1} (see Figure 3.1 b)). After presenting I_{m+1} the adversary has two options.

i) Present no more 2-intervals. The expected number of 2-intervals in σ_R is then at most $p' + (1-p')\cdot p_{m+1} + 1/\sqrt{d}$ as in Theorem 2.1. The optimal weight of the input is on the other hand 2, since I_m and I_{m+1} are disjoint.

Therefore,

$$c_R \geq \sup_d \frac{2}{p' + (1-p')\cdot p_{m+1} + 1/\sqrt{d}} = \frac{2}{p' + (1-p')\cdot p_{m+1}} .$$

If we denote the probability of scheduling some 2-interval overlapping $\mathcal{I}' = \bigcap_{i=1}^{m-1} I_i \cup I_{m+1}$ with p, then

$$c_R \geq \frac{2}{p' + (1-p')\cdot p_{m+1}} = 2/p . \tag{3.1}$$

ii) Use the area covered by both left and right segment of $\mathcal{I}'$ to present two interval $\sqrt{d}$-stacking constructions. The area covered by a single segment of $\mathcal{I}'$ is of length $\sqrt{d}$. By corollary 2.2, there exists a an intersection interval d-stacking construction with either a left-oriented interval $\mathcal{J}$, or a right-oriented intersection interval. The expected number of 2-intervals in σ_R is in this case at most $p + (1-p)\cdot(1+1/\sqrt[4]{d})\cdot 2 + \frac{1}{\sqrt{d}}$. This is because the expected number of 2-intervals σ_R is the unconditional probability of R scheduling the 2-interval I_m, plus the probability of scheduling some interval overlapping $\mathcal{I}'$ and the expected number of intervals in the d-stacking constructions.

The optimal weight is on the other hand 4 since both interval d-stacking constructions include 2 disjoint intervals. Therefore,

$$c_R \geq \sup_d \frac{4}{p+(1-p)(1+1/\sqrt[4]{d})\cdot 2 + \frac{1}{\sqrt{d}}} = \frac{4}{p+(1-p)\cdot 2} = \frac{4}{2-p}\,. \tag{3.2}$$

Comparing 3.4 and 3.2 we get that $c_R \geq \min_p(\max\{\frac{2}{p}, \frac{4}{2-p}\})$. Notice that $2/p$ is a strictly decreasing function and $4/(2-p)$ is a strictly increasing one. The solution to $\min_p(\max\{\frac{2}{p}, \frac{4}{2-p}\}) = 3$ since

$$\frac{2}{p} = \frac{4}{p+(1-p)\cdot 2} \Leftrightarrow p = 2/3 \text{ and } \frac{2}{2/3} = 3. \tag{3.3}$$

Therefore, $c_R \geq 3$.

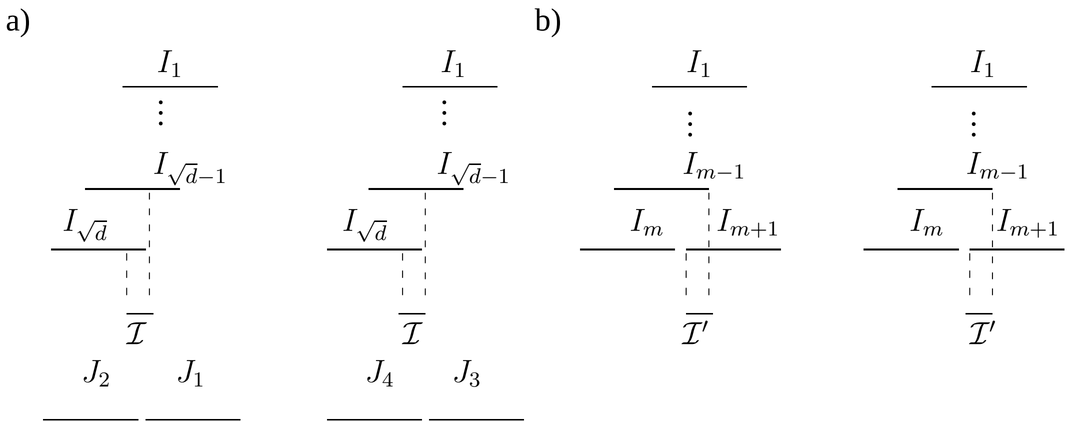


Figure 3.1: a) Input instance designed by the adversary in the proof Theorem 3.1, in the case of a left-oriented intersection 2-interval. b) Input instance designed by the adversary in the proof Theorem 3.1, in the case of a right-oriented intersection 2-interval.

■

The area covered by the input, designed by the adversary for given d in Theorem 3.1, is $14d$ since $I_1 = [d, 2d) \cup [10d, 11d)$ and every 2-interval presented after I_1, overlaps I_1. Since the length d is relative, the adversary can use an area of length 14 to design an input instance for $2ISP$ with unit segments, where the performance ratio is as close to 3 as desired. In other words, for a given number $0 < \delta$, there exists an input instance, such that the performance ratio is at least $\frac{3}{1+\delta}$.

3.1.2 Search for an Algorithm

Several randomized algorithms were tried for $2ISP$ with unit lengths but the competitive analyzis of them fell short. As for any online randomized algorithm our aim is prove upper bounds, at least lower than First Fit (FF).

Theorem 3.2: FF is 4-competitive.

Proof. For each scheduled 2-interval, there are at most four intervals in the optimal schedule that intersect a 2-interval scheduled by FF. Therefore FF is 4-competitive. ∎

Here is an algorithm which was considered as the best candidate for having a competitive ratio close to 3.

Algorithm 3.1: Let S be an instance of an online $2ISP$ with unit segments. When a 2-interval is presented and:

1. **the resource is free,** schedule it directly with probability $2/3$ and virtually with probability $1/3$.

2. **it does intersect a virtually scheduled 2-interval with two segments**, then do not schedule it.

3. **it intersects a virtually scheduled 2-interval with one segment**. Schedule it directly with probability $2/3$ and virtually with probability $1/3$.

The following example shows us that the algorithm fails to have competitive ratio 3.

Example 1: Consider Figure 3.2. In this online $2ISP$ instance the 2-intervals are presented in the order: I_1, I_2, J_1, J_2, J_3 and J_4. The expected number of 2-intervals in the schedule of Algorithm 3.1 is $2/3 \cdot 1 + 1/3 \cdot 2/3 + (1/3)^2 \cdot 4 \cdot 2/3 = 32/27$. On the other hand the optimal schedule has the weight of 4. The competitive ratio of the Algorithm 3.1 is therefore at least $108/32 = 3.375$.

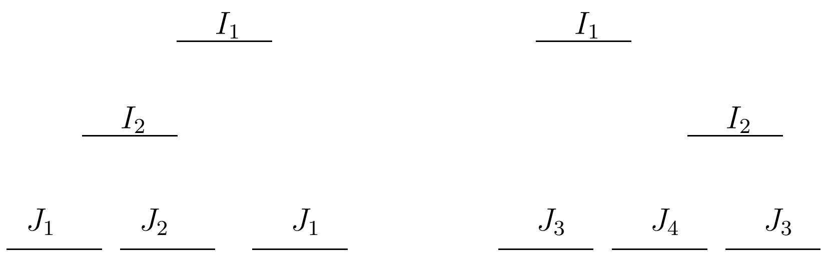


Figure 3.2: An input instance which proves that Algorithm 3.1 is at least 3.375 competitive. The 2-intervals are presented int the order I_1, I_2, J_1, J_2, J_3 and J_4.

3.1.3 $2ISP$ with Unit Segments and Depth Two

Recall that online $2ISP$ with unit segments has depth two, if the maximum clique size in the intersection graph induced by the 2-intervals is two. Lower bound for this problem is proved with Yao's Minimax Theorem.

Theorem 3.3: The competitive ratio of any randomized online algorithm for $2ISP$ with depth 2 is at least $11/6$.

Proof. With probability $2/3$ instance consisting of a single 2-interval, I_1, is presented. With probability $1/6$, instances of Figure 3.3 a) and b) are presented. A deterministic algorithm must do the same with I_1 and I_2 on all instances. Deterministic algorithms can be grouped by how they schedule I_1 and I_2. There are three cases.

1. A deterministic algorithm D_1 schedules I_1. The expected number of 2-intervals in σ_{D_1} is $2/3 + 1/6 + 1/6 = 1$.
2. A deterministic algorithm D_2 schedules I_2 but not I_1. The expected number of 2-intervals in σ_{D_2} is at most $1/6 \cdot 1 + 1/6 \cdot 4 = 5/6$.
3. A deterministic D_3 schedules neither I_1 nor I_2. The expected number of 2-intervals in σ_{D_3} is at most $1/6 \cdot 3 + 1/6 \cdot 3 = 1$.

However, the expected number of 2-intervals in the of the optimal schedule is $2/3 + 1/6 \cdot 3 + 1/6 \cdot 4 = 11/6$. By Yao's Minimax Theorem we get that a competitive ratio of a randomized algorithm is at least $11/6$.

∎

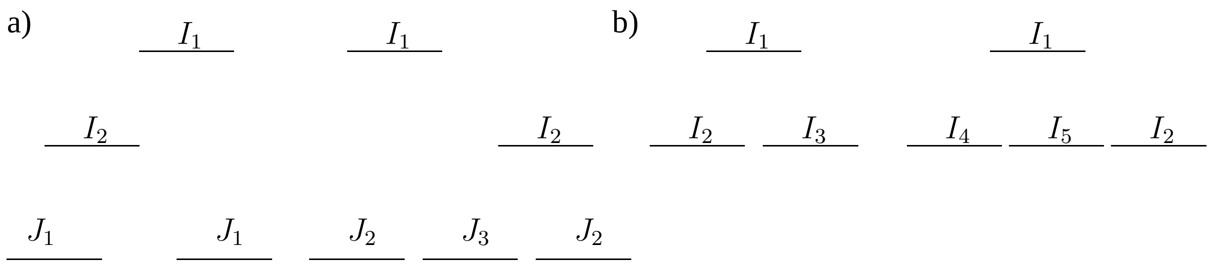


Figure 3.3: In Theorem 3.3 Yao's Theorem, is used with three input instances. With probability $2/3$, there is only a single 2-interval I_1 in the instance. With probability of $1/6$ instances in a) and b) are presented. a) The 2-intervals are presented in the order I_1, I_2, J_1, J_2 and J_3. b) The 2-intervals are presented in the order I_1, I_2, I_3 and I_4. The 2-intervals I_1 and I_2 do have the same status in a and b.

3.2 $2ISP$ Problem with Segments of Different Length

In this section we prove lower bound of 6 for $2ISP$ where the 2-intervals can have lengths 1 and $d \gg 1$. Finally we give a 8-competitive algorithm for this problem.

3.2.1 Randomized Lower Bounds

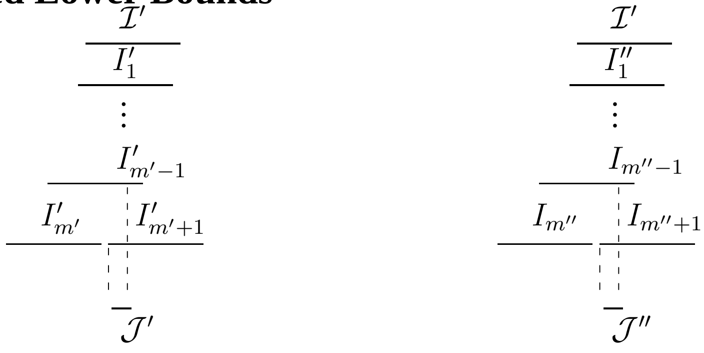


Figure 3.4: This picture represents the third option in the proof of Theorem 3.4. In order to avoid a competitive ratio higher than 6, the structure of *R* must be such that $\mathcal{J}'$ and $\mathcal{J}''$ are right-oriented intersection intervals, in option iii).

Theorem 3.4: Any online randomized algorithm for $2ISP$, where the 2-intervals are of lengths 1 and $d \gg 1$, has a competitive ratio at least 6.

Proof. Assume that a randomized algorithm *R* is given. Let d be a fourth power of an integer. By Theorem 2.2 there exists a 2-interval d-stacking construction.

Depending on the structure of R , the adversary can design an input instance such that, there exists a left-oriented intersection 2-interval $\mathcal{I}$ or a right-oriented intersection 2-interval $\mathcal{I}'$, as defined in the remarks before Theorem 2.2, both with segments of length $\sqrt{d}$.

Left-oriented intersection 2-interval. The probability that R schedules some 2-interval I_i, $1 \le i \le \sqrt{d}$, is one (see Figure 3.1 a)). The intersection 2-interval $\mathcal{I} = \bigcap_{i=1}^{\sqrt{d}} I_i$ is a 2-interval with both segments of length $\sqrt{d}$. Therefore, the adversary adds $2\sqrt{d}$ disjoint 2-intervals with unit segments, each intersecting one segment of $\mathcal{I}$. By selecting $d > 9$, then $c_R \ge 2\sqrt{d} > 6$.

Right-oriented intersection 2-interval. Assume m is the lowest number in, $1, ..., \sqrt{d}$, such that the unconditional probability of scheduling $I_m = [s_m, f_m) \cup [s_m, f_m)$ is less than $1/\sqrt{d}$. After presenting I_m, the adversary presents $I_{m+1} = [f_m, f_m + d) \cup [f_m, f_m + d)$.

Let p' be the probability that R schedules some 2-intervals I_k, $1 \le k \le m - 1$, and p_{m+1} the conditional probability that R schedules I_{m+1} (see Figure 3.1 b)). After presenting I_{m+1} the adversary has three options.

i) Present no more 2-intervals. The expected number of 2-intervals in σ_R is then at most $p' + (1 - p') \cdot p_{m+1} + 1/\sqrt{d}$ as in Theorem 2.1. The optimal weight of the input is on the other 2, since I_m and I_{m+1} are disjoint.

Therefore,

$$c_R \ge \sup_d \frac{2}{p' + (1 - p') \cdot p_{m+1} + 1/\sqrt{d}} = \frac{2}{p' + (1 - p') \cdot p_{m+1}} .$$

If we denote the probability of scheduling some 2-interval in $I_1, ..., I_{m-1}, I_{m+1}$ with p, then

$$c_R \ge \frac{2}{p' + (1 - p') \cdot p_{m+1}} = 2/p . \tag{3.4}$$

ii) The second option is to use the final remarks of Subsection 3.1.1. Notice that segments of $\mathcal{I}'$ are both of length $\sqrt{d}$. Therefore, the adversary designs multiple inputs of 2-intervals with unit segments, using an area of constant length 14 on the area covered by the segments of $\mathcal{I}'$. For each of these inputs, the performance ratio is 3 vs. $1 + \delta$, where $0 < \delta$. If $s^\star = \max\{s \in \mathbb{N} \mid \frac{1}{14} \cdot \sqrt{d} \ge s\}$, then $s^\star$ is the number of such inputs, which can be made by the adversary on the area of length $\sqrt{d}$.

Therefore,

$$\begin{aligned} c_R &\geq \sup_{d,\delta} \frac{3s^\star}{p' + (1-p')p_{m+1} + 1/d + (1-p')(1-p_{m+1})(1+\delta)s^\star} \\ &= \frac{3}{(1-p')(1-p_{m+1})} = \frac{3}{1-p}\,. \end{aligned} \tag{3.5}$$

If c_R is no more than 6, then 3.4 and 3.5 give that $1/3 \leq p \leq 1/2$.

iii) The third option is to use Corollary 2.1 and construct interval $\sqrt{d}$-design on both segments of $\mathcal{I}'$ with intervals of length d. Assume first that the structure of *R* is such that schedules the intervals in the design "too greedily" (see proof of Theorem 2.3) on either end. Then *R* schedules one interval covering a left-oriented intersection interval (an area of length $\sqrt[4]{d}$) with probability 1 (see proof of Theorem 2.3). In this case the adversary will present $\sqrt[4]{d}+1$ disjoint unit intervals in the area covered by the left-oriented intersection interval. In this case c_R can be made as large as possible by letting $d \to \infty$.

We can therefore assume that the structure of *R* is such that it schedules the intervals with "diminishing probability" on both segments of $\mathcal{I}'$ (see proof of Theorem 2.1). The interval $\sqrt{d}$-design on both segments must therefore be a right-oriented intersection intervals, $\mathcal{J}'$ and $\mathcal{J}''$ as in Figure 3.4. This means that the intervals $I'_{m'}$ and $I''_{m''}$ are scheduled both with unconditional probability at most $1/\sqrt[4]{d}$. Because the problem is symmetrical for both ends of $\mathcal{I}'$, we can assume that *R* schedules some interval in $I'_1, ..., I'_{m'}$ and some interval in $I''_1, ..., I''_{m''}$ with the same probability y. Therefore,

$$c_R \geq \sup_{d} \frac{4}{p + (1-p)\cdot(1+1/\sqrt{d})\cdot 2y} = \frac{4}{p+(1-p)\cdot 2y}. \tag{3.6}$$

$\bigcap_{i=1}^{m'} I'_i$ and $\bigcap_{i=1}^{m''} I''_i$ are both of length at least $\sqrt[4]{d}$ (see Figure 3.4 b)) as Corollary 2.1 suggests.

The adversary can on the other hand design multiple inputs of 2-intervals with unit segments, using an area of constant length 14 on the area covered by the segments of $\mathcal{J}'$ and $\mathcal{J}''$. For each of these inputs, the performance ratio is 3 vs. $1+\delta$, where $0<\delta$.

Therefore

$$\begin{aligned} c_R &\geq \sup_{d,\delta} \frac{3m^\star \cdot 2}{p+(1-p)2y+2\cdot(1-p)(1-y)(1+1/d)m^\star} \\ &= \frac{3}{(1-p)(1-y)}. \end{aligned} \tag{3.7}$$

If c_R is no more than 6, then $y \leq 1/4$ by 3.7, since $1/3 \leq p \leq 1/2$. If we define $x = p+(1-p)y$, then 3.6 becomes $c_R \geq \frac{4}{x+(1-p)y} \geq \frac{4}{x+\frac{1}{6}}$ and 3.7 becomes $c_R \geq \frac{3}{1-x}$. Therefore $c_R \geq \max_x \left\{\frac{3}{1-x}, \frac{4}{x+\frac{1}{6}}\right\}$. By solving $\max_x \left\{\frac{3}{1-x}, \frac{4}{x+\frac{1}{6}}\right\}$, then

$$\frac{3}{1-x} = \frac{4}{x+\frac{1}{6}} \Leftrightarrow x = \frac{1}{2} \text{ and } 3/(1-1/2) = 6.$$

We have shown that $c_R \geq 6$.

■

3.2.2 Randomized Upper Bounds

In this subsection intervals of length d are *long* and unit intervals *short*.

The following algorithm is named the *Virtual Algorithm* (*VA*) as similar algorithm in the paper (Lipton & Tomkins, 1994).

Algorithm 3.2: Let S be an instance of an online $2ISP$ where the length of the segments can be either 1 or d. The following algorithm is *the Virtual Algorithm*. The algorithm schedules a presented 2-interval with the following rules.

1. **The resource is free**. Schedule a 2-interval greedily if both segments are short. Otherwise schedule it directly with probability $1/2$ and virtually with probability $1/2$.

2. **A presented 2-interval does intersect a virtually scheduled 2-interval**. If the 2-intervals intersect in such a way that long segments overlap, then do nothing.

3. **A presented 2-interval intersects only virtually scheduled 2-intervals with a short segment**. Schedule it greedily if it contains two short segments but otherwise with probability $1/2$.

Competitive Analyzis of the Virtual Algorithm

In order to prove that Algorithm 3.2 is 8-competitive with the Bucket Method we need first to assign weights properly to $J \in \sigma_\star$.

If $I \notin \sigma_\star$, $J \in \sigma_\star$ and I overlaps J then:

1. $w(I, J) = 1/4$ if segments of same length overlap, or if a short segment of I overlaps a long segment of J.

2. J is *a terminal* 2*-interval* of I, if a long segment of I overlaps an endpoint of a short segment of J. If J is a terminal 2-interval of I then $w(I, J) = 1/4$.

3. $w(I, J) = 0$ if I overlaps a short segment of J with a long segment, and J is not a terminal 2-interval of I.

Notice that weight of $I \notin \sigma_\star$ can at most be assigned to four $J \in \sigma_\star$. Therefore $\sum_{J\in\sigma_\star} w(I, J) \leq w(I) = 1$ and weights are assigned properly.

The following theorem tells us that $E_{VA}\left[bucket(J)\right]$ is at least $\frac{1}{8} \cdot w(J) = \frac{1}{8}$. Theorem 1.1 gives us that this is enough to prove that *VA* is 8-competitive.

Theorem 3.5: The *VA* is 8-competitive for online $2ISP$ with segments of length 1 and d.

Proof. Assume that S is an instance of $2ISP$ with segments of length 1 and d. We use the Bucket Method to prove this. Assume that the weights are assigned as in 1-3 above.

Five possibilities can occur when $J \in \sigma_\star$ is presented.

1. **The resource is free when J is presented**. Then J will be scheduled with the probability at least $1/2$. Therefore the expected weight of $bucket(J)$ is at least $1/2 \cdot w(J) = 1/2 \cdot 1 = 1/2$

2. **A 2-interval I is scheduled directly/virtually prior to arrival of J and they intersect with segments of same length**. In this case $w(I, J) = 1/4$. With probability $1/2$, I is scheduled directly. The expected weight of $bucket(J)$ is at least $1/2 \cdot 1/4 = 1/8$.

3. **A 2-interval I is presented prior to arrival of J and intersects a long segment of J with a short segment**. Then $W(I, J) = 1/4$. Since I is scheduled with $1/2$ the expected weight of $bucket(J)$ is at least $1/2 \cdot 1/4 = 1/8$.

4. **A 2-interval I is presented prior to the arrival of J and intersects a short segment of J with a long segment**. If J is a terminal 2-interval of I then $w(I,J) = 1/4$. In this case the expected weight of $bucket(J)$ is at least $1/2 \cdot 1/4 = 1/8$.

   If, on the other hand, J is not a terminal 2-interval of I, then $w(I,J) = 0$. Three scenarios can come up.

   a) No 2-interval is presented in between I and J, or 2-intervals which intersect a long segment of J with a long segment. In this case, every 2-interval that comes in between J and I is blocked by I. Therefore, the expected weight of $bucket(J)$ is at least $1/2 \cdot 1/2 = 1/4$.

   b) A 2-interval, I', with two short segments is presented in between I and J and overlaps the short segment of J covered by I. Since $w(I',J) = 1/4$ and I' is scheduled greedily, the expected weight of $bucket(J)$ is $1/2 \cdot 1/4 = 1/8$.

   c) Number of 2-intervals with one segment long and the other short are presented in between I and J, and intersect J with a short segment as in Figure 3.5. Assume that these intervals are m in number. The probability that J is scheduled directly is at most $\frac{1}{2} \cdot \left(\frac{1}{2}\right)^{m+1}$ because I is scheduled virtually with probability $1/2$, and all the 2-intervals presented between I and J, are scheduled virtually with probability $1/2$. Since the assigned weight of all these intervals is $1/4$, the expected value of their total assigned weight is $\frac{1}{2} \cdot \sum_{i=1}^{m} \left(\frac{1}{2}\right)^{i} \cdot \frac{1}{4}$.

   The expected weight of $bucket(J)$ is therefore

$$\frac{1}{2} \cdot \left(\sum_{i=1}^{m} \left(\frac{1}{2}\right)^{i} \cdot \frac{1}{4} + \left(\frac{1}{2}\right)^{m+1} \right) \geq \frac{1}{8} = \frac{1}{8} \cdot w(J),$$

   for all $m \in \mathbb{N}$.

5. 2**-intervals, I' and I'' are presented before J, which has two short segments, and I' and I'' cover both segments of J with long segments.**

   a) No more intervals in between I and J are presented or 2-intervals that are blocked. In this case the expected weight of $bucket(J)$ is $1/2 \cdot 1/2 \cdot 1 = 1/4$ since I' and I'' are scheduled virtually with probability $1/2$ and J is scheduled greedily.

   b) Two different 2-intervals are presented and overlap both segments of J with a short segment. The assigned weight of both of them to J is $1/4$ and both are scheduled with probability at least $1/2$. Because of that the expected weight of $bucket(J)$ is at least $1/2 \cdot 1/2 \cdot 1/4 + 1/2 \cdot 1/2 \cdot 1/4 = 1/8$.

c) Greedily scheduled interval is presented in between I and J and overlaps both segments of J. In this case the assigned weight to J is $1/2$ ($1/4$ from both segments). The expected weight of $bucket(J)$ is therefore $1/2 \cdot 1/2 \cdot 1/2 = 1/8$.

d) All possible 2-intervals (not blocked) that are presented in between I and J intersect one segment of J as in 4c). For simplicity assume that this segment is covered by I'. The only influence I'' has on the expected weight of $bucket(J)$ is to make sure J is scheduled with probability at least $1/2$. This case is therefore identical to 4c).

By 1-5,

$$E_{VA}[w(bucket(J))] \geq 1/8 = 1/8 \cdot w(J) \ .$$

■

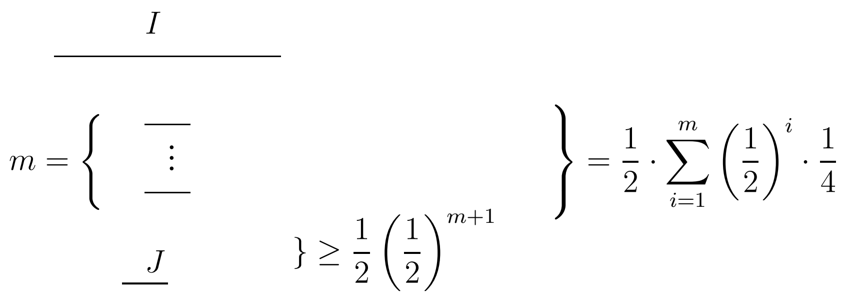

I
$m =$
J
$\} \geq \frac{1}{2}\left(\frac{1}{2}\right)^{m+1}$
$= \frac{1}{2} \cdot \sum_{i=1}^{m} \left(\frac{1}{2}\right)^{i} \cdot \frac{1}{4}$

Figure 3.5: Case 4c) in the proof of Theorem 3.5. Multiple 2-intervals are presented in between the arrivals of I and J, each having one long segment and intersect J with a short segment.

3.2.3 The Virtual Algorithm for Two Groups of Segments

Assume that we have a $2ISP$ where segments can either be of length from 1 to 2 or d to $2d$. Let the first group be the *short group* and the second one the *long group*. Segments from the long group are *long* and segments from the short group are *short*. For this problem we can redefine the Virtual Algorithm (Algorithm 3.2), such that it treats the segments from the short group as unit segments and the segments from the long group as segments of length d.

If $I \notin \sigma_\star$ and $J \in \sigma_\star$, then the following assignment of weights is proper because the weight of each 2-interval is assigned to at most six disjoint 2-intervals:

1. $w(I, J) = 1/6$ if I overlaps $J \in \sigma_\star$, and segments from the same length group overlap, or if a short segment of I overlaps a long segment of J.
2. J is *a terminal* 2*-interval* of I, if a long segment of I overlaps an endpoint of a short segment of J. If J is a terminal 2-interval of I, then $w(I, J) = 1/6$.
3. $w(I, J) = 0$ if J overlaps a short segment of J with a long segment, and J is not a terminal 2-interval of I.

Using the proper assignment of weights above we get the following Theorem.

Theorem 3.6: For an online $2ISP$ with segments from a short group and a long group, defined in this section, the redefinition of the Virtual Algorithm in this section is 12-competitive.

Proof. Assume that S be an instance of $2ISP$ with segments of length 1 and d. Assume that the weights are assigned as in 1-3 above. This Theorem is proved by using the same proof as in Theroem 3.5, with the redefinition of long segment and short segment. ■

We call this redefinition the *Redefined Virtual Algorithm* (*RVA*).

3.3 $2ISP$ with Segments of Various Lengths

In this section we examine $2ISP$ with various lengths. An important factor here is Δ, the ratio between the shortest and the longest segment. We can assume without a loss of generality that the shortest segment is of length 1 and the longest segment of length Δ.

We will examine two cases, when Δ is known in advance and when Δ is not known in advance.

3.3.1 Δ Known in Advance

We can group the first segments of the 2-intervals of a given instance S into $\lceil \log \Delta - 1 \rceil$ groups. In group $\lceil \log \Delta - 1 \rceil$ are the segments of length strictly greater than $\Delta/2$ and less than or equal to Δ. In group $\lceil \log \Delta - 1 \rceil - 1$ are segments of length strictly greater than $\Delta/4$ and less than or equal to $\Delta/2$ etc. Let the groups be υ_1, υ_2, ... , $\upsilon_{\lceil \log \Delta - 1 \rceil}$. In the same fashion we group the second segments of the 2-intervals, into the groups υ'_1, υ'_2, ... , $\upsilon'_{\lceil \log \Delta - 1 \rceil}$. These groups are partitions of the first and second segments.

We can combine groups in last paragraph and group the 2-intervals of input instance S into groups. These groups are symbolized by a $\lceil \log \Delta - 1 \rceil \times \lceil \log \Delta - 1 \rceil$ matrix S_{ij}. The entry S_{mn} stands for a group of 2-intervals from S where the first segment is in υ_m and the second segment in υ'_n. These grouping defines a partition of the 2-intervals of S.

The next theorem is a direct result of Theorem 3.6.

Theorem 3.7: The *RVA* is 12-competitive for each group represented by S_{nm}.

A simple randomized algorithm, R_1 , takes advantage of this by picking in advance a single group represented by S_{ij}. Each group is picked with equal probability $p = \dfrac{1}{(\log \Delta)^2}$, and 2-intervals from it scheduled, using *RVA* . Since this algorithm is 12-competitive for each group, the expected number of 2-intervals in σ_{R_1} is:

$$\begin{aligned} E[\sigma_{R_1}] &= \sum_{i=1}^{\lceil \log \Delta - 1 \rceil} \sum_{j=1}^{\lceil \log \Delta - 1 \rceil} \frac{1}{(\log \Delta)^2} \cdot E(\textit{RVA}\ (S_{ij})) \\ &\geq \sum_{i=1}^{\lceil \log \Delta - 1 \rceil} \sum_{j=1}^{\lceil \log \Delta - 1 \rceil} \frac{1}{(\log \Delta)^2} \cdot \frac{1}{12} \cdot |OPT(S_{ij})|. \end{aligned}$$

Since $\sum_i \sum_j |OPT(S_{ij})| \geq |OPT(S)|$ we get:

$$\sum_{i=1}^{\lceil \log \Delta - 1 \rceil} \sum_{j=1}^{\lceil \log \Delta - 1 \rceil} \frac{1}{(\log \Delta)^2} \cdot \frac{1}{12} \cdot |OPT(S_{ij})| \;\; \geq \;\; \frac{1}{(\log \Delta)^2} \cdot \frac{1}{12} \cdot |OPT(S)|.$$

Algorithm R_1 is therefore $O(\log^2 \Delta)$-competitive.

Theorem 3.8: R_1 is $O(\log^2 \Delta)$-competitive for $2ISP$ with intervals of various length, where Δ is known in advance.

3.3.2 Δ Not Known in Advance

If Δ is not known beforehand, we can still take advantage of a partition mentioned in last section. We motify R_1 from last section to get a new randomized algorithm R_2 . A presented 2-interval I is in the same group as a previously presented 2-interval I', if the ratio between the length of the first/second segment of I and I' is between 1 and 2. If not, I belongs to a new group.

We can keep track of the longest segment known each time, l and schedule 2-intervals in group i with probability
$c_i = \frac{1}{\zeta(1+\epsilon)((\log l)^2)^{1+\epsilon/2}} = \frac{1}{\zeta(1+\epsilon)(\log l)^{2+\epsilon}}$ where

$$\zeta(x) = \sum_{i=1}^{\infty} \frac{1}{i^x} < \infty, \text{ if } x > 1,$$

is the *Riemann zeta function*.

R_2 changes l at most $\log^2 \Delta$ times. Because of this, we get a probability distribution on S_{ij}:

$$\sum_{i=1}^{(\log\Delta)^2} c_i \leq \sum_{i=1}^{\infty} \frac{1}{\zeta(1+\epsilon/2)i^{1+\epsilon/2}} = \frac{1}{\zeta(1+\epsilon/2)} \sum_{i=1}^{\infty} \frac{1}{i^{1+\epsilon/2}} = \zeta(1+\epsilon/2)\cdot \frac{1}{\zeta(1+\epsilon/2)} = 1.$$

A 2-interval presented to the algorithm is either in a new group, a selected group or a rejected one. If it is a new group and no group has been selected, then either its group is chosen with probability p_i or not with probability $1 - p_i$.

The algorithm R_2 uses the probability c_i to find out with what probability, p_i, it needs to choose ith group when a 2-interval from it is presented. Before an 2-interval in group i arrives the algorithm has to refuse to schedule 2-intervals from certain number of groups. Therefore the algorithm needs to ensure that $c_i = p_i \cdot \Pi_{j=1}^{i-1}(1 - p_j)$. Then $p_1 = c_1$ and $p_i = \frac{c_i}{\Pi_{j=1}^{i-1}(1-p_j)}$. Notice that $0 < p_i < 1$ does always hold. This can be shown by induction. For $i = 1$ we have that $p_1 = c_1 < 1$. If we assume that $0 < p_i < 1$ for $i = 1, \ldots, k-1$ we get

$$\begin{aligned} 0 < p_k &= \frac{c_k}{\Pi_{j=1}^{k-1}(1-p_j)} = \frac{c_k}{(1-p_{k-1})\cdot\Pi_{j=1}^{k-2}(1-p_j)} = \frac{c_k}{(1-p_{k-1})\cdot\frac{c_{k-1}}{p_{k-1}}} = \frac{c_k}{c_{k-1}}\cdot\frac{p_{k-1}}{1-p_{k-1}} \\ &= \frac{\frac{1}{\zeta(1+\epsilon/2)k^{1+\epsilon/2}}}{\frac{1}{\zeta(1+\epsilon/2)(k-1)^{1+\epsilon/2}}}\cdot\frac{p_{k-1}}{1-p_{k-1}} = \left(\frac{k-1}{k}\right)^{1+\epsilon/2}\cdot\frac{p_{k-1}}{1-p_{k-1}} < \frac{p_{k-1}}{1-p_{k-1}} \le \frac{\frac{1}{2}}{1-\frac{1}{2}} = 1 \end{aligned}$$

since the function $f(x) = \frac{x}{1-x}$ has an absolute extreme in $x = \frac{1}{2}$, on the interval $]0,1]$.

The probability of *R*$_2$ choosing a single group is at least $c_{\log^{2+\epsilon}(\Delta)}$. After selecting a group as above, *R*$_2$ uses *RVA* to schedule the 2-intervals in the selected group. For a given group, S_{nm}, we have:

$$\begin{aligned} E[R_2(S_{nm})] &\ge c_{\log^{2+\epsilon}\Delta}\cdot E[RVA(S_{nm})] \\ &\ge \frac{1}{(\log\Delta)^{2+\epsilon}}\cdot\frac{1}{12}\cdot E[OPT(S_{nm})]. \end{aligned} \quad (3.8)$$

By 3.8, *R*$_2$ is $O(\log^{2+\epsilon}\Delta)$-competitive.

Theorem 3.9: *R*$_2$ is $O(\log^{2+\epsilon}\Delta)$-competitive for $2ISP$ with intervals of various length, where Δ is not known in advance.

Chapter 4

t-Interval Scheduling Problems

4.1 Randomized Lower Bound

Some graphs have the property that they can be represented by intervals, such that for each vertex there is one and only one half open interval on the real line, and two intervals intersect if and only if there is an edge between their vertices. Every complete graph K_n on n vertices can be represented by intervals by assigning the same interval to each vertex in K_n. Not all graphs can be represented by intervals. The simplest class of such graphs are cycles, C_n. However they can be represented by 2-intervals. This is the motive to the following definition.

Definition 4.1: A *t-interval representation* of a graph G is an assignment of t-intervals to each vertex of G such that:

a) for each vertex $v \in G$ there is one and only one t-interval.

b) for all edges in G, $\{v, w\} \in G$ if and only if the t-interval assigned to v intersects the t-interval assigned to w.

A graph is *s-interval representable* if there exists an integer $s \geq 0$, such that G has a s-interval representation. If every segment of the s-intervals in the representation is a unit interval, the graph is *s-unit-interval representable* and the representation *s-unit-interval representation*. If $s = 1$ then we say that the graph is *interval representable*. *Interval number* of a graph G, $i(G)$, is the smallest integer $s \geq 0$ such that G is s-interval representable.

For each graph G there exists a $s \geq 0$ such that G is s-interval representable. Griggs and West (Griggs & West, 1980) give lower bound of $i(G)$ in terms of number of vertices.

They show that for any graph G, $i(G) \leq \lceil \frac{n}{3} \rceil$. Therefore, every graph G with n vertices is n-interval representable. This can can be exploited in order to relate t-interval scheduling to *Online Independent Set Problem* ($OISP$)

An $OISP$ is the problem of finding maximum independent set in a graph, where the vertices are presented one at a time along with edges to previously presented vertices. Recall that Theorem 2.10 gives us that the competitive ratio of any online randomized algorithm is at least $\frac{n}{4}$ for online ISP. By definition of ISP, these lower bound hold for $OISP$ as well, since interval graphs are a special class of graphs. This fact is a key to proving lower bound for $tISP$ with unit segments.

Corollary 4.1: Any randomized algorithm for $OISP$ has a competitive ratio at least $n/4$.

In order to relate $OISP$ to online $tISP$ it is essential to show, that $OISP$ can be converted bit by bit to $tISP$.

Definition 4.2: A $OISP$ with vertices $v_1, ..., v_n$ presented in this order can be converted *adaptively* to $tISP$ if

a) each vertex, v_j is represented by one and only one t-interval, I_j, constructed and scheduled online by some online $tISP$ algorithm, before v_{j+1} is presented. No t-interval can be rescheduled or reconstructed.

b) $I_1, ..., I_j$ is a t-interval representation of the induced subgraph $v_1, ..., v_j$.

Theorem 4.1: Every $OISP$ can be converted adaptively to a $nISP$ with unit segments.

Proof. Assume we have a $OISP$ with n vertices $v_1, ..., v_n$, presented in this order. Let A be an online algorithm. We show how each vertex v_j can be represented with an n-interval with unit segments $I_j = (i_1^j, ..., i_n^j)$. These n-intervals are constructed based only on previously presented vertices. etc.

We define n intervals, such that $i_k^j = [(j-1)n + (k-1), (j-1)n + k)$ for all $1 \leq k \leq n$ and $1 \leq j \leq n$. This means that $I_1 = [0, 1) \cup [1, 2) \cup \cdots \cup [n-1, n)$, $I_2 = [n, n+1) \cup [n+1, n+2) \cup \cdots \cup [2n-1, 2n)$. All these intervals are disjoint.

As the vertex v_j is presented, I_j is altered, such that if $\{v_j, v_k\} \in G$ for $1 \leq k \leq j-1$, then I_j is redefined, such that $i_k^j = i_j^k$. After I_j has been altered and before v_{j+1} is presented, A makes a decicion whether to schedule I_j or not.

Each vertex v_j is then represented by one and only one I_j and there is an edge between v_j and v_k if and only if I_j and I_k intersect.

This proves that each $OISP$ can be converted adaptively to $nISP$.

■

Theorem 4.2: Any online randomized algorithm for $tISP$ with unit segments has a competitive ratio at least $\frac{t}{4}$.

Proof. We prove this theorem by contradiction. Assume that there exists an algorithm A with competitive ratio less than $\frac{t}{4}$ for $tISP$. If S is an instance of $OISP$ with n vertices, then it can be converted adaptively to $nISP$ by Theorem 4.1. This problem can be solved with A, such that the performance ratio is less than $n/4$. If we output the vertices represented by the n-intervals in σ_A, then S has been solved with performance ratio less than $n/4$. This is a contradiction to Corollary 4.1. ■

Bibliography

Agnarsson, G., & Greenlaw, R. (2007). *Graph theory modelling, applications and algorithms*. Upper Saddle River, New Jersey: Prentice Hall.

Bafna, V., Narayanan, B. O., & Ravi, R. (1995). Non-overlapping local alignments (weighted independent sets of axis parallel rectangles). In *Wads '95: Proceedings of the 4th international workshop on algorithms and data structures* (pp. 506–517). London, UK: Springer-Verlag.

Bar-Noy, A., Canetti, R., Kutten, S., Mansour, Y., & Schieber, B. (1995). Bandwidth allocation with preemption. In *STOC '95: Proceedings of the twenty-seventh annual ACM symposium on Theory of computing* (pp. 616–625). New York, NY, USA: ACM.

Bar-Yehuda, R., & Even, S. (1985). Local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, *25*, 27–46.

Bar-Yehuda, R., Halldórsson, M. M., Naor, J. S., Shachnai, H., & Shapira, I. (2002). Scheduling split intervals. In *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms* (pp. 732–741). Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.

Bar-Yehuda, R., & Rawitz, D. (2001). On the equivalence between the primal-dual schema and the local-ratio technique. In *APPROX '01: Proceedings of the 4th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems* (pp. 24–35). London, UK: Springer-Verlag.

Bar-Yehuda, R., & Rawitz, D. (2006). Using fractional primal-dual to schedule split intervals with demands. *Discrete Optimization*, *3*(4), 275 - 287.

Buchbinder, N., & Naor, J. S. (2009). Online primal-dual algorithms for covering and packing. *Math. Oper. Res.*, *34*(2), 270–286.

Canetti, R., & Irani, S. (1995). Bounding the power of preemption in randomized scheduling. In *STOC '95: Proceedings of the Twenty-Seventh Annual ACM Symposium on*

Theory of Computing (pp. 606–615). New York, NY, USA: ACM.

Chrobak, M. (2007). Competitiveness via primal-dual. *SIGACT News*, *38*(3), 100–105.

Chrobak, M., Jawor, W., Sgall, J., & Tichy´, T. (2007). Online scheduling of equal-length jobs: Randomization and restarts help. *SIAM J. Comput.*, *36*(6), 1709–1728.

Chrobak, M., & Noga, J. (1998). LRU is better than FIFO. In *SODA '98: Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms* (pp. 78–81). Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.

Crama, Y., & Spieksma, F. (2008). Scheduling jobs of equal length: complexity, facets and computational results. *Mathematical programming*, *72*(4), 207-277.

Goldman, S. A., Parwatikar, J., & Suri, S. (2000). On-line scheduling with hard deadlines. *Journal of Algorithms*, *34*, 370–389.

Graham, R. L. (1969). Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, *17*, 416–429.

Griggs, J. R., & West, D. B. (1980). Extremal values of the interval number of a graph. *SIAM Journal on Algebraic and Discrete Methods*, *1*(1), 1-7.

Halldórsson, M. M., & Szegedy, M. (1992). Lower bounds for on-line graph coloring. In *SODA '92: Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms* (pp. 211–216). Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.

Hocbaum, D. S. (Ed.). (1997). *Approximation algorithms for np-hard problems*. 20 Park Plaza, Boston: PWS Publishing company.

Karlsson, R. K. (2005). *Strip graphs and related scheduling problems*. Unpublished master's thesis, University of Iceland, Reykjavik, Iceland.

Keil, J. (1992). On the complexity of scheduling tasks with discrete starting times. *Operations research letters*, *12*(5), 293-295.

Kleinberg, J., & Tardos, E. (2005). *Algorithm design*. Addison Wesley.

Koutsoupias, E., & Papadimitriou, C. H. (2000). Beyond competitive analysis. *SIAM J. Comput.*, *30*(1), 300–317.

Lipton, R. J., & Tomkins, A. (1994). Online interval scheduling. In *SODA '94 Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms* (pp. 302–311).

Vialette, S. (2004). On the computational complexity of 2-interval pattern matching problems. *Theor. Comput. Sci.*, *312*(2-3), 223–249.

Woeginger, G. J. (1994). On-line scheduling of jobs with fixed start and end times. *Theor. Comput. Sci.*, *130*(1), 5–16.

Yao, A. C.-C. (1977). Probabilistic computations: Toward a unified measure of complexity. In *Sfcs '77: Proceedings of the 18th annual symposium on foundations of computer science* (pp. 222–227). Washington, DC, USA: IEEE Computer Society.

REYKJAVÍK UNIVERSITY
HÁSKÓLINN Í REYKJAVÍK

Department of Computer Science
Reykjavík University
Ofanleiti 2, IS-103 Reykjavík, Iceland
Tel: +354 599 6200
Fax: +354 599 6201
http://www.ru.is