Scheduling Splitting Intervals

Irina Shapira

Scheduling Splitting Intervals

Research thesis
submitted in partial fulfillment of
the requirements for the degree of
Master of Science in Computer Science

Irina Shapira

Submitted to the Senate of the Technion—Israel Institute of Technology
Tevet 5761 Haifa January 2001

The research thesis was done under the supervision of
Doctor Hadas Shachnai in the Department of Computer Science.

I wish to express my deepest gratitude and appreciation
to Dr. Hadas Shachnai
for her excellent guidance which made this work possible.

The generous financial help of the Technion is gratefully acknowledged.

Contents

List of Figures

Abstract

Modern Multimedia servers enable the transmission of video programs to individual clients upon request. In this operation mode, a client may wish to take a break from the program, and to resume viewing at some later time. We represent each client by a 'job', requesting the 'resource' (i.e., server capacity) for a time interval, which consists of a set of segments on the real line. Thus, the problem of scheduling client requests can be modeled as the following problem of finding a maximum independent set (IS) in a *splitting interval* graph:

> Given a sequence of intervals $I = \{I_1, ..., I_n\}$, in which I_i consists of a set of segments of the real line I_i^1, I_i^t, for some $t \geq 1$, select a subset $I' \subseteq I$ of non-intersecting intervals, such that the size of I' is maximal.

We focus in this work on the class of 2-split interval graphs, where each interval consists of at most two segments. We consider both the offline and the online variants of the problem. In the offline case, we show that the problem is APX-complete, even if all segments are of equal length. We give an $O(min(\log R, \log n))$-approximation algorithm, where R is the maximal length of any segment in the input. For the case where the number of distinct segment lengths is a fixed constant, we develop a $(4, \epsilon)$ bi-criteria polynomial time approximation scheme (PTAS), which finds for any $\epsilon > 0$ an IS of at least 1/4 of the optimal size, and a schedule in which the maximum delay is ϵ times the optimal makespan.

In the online case, we distinguish between two models, depending on the information available to the scheduler upon arrival of an interval. When the segment lengths and break times are known, we show that any deterministic algorithm has a competitive ratio $\Omega(R)$; a Greedy algorithm achieves this ratio to within factor 2. We then give a randomized $O(\log^2 R)$ -competitive algorithm. Finally, for the case where either the break times or segment lengths are unknown to the scheduler, we show a lower bound on the competitive ratio of any algorithm; this bound is achieved by a simple (deterministic) Greedy algorithm.

List of Symbols

$G = (V, E)$	a graph G with a set of vertices V and edges E
n	the number of vertices in G, $\|V\|$
m	the number of edges in G, $\|E\|$
IS	an independent set in a graph
k	a claw number of a graph
OffBA	the Offline Bucketing algorithm
VS	the Virtual Select algorithm
SB	the Select from Barrel algorithm
HL	the algorithm HalfLengths
SBA	the Short Breaks Algorithm
SBHL	the Short Breaks HalfLengths algorithm
CG	the Conservative Greedy algorithm
GD	the Greedy with Delays algorithm
VSD	the Virtual Select with Delays algorithm
SBD	the Select from Barrel with Delays algorithm
$PTAS$	Polynomial Time Approximation Scheme
OPT	an optimal offline algorithm
σ	an input sequence of intervals
$\|\sigma\|$	the length of an input sequence σ
$\mathcal{A}(\sigma)$	the gain of an algorithm $\mathcal{A}$ on an input sequence σ
$OPT(\sigma)$	the gain of an optimal offline algorithm on an input sequence σ
I	an interval
I^i	the ith segment of I
t	a split number, the maximal number of segments of the input intervals
R	a ratio between longest to shortest segments in the input
D	maximal break length

O .... upper bound
Ω .... lower bound

Chapter 1

Introduction

1.1 Problem Statement

In the well studied *maximum independent set (IS)* problem, we are given a graph, $G = (V, E)$, and we seek a subset of vertices $U \subseteq V$, such that no two vertices in U are adjacent in G, and $|U|$ is maximal. We consider the IS problem on the class of *t-split interval graphs*, in which any vertex can be represented as a *splitting interval*, which consists of at most t segments on the line, for some $t \geq 1$; two vertices u, v are *neighbors* if some segment of u intersects some segment of v.

Formally, let $G = (V, E)$ be an undirected graph. We say that G is a t-split interval graph if G can be represented by an interval graph $G' = (V', E')$, as follows:

(i) Each vertex $v \in V$ is replaced in G' by a set of vertices $v_1, \ldots, v_{r_v} \in V'$, where $max_v r_v = t$.

(ii) Any edge $(u, v) \in E$ is replaced by an edge $(u_i, v_j) \in E'$ for some $1 \leq i \leq r_u$, $1 \leq j \leq r_v$.

In the special case where $t = 1$, we get that G is an interval graph. We focus on 2-split interval graphs.

Example 1.1.1 *In Figure 1.1 we show an instance of five intervals, and corresponding split interval and interval graphs. The 2-split interval graph G is shown in (a). The corresponding interval graph G' is shown in (b). Note that interval e consists of only one segment. The sets $\{a, d\}$, $\{b, c\}$, $\{b, e\}$ and $\{c, d\}$ are independent sets.*

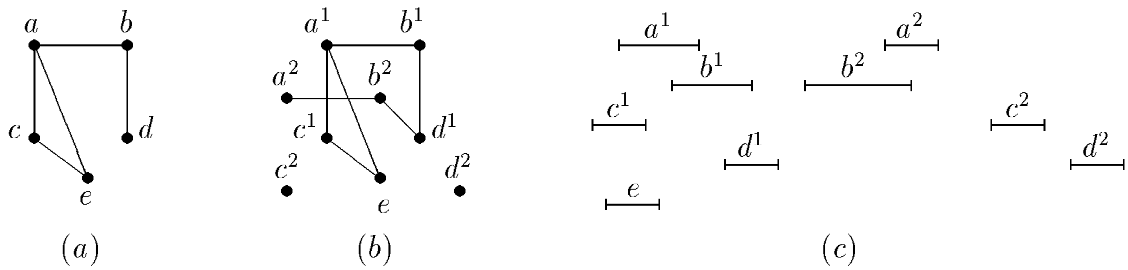


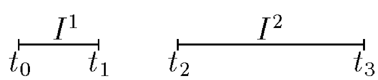
Figure 1.1: An example of a 2-split interval graph G and a corresponding interval graph G'.

1.2 Motivation

1.2.1 Scheduling in Multimedia Systems

Traditional multimedia servers transmit data to the clients by *broadcasting* video programs at prespecified times. Recent advances in high speed networking and compression technologies enabled to replace broadcasts with the allocation of video data streams to individual clients *upon request*, for some interval of time. In this operation mode a client may wish to take a break from the program, and to resume viewing at some later time. This scenario is natural, e.g., when video programs are used in remote education. Indeed, the service of a single request may now involve the allocation of several data streams, each corresponding to a time interval in which the program was viewed by the client continuously.

Suppose that a client sends a request to view a program at time t_0, and the system allocates a data stream to provide this service. At the time t_1 the client takes a break, and resumes viewing at t_2, till the end of the program (t_3). This scenario can be described by a *splitting interval* I, which consists of two segments: I^1 and I^2, of lengths $(t_1 - t_0)$ and $(t_3 - t_2)$ respectively. We define the total length of I, $\mid I \mid$, as $\mid I^1 \mid + \mid I^2 \mid$.



We say that the interval I *splits* at the time t_1 and *continues* at the time t_2. The gain of the scheduler from every scheduled interval is 1, and the scheduler has to be able to schedule both parts of the interval, otherwise the gain is zero.

The above problem, of allocating stream capacity to client requests, can be viewed as the problem of finding online a maximum independent set in a splitting interval graph:

> Given a sequence of intervals $I = \{I_1, \ldots, I_n\}$, in which I_i consists of a set of segments of the real line $I_i^1, \ldots, I_i^r$, for some $r \geq 1$, select a subset $I' \subseteq I$ of non-intersecting intervals, such that the size of I' is maximal.

1.2.2 Linear Resource Allocation

Another application is linear resource allocation. Requests to a linear resource can be modeled as intervals on a line; two requests for a resource can be scheduled together unless their intervals overlap. A disk drive is a linear resource when requests are for contiguous blocks [47]; a linear array network is a linear resource, since a request for bandwidth between processors i and j requires that bandwidth be allocated on all intervening edges. Such requests may be composed of several non-continuous requests, and, therefore, can be modeled as splitting intervals on a line. This problem is equivalent in the *offline* case (i.e., all the input is known to the algorithm in advance) to the problem of allocating stream capacity in multimedia systems. However, in the *online* case (where the algorithm may accept the input along its execution) we note a difference: while requests for stream capacity arrive in the order of their start times, i.e., t_0 in the above example, requests for linear resources are not sorted.

1.3 Related Work

IS in non-splitting interval graphs The problem of interval scheduling has been previously studied with several cost functions, as an offline and online problem. When the optimization goal is to maximize the number of scheduled intervals, the problem of scheduling (non-splitting) intervals can be optimally solved offline by a simple greedy algorithm [16]. (This algorithm sorts the input intervals by their finishing times, and then greedily selects intervals, that do not conflict with previously taken intervals). Another natural solution is to find a maximum independent set in the corresponding interval graph [22, 25].

IS in general graphs The problem of finding IS in general graphs was widely studied (a detailed survey appears in [30]): it is NP-complete [21] and for general graphs hard to approximate to within factor $n^{1-\epsilon}$, for some $\varepsilon > 0$ [34]. The best known approximation ratio for the problem is $O(n/\log^2 n)$ [13]. In the special case of $k+1$ claw free graphs[1], the *local improvements* technique described in [29] gives a $k/2+\epsilon$ performance ratio. The problem of finding IS admits a PTAS for planar graphs [8] and for unit disk graphs [35]. The case of degree bounded by Δ for $\Delta \geq 3$ is APX-complete [12, 45], and it is not approximable within Δ^ϵ for some $\epsilon > 0$ [3]. It is approximable within $(\Delta+3)/5$ [12] for small Δ, and within $O(\Delta \log\log\Delta/\log\Delta)$ for larger values (see, e.g., [39]). In the case of sparse graphs the problem is approximable within $O(d \log\log d/\log d)$ where d is the average degree of the graph [31].

[1]See in Section 2.2.

Splitting interval graphs Several previous studies addressed the problem of finding for a given graph G the minimal value of $1 \le t \le n$ (called the *interval number* of G), such that G is a t-split interval graph.[2] Trotter and Harary [52] showed that the interval number of a tree is at most 2, and the complete bipartite graph, $K_{m,n}$, has interval number $\lceil (mn+1)/(m+n) \rceil$. West and Shmoys [53] proved that any fixed value $k \ge 2$, determining whether the interval number of the given graph is at most k is NP-complete. Griggs and West [26] proved that if the maximum degree in G is $d \ge 1$, then its interval number is at most $\lceil \frac{1}{2}(d+1) \rceil$.

Halldórsson et al. showed in [33] that a union graph[3], in which the start times are *integral* corresponds to a line graph of bipartite graph. Since the problem of finding IS in a line graph can be solved optimally in polynomial time [21], IS in union graphs with integral start times can be solved optimally as well.

Bafna et al. considered in [7] the problem of finding *weighted* independent set in union graphs[4]. They proved that the problem is NP-hard even if all the weights are the same (i.e., in the unweighted case). They showed a 3.25-approximation algorithm for the weighted IS problem in proper union graphs, and a $(k-1+1/k)$-approximation algorithm for $(k+1)$-claw free union graphs[5].

Online scheduling non-splitting intervals Lipton and Tomkins considered in [42] the problem of online non-preemptive scheduling of intervals[6], using a single resource, where the objective is to maximize the resource *utilization*. When all the intervals have the same length, this problem can be solved optimally. For arbitrary interval lengths the paper presents an $O((\log \Delta)^{1+\varepsilon})$ competitive randomized algorithm for some $\epsilon > 0$, where Δ is the ratio of longest to shortest interval (that is not known to the algorithm in advance). Goldwasser considered in [24] a variant of this problem, where intervals are associated with *jobs* and each job offers a slack (the gap between its release time and the latest time it can be started), proportional to the job's processing time. He showed a simple greedy algorithm that is $(2+\frac{1}{k})$-competitive, where each job J has a slack of at least $k \cdot |J|$. Goldman et al. [23] extended the model of Lipton and Tomkins, allowing jobs to specify arbitrary slacks. They gave a 4-competitive randomized algorithm for the case where two job lengths are allowed, and a $6(\lceil \log \Delta \rceil + 1)$-competitive randomized algorithm for the general case, assuming that Δ is known to the online algorithm in advance.

Faigle and Nawijn [19] solved optimally the problem of scheduling intervals online on k identical service stations, for the model that allows preemptions, i.e., the currently serviced

[2] Clearly, any graph G of maximal degree d has a representation as a d-split interval graph: every edge (u, v) is represented by two intersecting segments, u^i and v^j, where u, v are the intervals representing the vertices u, v, respectively.

[3] See in Section 2.2

[4] See in Section 2.2

[5] See in Section 2.2

[6] that is, an interval cannot be interrupted or replaced by another interval while using the resource.

interval can be replaced by another interval. (In this case the preempted interval is considered as 'rejected').

Call admission Similar problems were studied in the area of call admission. The (non-splitting) interval scheduling problem can be viewed as a call admission problem on a line, where the objective is to maximize the number of accepted calls. The paper [6] presents a strongly $\lceil \log N \rceil$-competitive algorithm for the problem, where N is the number of nodes on the line. Garay et al. [20] considered the call control problem on the line, where the algorithm is allowed to *preempt* previously accepted calls. They gave a deterministic online algorithms for maximizing the *profit* from accepted calls in two cases: the value (or profit) of a call is constant ($O(\log N)$ competitive), and the value is given by the length of the call ($O(2g+1)$ competitive, where g is the golden ratio, that is $g = \frac{1+\sqrt{5}}{2} \approx 1.62$).

Awerbuch et al. [6] considered non-preemptive call control on *tree-like* switching networks. They showed an $\Omega(\log R)$ bound on the competitive ratio of any algorithm for calls of arbitrary duration on a *single* edge, where R is a ratio between the longest and shortest possible durations. Their proof can be used to show a lower bound of $\Omega(\log R)$ for online scheduling of non-splitting intervals, where R is the ratio between the longest and shortest intervals. Other related work deal with more generalized models of call control (see, e.g. [1, 5, 41] and the survey in [14]).

Flow Shop scheduling In general, the *flow shop* problem[7] is NP-complete even in the case of three machines ([21]). However, in the special case of the two-machine flow shop, where the objective is to minimize the makespan, a priority based ordering approach due to Johnson ([38]) yields the exact algorithm. The best known result is $O(log^2(m\mu)/\log\log(m\mu))$-approximation algorithm, where μ is the maximum number of operations per job, and m is the number of machines ([48, 51]). Hall developed in [27] a PTAS for the case where m is a fixed constant.

Coupled-task scheduling and scheduling with time lags The problem of scheduling splitting intervals (with two segments) was considered in the area of offline machine scheduling with the objective of minimizing the makespan. In this case, every input interval must be scheduled. Very little research has been conducted on this type of scheduling problem, called *coupled-task scheduling*. Orman and Potts in [44] showed that in general case this problem is unary NP-hard, and gave polynomial time algorithms for several restricted cases with assumptions on the segments and break lengths. Shapiro ([50]) gave simple heuristics and discussed numerous practical situations where the problem arises.

The time lags problems are relaxed versions of the corresponding coupled-tasks problems as they involve only a lower bound on the interval between the tasks. Rinnooy Kan showed

[7]See in Section 3.3.

in [46] that the permutation version of flow shop for two machines with time lags is solvable in $O(n \log n)$. Dell'Amico [18] proved some hardness results for two-machine shop problems with time lags. Brucker et al. considered in [15] also negative time lags.

Multimedia systems In the context of multimedia systems, several works considered the problem of allocation of server capacity to video programs, where the data is transmitted to the clients using broadcast (see, e.g., [2] and [4]). Also, some recent papers describe experimental studies, in which VCR-like operations can be used by the clients (see in [11, 17, 43, 55]); however, these studies focus on the efficient use of system resources, while supporting such operations, rather than the theoretical scheduling problem.

1.4 Main Results

We focus in this work on the IS problem in the class of 2-split interval graphs, where each interval consists of at most two segments. We consider both the offline and the online variants of the problem. In the offline case, the algorithm knows the whole input sequence in advance. In this case

- We show that the problem is *APX-complete*, even if all segments are of equal length.
- We give an $O(min(\log R, \log n))$-approximation algorithm, where R is the maximal length of any segment in the input.
- For the case where the number of distinct segment lengths is a fixed constant, we develop a $(4, \epsilon)$ bi-criteria PTAS, which finds for any $\epsilon > 0$ an IS of at least 1/4 of the optimal size, and a schedule in which the maximum delay is ϵ times the optimal makespan.

In our study of online algorithms we use competitive analysis [14]: the performance of an online algorithm is measured against an optimum offline algorithm, which knows the sequence of requests and the splitting times in advance. In the online problem, we distinguish between two models, depending on the information available to the scheduler upon arrival of an interval (the *complete information* and the *partial information* models).

- When the segment lengths and break times are known
  - We show that any deterministic algorithm has a competitive ratio $\Omega(R)$; a Greedy algorithm achieves this ratio.
  - We give a randomized $O(log^2 R)$-competitive algorithm.

- If R is unknown to the algorithm in advance, we use ideas similar to those of Lipton and Tomkins [42] to derive a randomized $O((\log R)^{2+\epsilon})$-competitive algorithm.

- When the scheduler knows only the *overall* length (including the break) of each arriving interval, but the length of each segment is unknown, we show a lower bound of $\Omega(R)$ on the competitive ratio of any (deterministic or randomized) algorithm. A matching upper bound is achieved by two deterministic algorithms for the problem.

- When the length of each segment is known, but the break length of each interval is unknown, we show that any (deterministic or randomized) algorithm has a competitive ratio $\Omega(D)$, where D is the maximal possible break length. A deterministic Greedy algorithm is shown to achieve the ratio $O(D)$ when $D \geq R$; when $D = o(R)$ we show an upper bound of $O(D \log R)$.

Our algorithm in Section 5.1 uses the greedy approach similar to algorithm given in [9], and our algorithms in Section 5.2 draw some ideas from [23].

1.5 Organization of this Work

In Chapter 2 we give some definitions and describe the performance measures used in our study. Then we present an observation that emphasizes the hardness of the problem of finding IS in split interval graphs, and how this problem differs from the problem of finding IS in interval graphs. In Section 2.4 we show that our problem is APX-complete.

In Chapter 3 we present our results for the offline case. We start with some simple algorithms in Section 3.1. In Section 3.2 we derive an $O(\log R)$ approximation algorithm. And in Section 3.3 we present a bi-criteria PTAS for the problem.

In Chapter 4 we engage in the online case. In Section 4.1 we present randomized algorithms for the complete information model. In this model, upon arrival of an interval, the scheduler knows the endpoints of the corresponding segments. In Section 4.2 we consider the model of partial information, where the scheduler does not know some of the interval endpoints. We derive lower bounds and show matching upper bounds for several cases.

In Chapter 5 we present some extensions, such as scheduling with delays. We conclude in Chapter 6 with summary and open problems.

Chapter 2

Preliminaries

2.1 Performance Measures

In the offline case we give approximation algorithms for our problem, and a bi-criteria approximation scheme. Denote by OPT an optimal offline algorithm.

Definition 2.1 *The* approximation ratio *of an offline deterministic algorithm $\mathcal{A}$ is r if for every finite input instance I,*

$$\frac{\mathcal{A}(I)}{OPT(I)} \geq \frac{1}{r},$$

where $\mathcal{A}(I)$ and $OPT(I)$ are the profits of $\mathcal{A}$ and the optimum on I.

Definition 2.2 *The* approximation ratio *of an offline randomized algorithm $\mathcal{A}$ is r if for every finite input instance I,*

$$\frac{E[\mathcal{A}(I)]}{OPT(I)} \geq \frac{1}{r},$$

where $E[\mathcal{A}(I)]$ is the expected profit of $\mathcal{A}$ and $OPT(I)$ is the profit of the optimum on I.

Definition 2.3 *A* polynomial time approximation scheme (PTAS) *for an optimization problem Π is an algorithm $\mathcal{A}$ which takes as input both the instance I and an error bound ϵ, has the performance guarantee*

$$R_{\mathcal{A}}(I, \epsilon) \leq (1 + \epsilon)$$

and runs in time polynomial in $|I|$.

In Chapter 3 we develop a *bi-criteria PTAS* for our problem, in which rather than obtaining an approximation with respect to one measure, we allow our solution to yield an approximation with respect to two measures. Formally,

Definition 2.4 *An* (β, ϵ) bi-criteria PTAS *is a PTAS which is an* β *approximation in one optimization criterion, and an* $(1 + \epsilon)$ *approximation in the other.*

In the online case, we use competitive analysis ([14]) to obtain performance bounds for the proposed algorithms.

Definition 2.5 *An online deterministic algorithm* $\mathcal{A}$ *is* c-competitive *if there is a constant* α *such that for all finite input sequences* σ,

$$\mathcal{A}(\sigma) \geq \frac{1}{c} OPT(\sigma) - \alpha,$$

where $\mathcal{A}(\sigma)$ *and* $OPT(\sigma)$ *are the profits of* $\mathcal{A}$ *and the optimum on* σ.

Definition 2.6 *An online deterministic algorithm* $\mathcal{A}$ *for some problem* P *is* strongly c-competitive *if any online deterministic algorithm for* P *is at least c-competitive.*

Definition 2.7 *A randomized online algorithm* $\mathcal{A}$ *is* c-competitive *if there is a constant* α *such that for all finite input sequences* σ,

$$E[\mathcal{A}(\sigma)] \geq \frac{1}{c} OPT(\sigma) - \alpha,$$

where $E[\mathcal{A}(\sigma)]$ *is the expected profit of* $\mathcal{A}$ *on* σ *and* $OPT(\sigma)$ *is the profit of the optimum on* σ.

Definition 2.8 *An online randomized algorithm* $\mathcal{A}$ *for some problem* P *is* strongly c-competitive *if any online (deterministic or randomized) algorithm for* P *is at least c-competitive.*

2.2 Graph-theoretic Definitions

In the next chapter we treat several sub-classes of graphs.

Definition 2.9 *A conflict graph of splitting intervals, in which the maximal number of segments composing an interval is at most* t*, is called a* t-split interval graph*. (See Figure 1.1). If* $t = 1$*, then the graph is an interval graph.*

Definition 2.10 *Given a t-split interval graph* G*, for some* $t \geq 1$*,* G *is called* proper*, if no segment contains properly another segment.*

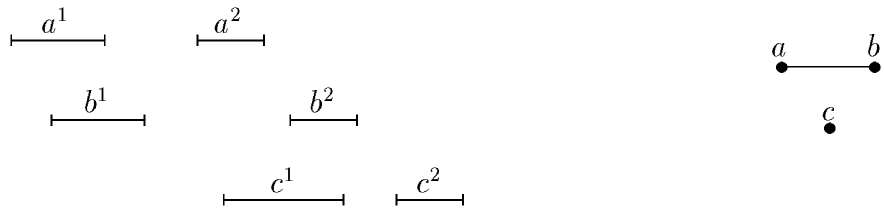


Figure 2.1: An example of an instance of splitting intervals and a corresponding 2-split union graph.

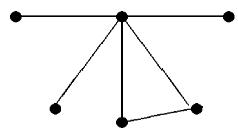


Figure 2.2: An example of a 5-claw free graph: the largest claw is of size 4.

Definition 2.11 *A graph G is called a t-split* union *graph, if there exists an interval graph $G' = (V', E')$, in which any vertex v is represented by a set of vertices $v_1, \ldots, v_{r_v}$, and there is an edge $(u, v) \in E$ iff there exists an edge $(u_i, v_i) \in E'$ for some $1 \leq i \leq \min\{r_v, r_u\}$. (See Figure 2.1).*

In this work we consider only *2-split* interval graphs and *2-split* union graphs, i.e., $t = 2$ in the above definitions. For short, we call these graphs *splitting interval graphs* and *union graphs*, respectively.

W.l.o.g., we assume that each interval in the input, I, consists of exactly two segments, I^1, I^2 (otherwise, if some interval consists of only one segment, we can add to it a segment of zero length). We call the two segments that compose the interval I *buddies*. We denote by R the ratio between the longest to shortest positive segments, or the length of the longest segment if we set the shortest segment to be of length 1, and by D a maximal break length.

Definition 2.12 $(k+1)$-claw *is a subgraph consisting of a center vertex adjacent to $k+1$ mutually non-adjacent vertices. A graph is called* $(k+1)$-claw free *if it does not contain an m-claw, where $m \geq k+1$. (See Figure 2.2).*

Note that any 2-split interval graph is $(2R+3)$-claw free.

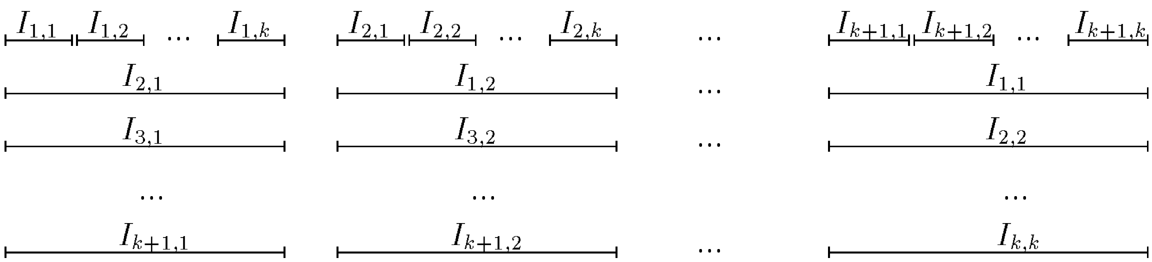


Figure 2.3: An example of a 2-split interval graph, in which every vertex has k independent neighbors.

Definition 2.13 *A graph G is* d-inductive *if the nodes of G can be numbered so that each node has at most d edges to higher-numbered nodes.*

Note that every d-inductive graph can be colored with $d + 1$ colors.

2.3 Finding a Maximum IS in Splitting Interval Graphs

The problem of finding maximum IS in interval graphs can be solved in polynomial time. The solution is based on the following property: every interval graph contains a simplicial vertex, i.e., a vertex, whose adjacency set is a clique [22]. However, in split interval graphs this property does not hold.

Observation 2.1 *For any $n \geq 2$ there exists a 2-split interval graph, where $|V| = n$, in which every vertex has $\Omega(\sqrt{n})$ independent neighbors, i.e., for each vertex v there exists a subset of the adjacency set of v of size $\Omega(\sqrt{n})$, which forms an independent set.*

Proof: For a given $n \geq 2$, let $k = \lfloor (\sqrt{4n+1} - 1)/2 \rfloor$. We show how to construct 2-split interval graph, in which every vertex has k independent neighbors. We construct the graph from $(k + 1)$ subsets of intervals; each subset consists of k intervals, and each interval is composed of two segments. We denote the jth interval in subset i by $I_{i,j}$.

The graph is constructed as follows: proceeding from left to right, we place under the intervals of subset i, $I_{i,1}, \ldots, I_{i,k}$ the ith intervals of subsets $1, \ldots, k + 1$, i.e., $I_{1,i}, \ldots, I_{k+1,i}$, excluding $I_{i,i}$. This is repeated for $i = 1, \ldots, k$. Finally, under the intervals of the $(k+1)$-th subset, we place the intervals $I_{i,i}$, $1 \leq i \leq k$ (see Figure 2.3).

Thus, we get that any interval $I_{i,j}$ with $i \neq j$, intersects k non-intersecting intervals of subset j, and, $I_{i,i}$ intersects k non-intersecting intervals of subset $k + 1$.

Note, that since $k(k + 1) \leq n$, we may get some remaining intervals, which were not included in the subsets. We can place each such interval I under any of the subsets i, $1 \leq i \leq k + 1$, providing that interval k independent neighbors. ∎

2.4 Hardness Result

The next theorem shows that unless $P = NP$ there is no PTAS for the IS problem in t-split interval graphs, already for $t = 2$ and the special case where the graph is proper, even when all the segments are of the same length.

Theorem 2.2 *The IS problem is APX-complete on a proper* 2*-split interval graph, even if all the segments are of equal length.*

Proof: Griggs and West [26] showed, that any graph with a maximum degree d, can be represented as a splitting interval graph G, where each vertex is represented by at most $\lceil \frac{1}{2}(d+1) \rceil$ segments. Moreover, G is proper, i.e., no segment contains another segment, and we can take segments of equal lengths. On the other hand (see, e.g., in [12, 45]) for any $d \geq 3$, finding a maximum IS in a graph of maximum degree d is APX-complete. This yields the statement of the theorem, by choosing $d = 3$. ∎

Chapter 3

The Offline problem

3.1 Simple Algorithms

Observation 2.1 shows that the greedy algorithm used for finding a maximum IS in interval graphs can perform poorly on split interval graphs. However, if we consider each *segment* as a different vertex in G, we get an interval graph. This suggests the following simple algorithm for our problem.

Naive Algorithm:
Given an instance of splitting intervals, construct a conflict graph, G, such that every segment is represented by a vertex in G.
Find an IS in G by collecting simplicial vertices.

Indeed, in selecting an IS in the transformed graph, we need to guarantee, that the two vertices representing the same interval in the original graph will be selected. An easy

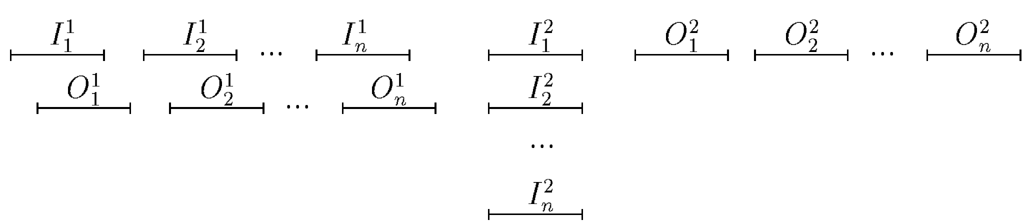


Figure 3.1: An instance in which the Naive algorithm has an approximation ratio of n. I_j are the intervals chosen by the Naive algorithm, and O_j the are intervals chosen by OPT.

example shows, that the algorithm may select for the IS segments, whose buddies intersect. In Figure 3.1 we show an instance in which the Naive algorithm has an approximation ratio of n, since we can choose any n, the Naive algorithm has an unbounded approximation ratio.

The next algorithm overcomes this problem.

Sort-and-Select Algorithm:

Order the input intervals from left to right,
by the right endpoint of the first segment.
Proceeding from left to right
While $G \neq \emptyset$
Select the interval I whose left segment ends first in the current graph
Add the two segments of I to the IS
Omit from G any interval that intersects with I.

Theorem 3.1 *The approximation ratio of the Sort-and-Select Algorithm is 3, if G is a proper 2-split interval graph, and k, if G is $k+1$-claw free.*

Proof: If G is a proper 2-split interval graph, then each segment conflicts with at most two independent intervals. When the algorithm chooses the first segment, I^1, of some interval, I, since the adjacency set of I^1 forms a clique, any optimal algorithm, OPT, can select one interval instead of segment I^1. Also, instead of the second segment, I^2, OPT can select two intervals which are independent neighbors of I^2. Hence, the approximation ratio is 3.

When the graph is $k+1$-claw free, OPT can choose at most k independent neighbors of an interval chosen by our algorithm.

■

Corollary 3.2 *The approximation ratio of the Sort-and-Select Algorithm is $R+2$, where R is the ratio between the longest to shortest segment.*

Proof: OPT can gain one interval instead of the first segment of interval I scheduled by the Sort-and-Select, and at most $R+1$ intervals instead of the second segment of I (in this case, I^2 is of length R, and the optimum schedules segments of length 1 instead). ■

Corollary 3.3 *The approximation ratio of the Sort-and-Select Algorithm is 4, if G is a 2-split union graph, and the ratio between the longest and shortest second segment is 2.*

Proof: For every interval I chosen by the algorithm, OPT can choose one interval instead of I's first segment, and at most three intervals, whose second segments conflict with I's second segment. ■

3.2 The Bucketing Algorithm

Let G be a 2-split interval graph, and let R denote the ratio between the longest to shortest segments. For convenience, assume that R is an integral power of two. We partition the intervals into $\log R$ sets, that we call *buckets*: bucket i contains all the intervals, I, in which the length of the second segment, I^2, is $\frac{R}{2^{i+1}} \leq |I^2| < \frac{R}{2^i}$, $1 \leq i \leq \log R - 1$, and $\frac{R}{2} \leq |I^2| \leq R$ in bucket 0. Therefore, in each bucket the ratio between the longest and shortest second segments is at most 2.

Note that we can represent the input intervals as $4n$ endpoints on the line, where n is the number of vertices in G. The length of each segment is the number of endpoints that lie between its left and right endpoints plus one[1]. Therefore, the maximal possible length of a segment is $4n - 1$, and the number of buckets is $B = min\{\log R, \log(4n - 1)\}$.

We now describe an algorithm that uses the above partition to buckets for finding an IS in G. The algorithm OffBA operates as follows. In each bucket i we construct a union graph H_i; we apply the Greedy algorithm to find an IS in H_i, then we construct $G[IS_i]$ a subgraph induced by the above independent set in G. Since $G[IS_i]$ is 3-inductive [2], we can color it with at most four colors. Then we select the largest color class I_i in each bucket $0 \leq i \leq \log R$, and finally, we take the set I_i of maximal size.

The following is a pseudocode of the algorithm:

Offline Bucketing Algorithm(OffBA):

Let $G = (V, E)$ be a 2-split interval graph.
for $i = 0$ to $B - 1$ do
  Let $H_i = (V_i, E_i^u)$ be the 2-split union graph
    derived from the subgraph of bucket i.
  Find an independent set IS_i in H_i using the Greedy algorithm.
  Let $G[IS_i] = (IS_i, E_i)$ be a subgraph induced by IS_i in G.
  Let $j = |IS_i|$.
  while $IS_i \neq \emptyset$ do
    Choose $v \in IS_i$, such that $deg(v) \leq 3$.
    $v_j \leftarrow v$
    $j \leftarrow j - 1$
    Remove v from $G[IS_i]$.
  end while
  Color the vertices $(v_1, v_2, \ldots, v_{|IS_i|})$ first-fit.
  Let I_i be the largest color class for bucket i.
end for
Choose the largest set among $I_0, \ldots, I_{B-1}$.

[1] Thus, if no segments intersect the segment I^i, $i \in \{1, 2\}$, then $|I^i| = 1$.

[2] See Definition 2.13 in Section 2.2

Theorem 3.4 *The approximation ratio of* OffBA *is* $O(min\{\log R, \log n\})$, *and its time complexity is* $O(m + n \log n)$.

We use in the proof the next Lemma due to Halldórsson [32].

Lemma 3.5 *Let* I *be an IS in a union graph. Then,* I *induces in the corresponding split interval graph a 3-inductive subgraph.*

Proof of Theorem 3.4: Let G_i denote the induced subgraph of G, which contains all the vertices from bucket i. By Corollary 3.3, $|IS_i|$ is at least 1/4 the size of a maximum independent set in H_i, the 2-split union graph derived from G_i. From Lemma 3.5, the subgraph induced by IS_i in G, $G[IS_i]$, is 3-inductive, and, therefore, from [37], the minimum degree greedy algorithm colors $G[IS_i]$ with at most four colors. Hence, $|I_i| \geq \frac{|IS_i|}{4}$, and I_i is a 16-approximate independent set for G_i. Let i^* be the index of the bucket, such that I_{i^*} is chosen by OffBA, then the size of an optimal IS, IS_{OPT}, satisfies

$$|IS_{OPT}| \leq 16 \cdot \sum_{i=0}^{B-1} |I_i| \leq 16 \cdot B \cdot |I_{i^*}|.$$

The construction of a union graph requires to examine every edge in the split graph, therefore, it requires $O(m)$ steps. Finding an IS in the union graph using the Greedy algorithm takes $O(n \log n)$, since we need to sort the intervals. Using the fact that the number of edges in the subgraph induced by IS_i, $G[IS_i]$, is linear in the number of vertices [32], finding an induced subgraph in G is linear in n, and the coloring takes $O(n \log n)$ if we use a heap (we store the vertices in a heap sorted by their degrees, therefore, finding a vertex v with $deg(v) \leq 3$ takes $O(1)$, and its removal takes $O(\log n)$). Hence, the overall complexity of the algorithm is $O(m + n \log n)$.

∎

3.3 A $(4, \varepsilon)$ bi-criteria Approximation Scheme

Assume now, that the number of distinct segment lengths is some fixed constant. In this section we derive a 4-approximation to the optimal solution for the IS problem, by allowing some delays in the schedule. Using these delays, our PTAS finds an *optimal* solution for the above problem on union graphs. Let c_1 and c_2 denote the number of distinct first and second segment lengths, respectively.

We use the relation of our problem to the *flow shop* scheduling problem. The input for this problem is a set of n jobs $J_1, ..., J_n$ that need to be processed on a given set of m machines $M_1, ..., M_m$. Each job, J_j, consists of a sequence of m operations $O_{1,j}, ..., O_{m,j}$ that need to be processed in this order: $O_{i,j}$ must be processed without interruptions on machine M_i for $p_{i,j}$ time units. Each machine can process at most one operation at a time, and each job may be processed by at most one machine at any time. For a given schedule, let C_j be the completion time of the job J_j. The objective is to minimize the *maximum completion time*, $C_{max} = max_j C_j$, known as the *makespan*.

If the number of operations that can be processed simultaneously on the machine M is unbounded, we call M a *non-bottleneck machine*.

Any instance of the problem of scheduling splitting intervals can be viewed as an instance of the *flow shop* problem: let t be the split number of the given instance. Each interval is a job, and each of its segments is an operation of the job. In addition, we simulate the breaks with operations of the same length that must be processed on non-bottleneck machines. Similarly, to include the release time, r_j, of the j-th interval, we add an operation $O_{1,j}$ to the job J_j, whose length equals to r_j, and the machine M_1 is a non-bottleneck machine. Thus, an interval I_j that consists of t segments will be represented by the job J_j that has $2t$ operations.

Example 3.3.1 *Let I be an interval from an input sequence to the problem of scheduling splitting intervals. I contains two segments. The release time of I is $r_I = 1$, $|I^1| = 2$, $|I^2| = 3$, and the break of I is of length 4. The corresponding job, J, in the input to the flow shop problem consists of four operations each to be processed on different machine. $|O_1| = 1$ simulating the release time of I. $|O_2| = 2$, $|O_4| = 3$, representing the segments. And $|O_3| = 4$, simulating the break. The machines M_1 and M_3 are non-bottleneck machines.*

In the problem of scheduling splitting intervals, each interval has a due date, that equals to its release time plus the sum of its processing times and break times. To simulate the due dates we define a *delivery time*, q_j, for each job, J_j. Following an approach of [28] and [40], we define $q_j = -d_j$, where d_j is a due date of J_j (i.e., the due date of the corresponding interval). Each job, J_j, gets an additional operation $O_{(2t+1),j}$, where $p_{(2t+1),j} = q_j$, and M_{2t+1} is a non-bottleneck machine. Our objective now is to minimize the maximum *delivery completion time*, which equals to $max_j\{C_j + q_j\} = max_j\{C_j - d_j\}$. Therefore, minimizing the maximum delivery completion time is equivalent to minimizing the maximum *lateness* of any job, $L_j = C_j - d_j$. Hence, our objective can be viewed now as minimization of $L_{max} = max_j L_j$.

However, note that the q_j's are negative values, therefore, we use the following transformation on the due dates: $\tilde{d}_j = d_j - D$; the value of D will be determined (see below), such that for any j, $\tilde{d}_j \leq 0$; by setting $q_j = -\tilde{d}_j$ we get that all the processing times are positive.

Hence, each job has $2t+1$ operations (i.e., $m = 2t+1$), the machines M_{2i+1}, $1 \leq i \leq t$, are non-bottleneck machines, and the objective is to minimize the maximum lateness. We can use as procedure a PTAS for finding a $(1+\varepsilon)$-approximation for the makespan in the flow shop scheduling problem (see, e.g., [27]).

We now return to 2-split interval graphs. Each interval is represented as a set of five operations, thus we can describe the j-th interval in the input as a vector with four entries $(t_{j,1}, t_{j,2}, t_{j,3}, t_{j,4})$, where $t_{j,1}$ is the release time, $t_{j,2}$ and $t_{j,4}$ are the lengths of the first and the second segments respectively, and $t_{j,3}$ is the break length. Note, that for each j, q_j can be calculated from the values of $t_{j,1}$, $t_{j,2}$, $t_{j,3}$ and $t_{j,4}$.

Suppose that we know the makespan of an optimal schedule, i.e., the time in which the last interval in this schedule terminates. Denote this value by D. Following the technique of Jansen et al. [36], we divide all processing and release times by D. Thus, we get that $0 \leq t_{j,i} \leq 1$ for $1 \leq i \leq 4$, $1 \leq j \leq n$. Next, we use a technique by Hall and Shmoys [28] and round each release time down to the nearest multiple of $\frac{\varepsilon}{4}$, and round up each break time to the nearest multiple of $\frac{\varepsilon}{4}$, for some value $\varepsilon > 0$. Now there are only $4/\varepsilon$ possible values for $t_{j,1}$ and $t_{j,3}$, and there are $(\frac{4}{\varepsilon})^2 \cdot c_1 c_2$ possible vectors $(t_{j,1}, t_{j,2}, t_{j,3}, t_{j,4})$.

Note that $L_{max} = max_j\{C_j - \tilde{d}_j\} = max_j\{C_j - d_j + D\}$, and since in an optimal schedule there are no "late" jobs, the minimal lateness is $L^*_{max} = D$.

We now summarize the steps of our approximation scheme, which gets as parameter some $\epsilon' > 0$.

The Scheme:

1. Guess the value of D,
   i.e., starting from the interval with latest completion time in the input,
   choose an interval to be the last to complete in an optimal schedule.
2. Guess $\mathcal{O}$, the number of intervals scheduled by OPT.
3. Guess the subset $S_{\mathcal{O}}$ of intervals of size $\mathcal{O}$ scheduled by OPT.
4. Run a PTAS for minimizing the makespan on the flow shop instance generated by the subset $S_{\mathcal{O}}$, and calculate L_{max}.
   If $(L_{max} \leq (1+\epsilon')\ L^*_{max})$ or (no more subsets $S_{\mathcal{O}}$), goto 1,
   else goto 3.
5. Among the sets $S_{\mathcal{O}}$ of largest size for which $L_{max} \leq (1+\epsilon')\ L^*_{max}$,
   choose the set for which L_{max} is minimal.

Note that initially we have transformed the original instance, I, to another instance I', in which all values $t_{j,1}$ and $t_{j,3}$ are multiples of $\frac{\epsilon}{4}$. We add $\frac{\epsilon}{4}$ to each release time, since we have rounded down the release times of the original instance.

Denoting by $L^*_{max}(I)$, $L^*_{max}(I')$ the minimal lateness for I and I' respectively, we get that $L^*_{max}(I') \leq (1+\frac{\epsilon}{2})L^*_{max}(I)$ (since each $t_{j,1}$ and $t_{j,3}$ can delay by $\frac{\epsilon}{4}$ the completion time of an optimal schedule for I'). Thus, taking $\epsilon' = \frac{\epsilon}{4}$ in our scheme, we have

$$L_{max} \leq (1+\epsilon')L^*_{max}(I') \leq (1+\frac{\epsilon}{4})(1+\frac{\epsilon}{2})L^*_{max}(I) \leq (1+\epsilon)L^*_{max}(I).$$

Finally, we need to add $D = L^*_{max}$ to every due date, and we have that the maximum lateness of any job in our schedule equals to εC^*_{max}.

Our PTAS for the flow shop instance of $S_{\mathcal{O}}$ allows to execute in parallel the operation $O_{l,2}$ and $O_{k,4}$ of some jobs J_l and J_k, respectively. In terms of our graph, this means that we ignore possible conflicts between the first segment of some interval and the second segment of another interval. In other words, our scheme finds an IS in the *union graph* that corresponds to our original split graph. As shown in the proof of Theorem 3.4, this set is a 4-approximation of the maximum IS in the original split graph.

We now analyze the complexity of our scheme. Steps 1. and 2. take $O(n)$ each, since the number of input intervals is n. In the third step the number of possible subsets is $O(n^{\frac{c_1 c_2}{\varepsilon^2}})$. Hence, overall our scheme takes $O(n^{2+O(\frac{c_1 c_2}{\varepsilon^2})})$ steps multiplied by the complexity of the PTAS for flow shop. We summarize in the next result.

Theorem 3.6 *Given a 2-split interval graph G where the number of distinct segment lengths is a fixed constant, let $\mathcal{O}$ be the size of an optimal IS, and let C^*_{max} be the latest completion time of any interval in this set. Then for any $\varepsilon > 0$ there is a PTAS, which schedules at least $\mathcal{O}/4$ intervals in G, such that each interval is late by at most εC^*_{max}.*

Chapter 4

Online Scheduling Splitting Intervals

4.1 The Complete Information Model

We now turn to study the online version of our problem, where the set of input intervals is not known to the scheduler in advance. We first discuss the *complete information model*: the intervals arrive in an online fashion, but the scheduler knows the endpoints of the corresponding segments. As before, we denote by R the ratio between longest to shortest segments; let k be the claw number of the graph.

4.1.1 Deterministic Algorithms

Recall that the Greedy algorithm schedules every arriving interval if it does not conflict with any of the previously scheduled intervals.

Theorem 4.1 *The Greedy algorithm is strongly* $min\{2R+1, k\}$ *competitive in the complete information model.*

Proof: Let m be a minimum between $2R+1$ and k. For any interval, I, scheduled by Greedy OPT can schedule at most m other non-conflicting intervals (at most R intervals instead of the first segment of I and at most $R+1$ instead of its second segment). Hence, the competitive ratio of the algorithm is m.

On the other hand, m is a lower bound for every deterministic algorithm. We now describe an adversary, that forces any online deterministic algorithm to achieve at least this bound. While the algorithm $\mathcal{A}$ does not schedule any intervals, the adversary presents to $\mathcal{A}$ intervals whose both segments are of the maximal length. When $\mathcal{A}$ chooses an interval to schedule, the adversary presents m intervals of the minimal length, that conflict with the interval scheduled by $\mathcal{A}$. $\mathcal{A}$ cannot schedule these small intervals, while OPT can, and the gain of OPT is m. We can continue with this scenario, hence the competitive ratio of any deterministic algorithm is at least m. ∎

4.1.2 Randomized Algorithms

In this section we study two randomized algorithms for our problem: the algorithm *Select-from-Barrel* assumes that we know a-priori the value of R, while the algorithm *HalfLengths* does not assume such knowledge.

We first present an algorithm for scheduling intervals, in which the first segment is of length 1 and the second of length l, for some $l \geq 1$ ($(1,l)$ intervals).

Virtual Select (VS):

Upon arrival of an interval I
  If I does not conflict with a previously scheduled interval
    Flip a fair coin
    If Heads: Take the interval.
    If Tails: Virtually take the interval, but do not work[1].
      Take no intervals whose long segment intersects I's long segment.

Theorem 4.2 *The algorithm* VS *is 6-competitive.*

Proof: We associate a *bucket* with each segment in the optimal schedule. Then we assign every segment of a feasible schedule to some bucket, under the condition that no part of any segment can be placed in more than one bucket. We show, that the expected weight of VS on each first segment bucket is $\frac{1}{6}$, and since the optimal weight is 1, VS is 6-competitive. (Note, that since any algorithm gains only 1 if it schedules both segments of some interval, we need not compute the weight of VS on buckets of second segments).

We now show how to assign segments to buckets. We say that the segment s_1 *blocks* the segment s_2 if s_1 and s_2 intersect, and s_1 starts before s_2. Each segment of the optimal schedule belongs to its own bucket. Recall that I^1, I^2 denote the first and the second segment of an interval I, respectively. Suppose that I^1 is not included in the optimal schedule (for short, I^1 is *non-optimal*), then I^1 is assigned to a bucket B, if it *blocks* the segment B. If I^2 is non-optimal, then I^2 is assigned to a bucket B, if (i) I^2 *intersects* the segment B, and (ii) B is not properly contained in I^2. Each of these non-optimal segments contributes 1/3 to any bucket to which it belongs. (Note that any non-optimal first segment can be assigned to only one bucket, and any non-optimal second segment can be assigned to at most two

[1] I.e., the resource is not taken, and in certain circumstances some other interval can be scheduled.

buckets. Thus, assigning the weight 1/3 to the segment I^1 or I^2, in each bucket to which it belongs, guarantees that the overall contribution of the interval I is at most 1). Each optimal segment contributes 1 to its own bucket. We say that the resource is *available* for some interval I, if the scheduler can consider I for scheduling (i.e., either the resource is free, or it was virtually taken by another interval, but VS allows to schedule I).

Now, we show that the expected weight of VS on each first segment bucket is at least 1/6. Let O be an interval included in some optimal schedule. Denote by O^1, O^2 the first and second segments of O, respectively. We partition the set of feasible schedules into disjoint groups of schedules, based on the treatment of the interval O in the schedule (see Figure 4.1):

1. In this group of schedules, O does not intersect with any interval for which the resource was previously available (Figure 4.1(a)); thus, the resource is available for O. VS schedules O with probability 1/2, it contributes 1 to its own first bucket, and the expected weight on this bucket is 1/2.

2. This group consists of all schedules in which the resource is available for an interval I, such that I^1 blocks O^1 (Figure 4.1(b)). I^1 belongs to the O^1 bucket, VS schedules it with probability 1/2, and it contributes 1/3 to the bucket. Hence, the expected weight of VS in O's buckets is at least 1/6.

3. This group of schedules consists of all schedules in which the resource is available for some interval, I, such that I^2 intersects O^2 (Figure 4.1(c)). Then VS schedules I with probability 1/2; it contributes 1/3 to the bucket of O^2, and the expected weight on O's buckets is at least 1/6.

4. This group consists of all schedules in which the resource is available for some interval I, such that I^2 intersects O^1, and O^1 is not properly contained in I^2 (Figure 4.1(d)). I is scheduled with probability 1/2, and it contributes 1/3 to the bucket of O^1. Hence, the expected weight on this bucket is 1/6.

5. In this group of schedules the resource is available for some interval I such that O^1 is properly contained in I^2 (Figure 4.1(e)). Now, VS virtually takes I with probability 1/2. Note that

   - the resource is not available for any interval whose second segment intersects I^2,
   - there is no interval $\hat{I}$ for which the resource is available, such that $\hat{I}^1$ *blocks* O^1 (otherwise, this schedule would belong to group 2), and
   - before we scheduled I, there was no interval J, such that J^2 intersects O^2 (otherwise, this schedule would belong to group 3).

   From the above we get that the resource is available for O; O is scheduled with probability 1/2, and we have at least the weight 1/4 on O's buckets.

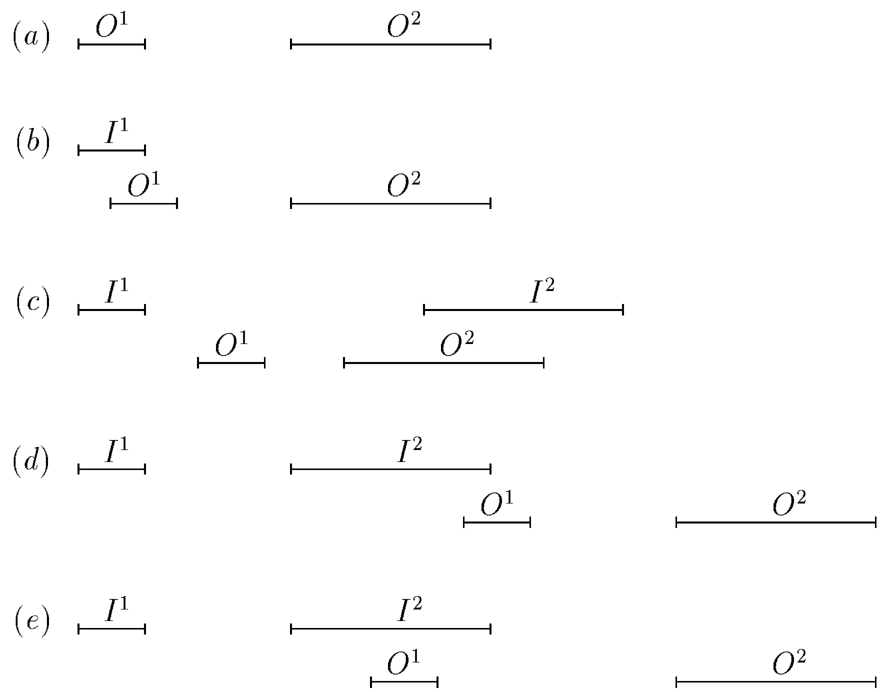


Figure 4.1: The disjoint groups of schedules in the proof of the Theorem 4.2.

Each feasible schedule that could be chosen by VS falls into one of the above groups. Assume that a schedule from group i occurs with probability p_i, for some unknown p_i, where $\sum_{i=1}^{5} p_i = 1$. Then, the gain of VS on the first segment bucket of O is at least $\frac{1}{6}\sum_{i=1}^{5} p_i = \frac{1}{6}$.

This completes the proof.

∎

Corollary 4.3 *If the length of the first segment is in the range* $[1, 2]$, *and the second segment is in the range* $[l/2, l]$, *then the algorithm* VS *is 10-competitive.*

Proof: In this case every first segment can block 2 segments. We modify the assignment of the second segments as follows: a non-optimal segment I^2 is assigned to a bucket B, if (i) I^2 intersects the segment B, and (ii) B is a second segment, or B is a first segment and it is not properly contained in I^2. Now, every non-optimal second segment can be assigned to at most three buckets. In addition, each first segment can belong to at most two buckets. Therefore, overall the segments of an interval, I, can be assigned to at most five buckets. Thus, each segment now contributes the weight $1/5$ in any bucket to which it belongs. Therefore, the minimum weight of VS on a bucket is now $\frac{1}{10}$. ∎

Now we present another algorithm for scheduling splitting intervals of the type $(1, k)$. Suppose we can schedule the intervals on two servers (i.e., the capacity of the resource is 2).

Double Select (DS):

Upon arrival of an interval I

If there is a resource available for I

Choose randomly one of the available resources;

Schedule the interval on this resource;

On the other resource schedule no interval whose long segment intersects I's long segment.

Lemma 4.4 *The competitive ratio of* DS *(when the optimal algorithm has only one server) is 2.5.*

Proof: For any $t \geq 1$, consider the first two intervals, I_1 and I_2, scheduled by DS at time t or later. Any optimal algorithm, OPT, can schedule at most 4 independent intervals, whose second segments intersect I_1^2 and I_2^2. Note that I_1^2 and I_2^2 do not intersect (as required by DS); therefore, OPT can schedule an interval, I, that is not scheduled by DS, only if I intersects both I_1^1 and I_2^1; otherwise, DS could schedule I on one of the two servers. Since all the first segments are of length 1, there can be only one such interval, I, whose arrival time is t or later. This gives the desired ratio. ∎

Unfortunately, we have only one server, so we randomly choose one. The expected gain in this case is 1/2 of the gain of DS.

Theorem 4.5 *The competitive ratio of the algorithm that randomly chooses one server and applies* DS *is 5.*

Corollary 4.6 *If the length of the first segment is in the range* $[1, 2]$*, and the second segment is in the range* $[l/2, l]$*, then the algorithm* DS *is 8-competitive.*

Proof: The proof is similar to the proof of Lemma 4.4. As before, we take the first pair of intervals scheduled by DS at time t or later. Now, OPT can take at most 6 independent intervals whose second segments intersect I_1^2 and I_2^2. Since (by the algorithm DS) I_1^2 and I_2^2 do not intersect, OPT can schedule an interval, I, that is not scheduled by DS, only if I intersects both I_1^1 and I_2^1; otherwise, DS could schedule I on one of its servers. There can be at most two such independent intervals, whose arrival times are t or later.

∎

Lemma 4.7 *The Greedy Algorithm is 5-competitive if the length of the first segment is in the range* $[l/2, l]$*, and the second segment is in the range* $[1, 2]$*.*

Proof: OPT can schedule at most two first segments of non-conflicting intervals instead of the first segment of an interval scheduled by Greedy, and at most 3 segments conflicting with the second segment. ∎

Now, we are ready to handle the general case. We present an $O(\log^2 R)$ competitive algorithm, which knows R in advance.

We partition the intervals into $\log^2 R$ *barrels*: the barrel (i,j), $1 \leq i,j \leq \log R - 1$, contains all the intervals in which the length of the first segment is at least $\frac{R}{2^{i+1}}$ and less than $\frac{R}{2^i}$, and the length of the second segment is at least $\frac{R}{2^{j+1}}$ and less than $\frac{R}{2^j}$. In $(0,j)$ buckets the length of the first segment is $\frac{R}{2} \leq |I^1| \leq R$, and in $(i,0)$ buckets the length of the segment segment is $\frac{R}{2} \leq |I^2| \leq R$,

We denote by A the set of barrels in which the first segment is longer, i.e., $\forall I \in A$ $|I^1| > |I^2|$.

Select-from-Barrel (SB):
Randomly choose one barrel $(i,j)^*$.
If $(i,j)^* \in A$, then schedule the intervals in this barrel greedily;
else apply the algorithm VS on this barrel.
Reject all other intervals.

Theorem 4.8 *The algorithm* SB *is* $O(\log^2 R)$ *competitive.*

Proof: We consider separately the competitive ratio obtained for two types of barrels: for the barrels in A, the greedy schedule achieves the competitive ratio 5 from Lemma 4.7, while on the other barrels VS has the ratio 10. Since half of the barrels are in A we get that the competitive ratio of SB is $7.5 \log^2 R$. ∎

Now we describe an algorithm for the case where R is unknown in advance. Note that when R is unknown, we can still partition the intervals into barrels, choosing a barrel k with probability $p_k = \frac{1}{f_\epsilon} \frac{1}{k \cdot \log^{1+\epsilon}(k)}$, where f_ϵ is chosen such that $\sum_{k \geq 1} p_k = 1$. We get the competitive ratio $O(c_\epsilon \cdot \log^2 R \log^{1+\epsilon} \log R)$ for any ϵ, where c_ϵ is increasing as ϵ decreases to 0. For more details see in [14]. We describe below an algorithm with competitive ratio of $O((\log R)^{2+\epsilon})$. Note that while the bound of $O(c_\epsilon \cdot \log^2 R \log^{1+\epsilon} \log R)$ is asymptotically better than ours, our bound is tighter for a fixed value of R.

The algorithm *HalfLengths (HL)* operates as follows. For each arriving interval, I, we examine separately the two segments of I, $I^i, i \in \{1,2\}$. The *depth* of I^i, denoted by $depth(I^i)$, is d, if there is a set of segments S_j, $1 \leq j < d$, such that

(i) $S_1, \ldots, S_{d-1}$ intersect with I^i.

(ii) $|S_j| \leq \frac{1}{2}|S_{j-1}|, \quad 1 \leq j < d-1.$

(iii) S_j was virtually taken, $1 \leq j < d-1$.

We use for coin tosses a probability distribution derived from the *Riemann zeta function* (see, e.g., [49]), defined as follows. For a complex number $z \in \mathcal{C}$, $\zeta(z) = \sum_{n \geq 1} \frac{1}{n^z}$. Note that $\zeta(z)$ converges when $|z| > 1$, thus we choose $z = 1 + \epsilon$ for some small $\epsilon > 0$. Let

$$c_d = \frac{1}{d^{1+\varepsilon}\zeta(1+\varepsilon)}, \quad p_d = \frac{c_d}{\prod_{j=1}^{d-1}(1-p_j)} \ . \tag{4.1}$$

The value of c_d denotes the probability that a segment in depth d will be taken, whereas p_d denotes the *conditional* probability that a segment of depth d will be taken, given that $S_1, \ldots, S_{d-1}$ were not taken.

We consider scheduling I^i in depth d, only if $|I^i| \leq \frac{1}{2}|S_{d-1}|$, in which case I^i is taken with probability p_d, and *virtually* taken with probability $1 - p_d$. This guarantees that if we virtually take an interval I, then we do not consider any other interval, J, that intersects with I^i, $i \in \{1, 2\}$, unless the segment of J which intersects with I^i is at least twice shorter than I^i.

Finally, we schedule the interval I only if both I^1 and I^2 were scheduled; if one of these segments was only virtually taken, then I is virtually taken, and if any of the segments was rejected, then I is rejected.

Algorithm HalfLengths (HL):

Let I be a new interval presented to the algorithm.
  If I intersects any other previously scheduled interval, do nothing.
  For i=1 to 2 do
    Let S be a virtually scheduled segment with the highest $depth$, that intersects I^i.
    If such S exists
      If $|I^i| \ \leq \ \frac{1}{2}|S|$
        $depth(I^i) \leftarrow \ depth(S) + 1;$
      else
        reject I^i.
    else
      $depth(I^i) \leftarrow \ 1.$
    With probability $p_{depth(I^i)}$ decide to schedule I^i, otherwise decide to virtually take I^i.
  end for
  If both I^1 and I^2 were scheduled, actually schedule I.
  If at least one of I^1 and I^2 was rejected, reject I.
  Otherwise, virtually schedule I, and do not work.

Theorem 4.9 *The competitive ratio of the algorithm HalfLengths is $O((\log R)^{2+\varepsilon})$.*

Proof: Let O be some interval scheduled by OPT. There are two possible cases: if HL has an opportunity to schedule O, (i.e., HL schedules O with some probability c_O), then we are done, since the maximal depth of any segment is $\log R$; thus, $c_O \geq c^2_{\log R}$. Otherwise, HL has an opportunity to schedule some other interval I, that blocks O, i.e., I intersects O and either $|O^1| > \frac{|I^i|}{2}$ or $|O^2| > \frac{|I^i|}{2}$, $i \in \{1, 2\}$. But in this case, OPT can schedule at most 5 intervals instead of I. For HL the probability to schedule any interval (if this interval is not blocked) is at least $c^2_{\log R}$, since each unblocked segment is scheduled with probability at least $c_{\log R}$, and scheduling of different segments of the same interval is independent.

Therefore, for every interval that belongs to the optimal schedule, the gain of HL on this interval is at least $\frac{c^2_{\log R}}{5}$.

Let $OPT(\sigma)$ and $HL(\sigma)$ denote the total gains of OPT and HL, for some input sequence σ, and $OPT(O)$ and $HL(O)$ denote the gain of OPT and HL on the interval O respectively. Then,

$$E[HL(\sigma)] \geq \sum_{O \in OPT(\sigma)} HL(O) \geq \sum_{O \in OPT(\sigma)} \frac{c^2_{\log R}}{5} OPT(O) = \frac{c^2_{\log R}}{5} \cdot OPT(\sigma)$$

■

4.2 The Partial Information Model

In this model, when an interval arrives, its *break times* are unknown. We consider two cases: (i) the break length is unknown and (ii) the lengths of the segments are unknown. Let D denote the maximal possible length of a break, and let R denote the maximal possible length of any segment.

4.2.1 Lower Bounds

Theorem 4.10 *The competitive ratio of any (randomized or deterministic) algorithm in the case of unknown break length is $\Omega(D)$.*

If the lengths of segments are unknown, the competitive ratio of any (randomized or deterministic) algorithm is $\Omega(R)$.

Proof: We show the bounds using Yao's minimax principle [54].

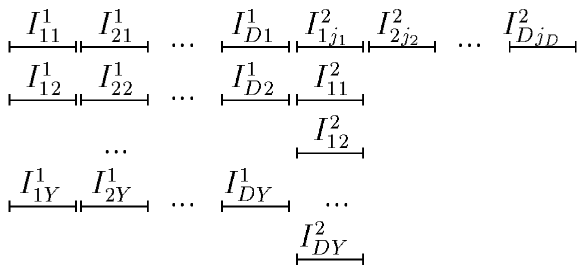


Figure 4.2: The input sequence for $\Omega(D)$ lower bound in Theorem 4.10.

1. We start with the proof of the first bound. The input contains $D \cdot Y$ intervals; each segment is of length 1. The first segments of all intervals form D cliques, each of size Y. From each clique $1 \leq c \leq D$, the adversary randomly chooses one interval I_{cj_c}, such that its break time is equal to D. The buddies of the remaining intervals from each clique arrive immediately after the end of clique D, and, therefore, intersect. (See Figure 4.2).

   The gain of OPT is clearly D. Any online deterministic algorithm $\mathcal{A}$ can choose one interval from each clique. Hence,

   $$Pr[\mathcal{A} \text{ schedules } x \text{ non-conflicting intervals}, x \leq D] = \binom{D}{x}(\frac{1}{Y})^x(1-\frac{1}{Y})^{D-x}$$

   Hence, $Y \geq D$ we get that

   $$E[\mathcal{A}(\sigma)] = \frac{D}{Y} \leq 1.$$

   In fact $\mathcal{A}$ always gains at least one interval, this implies that $E[\mathcal{A}(\sigma)] = 1$. Thus we have that

   $$\frac{E[OPT(\sigma)]}{E[\mathcal{A}(\sigma)]} = D\ .$$

2. Now we derive the second lower bound. The input consists of Y intervals with first segment of length R, and R intervals with first segment of length 1, that arrive as shown in Figure 4.3.

   The gain of OPT on such input is R. Any online deterministic algorithm chooses one of the $Y+1$ intervals, without knowing their lengths. Therefore,

   $$E[\mathcal{A}(\sigma)] = R \cdot \frac{1}{Y+1} + 1 \cdot \frac{Y}{Y+1} = \frac{R+Y}{Y+1} \longrightarrow_{Y\to\infty} 1.$$

   Hence, for random input σ, we have

   $$\frac{E[OPT(\sigma)]}{E[\mathcal{A}(\sigma)]} \geq R.$$

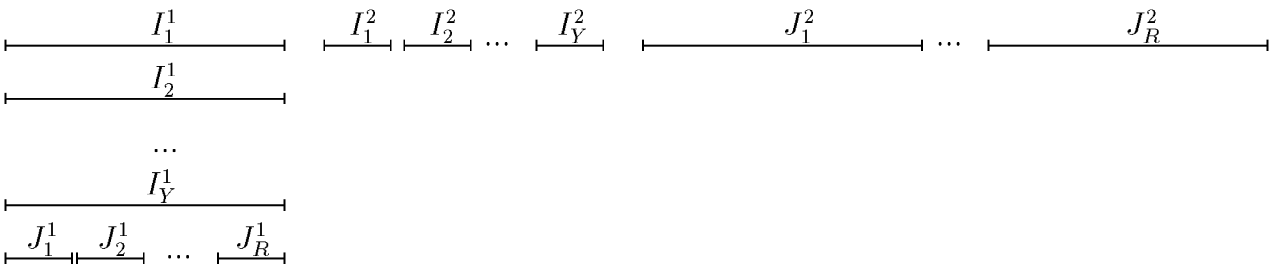


Figure 4.3: The input sequence for $\Omega(R)$ lower bound in Theorem 4.10.

■

Corollary 4.11 *If the break length or the segments lengths are unknown to the algorithm and unbounded, any algorithm has an unbounded competitive ratio.*

4.2.2 The U-Greedy Algorithm

We now derive some matching upper bounds.

Consider the following variant of the Greedy algorithm.

The U-Greedy Algorithm:
When an interval arrives, schedule it, if the resource is free.
When the interval splits,
wait for it and do not schedule any other interval during the break.

Theorem 4.12 U-Greedy *is $O(D+R)$ competitive.*

Proof: While Greedy schedules some interval, OPT can schedule at most $D+2R$ short intervals of length 1, and the theorem follows. ■

Note that if $D \geq R$ and the break length is unknown, then U-Greedy is $O(D)$ strongly competitive, and it is best possible algorithm in this model. Also, if $D \leq R$ and the lengths of the segments are unknown, then the U-Greedy algorithm is the best possible to within factor 3, since its competitive ratio is at most $3R$. In the two following sections we derive algorithms for two remaining cases: (i) the break length is unknown and $D < R$ (actually, we consider the case where $D = o(R)$, since even if $D < R$, it still can be the case where $R = O(D)$, and, therefore, U-Greedy can still be $O(D)$ strongly competitive), and (ii) the segment lengths are unknown and $R < D$ (actually, $R = o(D)$ as before). Note that if both the segment lengths and the break length are unknown, U-Greedy provides matching upper bound in both cases: $D \geq R$ and $D < R$.

4.2.3 Algorithms for Short Breaks

Now we consider the case in which the break length is unknown but bounded by some $D = o(R)$, and the segment lengths are known. In this case U-Greedy is $O(R)$ competitive, while the lower bound is $\Omega(D)$. Note that the lower bound of $\Omega(\log R)$ is still valid, and therefore, the lower bound is $\Omega(\max\{\log R, D\})$. We present an $O(D \log R)$ competitive randomized algorithm, when R is known to the algorithm in advance, and an $O(D \log^{1+\epsilon} R)$ competitive randomized algorithm, when R is not known to the algorithm.

We start with the case where R is known to the algorithm. In this case we can partition the intervals into $\log R + 1$ barrels according to their total lengths (the sum of the segment lengths). Barrel i, $1 \leq i \leq \log R$ contains all intervals of total length at least $\frac{2R}{2^{i+1}}$ and less than $\frac{2R}{2^i}$, and bucket 0 contains all the intervals of total length $R \leq |I| \leq 2R$.

Short Breaks Algorithm(SBA):
Randomly choose one barrel i^*.
When an interval $I \in i^*$ arrives
If the resource is free
Schedule I for the time $|I| + D$.
Reject all other intervals.

Theorem 4.13 *The algorithm* SBA *is* $O(D \log R)$ *competitive.*

Proof: Let I be an interval scheduled by SBA. It occupies the resource for $|I| + D$ units of time. The optimum can schedule at most $O(D)$ other non-conflicting intervals from this barrel that intersect I, since the maximal break is of length D. Therefore, the competitive ratio of the algorithm in one barrel is $O(D)$. There are $\log R + 1$ barrels, and the theorem follows. ∎

Now we consider the case when R is not known to the algorithm. In the next algorithm we slightly modify the algorithm HalfLengths, to include the unknown break length.

Algorithm Short Breaks HalfLengths (SBHL):
When an interval I arrives, and the resource is free
Let J be a virtually scheduled interval with the highest *depth* that intersects I.
If such J exists
If $|I| \leq \frac{1}{2}|J|$
$depth(I) \leftarrow depth(J) + 1;$
else
reject I.
else

$depth(I) \leftarrow 1.$
With probability $p_{depth(I)}$ schedule I for the time $|I| + D$,
otherwise virtually schedule I.

Theorem 4.14 *The competitive ratio of the algorithm* SBHL *is* $O(D \log^{1+\epsilon} R)$.

Proof: Every interval O has a probability at least $c_{\log R}$ to be scheduled, or it is blocked by some interval I of length at most $2|O|$. As before, OPT can schedule at most $O(D)$ intervals instead of I. The probability to schedule I is at least $c_{\log R}$, and the theorem follows. ∎

4.2.4 Unknown Segment Lengths

Consider now the following case. When an interval arrives the lengths of its segments are unknown (the total length of the interval is known), the length of the break is known, and $R = o(D)$. In this case U_Greedy is $O(D)$ competitive, while the lower bound is $\Omega(R)$. We describe another Greedy (deterministic) algorithm that achieves the competitive ratio of $O(R)$ both in the case where R is known and in the case where R is unknown to the algorithm in advance.

We denote the starting time of I by s_I, its break length by b_I, and its completion time by c_I. (Note that these parameters are known to the scheduler when I arrives).

Conservative Greedy Algorithm (CG):
When an interval I arrives
If the resource is free at the time intervals S=$[s_I,\ s_I + R]$ and T=$[c_I - R,\ c_I]$
Schedule I.
Do not schedule any other interval in S and T.

Theorem 4.15 *The competitive ratio of the algorithm* CG *is* $O(R)$.

Proof: Let I be some interval scheduled by the algorithm. Its length is at most $2R$, and OPT can schedule at most $2R + 1$ first segments of length 1 instead of I. In addition, after I is scheduled CG cannot schedule any first segment, J^1, at most $R - \epsilon$ units of time before the start of I^2, since CG can schedule J only if the resource is free at the time interval $[s_J, s_J + R]$; and it cannot schedule any second segment, J^2, at most $R - \epsilon$ units of time after the completion of I^2, since CG can schedule J only if the resource is free at the time interval $[c_J - R, c_J]$. OPT can schedule in these time intervals at most $2R$ intervals. Therefore, instead of I, OPT can schedule at most $4R + 1$ non-conflicting intervals.

If R is not known to CG in advance, the condition to schedule some interval I is that the resource must be free at $[s_I, s_I + |I|]$ and $[c_I - |I|, c_I]$. For every I, $|I| \leq 2R$, hence, in this case OPT can schedule at most $6R + 1$ intervals instead of I. ∎

Chapter 5

Extension to Scheduling with Delays

In this chapter we discuss a generalization of our model, in which every interval can be scheduled with some *delay*: each interval may offer a maximal possible delay, given as the *start deadline* of the first segment (the delay may also be equal to zero).

We show that with slight changes our algorithms can be applied in this model as well.

5.1 The Offline Problem

We use the following greedy algorithm (similar to [9]) to schedule intervals with delays:

Greedy with Delays(GD):
When the resource is free schedule an interval that
does not intersect with any previously scheduled interval;
If there are several such intervals
schedule the one whose first segment will complete first.

Theorem 5.1 *The approximation ratio of* GD *is 4 in proper split interval graphs, and in union graphs in which the ratio between longest to shortest second segment is 2.* GD *has the ratio $k+1$ in $(k+1)$-claw free split interval graphs.*

Proof: Let $\mathcal{G}$ be the set of intervals scheduled by GD, and let H be any feasible schedule among the intervals that are not in $\mathcal{G}$. First we show, that in proper graph and in union graph in which the ratio between longest to shortest second segment is 2, $|H| \leq 3|\mathcal{G}|$. Each interval in $\mathcal{G}$ intersects with at most 3 intervals in H, as shown in Figure 5.1. At most one interval in H intersects $I^1_{\mathcal{G}}$, since it is a simplicial vertex, and at most two intervals intersect

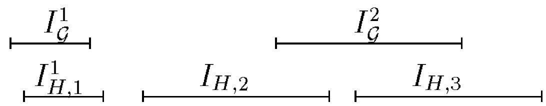


Figure 5.1: For every interval in $\mathcal{G}$ there are at most 3 intervals in H

$I^2_{\mathcal{G}}$. (In a union graph $I_{H,2}$ and $I_{H,3}$ must be second segments). There are no intervals in H that do not intersect any interval in $\mathcal{G}$, since such intervals would have been chosen by Greedy. Hence, in the above proper and union graphs we have $|H| \leq 3|\mathcal{G}|$. Similarly, in a $(k+1)$-claw free graph there are at most k intervals in H that intersect with every interval in G; thus, in such graphs $|H| \leq k|\mathcal{G}|$. In both cases, OPT can schedule at most both H and $\mathcal{G}$, and therefore $|OPT| \leq 4|\mathcal{G}|$ for the proper and union case, and $|OPT| \leq (k+1)|\mathcal{G}|$ in $(k+1)$-claw free graphs.

∎

Corollary 5.2 *The Offline Bucketing Algorithm is* $O(\min\{\log R, \log n\})$ *competitive in the model of scheduling with delays.*

5.2 The Online Problem

We start with the two length case, i.e., all the intervals are $(1, l)$ intervals, where $l \geq 1$.

We use a queue for intervals that have arrived but are still waiting to be scheduled. In the queue the intervals are given priorities by their start deadlines, i.e., the interval with the earliest start deadline has the highest priority. We say that the resource is available for some interval I, if I does not conflict any previously scheduled or virtually scheduled intervals, or, if I conflicts some previously virtually scheduled intervals, and I^2 intersects only first virtually scheduled segments.

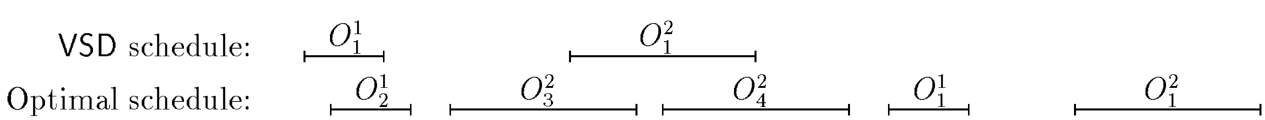


Figure 5.2: The interval O_1 of the VSD schedule belongs to the buckets of the intervals O_1, O_2, O_3 and O_4 of the optimal schedule.

Virtual Select with Delays(VSD):

Initialize Q to be an empty queue
When an interval I arrives
  If I does not conflict with previously scheduled intervals
    If the resource is available for I
      Flip a fair coin
      If Heads: Take the interval.
      If Tails: Virtually take the interval, but do not work.
            Take no intervals with the second segment intersecting I^2.
    Else put I in Q
When a scheduled or virtually scheduled interval completes
  If Q is not empty
    Let I be the interval with highest priority in Q
      If I does not conflict with previously scheduled intervals and
       the resource is available for I
        Flip a fair coin
        If Heads: Take the interval.
        If Tails: Virtually take the interval, but do not work.
              Take no intervals with the second segment intersecting I^2.
    Remove I from Q

Theorem 5.3 VSD *is 16-competitive for scheduling with delays.*

Proof: The proof is similar to the proof for VS. The only difference is that every interval of the optimal schedule must contribute to the buckets it intersects like any other (i.e., non-optimal) interval. In addition, it must contribute some weight to itself.

Therefore, for any I, a segment I^1 is assigned to a bucket B, if I^1 blocks the segment B. A segment I^2 is assigned to a bucket B, if (i) I^2 intersects the segment B, and (ii) B is not strictly contained in I^2. In addition, if I belongs to the optimal schedule, it is assigned to its own bucket (see Figure 5.2). Each of these segments contributes $1/4$ to any bucket to which it belongs. (The optimal interval contributes $1/4$ to itself only once, and not per each segment).

In addition to all the cases considered in the proof of VS, we have one more case: when an interval O was scheduled by OPT, the queue of VSD was empty. That means that VSD already scheduled O; O contributes $1/4$ to its own bucket; hence, in this case VSD's gain is $1/4$ of the optimal gain.

We calculate the gains in all the previous cases as well, and the theorem follows. ∎

Corollary 5.4 *If the length of the first segment is in the range* $[1,2]$*, and the length of the second segment is in the range* $[l/2,l]$*, then the algorithm* VSD *is 24-competitive.*

In the case of $(l, 1)$ intervals we can use the online Greedy algorithm, which selects the interval with the earliest start deadline.

Lemma 5.5 *If the length of the first segment is in the range $[l/2, l]$, and the length of the second segment is in the range $[1, 2]$, then the above Greedy algorithm is 6-competitive.*

Proof: Instead of an interval I scheduled by Greedy, OPT can schedule at most 5 non-conflicting intervals, and then schedule I some time later. ∎

Now we present an $O(\log^2 R)$ competitive algorithm for the case of arbitrary interval length. The algorithm knows R in advance. We partition the intervals into barrels as in the Select-from-Barrel Algorithm. We denote by A the set of barrels in which the first segment is longer, i.e., $\forall I \in A \; |I^1| > |I^2|$.

Select-from-Barrel-with-Delays (SBD):
Randomly choose one barrel $(i, j)^*$.
If $(i, j)^* \in A$, then schedule the intervals in this barrel greedily;
else apply the algorithm VSD on this barrel.
Reject all other intervals.

Theorem 5.6 *SBD is $O(\log^2 R)$ competitive.*

Proof: We consider separately the competitive ratio obtained for two types of barrels: for the barrels in A, the greedy schedule achieves the competitive ratio 6, while on the other barrels VSD has the ratio 24. Since half of the barrels are in A we get that the competitive ratio of SB is $15 \log^2 R$.

∎

Chapter 6

Summary and Open Problems

In this work we studied the problem of finding an IS in split interval graphs. We have shown that this problem differs significantly from the problem of finding an IS in interval graphs, and that it is APX-complete even on 2-split interval graphs, where all the segments are of the same length. We presented two algorithms for the offline case: an $O(\log R)$ approximation and a $(4, \epsilon)$ bi-criteria PTAS for the case where the number of distinct segment lengths is a fixed constant. In the online case, we developed a number of practical algorithms for different models. Some of these algorithms match our lower bounds for the corresponding models.

A main contribution of our work is in getting better intuition for this interesting problem, by studying several approaches to analyzing and solving it. To the best of our knowledge, the IS problem on splitting interval graphs was studied here for the first time.

Maximizing the Resource Utilization In our bi-criteria approximation scheme we need to make a single change to adapt the scheme to the resource utilization objective. When we choose the best subset, we need to take the subset that maximizes the resource utilization (and not the one of the maximal size). As before, we get a $(4, \varepsilon)$-approximation.

Open Problems The gaps between the upper and lower bounds are the main open problems.

- In the offline case we have a gap between APX-completeness and $O(\log R)$ approximation obtained by OffBA.
- In the online case, in the complete information model, there is a gap between $\Omega(\log R)$ lower bound and $O(\log^2 R)$ upper bound on the competitive ratio of randomized algorithms.
- When the break times are unknown and $D = o(R)$, we have the lower bounds of $\Omega(D)$ and $\Omega(\log R)$ and a randomized $O(D \log R)$ competitive algorithm.

The Weighted Case Another direction for future work is to study the *weighted* case. In this extension of our problem there is a weight associated with each interval. Using the Local-Ratio technique (see, e.g., [10]) we can solve the offline problem with approximation ratio of 3 on proper graphs, and k in $(k+1)$-claw free graphs. (In fact, OffBA can be applied with slight modification to yield the same approximation ratio as in the non-weighted case, for general split interval graphs).

In online scheduling of weighted non-splitting intervals there is gap between the lower bound of $\Omega(\log(R+W))$, where W is the ratio between the largest and smallest weights, and the upper bound of $O(\log R \log W)$ [6]. We believe, that this problem is closely related to the problem of scheduling splitting intervals, and that reducing one of the gaps can help reduce the other one.

General t Most of our algorithms can be used in the case of split interval graphs with $t > 2$. We chose to present only the results for $t = 2$, since this case is already a difficult problem, and the extension to $t > 2$ is straightforward.

Bibliography

[1] R. Adler, and Y. Azar."Beating the Logarithmic Lower Bound: Randomized Preemptive Disjoint Path and Call Control Algorithms". In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1999, pp. 1-10.

[2] A. Aggarwal, J. Garay, and A. Herzberg. "Adaptive Video on Demand". In *Proceedings of the 3th European Symposium on Algorithms (ESA)*, 1995, pp. 538-553.

[3] N. Alon, U. Feige, A. Widgerson, and D. Zuckerman. "Derandomized Graph Products". In *Computational Complexity*, vol. 5, 1995, pp. 60-75.

[4] B. Awerbuch, Y. Azar, A. Fiat, and T. Leighton. "Making Commitments in the Face of Uncertainty: How to Pick a Winner Almost Every Time". In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, 1996, pp. 519-530.

[5] B. Awerbuch, Y. Azar, and S. Plotkin. "Throughput-Competitive On-Line Routing". In *Proceedings of the IEEE 34th Annual Symposium on Foundations of Computer Science*, 1993, pp. 32-40.

[6] B. Awerbuch, Y. Bartal, A. Fiat, and A. Rosén. "Competitive Non-Preemptive Call Control". In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1994, pp. 312-320.

[7] V. Bafna, B. Narayanan, and R. Ravi. "Nonoverlapping Local Alignments (Weighted Independent Sets of Axis Parallel Rectangles)". In *Proceedings of the 4th Workshop on Algorithms and Data Structures (WADS'95)*, Lecture Notes in Computer Science, 955, Springer-Verlag, 1995, pp. 506-517.

[8] B.S. Baker. "Approximation Algorithms for NP-complete Problems on Planar Graphs". In *Journal of ACM*, vol. 41, 1994, pp. 153-180.

[9] A. Bar-Noy, S. Guha, S. Naor, and B. Schieber. "Approximating the Throughput of Multiple Machines in real-time scheduling". In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, 1999, pp. 622-631.

[10] A. Bar-Noy, R. Bar-Yehuda, A. Freund, S. Naor, and B. Schieber. "A Unified Approach to Approximating Resource allocation and Scheduling". In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, 2000, pp. 735-744.

[11] P. Basu, A. Narayanan, R. Krishnan, and T.D.C. Little. "An Implementation of Dynamic Service Aggregation for Interactive Video Delivery". In *Proceedings of SPIE - Multimedia Computing and Networking*, San Jose, CA, January 1998.

[12] P. Berman, and T. Fujito. "Approximating Independent Sets in Degree 3 Graphs". In *Proceedings of the 4th Workshop on Algorithms and Data Structures (WADS'95)* Lecture Notes in Computer Science, 955, Springer-Verlag, 1995, pp. 449-460.

[13] R. Boppana, and M.M. Halldórsson. "Approximating Maximum Independent Set by Excluding Subgraphs". In *Bit*, vol. 32, 1992, pp. 180-196.

[14] A. Borodin and R. El-Yaniv. Online Computation and Competitive Analysis. Cambridge University Press, 1998.

[15] P. Brucker, T. Hilbig, and J. Hurink. "A Branch and Bound Algorithm for a Single-Machine Scheduling problem with Positive and Negative Time-Lags". In *Discrete Applied Mathematics*, vol. 94, 1999, pp. 77-99.

[16] T. Cormen, C. Leiserson, and R. Rivest. Introduction to Algorithms. The MIT Press, 1990.

[17] A. Dan, P. Shahabuddin, and D. Sitaram. "Channel Allocation Under Batching and VCR Control in Movie-On-Demand Servers", IBM Research Report RC19588, May 1994.

[18] M. Dell'Amico. "Shop Problems with Two machines and Time Lags". In *Operations Research*, vol. 44, no. 5, 1996, pp. 777-787.

[19] U. Faigle and W.M. Nawijn. "Note on Scheduling Intervals On-line". In *Discrete Applied Mathematics*, vol. 58, 1995, pp. 13-17.

[20] J. Garay, I. Gopal, S. Kutten, Y. Mansour, and M. Yung. "Efficient On-Line Call Control Algorithms". In *Proceedings of the 2nd Israel Symposium on the Theory of Computing and Systems*, 1993, pp. 285-293.

[21] M. R. Garey and D. S. Johnson. Computers and Intractability: a Guide to the Theory of NP-completeness. Freeman, 1979.

[22] F. Gavril. "Algorithms for Minimum Coloring, Maximum Clique, Minimum Coloring by Cliques, and Maximum Independent Set of a Chordal Graph". In *SIAM Journal of Computing*, vol. 1, No. 2, 1972, pp. 180-187.

[23] S. Goldman, J. Parwatikar, and S. Suri. "On-line Scheduling with Hard Deadlines". In *Proceedings of the Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science*, vol. 1272, pp. 258-271. Springer-Verlag, 1997.

[24] M. Goldwasser. "Patience is a Virtue: the Effect of Slack on Competitiveness for Admission Control". In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1999, pp. 396-405.

[25] M. Golumbic. Algorithmic Graph Theory and Perfect Graphs. Academic Press, 1980.

[26] J.R. Griggs, and D.B. West. "Extremal Values of the Interval Number of a Graph". In *SIAM Journal on Algebraic and Discrete Methods*, 1980, vol. 1, No. 1, pp. 1-7.

[27] L.A. Hall. "Approximability of Flow Shop Scheduling". In *Mathematical Programming*, 1998, vol. 82, pp. 175-190.

[28] L.A.Hall, and D.B. Shmoys. "Approximation Algorithms for Constrained Scheduling Problems". In *Proceedings of the IEEE 30th Annual Symposium on Foundations of Computer Science*, 1989, pp. 134-139.

[29] M. M. Halldórsson. "Approximating Discrete Collections via Local Improvements". In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1995, pp. 160-169.

[30] M. M. Halldórsson. "Approximations of Independent Sets in Graphs". Invited survey paper in *the First International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, Lecture Notes in Computer Science 1444, Springer-Verlag, 1998, pp. 1-14.

[31] M. M. Halldórsson. "Approximations of Weighted Independent Set and Hereditary Subset Problems". In *Proceedings of the 5th Annual International Conference on Computing and Combinatorics*, Lecture Notes in Computer Science, Springer-Verlag, 1999, pp. 261-270.

[32] M. M. Halldórsson. "Split Interval Graphs and Union Graphs". Private communication.

[33] M. M. Halldórsson, S. Rajagopalan, H. Shachnai, and A. Tomkins. "Scheduling Multiple Resources". Manuscript, 1999.

[34] J. Håstad. "Clique is hard to approximate within $n^{1-\epsilon}$". In *Acta Mathematica*, vol. 182, 1999, pp. 105-142.

[35] H.B. Hunt III, M.V. Marathe, V. Radharkrishnan, S.S. Ravi, D.J. Rosenkrantz, and R.E. Stearns. "NC-approximation Schemes for NP- and PSPACE-hard Problems for Geometric Graphs". In *Journal of Algorithms*, vol. 26, 1998, pp. 238-274.

[36] K. Jansen, R. Solis-Oba, and M. Sviridenko. "Makespan Minimization in Job Shops: a Polynomial Time Approximation Scheme". In *Proceedings of the 31th Annual ACM Symposium on Theory of Computing*, 1999, pp. 394-399.

[37] T.R. Jensen, and B. Toft. Graph Coloring Problems. John Wiley & Sons, 1995.

[38] S.M. Johnson. "Optimal Two- and Three-stage Production Schedules with Setup Times Included". In *Naval Research Logistics Quarterly*, 1954, pp. 61-68.

[39] D. Karger, R. Motwani, and M. Sudan. "Approximate Graph Coloring by Semidefinite Programming". In *Journal of ACM*, vol. 45, 1998, pp. 246-265.

[40] D. Karger, C. Stein, and J. Wein. "Scheduling Algorithms". Algorithms and Theory of Computation Handbook, CRC Press, 1997.

[41] S. Leonardi, A. Marchetti-Spaccamela, A. Presciutti, and A. Rosén. "On-line Randomized Call Control Revisited". In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1998, pp. 323-332.

[42] R. Lipton and A. Tomkins. "Online Interval Scheduling". In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1994, pp. 302-311.

[43] C. Martin, P. S. Narayanan, B. Ozden, R. Rastogi, and A. Silberschatz. "The Fellini Multimedia Storage Server". In *Multimedia Information Storage and Management*, Kluwer Academic publishers, 1996.

[44] A.J. Orman, and C.N. Potts. "On the Complexity of Coupled-Task Scheduling". In *Discrete Applied Mathematics*, vol. 72, 1997, pp. 141-154.

[45] C.H. Papadimitriou, and M. Yanakakis. "Optimization, approximation, and complexity classes". In *Journal of Computer and System Sciences*, 1991, vol. 43, pp. 425-440.

[46] A.H.G. Rinnooy Kan. Machine Scheduling Problems. Martinus Nijhoff, The Hague, 1976.

[47] D. Rotem, and S. Seshadri. "Analysis of Disk Arm Movement for Retrieval of Large Objects". In *Proceedings of Principles of Database Systems*, 1992.

[48] J.P. Schmidt, A. Siegel, and A. Srinivasan. "Chernoff-Hoeffding Bounds for Applications with Limited Independence". In *SIAM Journal on Discrete Mathematics*, vol. 6, 1995, pp. 223-250.

[49] R. Sedgewick and P. Flajolet. An Introduction to the Analysis of Algorithms. Addison-Wesley Publishing, 1996.

[50] R.D. Shapiro. "Scheduling Coupled Tasks". In *Naval Research Logistics Quarterly*, vol. 27, 1980, pp. 489-498.

[51] D.B. Shmoys, C. Stein, and J. Wein. "Improved Approximation Algorithms for Shop Scheduling Problems". In *SIAM Journal on Computing*, vol. 23, 1994, pp. 617-632.

[52] W.T. Trotter, Jr. and F. Harary. "On Double and Multiple Interval Graphs". In *Journal of Graph Theory*, vol. 3, 1979, pp. 205-211.

[53] D.B. West, and D.B. Shmoys. "Recognizing Graphs with Fixed Interval Number is NP-complete". In *Discrete Applied Mathematics*, vol. 8, 1984, pp. 295-305.

[54] A. C.-C. Yao. "Probabilistic Computations: towards a unified measure of complexity". In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, 1977, pp. 222-227.

[55] P.S. Yu , J.L. Wolf, and H. Shachnai. "Design and Analysis of a Look_ahead Scheduling Scheme to Support Pause-Resume Video-on-Demand Applications". In *ACM Multimedia Systems Journal*, vol. 3, 1995, pp. 137-149.