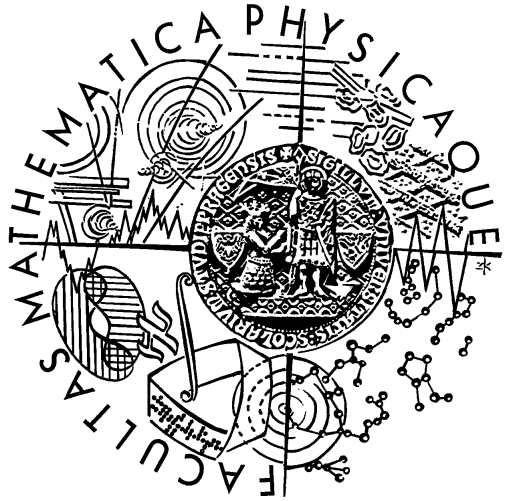
KARLOVA UNIVERZITA
MATEMATICKO–FYZIKÁLNÍ FAKULTA

DOKTORSKÁ DISERTAČNÍ PRÁCE

Tomáš Tichý

Aproximační a online algoritmy



Praha, 2008

Matematický ústav Akademie věd České Republiky
Institut teoretické informatiky
Matematicko–fyzikální fakulta
Univerzita Karlova v Praze

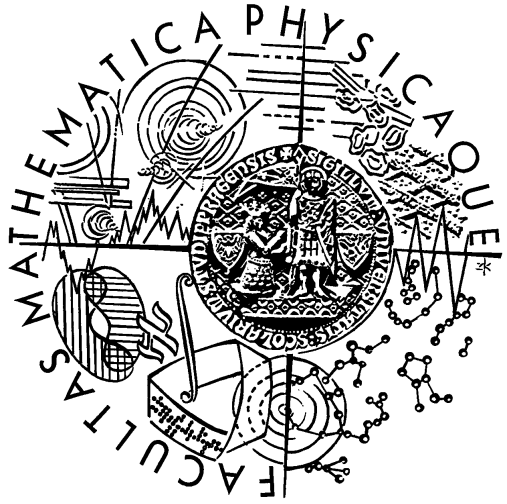
Školitel: Doc. RNDr. Jiří Sgall, DrSc.
Obor: I4 – Diskrétní modely a algoritmy

CHARLES UNIVERSITY
FACULTY OF MATHEMATICS AND PHYSICS

DOCTORAL THESIS

Tomáš Tichý

Approximation and Online Algorithms



Prague, 2008

Institute of Mathematics of the Academy of Sciences
of the Czech Republic
Institute for Theoretical Computer Science
Faculty of Mathematics and Physics
Charles University in Prague

Advisor: Doc. RNDr. Jiří Sgall, DrSc.
Branch: I4 – Discrete Models and Algorithms

Disertační práce byla vypracována v rámci doktorského studia, které uchazeč absolvoval na Matematicko–fyzikální fakultě Univerzity Karlovy v Praze v letech 2001–2008.

Doktorand:	RNDr. Tomáš Tichý
Školitel:	Doc. RNDr. Jiří Sgall, DrSc.
Školící pracoviště:	Matematický ústav AV ČR Žitná 25 115 67 Praha 1

Preface

The area of approximation and online algorithms and problems is a wide area of computer science. The nature of the problems arises from the optimization problems of the real world—they are mostly logistical, transportation, task scheduling, load balancing and similar problems. Together with them there are also purely theoretical applications in areas like graph theory or theory of complexity.

The first well–known researcher in the area is R. L. Graham who published results on scheduling problems in sixties of the 20th century. Various results on scheduling problems were published before him, but Graham is considered to be pioneer of the approximation and online algorithms area. Althrough the area has been studied intensively almost fifty years, we still study simple abstractions of the real problems and the area is rich on the interesting open problems remaining to be solved. On the other hand there are also practical applications of the theoretical results. The most important practical applications are in the area of computer networks, for example, algorithms for network switches, network routing, packet scheduling, buffer management, guaranting of quality of service etc. Because of these practical applications and lots of interesting and natural open problems with very simple definitions the area has been popular and intensively studied in recent years. Nevertheless it seems there is a lot of hard work for many years.

This thesis is focused on variants of the online scheduling problems. First the thesis gives an overview of basic definitions, common methods and variants of studied problems. Remaining chapters are based on my research. Most of the introduced results were already published, see List of publications. Some of the published papers are joint papers which are compilations of results of a few coauthors or joint research. The chapters consist of detailed description of my results and also they separately introduce other results from joint papers.

I am grateful to my advisor and coauthor, Jiří Sgall, for his guidance through my Ph.D. studies and his advice regarding my research, publish-

ing and thesis writing. I am also grateful to my coauthors, especially to my advisor Jiří Sgall and colleagues from Riverside University Marek Chrobak and Wojtek Jawor for their great cooperation, exchanging ideas and productive discussions. I would like to thank Institute for Theoretical Computer Science (ITI) of the Charles University in Prague and Mathematical Institute of the Academy of Sciences of the Czech Republic for their great support during my studies.

As required by the Charles University, I hereby declare I wrote this thesis on my own and that the references include all the sources of information which I exploited. I also authorize the Charles Univerity to lend this thesis to other institiutions or individuals for academic and research purposes.

Prague, April 2008 Tomáš Tichý

List of publications

Publications related to this thesis

Published papers

[Pub–1] M. Chrobak, W. Jawor, J. Sgall, T. Tichý: *Online Scheduling of Equal–Length Jobs: Randomization and Restarts Help.* SIAM Journal of Computing, 36(6):1709–1728, 2007. A preliminary version appeared in Proceedings of the 31st International Colloquium on Automata, Languages, and Programming (ICALP). Berlin, Springer 358–370, 2004.

[Pub–2] F. Y. L. Chin, M. Chrobak, S. P. Y. Fung, W. Jawor, J. Sgall, T. Tichý: *Online Competitive Algorithms for Maximizing Weighted Throughput of Unit Jobs.* Journal of Discrete Algorithms, 4(2):255–276, 2006..

[Pub–3] Y. Bartal, F. Y. L. Chin, M. Chrobak, S. P. Y. Fung, W. Jawor, R. Lavi, J. Sgall, T. Tichý: *Online Competitive Algorithms for Maximizing Weighted Throughput of Unit Jobs.* Proceedings of the 21st Annual Symposium on Theoretical Aspects of Computer Science (STACS). Berlin, Springer–Verlag 187–198, 2004.

[Pub–4] M. Chrobak, L. Epstein, J. Noga, J. Sgall, R. van Stee, T. Tichý, N. Vakhania: *Preemptive Scheduling in Overloaded Systems.* Journal of Computer and System Sciences, 67 (1) 183–197, 2003; Proceedings of the 29th International Colloquium on Automata, Languages, and Programming. Berlin, Springer–Verlag 800–811, 2002.

[Pub–5] M. Chrobak, W. Jawor, J. Sgall, T. Tichý: *Improved Online Algorithms for Buffer Management in QoS Switches.* To appear in ACM Transactions on Algorithms. A preliminary version appeared in Proceedings of the 12th European Symposium on Algorithms. Berlin, Springer 204–215, 2004.

Other publications

[Pub–6] D. Král', J. Sgall, T. Tichý: *Randomized Strategies for the Plurality Problem.* To appear in Discrete Applied Mathematics. KAM–DIMATIA Series 2005–722 and ITI Series 2005–238, Charles University, Prague, 2005.

[Pub–7] T. Tichý: *Randomized Online Scheduling on 3 Processors.* Operation Research Letters, 32 (2) 152–158, 2004.

[Pub–8] D. Král', V. Majerech, J. Sgall, T. Tichý, G. Woeginger: *It Is Tough to Be a Plumber.* Theoretical Computer Science, 313 (3) 473–484, 2004.

[Pub–9] T. Tichý: *Multiprocessor Randomized Online Scheduling.* ITI Series, 2002–069, Charles University, Prague, 2002.

[Pub–10] T. Tichý: *A Lower Bound for Restricted Randomized Online Algorithms for Scheduling.* Proceedings of the Week for Doctoral Students 2002. Praha, MATFYZPRESS 21–26, 2002.

Contents

Abstract

This thesis presents results of our research in the area of optimization problems with incomplete information—our research is focused on the online scheduling problems. Our research is based on the worst–case analysis of studied problems and algorithms; thus we use methods of the competitive analysis during our research.

Althrough there are many "real–world" industrial and theoretical applications of the online scheduling problems there are still so many open problems with so simple description. Therefore it is important, interesting and also challenging to study the online scheduling problems and their simplified variants as well.

In this thesis we have shown the following our results of our research on the online scheduling problems:

- A 1.58–competitive online algorithm for the problem of randomized scheduling of unit jobs on a single processor, where the jobs are arriving over time and the total weight of processed jobs is maximized.
- A lower bound 1.172 on the competitive ratio for the problem of randomized scheduling of 2–uniform unit jobs on a single processor, where the jobs are arriving over time and the total weight of processed jobs is maximized.
- A lower bound 1.25 on the competitive ratio for the problem of randomized scheduling of s–uniform unit jobs on a single processor where s is tending to infinity, the jobs are arriving over time and the total weight of processed jobs is maximized.
- A 1.5–competitive online algorithm for the problem of deterministic scheduling of equal–length jobs on a single processor, where restarts of jobs are allowed, the jobs are arriving over time and the total weight of processed jobs is maximized.
- There is no online 1–competitive algorithm with speed–up $s < 2$ for the problem of deterministic scheduling of tight jobs on a single

processor, where the preemptions of jobs are allowed, the jobs are arriving over time and the total weight of processed jobs is maximized.

- There is no 1–competitive k–relaxed online algorithm for any k for the problem of deterministic scheduling of jobs arriving over time with maximizing total weight of processed jobs.
- A lower bound 1.05099 on the competitive ratio for the problem of 1–relaxed deterministic scheduling of jobs arriving over time with maximizing total weight of processed jobs. We have shown a generalized lower bound for k–relaxed algorithms.

We present also some other results related to studied problems that are products of a joint work with other researchers.

Outline of this thesis

In the first part of the thesis—"Introduction"—we provide an overview of the methods of the competitive analysis in the context of various online scheduling problems and algorithms. We discuss the importance, advantages and disadvantages of the competitive analysis and its extensive usage by researchers. We also discuss extensions of the competitive analysis—especially we mention the resource augmentation framework because some of our results are developed under this framework.

Except the mentioned discussion on variants of scheduling problems and algorithms we also provide a detailed taxonomy of the online scheduling problems—including problems that we do not study.

The second part of the thesis—"Problems and results"—enumerates the problems that are studied in this thesis and the problems that are closely related to studied problems. For each problem we always provide detailed description and it's formal description—according to introduced taxonomy as well.

For each problem we also attach a list of previously best known results, we show our results and separately we show a list of results of our joint research with other co–authors. The problems presented in this thesis are mostly focused on scheduling problems running jobs on a single machine, where jobs arrive over time and are specified by their release times and deadlines, using the total weight of processed jobs as the objective function, considering deterministic and randomized computational model, usually using standard measure model. Often we consider additional restrictions on jobs—jobs with unit processing times or equal processing times.

The types of presented results follow from the methods of the competitive analysis. The main general goal is to find the exact competitive ratio of studied problems or algorithms. Usually we develop at least an approach for this general goal—a lower bound or an upper bound on the exact competitive ratio. The lower bound on the competitive ratio usually follows from the properties of some hard input instance. The upper bound on the competitive ratio of an problem usually follows from the properties of some algorithm.

The third part of the thesis presents our results. The overview of the necessary common preliminaries and terminology is presented in the chapter "Preliminaries". Our results are presented in the chapters "Scheduling", "Resource augmentation" and "Online scheduling of equal–length jobs". Parts of these chapters are taken from the existing papers presenting these results.

Some results have been published in joint papers with other co–authors and are presented in a context of these papers. What are our original results and what are the joint results is exactly mentioned in the chapter "Problems and results". Most of the presented results in this thesis have been already published.

Chapter 1

Introduction

Our area of interest arises from a traditional combinatorial area—the area of optimization problems. The optimization problems are widely studied for a very long time, but there are many pretty hard and famous problems. Because of the huge scope of the optimization problems there arise some subareas like scheduling which try to solve some classes of optimization problems.

We focus on variants of the optimization problems with incomplete information. In such problems, the information about the problem arrives in steps and we are forced to make decisions while the information is still incomplete.

In the usual definition of the optimization problems it is assumed that the whole information is known and fixed. However it is not the case of the real world, where information arrives incrementally in time and we have to make decisions continuously. This is exactly the case when the *online algorithms* can help us. The reason is that they are defined to make decisions using partial knowledge of their input instance.

In the next sections we discuss basic definitions, methods and the current state of art. We also give an overview of the studied problems.

1.1 Optimization and approximation

Let us describe what an optimization problem is. Consider an input instance for the problem. This input instance together with the problem definition gives a set of discrete objects. These objects are feasible solutions of the problem for a given input instance. An objective (cost or profit) function of the problem measures the quality of any particular solution. The goal is to find an optimal solution among the feasible solutions for a

fixed input instance, such that it minimizes its cost or maximizes its profit.

For many interesting optimization problems it is really hard to find an optimal solution in reasonable time or space, because of huge complexity in time or space. Interesting optimization problems are mostly NP–hard.

In practice there are many industrial applications of the optimization problems like making plans, working or logistic schedules etc. The advantage of practice is that we usually do not need an optimal solution but it is enough to find a sufficiently reasonable feasible solution.

One of the methods to find a reasonable solution is a method of approximation algorithms. The approximation algorithms are fast (with low complexity) and are provably close to the optimum.

1.2 Online algorithms

The concept of an online algorithm formalizes the real–world scenario, where a real algorithm does not know the whole input instance while offline algorithms do. Instead of this the online algorithm gets pieces of the input instance in steps (in time) and the algorithm has to react to the new requests with only partial knowledge of the input.

Moreover, many heuristical or approximation algorithms applied on hard optimization problems are actually online algorithms. The main reason is that we need to design simple algorithms because of realizable analysis and implementation. Simple algorithms are usually unable to use the complete information of the input instance. They split input instance into smaller pieces, for example they sort objects of the input instance and process it one by one.

In the offline world we are usually interested in the time and/or space complexity of the studied algorithms. Instead, in the online world we are interested in the quality of the produced solution. When we have sufficient time and space resources in the offline world, we always find the optimal solution. But this does not hold in the online world because of the important role of the partial knowledge of the input instance. Moreover, online algorithms are usually simple and fast, hence there is much more interesting challenge in the quality of produced solutions than in their complexity.

The online algorithms are usually quite simple heuristical algorithms with very low time and space complexity. Nevertheless the analysis of the online algorithms is sometimes really complicated. Let us take a look on the common methods used for measuring the quality of the online algorithms.

1.2.1 Competitive analysis

The *competitive analysis* was introduced by Sleator and Tarjan [57] and it is a variation on the traditional *worst–case analysis* of optimization algorithms. The worst–case analysis studies the performance of an algorithm in the worst case. We have to define what does it mean exactly.

We measure the cost of an online algorithm on an input instance by a cost function. The cost function is unbounded and we are interested in the behaviour on all input instances. Therefore we compare the cost to the cost of another algorithm for each fixed input instance. We want to bound the ratio of these costs over all input instances.

We are interested in how much worse is the online algorithm against the optimal solution. Since we give no restrictions on time or space complexity we can assume that the offline algorithm always produces an optimal solution. Hence in the competitive analysis it is natural to compare the cost of the online algorithm to the cost of the offline algorithm on the same input instances.

According to the worst–case analysis we look for the comparison of an online algorithm and an offline algorithm on the worst instance. Formally we define this by the *competitive ratio* as follows.

We consider minimization or maximization problems. In the minimization problems an algorithm minimizes cost which it has to pay. In the maximization problems an algorithm maximizes profit which it gets. Let us denote an online algorithm as $\mathcal{A}$, an input instance as σ and the cost or profit of the algorithm on the instance as $\mathcal{A}(\sigma)$. Let OPT denote an (arbitrary fixed) offline (optimal) algorithm and $\mathrm{OPT}(\sigma)$ the optimal cost or profit on the instance.

For minimization problems we define the competitive ratio (over all possible input instances) as follows:

$$\mathcal{R}(\mathcal{A}) = \inf_{R \in \mathbb{R}} \{R : (\forall \sigma), \mathcal{A}(\sigma) \leq R \cdot \mathrm{OPT}(\sigma)\}$$

and similarly for the maximization problem as follows:

$$\mathcal{R}(\mathcal{A}) = \inf_{R \in \mathbb{R}} \{R : (\forall \sigma), R \cdot \mathcal{A}(\sigma) \geq \mathrm{OPT}(\sigma)\}.$$

Because of these definitions, the competitive ratio is always greater than or equal to 1. The best online algorithm is the online algorithm with the lowest possible competitive ratio. An online algorithm is an optimal algorithm for the problem if and only if its competitive ratio is exactly 1.

When we consider a fixed scheduling problem then we also define the **competitive ratio of the problem** as

$$\mathcal{R} = \inf_{\mathcal{A}} \mathcal{R}(\mathcal{A}),$$

which is going over all online algorithms for the problem.

Obviously, the competitive ratio is a worst–case measure of online algorithms. It shows the strong influence of uncertain input instance (partial knowledge of instance) on the profit. Another userful feature of the measure is the following claim. Let us consider an algorithm $\mathcal{A}$ with the competitive ratio $c = \mathcal{R}(\mathcal{A})$ and any other better d–competitive algorithm $\mathcal{B}$ for some $d < c$. Then there exists an input instance such that the better algorithm is at least c/d–times better on the instance. Obviously the worst case input instance of the algorithm $\mathcal{A}$ proves this claim.

On the other hand the disadvantage of the method is that it sometimes gives competitive ratio which does not correspond to the empirical performance of some algorithms which perform very well in practice.

Let us consider two well–known algorithms for the paging problem—the LRU algorithm and the FIFO algorithm. A paging algorithm is a strategy which chooses which page may be evicted from the cache in the case of a fault in the cache. The LRU—least recently used—algorithm always evicts the page whose last access was earliest. The FIFO—first in, first out—algorithms always evicts the page that has been in the cache for the longest time. This pair of LRU and FIFO algorithms is such an example that the competitive ratio of both is the same, but the LRU algorithm is much better then the FIFO algorithm. When we denote k as a number of pages in the paging problem then both algorithms are exactly k–competitive. In practice the results produced by the FIFO algorithm are approximately k–times worse than the optimum but the results produced by the LRU algorithm are much better than k–times the optimum. These two algorithms can be distinguished using the access graphs for the analysis—one of extensions of the competitive analysis.

This example illustrates the known defect of the competitive analysis and shows an effective way how to bypass this defect—using some extension of the competitive analysis. We should not consider this as a general method how to distinguish between algorithms with the same competitive ratio but different results in practice. This example shows the method which can help but does not have to.

1.2.2 Randomization in competitive analysis

Randomization is a standard extension of the competitive analysis. It gives us ways of significant improvement of the competitive performance of our algorithms while we remain in the worst–case world. The basic idea of randomization is to allow the algorithm to use random bits in its decision process. Instead of the objective function on a fixed input instance we consider the expectation of the objective function on the fixed input instance.

The competitive ratio for a minimization problem is defined as

$$\mathcal{R}(\mathcal{A}) = \inf_{R \in \mathbb{R}} \{R : (\forall \sigma), \mathbb{E}[\mathcal{A}(\sigma)] \leq R \cdot \mathrm{OPT}(\sigma)\}$$

and similarly for the maximization problem as follows:

$$\mathcal{R}(\mathcal{A}) = \inf_{R \in \mathbb{R}} \{R : (\forall \sigma), R \cdot \mathbb{E}[\mathcal{A}(\sigma)] \geq \mathrm{OPT}(\sigma)\}.$$

The important aspect is that we do not consider randomization over input instances but we consider randomization for each fixed input instance separately.

Therefore the nature of the analysis remains worst–case. The randomized computation model is stronger than deterministic model, hence the algorithm is more powerful. The model can significantly improve the competitive ratio of some problems.

1.2.3 Resource augmentation in competitive analysis

The resource augmentation is one of the common techniques used in the competitive analysis. Generally, results deloped under this technique allow us to better understand the studied problems—give use more complex view on the problems—and possibly allow us to design better algorithms for the problems.

For the first time, this technique was already used in 1966 by Graham in [32]. The technique was officially introduced and entitled in 1995 by Kalyanasundaram and Pruhs [37]. They demonstated this method on a certain scheduling problem.

The basic idea of the resource augmentation technique is to allow more resources to the online algorithm than to the corresponding offline algorithm on the same problem. For example, the online algorithm can be allowed to use more processors, faster processors, later deadlines, etc. Under this technique we study the problems from the point of view of the sufficiency of resources. We study changes of the competitive ratio when we break some of the resource constraints.

Motivation

In the following lines we describe the basic reasons why this technique was developed. When we study a problem using the competitive analysis then we always compare results of an online algorithm and an optimal offline algorithm for the same problem. Both have the same resources like memory, number of processors, speed of procesors, weights, profits, etc. But the online algorithm has an important handicap—the uncertainty of the input instance.

Let us recall the goal of the competitive analysis—we strongly prefer the online algorithms with a competitive ratio bounded by a constant. It means that we do not want to be dependent on any parameters of studied problems. But for some of the studied problems it can be really hard to find such an online algorithm or such an algorithm does not exist.

When it is hard to find a constant competitive online algorithm for a studied problem then this technique simplifies the problem, gives us better understanding of the problem and moves us closer to the solution of the more general problem.

The most imporatant application of this technique is when the constant competitive online algorithm does not exist. Then this technique shows us which resources it is necessary to break to get a constant–competitive online algorithms for the studied problem.

Kalyanasundram and Pruhs in [37] show this technique on a certain scheduling problem. This problem has been extensively studied earlier using "common" methods—Koren and Shasha introduced an non–constant competitive online algorithm which is optimal for the problem. Whenever we prove that there is no constant–competitive algorithm and we still insist on the "constant–competitiveness" then we must break some constraints of the studied problem. In their problem the objective function was the total flow time. Kalyanasundram and Prush broke the speed of the processor—their online algorithm uses s–times faster processor than the offline algorithm. They showed that the competitive ratio of the online algorithm is significantly improved when the speedup s is slightly greater than 1 for their scheduling problem.

Under the resource augmentation technique we are interested in:

- the behaviour of the competitive ratio of an online algorithm when we slightly break a resource constraint,
- sufficient amount of resources to obtain an 1–competitive online algorithm for the problem.

With respect to the second interest—Kalyansundram and Pruhs showed an 1–competitive online algorithm using 32–times faster processor for the studied problem.

1.2.4 Probabilistic analysis

In practice we are usually satisfied with algorithms which are good on average and we do not care so much about the performance of such algorithms in the worst cases. It means that we are satisfied with an algorithm which performs well on the most of the input instances and performs badly only occasionally—on some singular input instances.

One of the possible approaches how to get a practically usable algorithm is to use the probabilistic analysis method. This method provides weaker and not so interesting results. Thus we must be aware with the limitations of this method.

When we want to develop an algorithm for practice we are able to imagine probabilistic assumptions on the input instances. Also we can formalize our ideas and define the probabilistic distribution on the input instances for the problem. Then we can design some heuristical online algorithm according to the probabilistic distribution. Then we study the performance of the algorithm on the average over the probabilistic distribution of the input instances.

It is not easy to find a good probabilistic distribution of input instances. Actually it is really hard problem to define a distribution that does not suppress bad singular cases—such cases that are important in the worst case analysis. Althrough the performance of our heuristical algorithm is good on average there can remain some hard cases on which the heuristical algorithm performs badly. We must be very careful when we want to claim something on the performance of our algorithm.

These assumptions are quite different from our assumptions in problems which we study. We consider that such assumptions are too strong. Results based on such assumptions can look very nice and strong, but they tell nothing about behaviour of studied algorithms in bad cases. In this model we can get an algorithm usable in applications in real world—usually called good on average. Such an algorithm works fine but for some special input instances it returns very bad solutions. Instead our philosophy is that an algorithm should always return results with some guaranties on its quality. Therefore this thesis does not present results based on average assumptions.

1.2.5 Adversaries

Let us view the online problems as a game. The first player of the game is the online algorithm, its goal is to find good solution while it has partial knowledge of the input instance. Its opponent generates the input instance "on the fly" and gives the pieces to the first player. The opponent is called an adversary in the terminology of the online algorithms. We distinguish the following types of adversaries according to their power:

- **Oblivious adversary**—he gives pieces of the input instance according to his strategy ignoring the real actions of the online algorithm, he chooses the strategy in advance;
- **Online adaptive adversary**—he gives pieces of the input instance using his knowledge of the all previous answers of the online algorithm in the game, continuously generates his own answers;
- **Offline adaptive adversary**—he gives pieces of the input instance using his knowledge of the all previous answers of the online algorithm in the game. He gives his, obviously optimal, solution when the whole input instance is generated.

We can view the randomized online algorithms as the games for two players as we do for the deterministic online algorithms. In such a case we never allow reading random bits of the algorithms to the adversaries. According to the method of randomization in the competitive analysis—which we presented above—we can consider the oblivious adversary only. It means that we fix the input instance (it is the the strategy of the oblivious adversary) and then we are interested in the expected objective.

1.3 Overview of online scheduling problems

In this section we show a taxonomy of online scheduling problems. Of course the taxonomy cannot cover all such problems but it covers most of them except some unusual ones. First we classify the problems according to basic concepts of the algorithms—the online paradigm, the objective function and the computational model. Second we show some variants—the problems with additional various constraints or extentions.

1.3.1 Basic classification

The online scheduling problems are classified from the four points of view:

- machine environment,
- the paradigm in which the problem is online,
- objective function to maximize/minimize,
- computational model—deterministic/randomized.

First for all we distinguish the online scheduling problems according to the machine environment—number of available machines working on given jobs. There are big differences in scheduling problems using single machine and scheduling problems using more than one machine.

The online scheduling problems are online in the paradigm how the algorithm gets information about arriving jobs. In all of the paradigms the algorithm does not know the future jobs. Each paradigm gives some restrictions on the processing of arriving jobs.

There are these basic paradigms for the online scheduling problems:

- **Arriving one by one.** The jobs arrive one by one in steps. In each step the algorithm picks an arriving job. The algorithm gets complete information about the arriving job. The algorithm schedules the job earlier than it picks the next job. The schedule is generally an assignment of jobs to machines and time slots. These assignments cannot be changed later. Note that it is usually allowed to assign a job to some future time slots.
- **Arriving over time.** The jobs arrive over time, each job arrives at its release time. The algorithm gets complete information about the arriving job. The algorithm does not need to schedule jobs immediately. The algorithm maintains a set of available jobs and schedules available jobs over time independently of their arrivals. There can be allowed preemptions or restarts of already running jobs.
- **Unknown running times.** This is same as "Arriving over time", but the algorithm does not get information about the running times of jobs on their arrival. The algorithm gets the running time when the job is finished.
- **Interval scheduling.** In this paradigm the jobs arrive over time or one by one. The imporatant difference to the other paradigms is that the algorithm does not need to process all jobs. The objective is the number of processed jobs or the total weight of processed jobs according to the studied problem. There is a restriction that jobs are

tight, which means that for each job the length of the interval between its deadline and its release time is equal to its processing time.

- **Real–time scheduling.** Modern operating systems need to schedule periodic and aperiodic tasks in real time. Each execution of a task is called a job. Each task is specified by its offset, period, worst case execution time and (relative) deadline. In such a problem we study the feasibility of input instances (whether all deadlines can be satisfied).

The objective function measures the operational expenses or the satisfaction of the owner of the system. There most usual objective function for scheduling problems is makespan. The most common objective functions are:

- **makespan,** the time when the last job is completed;
- **total completion time,** the sum of the completion times of all jobs, the completion time is the time when the job is completed;
- **total flow time,** the sum of flow time of all jobs, flow time is the completion time minus the release time;
- **total waiting time,** the sum of the waiting times of all jobs, the waiting time is the flow time minus the running time;
- **number of jobs,** the number completed jobs;
- **total weight of jobs,** the sum of weights of all completed jobs.

There are also other more complicated objective function which are usually problem specific. Such objective function can minimize e.g. number of preemptions, give some penalties for rejections etc.

The third basic classification of the online scheduling problems is the computational model. We choose between deterministic and randomized algorithms. Each deterministic online scheduling problem can be naturally extended to its randomized version—allowing randomized algorithms and using expected objective value instead of the deterministic one.

1.3.2 Variants

In this section we shortly introduce some variants of the online scheduling problems. Some of these variants are of course also variants of the (offline) scheduling problems.

- **Job constraints**—the additional constraints which must be considered by the online algorithm while it makes schedule. There can be specified release time (when the job becomes to be available the first time), deadline (must be finished before deadline or thrown away), dependecies of jobs (a job becomes available when all its predecesors are finished), conflicts of jobs (some jobs cannot be processed at the same time).

- **Preemptions and restarts**—in some problems a running job can be preempted—it can be stopped and resumed or restarted later. Restarts are interesting for the problems where we cannot allow interruptions of a running job.

- **Parallel jobs**—such jobs can be processed simultaneously on a few processors. We use parallel processing because of reducing the completion times. Usually the parallel processing splits the processing time of a job to a few processors. There are two basic types of parallel jobs. The non–malleable jobs are in the first type—such jobs specify the maximal number of processor that can be used simultaneously. The rest are the malleable jobs—that can be simultaneously processed on an exact number of processors.

- **Different speeds**—machines may have different speeds, there are two variants—fixed speeds for all machines (uniformly related machines) and different speeds for different jobs (unrelated machines).

- **Job consists of tasks**—there is a set of tasks in the problem and each job consists of several tasks on different machines. In "open shop" the order of processing tasks is arbitrary. In "flow shop" the jobs are processed according to one fixed ordering of tasks for all jobs. In "job shop" the ordering is fixed for each job separately.

Note that there are also lots of minor variants of scheluding problems, which are not shown here.

1.3.3 Standard vs. metered model

Whenever we consider the scheduling problems and we allow to the algorithm to use preemptions of running jobs then we may also think about the model of the computation of the profit in the problem. The important factor which affects the model of the computation is the motivation of the studied problem. We distinguish two models:

- **Standard model**—In the standard model the algorithm obtains whole profit for each completed job. The algorithm does not obtain any profit for preempted jobs.
- **Metered model**—In the metered model the algorithm obtains at least partial profit for each job. For each job the obtained profit is proportional to the size of processed part of the job. We always assume that the profit grows linearly—that the ratio of obtained profit over whole profit is equal to the ratio of the size of processed part of the job over the whole size (processing time) of the job.

Note that the first one—the standard model is applicable in some scheduling problems like the packet scheduling, network switching, etc. In these problems the uncompleted (especially preempted) jobs cannot be used for later processing because of the nature of these problems. Therefore the algorithm does not obtain any profit for preempted jobs.

On the other hand—the metered model is applicable in some other scheduling problems like scheduling computational tasks on processors, realtime scheduling or planning of manufacturing in factories, etc. In such problems the jobs can be suspended and also resumed later or already processed work on uncompleted jobs can be efficiently used later.

Chapter 2

Problems and results

The basic purpose of this chapter is to show a compact overview of all problems studied in this thesis. This overview also includes enumeration of our results related to the studied problems.

We do not intend this thesis to be a detailed description of a new entire consistent theory on scheduling and approximation algorithms. The area of the approximation and online algorithms is so wide and contains lots of interesting open problems that are waiting for their resolution. We focused our research effort on some of these problems and we tried to solve them or at least to solve its subproblems. We were successful in some cases and of course in some cases we were not successful. Instead of developing a consistent theory we are helping to assemble the mosaic of open problems of the online and approximation algorithms by providing our fragments of the mosaic—we provide a set of various results of our research.

When we look at the problems which we study we will see that we are solving similar scheduling problems for various special cases. Basically there are two reasons:

- the general case is usually too hard to be solved directly, we are getting closer by solving special cases,
- the general case is already solved, but the results are not satisfactory—they are too weak to be used in practice, then we are solving special cases to get stronger results for reasonable restrictions.

In the introduction we have described and discussed the taxonomy of scheduling problems. Let us continue the discussion. But now the discussion will be focused on our research, especially on the problems studied in this thesis.

We show a detailed overview on these problems, we mention related problems and known related results from other authors—the current state

of art—and also results of our research. Of course, we bind these problems to the formal classes of the described taxonomy of the scheduling problems.

2.1 Online scheduling of unit jobs

As the importance of the Internet is growing, people are searching for improvements of the general performance of the global network. In the network the packets are forwarded by network routers and switches. Unfortunatelly, the most of them implements the First–In–First–Out (FIFO) strategy for packet forwarding. However the communication protocols based on IP (Internet Protocol) are sufficiently robust—assumes unpredictable packet flows and heterogenous networks as well. Thus the research of the strategies based on QoS (Quality of Service) for network routers snad switches become more important.

We discuss the problem of online scheduling of unit jobs which arises from the area of buffer management problems in this section. In the buffer management problems we study how to manage buffers for storing network packets in the QoS networks. In such networks packets arrive and are buffered at network switches. Each packet has its QoS value which is the profit gained by forwarding the packet. The network switch works in steps—the switch can receive and transmit only one packet at each step. When the system is overloaded then some packets will not be delivered before their deadline (dropped packets). Such a buffer management problem can be formally described at the following problem of online scheduling of unit jobs.

2.1.1 Problem description

In the model the processing time of each job is equal to 1. Each job is specified by its release time, deadline and weight, where release times and deadlines are integral values and weight is a non–negative real value. These jobs are processed on a single processor—at most one job can be processed at each integral time. It is allowed to drop jobs which cannot be processed before their deadline. The profit is the total weight of all jobs completed before their deadline. The goal is to maximize the obtained profit.

2.1.2 Classification

According to the taxonomy of the online scheduling problems we can describe the problem of the online scheduling of the unit jobs as follows:

- **Machines:** $m = 1$,
- **Online paradigm:** Arriving over time,
- **Objective function:** Maximized total weight of jobs,
- **Computational model:** Deterministic or randomized,
- **Measure:** Standard model,
- **Variants:**
  - Job constraints—processing time of each job is equal to 1.

2.1.3 Previous results

We consider the problem in the randomized computational model, thus the deterministic lower bounds apply for the problem. We also mention the deterministic upper bound—the best randomized algorithm must be better (or equal) than the deterministic algorithm.

- **Upper bound for deterministic algorithms**—The best known upper bound was recently presented in [24]—the 1.828–competitive deterministic algorithm. For a long time the best upper bound was a 2–competitive deterministic algorithm presented in [39]. This result was improved in our joint paper [8]—the 1.939–competitive algorithms, the first algorithm with the ratio strictly below 2. This was recently improved in two independent papers. The first was mentioned as the best known, the second one is the 1.854–competitive algorithm which is presented in [48].
- **Upper bound for memoryless deterministic algorithms**—The best known upper bound for memoryless deterministic algorithms is the 1.893–competitive algorithm presented in [24]. It is the first algorithm with the competitive ratio strictly below 2 for the memoryless algorithms for the problem.

- **Lower bound for deterministic algorithms**—The best known deterministic lower bound on the competitive ratio for the problem is the $\phi \approx 1.618$—this lower bound is based on the 2–bounded input instances and is presented in [4] and [20].

- **Lower bound for randomized algorithms**—The best known lower bound on the competitive ratio for the problem in the randomized model is 1.25, it has been shown in [20] and also based on the 2–bounded instances.

- **Upper bound for randomized algorithms**—There are none, bounded by the deterministic upper bounds.

2.1.4 Our results

We have shown in Section 4.5.1 the following result for the studied problem, this result is published in [Pub–2]:

- **Upper bound for randomized algorithms**—We have improved the upper bound for the problem in the randomized model. We have shown the $\frac{e}{e-1} \approx 1.58$–competitive algorithm which is still the best known upper bound for the problem in the randomized model. This result is published in a joint paper [Pub–2] together with other results of other co–authors. This result is proven in Theorem 4.5.1.

2.2 Online scheduling of uniform jobs

The problem of the online scheduling of uniform jobs arises from the buffer management problems as the problem of the online scheduling of the unit jobs—the problems are closely related and have the same motivation.

This problem introduces a restriction on the input instances for the problem which can be considered as reasonable for practical point of view. Thus it is natural that we can develop stronger algorithms—with better competitive ratio than in the general case. In the considered problem the attribute "uniform jobs" means that the input instances are restricted—these consist of s–uniform jobs, where s is a parameter of considered problem, this means that the span of each of the jobs is equal to s.

2.2.1 Problem description

The problem of the online scheduling of uniform jobs is about the online scheduling of s–uniform input instances of unit jobs, where s is a fixed integer—the parameter of the considered problem. Each job has its span (difference of its deadline and release time) equal to s. Each job is specified by its release time and weight. Jobs are processed on a single processor, some jobs can be dropped. The objective is the total weight of scheduled jobs.

2.2.2 Classification

According to the taxonomy of the online scheduling problems we can describe the problem of the online scheduling of s–uniform jobs as follows:

- **Machines:** $m = 1$,
- **Online paradigm:** Arriving over time,
- **Objective function:** Maximized total weight of jobs,
- **Computational model:** Deterministic or randomized,
- **Measure:** Standard model,
- **Variants:**
  - **Job constraint:** Processing time is equal to 1,
  - **Job constraint:** Span is equal to s.

The span s is a fixed positive integer—it is a parameter of the problem.

2.2.3 Previous results

Because of the parameter s for the problem we can consider the problem in two ways—we can consider the problem for some values of s and we can consider the problem in general—the worst case over all possible values of s as well. Naturally, we are interested in both cases. Thus we distinguish the lower and upper bounds according according to these two cases. Obviously, a lower bound for a special case applies in the general case.

- **Upper bound for deterministic algorithms on general input instances**—The best known general upper bound in the deterministic model is the 1.75–competitive algorithm and it was shown in [6].

- **Upper bound for deterministic algorithms on 2–uniform input instances**—The best known upper bound in the deterministic model for input instances restricted on 2–uniform jobs is the $\sqrt{2} \approx 1.41$–competitive algorithm and it was shown in [4].

- **Lower bound for deterministic algorithms on 2–uniform input instances**—The best known lower bound in the deterministic model for the 2–uniform instances is approximately 1.366, this result was shown in [4].

- **Lower bound for deterministic algorithms on general input instances**—The currently best known upper bound is 1.36. This result follows from the lower bound for the 2–uniform instances shown in [4].

2.2.4 Our results

We have shown in Section 4.6.1 the following results for the studied problem:

- **Lower bound for randomized algorithms on 2–uniform input instances**—We have shown in [Pub–2] a lower bound 1.172 on the competitive ratio for the problem for 2–uniform instances in the randomized model. This is still the best known lower bound in randomized model. This result is proven in Theorem 4.6.2.

- **Lower bound for randomized algorithms on general input instances**—We have shown a lower bound 1.25 on the competitive ratio for the problem with the s–uniform instances, where the span s is tending to infinity. We have presented this result in [Pub–2]. This result is proven in Theorem 4.6.3.

2.2.5 Joint results

The following interesting results are the products of our joint research on the problem with other co–authors. These results were shown in paper [Pub–2] and [Pub–3]. There results are presented in Section 4.6.2.

- **Matching lower bound on the competitive ratio for deterministic algorithms on 2–uniform input instances**—We have shown that there is no deterministic online algorithm for the problem of online scheduling of 2–uniform input instances with competitive ratio smaller than approximately 1.376. This result is presented in Theorem 4.6.5 and matches the following upper bound.

- **Upper bound on the competitive ratio for deterministic algorithms on 2–uniform input instances**—We have shown 1.377–competitive algorithm for the problem of online scheduling of 2–uniform input instances in deterministic model. This is presented in Theorem 4.6.6.

2.3 Online scheduling of bounded jobs

The problem of online scheduling of bounded jobs is very similar to the previous problem of online scheduling of uniform jobs. This problem is also parametrized—again the parameter s restricts the span of jobs in the input instances. The difference is that in the problem with uniform jobs the span must be equal to the parameter s and in the problem with bounded jobs the span must be at most s.

2.3.1 Problem description

The problem of online scheduling of bounded jobs is about online scheduling of s–bounded input instances of unit jobs, where s is a fixed integer—the parameter of the considered problem. Each job has its span (difference of its deadline and release time) at most s. Each job is specified by its release time and weight. Jobs are processed on a single processor, some jobs can be dropped. The objective is the total weight of scheduled jobs.

2.3.2 Classification

According to the taxonomy of online scheduling problems we can describe the problem of the online scheduling of s–uniform jobs as follows:

- **Machines:** $m = 1$,

- **Online paradigm:** Arriving over time,

- **Objective function:** Maximized total weight of jobs,
- **Computational model:** Deterministic or randomized,
- **Measure:** Standard model,
- **Variants:**
  - **Job constraint:** Processing time is equal to 1.
  - **Job constraint:** Span is at most s.

The maximal span s is a fixed positive integer—it is a parameter of the problem.

2.3.3 Previous results

For the problem of the online scheduling of uniform jobs we consider the problem from two points of view—as the worst case over all possible values of the parameter s of the problem and the problem for a fixed parameter s. Observe that in the studied problem of the online scheduling of bounded jobs the set of input instances for a parameter s contains also all input instances for all parameters smaller than s. Thus the worst case over all possible values of the parameter s is given by the parameter s tending to infinity.

Moreover the problem for the parameter s tending to infinity colapses to the previously described problem of online scheduling of unit jobs because with growing parameter s we lose boundaries.

- **Upper bound for deterministic algorithms on general input instances**—the best known general deterministic upper bound for the problem—for unlimited value of parameter s—is the 2–competitive algorithm that has been shown in [39].
- **Upper bound and lower bound for deterministic algorithms on 2–bounded input instances**—The optimal algorithm is known for the problem for the 2–bounded instances—this algorithm is $\phi \approx 1.618$–competitive. This algorithm was introduced in [39] and corresponding lower bound was shown independently in [4] and [20].
- **Lower bound for randomized algorithms on 2–bounded input instances**—The best known lower bound on the competitive ratio in the randomized model is 1.25 and was shown in [20].

2.3.4 Joint results

The following interesting results are the products of our joint research on the problem with other co–authors. These results were shown in paper [Pub–2]. There results are presented in Section 4.7.

- **Upper bound for deterministic algorithms on s–bounded input instances**—We have shown an algorithm for the problem in the deterministic model, its competitive ratio is given by formula $2 - 2/s + o(1/s)$. The competitive ratio tends to general upper bound 2 for the growing parameter s. Presented in Theorem 4.7.5.

- **Upper bound for deterministic algorithms on 4–bounded input instances**—We have shown the approximately 1.732–competitive algorithm for the problem with 4–bounded input instances in the deterministic model. Presented in Theorem 4.7.4.

- **Upper bound for deterministic algorithms on 3–bounded input instances**—We have shown the $\phi = 1.618$–competitive algorithm for the problem with 3–bounded input instances in the deterministic model. Presented in Theorem 4.7.2.

- **Upper bound for randomized algorithms on 2–bounded input instances**—We have shown the 1.25–competitive algorithm for the problem with 2–bounded input instances in the randomized model. This upper bound matches the best known lower bound mentioned above thus the algorithm is optimal for the case of 2–bounded instances in the randomized model. Presented in Theorem 4.7.1.

2.4 Online scheduling of equal–length jobs

The problem of online scheduling of equal–length jobs is one of the fundamental problems in the area of real–time scheduling. The motivation for the problem is in the real–time scheduling of jobs in overloaded systems where the matching deadlines is very important. We can find practical application in packet switched networks (with and without preemptions)—various streaming and processing applications, when weights are allowed the problem is related to quality of service problems. Our motivation is that the studied problem is a simplified version of fundamental scheduling problem where only a little is known about its competitiveness.

The jobs in the considered problem are almost the same—the jobs have equal processing times and equal weights (weights are not specified). The goal is to maximize the total number of jobs completed before their deadlines.

2.4.1 Problem description

The studied problem is about online scheduling of jobs where the processing time of each job is equal to p where p is a parameter of the problem. Each job is specified by its release time, deadline, where release times and deadlines are integral values. Weights are not specified. The jobs are processed on a single processor. It is allowed to drop jobs that cannot be processed before their deadlines. The considered profit is the total number of jobs completed before their deadlines. The goal is to maximize the obtained profit.

The resulting schedule—produced by the offline and online algorithms—has to be non–preemptive. Althrough we allow preemptions with restarts to the online algorithm in one of studied cases the requirement on the non–preemptive resulting schedule is still satisfied, because we obtain profit only for such a job that is completed before its deadline and its processing was not preempted because of the nature of restarts.

2.4.2 Classification

- **Machines:** $m = 1$,
- **Online paradigm:** Arriving over time,
- **Objective function:** Maximized total number of completed jobs,
- **Computational model:** Deterministic or randomized,
- **Measure:** Standard model,
- **Variants:**
  - **Job constraint:** Processing time is equal to p.

The processing time p is a fixed positive integer—it is a parameter of the problem.

2.4.3 Previous results

Because of the importance of the problem for the area of real–time scheduling, the problem was extensively studied also in its offline version. The feasibility version of the problem was studied in [28]—to goal is to check whether it is possible to schedule all jobs of given input instance. They have shown a deterministic algorithm for the feasibility problem with time complexity $O(n \log n)$. The maximization version of the problem was studied in [21] and [7]—they have shown polynomial but very slow algorithm.

The following results are known for the topic of our interest—the online version of the problem:

- **Upper bound for deterministic algorithms**—In the paper [13] and [12] it was shown that the Greedy algorithm for the studied problem is 2–competitive. The shown result is more stronger—there was shown that any non–preemptive deterministic algorithm that never idles when jobs are available is also 2–competitive.

- **Lower bound for deterministic algorithms**—The fact that the Greedy algorithm is optimal was shown in [29], they have shown a lower bound 2 on the competitive ratio for the studied problem in the deterministic model.

- **Lower bound for randomized algorithms**—The lower bound $4/3 \approx 1.333$ on the competitive ratio for the studied problem in the randomized model was shown in [29].

- **Upper bound for deterministic algorithms on input instances with large slack**—The lower bound 2 on the competitive ratio for the problem in the deterministic model can be beaten when we require sufficiently large slack of jobs. In the paper [29] was shown a 1.5–competitive algorithm for input instances consisting of jobs with slack at least p, it means that each job j satisfies $d_j - r_j \geq 2 \cdot p$. An improvement of this result is presented in [30]—a $(1 + 1/\lambda)$–competitive algorithm for input instances consisting of jobs such that each job j satisfies $d_j - r_j \geq \lambda \cdot p$.

2.4.4 Our results

We have presented the following results for the studied problem in [Pub–1], these results are presented in Section 6.5:

- **Upper bound on deterministic algorithms allowed to restart jobs**—We have shown a 1.5–competitive algorithm for the studied problem such that it is an online scheduling algorithm for the problem which is allowed to restart jobs. Allowed restarts means that the algorithm is allowed to preempt running jobs and the preempted jobs can be completed later—the processing of the job cannot be continued but the job can be again processed from scratch. This result is presented in Theorem 6.5.2.

- **Lower bound on deterministic algorithms allowed to restart jobs**—We have shown a lower bound 1.5 on the competitive ratio of deterministic algorithms allowed to restart jobs. Thus our algorithm is optimal for the studied problem. This result is presented in Theorem 6.5.4.

- **Lower bound on randomized algorithms allowed to restart jobs**—We have shown a lower bound 1.2 on the competitive ratio of randomized algorithms allowed to restart jobs. This result is presented in Theorem 6.5.4.

2.4.5 Joint results

The following interesting results are the products of our joint research on the problem with other co–authors. These results were shown in paper [Pub–1].

- **Upper bound for randomized non–preemptive algorithms**—We present a barely random algorithm which uses only one random bit to choose between two deterministic non–preemptive algorithms. We show that this algorithm is $5/3 \approx 1.667$–competitive for the studied problem. This result is presented in Theorem 6.6.1.

- **Lower bound for barely random algorithms**—We show a lower bound on the competitive ratio of barely random algorithm such that randomly chooses between two deterministic algorithms. The competitive ratio of such an algorithm is at least 1.5. Moreover when the algorithm chooses between the deterministic algorithms with equal probability, then its competitive ratio is at least 1.6. These two results are presented in Theorem 6.6.5 and Theorem 6.6.6.

2.5 Overloaded systems

In this section we focus on preemptive online scheduling problems for overloaded systems. Recall that there are two types of task scheduling problems—first the minimization problems where we have to schedule all jobs and minimize objective like makespan and second the maximization problems where we do not need to schedule all jobs but our goal is to maximize our profit. This problem is about the second case—in the overloaded systems the number of jobs and their processing times exceeds the capacity of machines which we use to process these jobs and thus not all jobs can be completed.

2.5.1 Problem description

In the problem each job is specified by its release time, deadline, processing time and its weight which represents corresponding profit rate. We allow preemptions—the online algorithms are allowed to split each job into several pieces—arbitrary number of pieces with arbitrary granularity. There is only one machine, which can be used for processing of these jobs. The goal is to find schedule for 1 processor, that maximizes the total profit.

We studied the problem in two models with different measures—the standard model and the metered model. In the standard model we get the whole profit for completed tasks only. In the metered model we get the proportional part of the whole profit—given by the fraction of the execution time and processing time.

We study this problem in the deterministic model. For the analysis of the problem we used the methods of the competitive analysis including resource augmentation.

2.5.2 Classification

According to the taxonomy of the online scheduling problems we can describe the problem of the online scheduling in overloaded systems as follows:

- **Machines:** $m = 1$,
- **Online paradigm:** Arriving over time,
- **Objective function:** Maximized total weight of processed jobs,
- **Computational model:** Deterministic,

- **Measure:** Standard model or metered model,
- **Variants:**
  - **Job constraint:** Preemptions are allowed, each job can be divided into arbitrary number of pieces (disjoint intervals) and granularity.
  - **Resource augmentation:** When we study the problem under the resource augmentation, then the online algorithms are allowed to use faster processor—with a speed–up s. It means that the online algorithm uses s times faster processor than the offline algorithm.

2.5.3 Previous results

Standard model

The problem in the standard profit model has been extensively studied. It was shown in [44] and [14] that there is no constant competitive online algorithm for the problem because the shown lower bound depends on so called importance factor—the ratio of jobs with maximum and minimum weights which is unbounded for general input instances. Since there is no constant competitive online algorithm it is natural to study the problem under the resource augmentation framework.

The following results related to the studied problem in the standard profit model were published:

- **Upper bound on the competitive ratio in the standard model**—The $(\sqrt{\xi}+1)^2$–competitive online algorithm for the studied problem, where $\xi = \max_j w_j / \min_j w_j$ is called the *importance factor*, was shown in [44].
- **Lower bound on the competitive ratio in the standard model**—It was shown in [44] and [14] that the algorithm showing the upper bound is optimal algorithm for the studied problem.
- **Upper bound on speed–up in the standard model**—The first online algorithm with constant competitive ratio with speed–up 32 for the problem in the standard model was shown in [37].
- **Upper bound on speed–up in the standard model for 1–competitiveness**—A 1–competitive online algorithm with speed–up

$O(\xi)$ was shown in [46]. The parameter ξ is again the importance factor.

- **Upper bound on speed–up in the standard model for 1–competitiveness and tight jobs**—A 1–competitive online algorithm with speed–up $O(1)$ was shown in [43].
- **Lower bound on speed–up in the standard model for 1–competitiveness and tight jobs**—A lower bound $\phi \approx 1.618$ on the competitive ratio was shown in [45].

Metered model

The problem in the metered profit model has been studied in [17], however, in a different terminology. They studied the problem in the context of thinwire visualization—user is viewing a low–resolution image and uses cursor to generate requests for higher resolution at specified positions. The problem is overloaded because of limited network bandwidth (thinwire). This is a good example of practical application where the partial processing of jobs is benefical for the user.

The following results related to the studied problem in the metered profit model were published:

- **Upper and lower bound on FIRSTFIT and ENDFIT algorithms in the metered model**—In the paper [17] was shown that the online scheduling algorithms known as FIRSTFIT and ENDFIT are both 2–competitive.
- **Lower bound on the competitive ratio in the metered model**—A lower bound on the competitive ratio for the problem was shown in [17]—there is no online algorithm with better competitive ratio than $2(2-\sqrt{2}) \approx 1.17$.

2.5.4 Our results

We have shown the following result for the studied problem, which we has been published in [Pub–4] and which we present in this thesis in Section 5.4.2:

- **Lower bound on speed–up for tight jobs in the standard model**—We have shown that in the standard profit model there is no online 1–competitive algorithm with speed–up $s < 2$ for the problem with the input instances consisting of tight jobs only. Presented in Theorem 5.4.1.

2.5.5 Joint results

The following interesting results are the products of our joint research on the problem with other co–authors. These results were shown in paper [Pub–4] and also these results are mentioned in Section 5.4.3.

- **Upper bound in the metered model**—We have shown a $e/(e-1) \approx 1.58$–competitive algorithm for the problem in metered profit model. Presented in Theorem 5.4.2.
- **Lower bound in the metered model**—We have shown a lower bound $\sqrt{5}-1 \approx 1.236$ on the competitive ratio for the problem in metered profit model. Presented in Theorem 5.4.3.
- **Lower bound on speed–up in the metered model for 1–competitiveness**—We have shown that there is no 1–competitive online algorithm with speed–up better than $\Omega(\log\log\xi)$. Presented in Theorem 5.4.4.
- **Lower bound in standard for 1–competitiveness**—Another resource augmentation result for the model where the online algorithm is allowed to use more machines than the offline algorithm. We have shown that the competitive ratio of online algorithms is at least $\Omega(\sqrt[m]{\xi}/m)$ where m is number of machines used by the online algorithm even if all jobs are tight. Presented in Theorem 5.4.5.

As we have shown that the online algorithms in the standard model cannot achieve the competitive ratio 1 using constant number of machines but these algorithms can achieve the competitive ratio 1 using constant speed–up. Thus the speed–up is more powerful than increasing of number of machines.

2.6 Resource augmentation in online scheduling problems

In this section we consider online scheduling problems with resource augmentation. The resources can be augmented in various ways—usually in time, speed, number of processors or simply by breaking or weaking some other constraints.

The resource augmentation techniques we use when we study interesting or important problems and we are not satisfied with the competitive

ratio of the studied problem. It can happen that the competitive ratio is unbounded for some scheduling problems—then we are searching for the "strong" constraint which is forcing the unboundness of the competitive ratio. Or the competitive ratio is too high—altrough we have proven that the problem is constant competitive but the competitive ratio is too big that it cannot be applied in practice. Especially in the practical applications we need online algorithms because of their simplicity and we require similar quality of results like it is provided by the offline algorithms. Thus we are only satisfied with competitive ratio really close to 1. Therefore it is necessary to break or make weaker some of not so important constraints. Then the result is usually an online algorithm applicable in practice with very good performance, but breaking some minor constraints.

In previous section on equal–length jobs we have presented an online scheduling problem with resource augmentation—augmentation of speed of processors used by the online algorithm. In this section we present the augmentation of deadlines—this means that the online algorithm has more time for processing given jobs.

2.6.1 Problem description

In this problem we study the influence of the resource augmentation on the competitive ratio of one of common scheduling problems - the problem with equal processing times of jobs. The problem is a single processor online scheduling problem and the problem is considered in the deterministic computational model. The time axis is assumed to be integral and jobs arrive one by one in time. In the problem each job is specified by its integral release time and integral deadline and non–negative weight. The processing times of all jobs are equal to a constant.

In this problem we allow augmentation of time—we break the constraints given by deadlines. We consider some constant k and the deadlines for the online algorithm are shifted by k time units to the future while the offline algorithm must process the jobs before their deadlines.

As it is usual in the problems with allowed resource augmentation we are interested in the influence of the resource augmentation on the competitive ratio of the problem.

2.6.2 Classification

Fixed integral constant—the relaxation $k > 0$.

- **Machines:** $m = 1$,

- **Online paradigm:** Arriving over time,
- **Objective function:** Maximized total weight of jobs,
- **Computational model:** Deterministic,
- **Measure:** Standard model,
- **Variants:**
  - **Job constraint:** Consider job j, then
    - for offline algorithm the deadline is d_j,
    - for online algorithm the deadline is $d_j + k$.

2.6.3 Previous results

Very similar problem was studied in [3], which is almost the same problem but in different terminology. They studied the problem of packet buffering and the problem matches our problem for unbounded buffers.

- **Upper bound for greedy algorithm with k–time faster transmissions**—They shown that the greedy algorithm is $(1 + 1/k)$–competitive when such a resource augmentation of speed is considered.

2.6.4 Our results

We present the following results for the studied problem in Section 5.3:

- **Lower bound for 1–competitiveness**—We have shown that there is no online 1–competitive k–relaxed algorithm for any k. Presented in Theorem 5.3.2.
- **Lower bound on the competitive ratio for 1–relaxed**—We have shown a lower bound ≈ 1.05099 on the competitive ratio for 1–relaxed online algorithms for the studied problem. Presented in Theorem 5.3.3.
- **Lower bound on the competitive ratio**—We have shown a general lower bound $1 + \frac{1}{2^{k+1}(k^2+3k+5)}$ for the online k–relaxed algorithms for the problem. Presented in Theorem 5.3.4.

2.7 List scheduling—Previous research

The topic of this thesis—the scheduling and optimization problems—arises from my Master's thesis. Althrough some problems that are presented to in this thesis are very similar to problems studied in the Master's thesis it is not just an extension of the Master's thesis. We are still in the same are but we are solving more more complicated and more general problems.

Let us mention some of the main differences of Master's thesis and this thesis in the area of the scheduling problems—in Master's thesis there are no deadlines, weights of jobs, and there are no other features like preemptions, restarts which can be applied in the studied problems, etc.

We can conclude that we are interested in the same area but we are working on quite different problems.

2.7.1 Problem description

My Master's thesis is focused on randomized online scheduling problem, where jobs arrive one by one. Each job is specified by its release time (time of arrival). Each job is scheduled on one of m processors. The objective is the makespan function (length of the obtained of schedule).

2.7.2 Classification

- **Machines:** arbitrary $m > 0$,
- **Online paradigm:** Arriving one by one,
- **Objective function:** Minimized makespan,
- **Computational model:** Deterministic,
- **Measure:** Standard model.

2.7.3 Our results

The most important result is a negative result, which disproves an old conjecture on the competitive ratio of the problem. Chen, van Vliet and Woeginger [18] and Sgall [54] proved a simple general lower bound of

$$\sigma_m = 1 + \frac{1}{\left(\frac{m}{m-1}\right)^m - 1} \tag{2.1}$$

on the competitive ratio for the randomized online scheduling of jobs on $m \geq 2$ processors.

It seemed natural and reasonable to believe that this lower bound of σ_m on the competitive ratio for m processors indeed is the optimal competitive ratio for every $m \geq 2$. There are at least two facts that support this conjecture: First, for $m = 2$ processors this lower bound of $\sigma_2 = 4/3$ has been proved to be the optimal competitive ratio (Bartal, Fiat, Karloff and Vohra [10]). Secondly, for any number $m \geq 2$ of processors the bound σ_m describes the optimal competitive ratio for the problem variant where *job preemption* is allowed (Chen, van Vliet and Woeginger [18]). This conjecture on σ_m was well–known in the online algorithms community.

We have shown that the conjecture does not hold for $m = 3$ processors. This result was published in [Pub–7], [Pub–9].

Some published algorithms for the online scheduling problem are restricted because of their simplicity—they schedule an arriving job on a processor with the smallest load or on a processor with the second smallest load. Hence in the rest of the Master's thesis we focused on this class of algorithms and we have shown some lower bounds. These results have been published in [Pub–10].

Chapter 3

Preliminaries

This thesis is focused on the approximation and online algorithms, especially on the online scheduling problems and algorithms. We introduce terminology, definitions and notations in the following lines. We state some basic observations related to these topics as well.

This chapter is an overview thus all necessary definitions, notations, etc. are repeated in the following chapters in the context of presented results.

3.1 Jobs

Whenever we consider any scheduling problem then the job is the most important thing which we are thinking about. Sometimes we use the term "task" in the equivalent meaning as the term job.

In the scheduling problems there is one or more machines which are able to process given jobs. As we already discussed in the introduction of this thesis there can be considered lots of variants or constraints to model various scheduling problems.

Usually we define the sequence of jobs as $1, \ldots, n$. Then a single **job** is denoted as j. Each job j is specified by some other quantities—we denote the release time of the job as r_j, the deadline of the job as d_j, its processing time p_j and the weight of the job as w_j. Often the job is specified by such a quadruple (r_j, d_j, p_j, w_j).

In some problems the job j can be specified for example by a triple. Usually it is the case that processing time or weight is equal for all jobs in studied problem. These are the problems when we consider the scheduling of unit jobs or we consider scheduling when we do not care about weights of jobs.

Release time

The **release time** r_j is the first time when the processing of the job j can be started. In the online variant of the scheduling problems it is usually the time, when the job occurs for the online algorithm, as well.

Deadline

Similarly, the **deadline** d_j is the latest time when the processing of the job can be finished. After this moment the job may not be scheduled. In scheduling problem with the standard profit model we do not get any profit for jobs that are not finished up to its deadline. Similarly in scheduling problems with the metered profit model we do not get any profit for processing a job after its deadline.

Processing time

The **processing time** p_j is the amount of processing time, which is necessary to process the job j. Whenever there is $p_j = 1$ for each job then we consider scheduling of unit jobs. Whenever there is a constant p such that $p_j = p$ for each job then we consider scheduling of equal–length jobs.

Span

The difference between the deadline and release time is called **span**. It is sometimes denoted as s_j. Formally, it is defined as $s_j = d_j - r_j$. It is exactly the gap in which the job can be processed.

Weight

The **weight** w_j is the weight of the job j. It is exactly the profit which we gain for the processing of the job—when the job is finished up to the deadline d_j.

Sometimes a job is identified by its weight to simplify the terminology and notation. Thus we can say "job w" meaning "the job with weight w". When there are more jobs with the same weight it is clear from the context which job we refer to or it does not matter.

It is useful for us to define some ordering of the jobs according to their weights. But we need to break ties between jobs with the same weight. Formally we say that a job j is heavier than a job j' it either $w_j > w_{j'}$ or $w_j = w_{j'}$ and $j < j'$.

Weight rate

When we study scheduling problems with allowed preemptions in the metered profit model then it is better to use weight rates of jobs instead of weights of jobs. The notation and usage is almost the same as for weights. The basic difference is the computation of the gained profit for the processing of jobs.

The weight of job j specifies the profit gained for the processing of the whole job j. Opposite to this the weight rate of the job j specifies the profit gained for the processing of the job j for one unit of time. Thus the profit for the whole for job j is $w_j \cdot p_j$ (product of weight rate and processing time).

Tight jobs

Sometimes it is useful to mark a job j as a **tight job** when the scheduling of the job is strongly restricted by given its release time r_j, its deadline d_j and its processing time p_j. The idea is to mark the job whenever the intersection of all possible scheduling intervals of the job j is not empty.

This can be formalized as follows: the job j is **tight** whenever

$$d_j - r_j < 2 \cdot p_j.$$

Expiration time

The **expiration time** is considered in the scheduling problems without preemptions. The expiration time of a job j is the latest moment when the job j can be started to be finished before its deadline.

We denote the expiration time as x_j and it is defined as:

$$x_j = d_j - p_j.$$

Slack of job

The **slack** of a job j specifies the strongness of scheduling constraints given by its release time, deadline and processing time. When the gap between release time and deadline is much larger than the processing time then we have large freedom for scheduling of the job. When size of the gap is equal to the processing time then we have no freedom—we must decide to schedule or not to schedule the job, we cannot postpone the start of processing of the job.

Formally, the slack of the job j can be defined as:

$$d_j - r_j - p_j.$$

The slack of j is zero if and only if we have no freedom for scheduling of the job.

Admissible jobs

Let us consider the scheduling problem without preemptions. We can refer a job j as **admissible job** at time t whenever it was already released and it is not expired yet. Formally the job j is admissible whenever it satisfies:

$$r_j \leq t \leq x_j.$$

Earliest–deadline job

Let us consider a set of jobs J. We mark a job j as **earliest–deadline** in J when the deadline is minimal deadline in jobs in J. Because of technical reasons it is important to break ties between jobs with the same minimal deadline. It means that there is always just one earliest–deadline job for arbitrary non–empty set of jobs. Formally the earliest–deadline job is defined as:

$$\min_{j \in J} \{j; d_j = \min_{j' \in J}(d_{j'})\}.$$

Sometimes we use the abreviation "ED job" instead of the earlist–deadline job.

3.2 Schedules

A **schedule** $\mathcal{S}$ specifies which jobs are executed and at what time they are executed. We can consider such a schedule in several contexts like:

- a description of the behaviour of an online or offline scheduling algorithm on an input instance,
- an optimal schedule on an input instance,
- another schedule in a normalized form—we apply some rules to modify a schedule, the resulting schedule is almost equivalent (improved or worsen according to its purpose) but better to analyze.

3.2.1 The schedule

In the case of a single processor problem the schedule $\mathcal{S}$ is a partial mapping from a set of real numbers to a set of integers. The set of real numbers represents the time axis and the set of integers represents set of jobs.

For a real number t we define the schedule $\mathcal{S}$ as follows:

- $\mathcal{S} : \mathbb{R} \mapsto \mathbb{N}$—the mapping from reals to integers,
- $\mathcal{S}(t) = j$—when the processor is processing the job j at time t,
- $\mathcal{S}(t) =$ *undefined*—when the processor is idling at time t—the processor is not processing any job at time t.

We also require that the schedule must satisfy all constraints given by the definition of studied problem. Especially it must satisfy the basic constraints given by release times, deadlines, processing times, allowed/denied preemptions, etc.

Observations

The definition denies simultaneous processing of two jobs at the same time on the same processor, because of the nature of the mapping.

Let us consider the preimage of the job j—the image $\mathcal{S}^{-1}(j)$ is

- **empty**—when the job j was not scheduled in $\mathcal{S}$,
- **time interval of length** p_j—when the job was scheduled in $\mathcal{S}$ in a scheduling problems without preemptions, we usually denote this interval as $[S_j; C_j)$,
- **union of time intervals**—when the job was scheduled in $\mathcal{S}$ in a scheduling problem with preemptions.

Multiprocessor schedules

Recall that in the area of scheduling we do not distinguish between terms a **processor** and a **machine**.

In the case of studying scheduling problems with multiple processors we must use more complicated notation and definition. In this case the schedule $\mathcal{S}$ represents a collection of mappings—there is a separate mapping for each of the processors.

We use the following notation:

- m—the number of the processors, this notation comes obviously from the term "machine",
- $\mathcal{S} = \{\mathcal{S}_1, \ldots, \mathcal{S}_m\}$—the collection of schedules—one mapping for each processor,
- $\mathcal{S}_i(t) = j$—when the i–th processor is processing the job j at time t,
- $\mathcal{S}_i(t) =$ *undefined*—when the i–th processor is idling at time t.

Obviously, there is an alternative to look at the schedule $\mathcal{S}$ for the multiple processors as on the mapping from the product of integers (representing the set of processors) and real numbers (representing the time axis) to the integers (representing the set of jobs).

The schedule $\mathcal{S}$ must satisfy the additional constraints following from the problem definition. Except the constraints considered for the problems with a single processor we must also consider parallel processing of jobs—whether it is allowed or denied. When parallel processing of jobs is not allowed, then the schedule $\mathcal{S}$ has to satisfy:

$$(\forall t, i, i')\ i \neq i' \Rightarrow \mathcal{S}_i(t) \neq \mathcal{S}_{i'}(t),$$

which exactly means that simultaneous execution of the same job at the same time t on two different processors i and i' is not allowed.

3.2.2 Jobs in schedules

Let us introduce a few terms related to jobs in the context of our consideration on schedules. We describe the terms on a schedule $\mathcal{S}$.

Starting time

We denote the first moment of the scheduling of the job j in the schedule $\mathcal{S}$ as S_j—the **starting time** of the job j. Formally we define the start time as:

$$S_j = \min_{t:\mathcal{S}(t)=j}(t).$$

Completion time

We denote the last moment of the scheduling of the job j in the schedule $\mathcal{S}$ as C_j. In the scheduling problems without preemptions it is exactly

the moment when the job j becomes to be completed—its processing is finished. Formally we define the completion time as:

$$C_j = \sup_{t:\mathcal{S}(t)=j} (t).$$

Execution time

For the schedule $\mathcal{S}$ and a job j we define the **execution time** of the job j as the amount of time which is spent during the processing of the job j.

In the case of studying problems without preemptions the value of execution time of the job j is equal to 0 when the job is not scheduled, or the processing time p_j when the job is scheduled. In the problems with allowed preemptions the execution time of job j is obviously between 0 and the processing time p_j according to the spent time. In the problems with allowed restarts of jobs we can get executaion time greater than p_j.

Formally, the execution time is defined as the size of interval (or set of intervals with preemptions) $\mathcal{S}^{-1}(j)$.

Pending jobs

We mark a job j at time t for a schedule $\mathcal{S}$ as a **pending job** or as an **available job** whenever it can be scheduled at time t. The job j can be scheduled at time t is it is allowed by contraints given by release times, deadlines, etc. and the job was not scheduled before time t in s.

The explicit formal definition based on the release time, the deadline, and the processing time is different for different problems. For example when we allow or deny preemptions it is obvious that we obtain quite different definitions.

In a simple case—in an online scheduling problem without preemptions—the job j can be marked as a pending job when $t \geq r_j$ and $t + p_j \leq d_j$ and the job j was not scheduled in $\mathcal{S}$ before t.

Sometimes we refer to the **pending jobs**—then we mean the set of jobs which are pending at time t in the schedule $\mathcal{S}$.

When we consider scheduling problems with weighted jobs then we can assume without loss of generality that there always exists a pending job with weight equal to 0 in the input instance. Obviously, this assumption cannot change the profit of the optimal schedule and cannot change the profit which can be obtained by an online algorithm as well.

Feasible sets of jobs

Sometimes during the analysis we are interested in analyzing some sets of jobs. For this purpose we define this term—**feasible jobs**. Some set of jobs is feasible when it is possible to schedule all its jobs with respect to all the constraints given by the studied scheduling problem.

Let us assume for a while a scheduling problem without preemptions and some set of jobs J. We say that the set J is **feasible** at time t when there exists a schedule $\mathcal{S}$ such that all jobs of J are completed in $\mathcal{S}$, do not start before t in $\mathcal{S}$ and $\mathcal{S}$ is a valid schedule for the studied scheduling problem.

Flexible jobs

Again we consider the set of jobs J where the processing time of each job is equal to p and we are interested in the feasibility of the set of jobs. We consider the set J that is feasible at time t and we analyse whether it is kept feasible in near future.

The motivation arises in the scheduling problems without preemptions—in such problems we must consider that some heavy (big profit) jobs can arrive in near future. So we postpone the processing for the longest time without losing any of known jobs.

For our purposes it is sufficient to define the term for the scheduling problems where the processing time of all jobs is equal to p. In this case we denote that the set of jobs is **flexible** at time t if and only if the set is feasible at time $t + p$.

We mark a job j as a **flexible job** at time t when the set of pending jobs is flexible at time t. Otherwise we mark the job j as an **urgent job**.

3.2.3 Throughput

In the scheduling problem there is some objective which we want to optimize—maximize or minimize—the main reason for spending our effort during the scheduling.

When we discuss a scheduling problem we usually refer to this objective. People use the following terms for this objective:

- throughput,
- benefit,
- profit,
- gain.

All these terms are really used in the same meaning—to represent the objective. This is simply the profit which is payed to the owner of the algorithm. This payment is our motivation to make the best schedule as we are able to do.

As we mentioned in the taxonomy of the online scheduling problems, there is a lot of common objective functions. But in the problems which we study we use only the following:

- **Total weight of jobs**—The sum of weights of all completed jobs. For the scheduling problems with preemptions in the metered model we use the weight rates and we gain the profit proportionally to the processed part of each job.
- **Makespan**—The time when the last job is completed (the latest completion time). We use this objective in the scheduling problems where the weights of jobs are not specified.

3.2.4 Earliest–deadline schedules for equal–length schedules

We define earliest–deadline schedules for the scheduling problems restricted on the input instances with equal–length jobs. This restriction follows from the definition of flexible jobs where we assume jobs with equal lengths (processing times).

During the analysis of some scheduling problems it is useful to assume that the studied schedule is in some special form. Thus we define the **normal** schedule—each schedule can be simply transformed into such form.

We assume a scheduling problem of equal–length jobs on a single processor.

Equivalent schedules

Let us assume two schedules $\mathcal{S}$ and $\mathcal{S}'$ for an input instance J. These two schedules are considered to be **equivalent** when the following conditions are satisfied for each time t:

- $\mathcal{S}$ starts a job at t if and only if $\mathcal{S}'$ starts a job at t,
- Suppose that $\mathcal{S}$ starts a job j at time t and $\mathcal{S}'$ starts a job j' at time t. Then j is flexible in $\mathcal{S}$ if and only if j' is flexible in $\mathcal{S}'$. Moreover, if the schedules are both flexible then $j = j'$.

When $\mathcal{S}$, $\mathcal{S}'$ are equivalent, then it is obvious that their profits are equal.

Normal schedule or EDF

A schedule $\mathcal{S}$ is **normal** if it satisfies the following two properties:

- when the schedule $\mathcal{S}$ starts a job, it chooses the earliest–deadline job from the set of jobs which are scheduled in $\mathcal{S}$ and pending at time t,
- when the set of all pending jobs (from the scheduled jobs) in $\mathcal{S}$ at some time t is not flexible, then some job is running at t.

It is obvious that an arbitrary schedule can be simply transformed into an equivalent normal schedule. This observation can be proven by mathematical induction. We always find the first job which breaks the earliest–deadline condition and we switch this job with another job (scheduled later) which satisfies this condition. This step can be repeated until we get normal schedule.

3.2.5 Time–sharing

The time–sharing is one of useful metods which can help us with analysis of studied online scheduling problems. Especially it is very useful for online scheduling problems with allowed preemptions.

This technique provides generalization of the schedule which is produced by the online and offline algorithms. Recall that in the standard schedule, we always have to specify the running job—for each time moment and each processor.

The basic principle of this technique is the generalization of scheduling of a job by allowing simultaneous running of two or more jobs in the same time unit. These jobs share the same time unit. Naturally, the performace of considered processors is limited, thus while we consider sharing of time unit we also need to consider sharing of this performance. For each job and each time moment we also specify the running speed. The sum of speeds of all jobs for each fixed time moment and considered processor cannot exceed the speed of the processor.

Let us consider an online scheduling problem on a single processor and real time axis. Then the generalized schedule can be defined as the following mapping: $V(j,t)$ is the running speed of the job j at time t and satisfies following conditions. For each time moment the total speed of all

jobs cannot exceed the speed of the processor:

$$(\forall t) \sum_j V(j,t) \leq 1.$$

The speed cannot be negative:

$$(\forall j,t) V(j,t) \geq 0.$$

For each job the total processing time cannot exceed the processing time of the job:

$$(\forall j) \int_0^\infty V(j,t)dt \leq p_j,$$

Each job can be scheduled between release time and deadline only:

$$(\forall j,t)\ V(j,t) > 0 \Rightarrow t \in [r_j; d_j).$$

3.3 Competitive analysis

The competitive analysis has been already widely discussed in this thesis. Let us remind the basic facts and definitions related to the competitive analysis and the competitive ratio.

The competitive analysis is a variation of the traditional worst–case analysis applied on the online scheduling problems. The method allows us to reasonably compare the performace of the online algorithms for the online scheduling problems and also compare them to the optimal offline algorithms.

The "competitive ratio" is the measure used for comparing the performance of the algorithms. The competitive ratio is the ratio of the quality of the results produced by an online algorithm and the optimal offline algorithm. The competitive ratio of the algorithm is the worst competitive ratio over all input instances. We distinguish the definitions for the minimalization and maximalization problems. The competitive ratio for the minimalization problem is defined as:

$$\mathcal{R}(\mathcal{A}) = \inf_{R \in \mathbb{R}} \{R : (\forall \sigma), \mathcal{A}(\sigma) \leq R \cdot \mathrm{OPT}(\sigma)\}$$

and similarly for the maximization problem as follows:

$$\mathcal{R}(\mathcal{A}) = \inf_{R \in \mathbb{R}} \{R : (\forall \sigma), R \cdot \mathcal{A}(\sigma) \geq \mathrm{OPT}(\sigma)\},$$

where $\mathcal{A}$ denotes objective of the algorithm, OPT denotes the objective of the optimal offline algorithm.

When we consider a fixed scheduling problem then we also define the **competitive ratio of the problem** as

$$\mathcal{R} = \inf_{\mathcal{A}} \mathcal{R}(\mathcal{A}),$$

which is going over all online algorithms for the problem.

There are some extensions of the competitive analysis. One of the most important extensions is the randomized version of the competitive analysis. When we consider the randomized online scheduling algorithms, then we can define the competitive ratio for the minimalization problem as follows:

$$\mathcal{R}(\mathcal{A}) = \inf_{R \in \mathbb{R}} \{R : (\forall \sigma), \mathbb{E}[\mathcal{A}(\sigma)] \leq R \cdot \mathrm{OPT}(\sigma)\}$$

and similarly for the maximization problem as follows:

$$\mathcal{R}(\mathcal{A}) = \inf_{R \in \mathbb{R}} \{R : (\forall \sigma), \mathbb{E}[\mathcal{A}(\sigma)] \geq R \cdot \mathrm{OPT}(\sigma)\}.$$

Recall the important fact that we do not consider the randomization over input instances but we consider randomization over random bits provided to the randomized online algorithms. Thus we do not study "average behaviour in average cases" of the online algorithms but we study the real expected worst case behaviour of the randomized online algorithms.

Chapter 4

Scheduling

In this chapter we study restrictions of the traditional scheduling problems. First of all we study the problem of scheduling of unit jobs—the input instances contain jobs with processing time equal to 1, each job is specified by its release time, deadline and non–negative weight. We consider this problem in the deterministic and randomized computational model.

In the second part of this chapter we study the problem of scheduling of uniform jobs. It is another restriction of the traditional scheduling problems—again the input instances are restricted on unit jobs, moreover the span (difference of deadline and release time) is fixed and equal for all jobs. For example, the problem of scheduling of s–uniform jobs means the problem of scheduling of jobs with span equal to s.

The third part of this chapter is about the problems of scheduling of s–bounded jobs. This is very similar to previous problem of scheduling of s–uniform jobs. In the uniform case there is given the span exactly as s but in the bounded case the span is bounded by s—it means that span is at most s.

4.1 Results overview

This chapter is divided into three parts as we mentioned above. According to this we also divide results—we separate our results and joint results and we also separate results of research on the problem of online scheduling of unit jobs, uniform jobs and bounded jobs. We have selected only the most interesting joint results.

Our results

Here we mention results which are products of our own research.

Scheduling of unit jobs

We have shown in Theorem 4.5.1 a randomized online scheduling algorithm for the problem of online scheduling of unit jobs in standard profit model. This algorithm is $e/(e-1) \approx 1.58$–competitive and this has been published in [Pub–2].

Scheduling of s–uniform jobs

In the case of online scheduling of s–uniform jobs we are focused on the randomized again. We have shown two lower bounds and both of these results have been published in [Pub–2]. We have shown in Theorem 4.6.2 a lower bound 1.172 on the competitive ratio for the problem of online scheduling of 2–uniform jobs in the randomized model.

Regarding the general case we have shown a lower bound for randomized algorithms 1.25 on the competitive ratio, again in the standard profit model. See Theorem 4.6.3.

Joint results

Here we mention results which are products of joint research with co–authors in the studied areas.

Scheduling of s–uniform jobs

The following results were published in [Pub–2] and [Pub–3]. As a product of joint research we have shown upper bound and matching lower bound on the competitive ratio of the online algorithms for the problem of online scheduling of 2–uniform jobs in deterministic model.

In Theorem 4.6.5 we have shown a lower bound on the competitive ratio approximately 1.376. And also in Theorem 4.6.6 we have shown a matching deterministic online algorithm for the problem of scheduling of 2–uniform jobs. Thus the problem for 2–uniform jobs have been fully solved.

Scheduling of s–bounded jobs

The following results were publised in [Pub–2] and [Pub–3] and are the products of joint research. In Theorem 4.7.5 we have shown a deterministic algorithm on general s–bounded input instances with competitive ratio $2 - 2/s + o(1/s)$.

Except the general case we were interested also in the special cases. In Theorem 4.7.4 we are focused on the special case for $s = 4$, we are focused on online deterministic scheduling of 4–bounded input instances. We have shown approximately 1.732–competitive algorithm for this case. In Theorem 4.7.2 we have shown approximately 1.618–competitive online deterministic algorithm for 3–bounded instances.

We were also working on the randomized model. In Theorem 4.7.1 we have shown a 1.25–competitive online algorithm for 2–bounded input instances. This results matches the lower bound for 2–uniform input instances (2–uniform instance is also 2–bounded). Thus the problem of 2–bounded online scheduling in the randomized model is solved.

4.2 Introduction

We are focused on problems of online scheduling of unit jobs that arise from the buffer management problems. The online bounded delay buffer problem has been introduced in [39, 4] to model the trade–offs arising in managing buffers for storing packets in QoS networks. In the problem the packets arrive to network switches and can be buffered at the network switches. At each step several packets can be received (e.g. on different ports) but at most one packet can be transmitted. Each packet has its QoS value—this is the profit gained by forwarding it. The delay between the endpoints of communication is limited, therefore a deadline is specified for each packet. The deadline specifies the latest time when packet can be transmitted. Obviously, each network switch can be easily overloaded by received packets—packets arrive faster than can be transmitted. Due to the overloading conditions the switch is allowed to drop jobs. The goal is to maximize the profit—gained QoS values of packets transmitted before their deadlines.

The following problem of online scheduling of unit jobs is equivalent to the described buffer management problem. The packets correspond to jobs—each job is specified by its release time, deadline and weight, where release times and deadlines are integers and weight is a non–negative value, the processing time of each job is equal to 1. The switch is rep-

resented by a single processor (at most one job can be processed at each integral time). The profit is the total weight of all jobs completed before their deadlines. The natural goal is to schedule jobs in such a way to maximize the profit.

Note that we focus on online scheduling problems but it is possible to consider an offline variant of the scheduling problem and buffer management problem as well.

In the online variant of the scheduling of unit jobs each job arrives at its release time. The algorithm performs in steps and the online algorithm has to schedule one of pending jobs without any knowledge of the jobs that will arrive in the future.

This problem of online scheduling of unit jobs is also closely related to other QoS problems. One such related problem is online scheduling of jobs in the metered model, where each job is specified by its release time, deadline, processing time and weight. Preemptions are allowed in the problem. Unlike in the standard model each non–completed job gains proportionally—it means that when the algorithm processes p percents of scheduled job then the algorithm gains p percents of profit assigned to the job. It is easy to see that both problems are equivalent. Applications of the problem arise from transfering large images over slow networks (with low bandwidth) or imprecise computations in the real–time systems, where tasks can be completed partially (it can cause worse quality of result).

4.3 Previous work

Although the problem of scheduling of unit jobs was widely studied, the best known algorithm for the general deterministic and randomized computational model was the Greedy algorithm. The Greedy algorithm for the problem is naturally defined—it always schedules the heaviest job.

It is easy to show that the competitive ratio of the Greedy algorithm is equal to 2. The worst input instance showing the lower bound of the competitive ratio is really simple—the Greedy algorithm prefers heaviest job even if the input instance contains more urgent job with almost the same weight. When we consider such an instance with two jobs where the smaller job must be processed immediately otherwise it is lost, then the Greedy algorithm loses almost half of optimal profit.

A lower bound of $\phi \approx 1.618$ for the problem in determinitic model was shown in [4, 20, 5].

Some restrictions on instances of the problem have been studied in the literature [38, 39, 4, 20, 48]. Let the *span* of a job be the difference between

its deadline and the release time. In *s–uniform instances* the span of each job is equal exactly s. In *s–bounded* instances, the span of each job is at most s. The lower bound of $\phi \approx 1.618$ in [4, 20, 5] applies even to 2–bounded instances. A matching upper bound for the 2–bounded case was presented in [38, 39]. Algorithms for 2–uniform instances were studied by Andelman *et al.* [4], who established a lower bound of $\frac{1}{2}(\sqrt{3}+1) \approx 1.366$ and an upper bound of $\sqrt{2} \approx 1.414$. This upper bound is tight for memoryless algorithms [8], that is, algorithms which base their decisions only on the weights of pending jobs and are invariant under scaling of weights. Finally, the first deterministic algorithms with competitive ratio lower than 2 for the s–bounded instances appear in [8]. These ratios, however, depend on s, and approach 2 as s increases. Kesselman *et al.* [40] gave an algorithm for s–uniform instances with ratio 1.983, independent of s.

Kesselman *et al.* [38, 39, 40] consider a different model related to the s–uniform case: packets do not have individual deadlines, but instead they are stored in a FIFO buffer of capacity s (i.e., if a packet is served, all packets in the buffer that arrived before the served one are dropped). Any algorithm in this FIFO model applies also to s–uniform model and has the same competitive ratio [40]. Bansal *et al.* [6] gave a deterministic 1.75–competitive algorithm in the FIFO model; this implies a 1.75–competitive algorithm for the s–uniform case.

We say that jobs are *similarly ordered* if $r_i < r_j$ implies $d_i \leq d_j$ for all jobs i, j. Note that this includes s–uniform and 2–bounded instances, but not s–bounded ones for $s \geq 3$. Recently, Li *et al.* [48] gave a ϕ–competitive algorithm for instances with similarly ordered jobs; this matches the lower bound for 2–bounded instances, and thus it is an optimal algorithm for this case. (Li *et al.* [48] use a different terminology for the same restriction on inputs and say that the jobs have *agreeable deadlines*.)

4.4 Preliminaries

We present our results in terms of scheduling of unit jobs, as explained in the introduction. A *schedule* S specifies which jobs are executed, and for each executed job j it specifies an integral time t, $r_j \leq t < d_j$, when it is scheduled. (When we say that j is scheduled at time t, we mean that j is started at time t, and thus it occupies the processor through the time interval $[t, t+1)$.) Only one job can be scheduled at any time t. The *throughput* or *profit* of a schedule S is the total weight of the jobs executed in S. If $\mathcal{A}$ is a scheduling algorithm, by $\mathit{profit}_{\mathcal{A}}(I)$ we denote the profit of the schedule computed by $\mathcal{A}$ on an instance I. The optimal profit on I is

denoted by $opt(I)$. A job i is *pending* in S at time t if $r_i \leq t < d_i$ and i has not been scheduled in S before t. (Thus all jobs released at time t are considered pending.) An instance is *s–bounded* if $d_j - r_j \leq s$ for all jobs j. Similarly, an instance is *s–uniform* if $d_j - r_j = s$ for all j. The difference $d_j - r_j$ is called the *span* of a job j. An instance is *similarly ordered* if the release times and deadlines are similarly ordered, that is $r_i < r_j$ implies $d_i \leq d_j$ for any two jobs i and j.

Given two jobs i, j, we say that i *dominates* j if either (i) $d_i < d_j$, or (ii) $d_i = d_j$ and $w_i > w_j$, or (iii) $d_i = d_j$, $w_i = w_j$ and $i < j$. (Condition (iii) only ensures that ties are broken in some arbitrary but consistent way.) Given a non–empty set of jobs J, the *dominant* job in J is the one that dominates all other jobs in J; it is always uniquely defined as 'dominates' is a linear order.

A schedule S is called *canonical earliest–deadline* if for any jobs i and j in S, where i is scheduled at time t and j is scheduled later, either j is released strictly after time t, or i dominates j. In other words, at any time, the job to be scheduled dominates all pending jobs that appear later in S. Any schedule can be easily converted into a canonical earliest–deadline schedule by rearranging its jobs. Thus we may assume that offline schedules are canonical earliest–deadline.

We often view the behavior of an online algorithm $\mathcal{A}$ as a game between $\mathcal{A}$ and an adversary. Both algorithms schedule jobs released by the adversary whose objective is to maximize the ratio $opt(I)/profit_{\mathcal{A}}(I)$. Several of the upper bound proofs are based on a *potential function* argument. In such proofs, we define a potential function Φ that maps all possible configurations into real numbers. (In general, a configuration at a given step may include all information about the computation before and after this step, both for the algorithm and the adversary. In most arguments, however, it is sufficient to include only the set of pending jobs in both schedules.) Intuitively, the potential represents $\mathcal{A}$'s savings at a given step. At each time step, an online algorithm and the adversary execute a job. The proofs are based on the following lemma which can be proven by a simple summation over all steps.

Lemma 4.4.1 *Let $\mathcal{A}$ be an online algorithm for scheduling unit jobs. Let Φ be a potential function that is 0 on configurations with no pending jobs, and at each step satisfies*

$$R \cdot \Delta profit_{\mathcal{A}} \geq \Delta adv + \Delta\Phi, \tag{4.1}$$

where $\Delta\Phi$ represents the change of the potential, and $\Delta profit_{\mathcal{A}}$, Δadv represent $\mathcal{A}$'s and the adversary profit in this step. Then $\mathcal{A}$ is R–competitive.

The lemma above applies to randomized algorithms as well. In that case we need to prove the inequality on average with respect to the algorithm's random choices at the given step, i.e., to prove that $\mathbf{Exp}[R \cdot \Delta \mathit{profit}_{\mathcal{A}} - \Delta\Phi] \geq \Delta \mathit{adv}$, as both the profit of the algorithm and the change of the potential are influenced by the random choices.

In some proofs, in particular for deterministic algorithms, we use a different approach called *charging*. In a charging scheme, the weight of each of the jobs in the adversary schedule is charged to some time in our schedule, in such a way that for each time t the weight of all jobs charged to t is at most R times the total profit of the job(s) scheduled in our schedule at t. If such a charging scheme exists, by simple summation over all time steps, it implies that our algorithm is R–competitive. A charging scheme can be transformed into a potential method argument, with the potential function at a given time equal to the sum of the charges going backward in time across this time minus the sum of the charges going forward. However, proofs based on charging schemes tend to be more illuminating.

4.5 Unit jobs scheduling

In this section we focus on unrestricted version of the problem of online scheduling of unit jobs. In this problem, each job j is specified by (r_j, d_j, w_j), release time r_j is integer, deadline d_j is integer, weight w_j is non–negative real, unit processing time $p_j = 1$. Jobs are processed on a single processor.

4.5.1 Randomized model

We develop a randomized algorithm called RMIX for the problem of scheduling of unit jobs described below. We will prove that the competitive ratio of this algorithm is $e/(e-1) \approx 1.582$. This is the first algorithm with the competitive ratio smaller than 2. Previously the best known algorithm was the Greedy algorithm, its competitive ratio is 2. The lower bound for the Greedy algorithm obviously follows from the fact that the algorithm always schedules the heaviest job.

The basic principle of our algorithm RMIX is based on similar ideas as the algorithm MIXED presented in our joint paper [Pub–4]. But the algorithm MIXED is a scheduling algorithm for quite different problem—the problem of scheduling of metered jobs in a single processor system which can run simultaneously several jobs (the performance is split among running jobs in according to specified speeds).

The only way to improve the competitive ratio is to break this Greedy rule. We found during our considerations on the lower bound of the Greedy algorithm that sometimes it is better to prefer a more urgent job over another heavier job. Thus our algorithm balances between heavy jobs and urgent jobs. We were considering scheduling of an earliest–deadline pending job among some heavy jobs. We used this idea to develop our algorithm EDF$_\alpha$ for the deterministic model, this algorithm schedules the earliest–deadline job among jobs with weight at least α–times weight of the heaviest pending job. The EDF$_\alpha$ improves the competitive ratio for s–bounded instances.

We consider the randomized computational model thus randomization can help us to improve the competitive ratio. We have to define probabilities for each of pending jobs. Algorithm RMIX works in steps and in each step it randomly chooses a job from a set of pending jobs where each job has assigned its probability. The algorithm considers sequence of heavy pending jobs such that the deadlines of jobs are decreasing and weight of each job is at least $1/e \approx 0.368$ times weight of the heaviest job and each job is the heaviest job among the pending jobs with same or smaller deadline. Non–zero probability is assigned to jobs in the sequence according to their weights. The probability of other pending jobs is zero. Obviously the sequence tries to balance between urgency and weight. The next job in the sequence is always more urgent because of smaller deadline but has still reasonable weight—in comparison with the weight of the heaviest pending job. Formally we define the RMIX algorithm as follows:

Algorithm RMIX**.** At each step, we inductively select a set of jobs $h_1, \ldots, h_k$ as: h_1 is the heaviest pending job, h_{i+1} is the heaviest pending job such that $w_{h_{i+1}} > w_{h_1}/e$ (limited weight) and $d_{h_{i+1}} < d_{h_i}$ (earlier deadline). We define $v_i = w_{h_i}$ and $v_{k+1} = w_{h_1}/e$. The algorithm executes one of $h_1, \ldots h_k$, it executes h_i with probability $\ln(v_i) - \ln(v_{i+1})$.

Theorem 4.5.1 *Let us consider the problem of online scheduling of unit jobs in the randomized model with the standard profit model. Above we described the randomized algorithm* RMIX *for considered problem.*

Then the algorithm RMIX *is* $\frac{e}{e-1} \approx 1.58$*–competitive.*

Proof. We prove the competitive ratio of the algorithm using one of the methods of the competitive analysis—we use the potential function to prove algorithm's competitive ratio. The basic idea of this method is to analyze the algorithm in steps. When we want to prove that the competitive ratio is c then we must show that the adversary gains at most

c times more than the algorithm gains and we can help us by lending some "credit" from the potential function. Of course the potential function must be well–defined—must start and end with the same value, usually zero.

It is obvious that any schedule can be modified to earliest–deadline schedule. In the earliest–deadline schedule whenever two jobs are pending at the same time then they occure in the schedule in the order given by their deadlines—a job with later deadline cannot be processed before another job with earlier deadline in such schedule. Every schedule can be modified to be earliest–deadline without change of its profit and without breaking constraints given by release times, deadlines. Thus we assume without loss of generality that the adversary's schedule is earliest–deadline.

Let M be the set of pending jobs in RMIX. Also, by A we denote the set of pending jobs in the adversary schedule. The jobs are removed from A either when they are executed, or when a job with a later deadline is executed.

Define the potential function

$$\Phi = \sum_{i \in A-M} w_i.$$

Job arrivals and expirations cannot increase the potential. So we only need to analyze how the potential changes after job execution.

Note a fact for all $a \geq b > 0$

$$a(\ln a - \ln b) \geq a - b. \tag{4.2}$$

Consider a time step t when the adversary executes a job j. The expected weight gained by RMIX is

$$\omega = \sum_{i=1}^{k} w_{h_i}(\ln v_i - \ln v_{i+1}).$$

Notice that (4.2) implies

$$w_{h_1} \leq \frac{e}{e-1}\omega.$$

Suppose first that $j \in A - M$. Executing j by the adversary, decreases the potential function by w_j. At the same time, one of $h_1, \ldots, h_k$ can be added to $A - M$, h_i with probability $\ln v_i - \ln v_{i+1}$, increasing the potential by at most ω.

So $\Delta\Phi \leq -w_j + \omega$. It follows that

$$w_j + \Delta\Phi \leq \omega \leq \frac{e}{e-1}\omega.$$

Now assume that $j \in M$. If $w_j \leq w_{h_1}/e$, then

$$w_j + \Delta\Phi \leq w_{h_1}/e + \omega \leq \omega/e + \omega = \frac{e}{e-1}\omega.$$

So we can assume that $w_j > w_{h_1}/e$. Since $j \in M$ there exists p such that

$$v_p \geq w_j > v_{p+1},$$

hence

$$d_{h_p} \leq d_j.$$

By the assumption on the adversary, he will not execute $h_p, \ldots h_k$ in the future, so these are removed from A. The expected potential increase is then at most

$$\sum_{i=1}^{p-1} w_{h_i}(\ln v_i - \ln v_{i+1}).$$

Note that

$$w_{h_i}(\ln v_i - \ln v_{i+1}) - (v_i - v_{i+1}) \geq 0$$

and obviously

$$w_{h_1} = \frac{e}{e-1}\sum_{i=1}^{k}(v_i - v_{i+1}).$$

So

$$w_j + \Delta\Phi \leq w_{h_p} + \sum_{i=1}^{p-1} w_{h_i}(\ln v_i - \ln v_{i+1}) =$$

$$= \sum_{i=1}^{p-1}(w_{h_i}(\ln v_i - \ln v_{i+1}) - (v_i - v_{i+1})) + v_1$$

$$= \sum_{i=1}^{p-1}(w_{h_i}(\ln v_i - \ln v_{i+1}) - (v_i - v_{i+1})) + \sum_{i=1}^{p-1}(v_i - v_{i+1}) + w_{h_p}$$

$$= \sum_{i=1}^{p-1}(w_{h_i}(\ln v_i - \ln v_{i+1}) - (v_i - v_{i+1})) + \frac{e}{e-1}\sum_{i=1}^{k}(v_i - v_{i+1})$$

$$\leq \frac{e}{e-1}\sum_{i=1}^{k} w_{h_i}(\ln v_i - \ln v_{i+1}) = \frac{e}{e-1}\omega,$$

completing the proof.

We have shown that whenever the adversary schedules any job denoted as w_j the profit of adversary plus the average change of potential function

(loan or refund) denoted as $\Delta\Phi$ is not more then $\frac{e}{e-1}$ times the average profit of the RMIX algorithm denoted as ω. Obviously the potential function Φ starts end ends with zero value. Thus we have shown that the competitive ratio of RMIX is $\frac{e}{e-1} \approx 1.58$.

□

4.6 Uniform jobs scheduling

We consider the following online scheduling problem of unit jobs. Each job j is specified by (r_j, w_j), release time r_j is integer, weight w_j is non–negative real, unit processing time $p_j = 1$. Deadline $d_j = r_j + s$ for the s–uniform jobs.

4.6.1 Randomized lower bounds

In this section we consider s–uniform instances, where $d_j = r_j + s$ for each job j. We first prove a lower bound on the competitive ratio of randomized algorithms which increases with s and tends to 1.25 for large s. This improves the (deterministic) lower bound of $4 - 2\sqrt{2} \approx 1.172$ from [38] for $s \to \infty$.

Moreover this implies lower bound $4 - 2\sqrt{2}$ on the competitive ratio for 2–uniform instances.

Theorem 4.6.1 *Let us consider a fixed integer $s > 0$ and the problem of online scheduling of s–uniform instances of unit jobs in the randomized model with the standard profit model.*

We define

$$R_s \;=\; 1 + \frac{s-1}{2s - 1 + 2\sqrt{s^2 - s}}\,.$$

Then there is no algorithm solving this problem better than R_s–competitive.

Proof: We use Yao's minimax principle, in the form applicable to the lower bounds on the competitive ratios [15]. Following this principle, it is sufficient to give a distribution on instances for which the ratio between the expected profit of any deterministic online algorithm and the expected optimal profit is no better than R_s.

We generate an instance randomly as follows. Fix a large integer n and let $a = 1 + \sqrt{s/(s-1)}$ and $p = 1/a$. (Note that $a = 1 + \sqrt{2}$ for $s = 2$ and $a \to 2$ for $s \to \infty$.) Each instance consists of stages $0, 1, \ldots$, where in stage

i we have s jobs of weight a^i released at time si and $s-1$ jobs of weight a^{i+1} released one by one at times $si+1$, $si+2$, ..., $si+s-1$. After each stage $i \leq n$, we continue with probability p or stop with probability $1-p$. After stage n, if the process has not yet terminated, then at time $(s+1)n$, we release s jobs of weight a^{n+1} and stop.

Fix a deterministic online algorithm $\mathcal{A}$. We compute the expected profit of $\mathcal{A}$ and the adversary in stage $i \leq n$, conditioned on stage i being reached. More precisely, in stage i we include the contributions of jobs scheduled at times si, $si+1$, ..., $si+s-1$ plus the expected profit of the jobs that remain pending at time $s(i+1)$ in case this is the last stage.

Suppose that $\mathcal{A}$ reaches stage i. Let x be the number of jobs with weight a^i executed by $\mathcal{A}$. Then the profit of $\mathcal{A}$ is $xa^i + (s-x)a^{i+1}$. In addition, there are $x-1$ pending jobs of weight a^{i+1} at the end of the stage, which contribute if the instance ends by this stage, i.e., with probability $1-p$. Since the probability of reaching stage i is p^i, the expected profit for stage i is

$$\begin{aligned} & p^i(xa^i + (s-x)a^{i+1} + (1-p)(x-1)a^{i+1}) \\ &= \quad x + sa - xa + (a-1)(x-1) \;=\; 1 + (s-1)a. \end{aligned}$$

Note that this is independent of x and i.

We now calculate the expected adversary profit in stage i. If we stop after this stage, the contribution of stage i towards adversary's profit is $sa^i + (s-1)a^{i+1}$, otherwise it is $a^i + (s-1)a^{i+1}$, so the expected contribution of stage i is

$$p^i(a^i + (s-1)a^{i+1} + (1-p)(s-1)a^i) \;=\; 1 + (s-1)(a+1-p).$$

Summarizing, for each stage, except the last one, the contributions towards the expected value are constant. The contributions of stage $n+1$ are different, but they are also constant (independent of n). So the overall ratio will be, in the limit for $n \to \infty$, the same as the ratio of the contributions of stages $0, \ldots, n$, which is (after some calculation)

$$\frac{1 + (s-1)(a+1-p)}{1 + (s-1)a} \;=\; 1 + \frac{(s-1)(1-p)}{1 + (s-1)a} \;=\; R_s.$$

□

Theorem 4.6.2 *Let us consider the problem of online scheduling of 2–uniform instances of unit jobs in the randomized model with the standard profit model.*

Then there is no algorithm solving this problem better than ≈ 1.172*–competitive.*

Proof: This theorem is special case of Theorem 4.6.1 for $s = 2$. In this case we compute $R_2 \approx 1.172$. □

In the following theorem we consider the generalization of the problem of online scheduling of s–uniform instances of unit jobs, we consider scheduling of uniform instances of unit jobs. Naturally an instance of unit jobs is uniform if there exists s such that the instance is s–uniform. Obviously we are looking for the worst competitive ratio over all possible s.

Theorem 4.6.3 *Let us consider the problem of online scheduling of uniform instances of unit jobs in the randomized model with the standard profit model.*

Then there is no algorithm solving this problem better than 1.25*–competitive.*

Proof: This theorem is special case of Theorem 4.6.1 where we maximize the competitive ratio over all possible s. Simply we compute

$$\sup_{s \in \mathbb{N}} R_s = 1.25.$$

Obviously for any $c < 1.25$ there exists s such that $R_s > c$. Thus there is no algorithm for general uniform instances with better competitive ratio than 1.25. □

4.6.2 2–Uniform jobs in deterministic model

We completely solve the 2–uniform case: we give an algorithm with competitive ratio ≈ 1.377 and a matching lower bound. This ratio is strictly in–between the previous lower and upper bounds from [4]. Our algorithm is rather technical, which is not really surprising, given the lower bound of $\sqrt{2}$ for memoryless algorithms [8]. To our knowledge, the lower bound of ≈ 1.377 is the best lower bound for the s–uniform case for any s.

We consider 2–uniform instances—in such instances each job j satisfies $d_j = r_j + 2$. Let $Q \approx 1.377$ be the largest root of $Q^3 + Q^2 - 4Q + 1 = 0$. First, we prove that no online algorithm for this problem can be better than Q–competitive. Next, we show that this lower bound is in fact tight.

Lower bound

The proof is by constructing an appropriate adversary strategy. Given an online algorithm $\mathcal{A}$, the adversary releases a sequence of jobs on which the profit of $\mathcal{A}$ is less than Q times the optimal profit.

At each step t, we distinguish *old pending jobs*, that is, those that were released at time $t-1$ but not executed, from the newly released jobs. We can always ignore all the old pending jobs except for the heaviest one, as only one of the old pending jobs can be executed. To simplify notation, we identify jobs by their weight. Thus "job x" means the job with weight x. Such a job is usually uniquely defined by the context, possibly after specifying if it is an old pending job or a newly released job.

For simplicity, we assume first that the additive constant in the definition of competitiveness is 0. We show later how this assumption can be eliminated.

Fix some $0 < \epsilon < 2Q-2$. We define a sequence Ψ_i, $i = 1, 2, \ldots$, as follows. For $i = 1$, $\Psi_1 = Q - 1 - \epsilon$. Inductively, for $i \geq 1$, let

$$\Psi_{i+1} \;=\; \frac{(2-Q)\Psi_i - (Q-1)^2}{2 - Q - \Psi_i}.$$

Lemma 4.6.4 *For all i, we have $|\Psi_i| < Q-1$. Furthermore, the sequence $\{\Psi_i\}$ converges to $1-Q$.*

Proof: Substituting $z_i = \Psi_i + Q - 1$, we get a recurrence $z_{i+1} = \frac{(3-2Q)z_i}{1-z_i}$. Note that $z_1 = 2Q - 2 - \epsilon$ and that $0 < z_i \leq 2Q - 2 - \epsilon$ implies

$$0 \;<\; z_{i+1} \;\leq\; \frac{3-2Q}{3-2Q+\epsilon} z_i \;<\; z_i.$$

Thus, by induction, $0 < z_i \leq 2Q-2-\epsilon$ for all i and, furthermore, $\lim_{i\to\infty} z_i = 0$. The lemma follows immediately. □

Theorem 4.6.5 *Let us consider the problem of online scheduling of 2–uniform instances of unit jobs in the deterministic model with the standard profit model.*

Then there is no algorithm solving this problem better than $Q \approx 1.377$–competitive.

Proof: Let $\mathcal{A}$ be some online algorithm for the 2–uniform case. We develop an adversary strategy that forces $\mathcal{A}$'s ratio to be bigger than $Q - \epsilon$.

Let Ψ_i be as defined before Lemma 4.6.4. For $i \geq 1$ define

$$a_i \;=\; \frac{1-\Psi_i}{Q-1} \quad \text{and} \quad b_i \;=\; \frac{Q(2-Q-\Psi_i)}{(Q-1)^2}.$$

By Lemma 4.6.4, for all i, $b_i > a_i > 1$ and, for large i, $a_i \approx 3.653$ and $b_i \approx 9.688$.

Our strategy proceeds in stages. It guarantees that at the beginning of stage $i = 1, 2, ...$, both $\mathcal{A}$ and the adversary have one or two old pending job(s) of the same weight x_i. Note that it is irrelevant if one or two old pending jobs are present, and also if they are the same for $\mathcal{A}$ and the adversary.

Each stage $i \geq 1$ except last consists of three time steps. The last stage can consist of one, two, or three steps. We will also have an initial stage numbered 0 that consists of one time step.

Initially, in stage 0, we issue two jobs of some arbitrary weight $x_1 > 0$ at time 0. Both $\mathcal{A}$ and the adversary execute one job x_1, and at the beginning of stage 1 both have an old pending job with weight x_1.

At the beginning of stage $i \geq 1$, $\mathcal{A}$ and the adversary start with an old pending job x_i. The adversary now follows this procedure:

issue one job $a_i x_i$
(A) `if` $\mathcal{A}$ executes $a_i x_i$ `then` execute x_i, $a_i x_i$ and halt
`else` ($\mathcal{A}$ executes x_i)
at the next time step issue $b_i x_i$
(B) `if` $\mathcal{A}$ executes $b_i x_i$ `then` execute x_i, $a_i x_i$, $b_i x_i$, and halt
`else` ($\mathcal{A}$ executes $a_i x_i$)
(C) at the next time step issue two jobs $x_{i+1} = b_i x_i$
execute $a_i x_i$, $b_i x_i$, $b_i x_i$

If $\mathcal{A}$ executes first x_i and then $a_i x_i$, then after step (C) it executes one job $b_i x_i$, either the old pending one or one of the two newly released jobs. After this, both $\mathcal{A}$ and the adversary have one or two newly released jobs $b_i x_i$ pending, and the new stage starts with $x_{i+1} = b_i x_i$.

A single complete stage of the adversary strategy is illustrated in Figure 4.1.

If the game reaches stage i, then define profit_i and adv_i to be the total profit of $\mathcal{A}$ and the adversary, respectively, in stages $0, 1, \ldots, i-1$. By ρ we denote the sequence of all jobs released by the adversary.

Claim A: For any i, either the game stops before stage i and the algorithm fails to be $(Q - \epsilon)$–competitive on the input sequence ρ, i.e., $(Q - \epsilon)\mathit{profit}_{\mathcal{A}}(\rho) - adv(\rho) < 0$, or else at the beginning of stage i we have

$$(Q - \epsilon)\mathit{profit}_i - adv_i \quad \leq \quad \Psi_i x_i. \tag{4.3}$$

The proof of Claim A is by induction on the number of stages. For $i = 1$, $(Q - \epsilon)\mathit{profit}_1 - adv_1 \leq (Q - \epsilon - 1)x_1 = \Psi_1 x_1$, and the claim holds.

In the inductive step, suppose that stage i has been reached and is about to start. Thus now $\mathcal{A}$ and the adversary have an old pending job

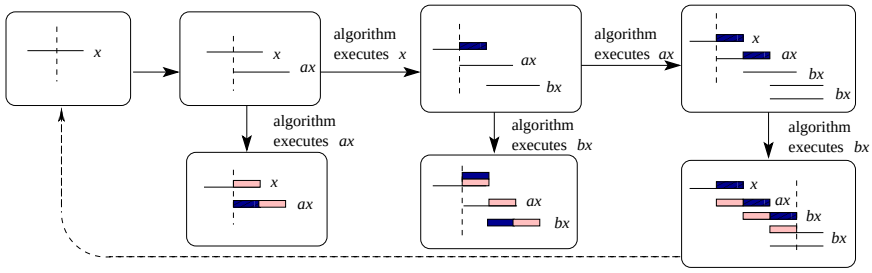


Figure 4.1: The adversary strategy. We denote $x = x_i$, $a = a_i$, and $b = b_i$. Line segments represent jobs, dark rectangles represent slots when the job is executed by $\mathcal{A}$, and lightly shaded rectangles represent executions by the adversary.

with weight x_i and (4.3) holds. If $\mathcal{A}$ executes $a_i x_i$ in step (A), then, denoting by ρ the sequence of all released jobs (up to and including $a_i x_i$), using the inductive assumption, and substituting the formula for a_i, we have

$$\begin{aligned}(Q-\epsilon)\mathit{profit}_{\mathcal{A}}(\rho) - adv(\rho) &= (Q-\epsilon)\mathit{profit}_i - adv_i + (Q-\epsilon)a_i x_i \\ &\quad -(x_i + a_i x_i) \\ &\leq [\Psi_i - 1 + (Q-\epsilon-1)a_i]x_i \;=\; -\epsilon a_i x_i \;<\; 0,\end{aligned}$$

as claimed.

If $\mathcal{A}$ executes x_i and then $b_i x_i$ in (B), then, again, denoting by ρ the sequence of all released jobs, using the inductive assumption, and substituting the formulas for a_i, b_i, we have

$$\begin{aligned}(Q-\epsilon)\mathit{profit}_{\mathcal{A}}(\rho) - adv(\rho) &= (Q-\epsilon)\mathit{profit}_i - adv_i + (Q-\epsilon)(x_i + b_i x_i) \\ &\quad -(x_i + a_i x_i + b_i x_i) \\ &\leq [\Psi_i + Q - \epsilon - 1 + (Q-\epsilon-1)b_i - a_i]x_i \\ &= -\epsilon(1+b_i)x_i \;<\; 0.\end{aligned}$$

In the remaining case (C), $\mathcal{A}$ executes first x_i, then $a_i x_i$, and then $b_i x_i$. Using the formulas for a_i, b_i, Ψ_{i+1}, and the defining equation $Q^3 + Q^2 - 4Q + 1 = 0$, we have

$$\begin{aligned}(Q-\epsilon)\mathit{profit}_{i+1} - adv_{i+1} &\leq (Q-\epsilon)\mathit{profit}_i - adv_i \\ &\quad +(Q-\epsilon)(x_i + a_i x_i + b_i x_i) - (a_i x_i + 2b_i x_i) \\ &\leq [\Psi_i + Q + (Q-1)a_i - (2-Q)b_i]x_i \\ &= b_i \Psi_{i+1} x_i \;=\; x_{i+1}\Psi_{i+1}.\end{aligned}$$

This completes the proof of Claim A.

Lemma 4.6.4 and Claim A imply that, for i large enough, we have $(Q-\epsilon)\mathit{profit}_i - adv_i \leq \Psi_i x_i < (1-Q+\epsilon)x_i$. For this i, if the game has not stopped earlier, the adversary ends it after stage i. Denoting by ϱ the sequence of all jobs (including the pending jobs x_i), we have

$$\begin{aligned}(Q-\epsilon)\mathit{profit}_{\mathcal{A}}(\varrho) - adv(\varrho) &= (Q-\epsilon)\mathit{profit}_i - adv_i + (Q-\epsilon)x_i - x_i \\ &< (1-Q+\epsilon)x_i + (Q-\epsilon)x_i - x_i \;=\; 0.\end{aligned}$$

This completes the proof of the lemma, except that in the argument so far we assumed that the additive constant is 0. This assumption is easy to eliminate: For any given additive constant B, simply choose the initial job $x_1 \gg B$. The remainder of the proof is a simple modification of the presented argument. □

Upper bound

We now present our Q–competitive algorithm for the 2–uniform case. Given that the 2–uniform case seems to be the most elementary case of unit job scheduling (without being trivial), our algorithm (and its analysis) is surprisingly difficult. Recall, however, that, as shown in [8], any algorithm for this case with competitive ratio below $\sqrt{2}$ needs to use some information about the past. Further, when the adversary uses the strategy from Theorem 4.6.5, any Q–competitive algorithm needs to behave in an essentially unique way. Our algorithm was designed to match this optimal strategy, and then extended (by interpolation) to other adversarial strategies. Thus we suspect that the complexity of the algorithm is inherent in the problem and cannot be avoided.

We start with some intuitions. Let $\mathcal{A}$ be our online algorithm. Suppose that at time t we have one old pending job z, and two new pending jobs b, c with $b \geq c$. In some cases, the decision which job to execute is easy. If $c \geq z$, $\mathcal{A}$ can ignore z and execute b in the current step. If $z \geq b$, $\mathcal{A}$ can ignore c and execute z in the current step. If $c < z < b$, $\mathcal{A}$ faces a dilemma: it needs to decide whether to execute z or b. For $c = 0$, the choice is based on the ratio z/b. If z/b exceeds a certain threshold (possibly dependent on the past), we execute z, otherwise we execute b. Taking those constraints into account, and interpolating for arbitrary values of c, we can handle all cases by introducing a parameter η, $0 \leq \eta \leq 1$, and making the decision according to the following procedure:

Procedure CHOOSE$_\eta$**:** If $z \geq \eta b + (1-\eta)c$ schedule z, otherwise schedule b.

To derive an online algorithm, say $\mathcal{A}$, we need to determine what values of η to use at each step. To this end, we examine the adversary strategy in the lower bound proof. Consider the limit case, when $i \to \infty$, and let $a_* = \lim_{i\to\infty} a_i = Q/(Q-1)$ and $b_* = \lim_{i\to\infty} b_i = Q/(Q-1)^2$.

Suppose that in the previous step two jobs z were issued. If the adversary now issues a single job a, then $\mathcal{A}$ needs to do the following: if $z \geq a/a_*$, execute z, and if $z \leq a/a_*$, then execute a. (The tie for $z = a/a_*$ can be broken either way.) Thus in this case we need to apply CHOOSE$_\alpha$ with the threshold $\alpha = 1/a_* = (Q-1)/Q$.

Now, suppose that in the first step $\mathcal{A}$ executed z, so that in the next step a is pending. If the adversary now issues a single job b, then (assuming in the previous step the optimal value of $a \approx a_*$ was used) $\mathcal{A}$ must to do the following: if $a \geq b/b_*$, execute a, and if $a \leq b/b_*$, then execute b. Thus in this case we need to apply CHOOSE$_\beta$ with the threshold $\beta = a_*/b_* = Q-1$.

Suppose that we execute a. In the lower–bound strategy, the adversary would now issue two jobs b in the next step, in which case we can use $\eta = \alpha$. But what happens if he issues a single job, say c? Calculations show that $\mathcal{A}$, in order to be Q–competitive, needs to use yet another parameter η in CHOOSE$_\eta$. This parameter is not uniquely determined, but it must be at least $\gamma = (3-2Q)/(2-Q) > Q-1$. Further, it turns out that the same value γ can be used on subsequent single–job requests.

Our algorithm is derived from the above analysis: on a sequence of single–job requests in a row, use CHOOSE$_\eta$ with parameter α in the first step, then β in the second step, and γ in all subsequent steps. In general, of course, two jobs can be issued at each step (or more, but only the two heaviest jobs need to be considered). We think of an algorithm as a function of several arguments. The values of this function on the boundary are determined from the optimal adversary strategy, as explained above. The remaining values are obtained through interpolation.

We now give a formal description of our algorithm. Let

$$\alpha = \frac{Q-1}{Q} \approx 0.27, \qquad \beta = Q-1 \approx 0.38, \qquad \gamma = \frac{3-2Q}{2-Q} \approx 0.39,$$

$$\lambda(\xi) = \min\left\{1, \frac{\xi-\alpha}{\beta-\alpha}\right\}, \qquad \delta(\mu,\xi) = \mu\alpha + (1-\mu)[\beta + (\gamma-\beta)\lambda(\xi)],$$

where $0 \leq \mu \leq 1$ and $\alpha \leq \xi \leq \gamma$. Note that for the parameters μ, ξ within their ranges, we have $0 \leq \lambda(\xi) \leq 1$, $\alpha \leq \delta(\mu,\xi) \leq \gamma$. Function δ also satisfies $\delta(1,\xi) = \alpha$, $\delta(0,\xi) \geq \beta$ for any ξ, and $\delta(0,\alpha) = \beta$, $\delta(0,\beta) = \delta(0,\gamma) = \gamma$.

Algorithm SWITCH**.** Without loss of generality, we assume that at each step exactly two jobs are released. If more jobs are released, consider only

the two heaviest jobs. If fewer jobs are released, create dummy jobs with weight 0.

Fix a time step t. Let b, c (where $b \geq c$) be the two jobs released at time t, and u, v (where $u \geq v$) be the two jobs released at time $t-1$. (Initially, at $t = 0$, let $u = v = 0$.)

We distinguish two cases. If $u = v$, or if u was scheduled at time $t-1$, then run the job selected by CHOOSE$_\alpha$. (Note that this includes the case $t = 0$.) Otherwise, denoting by ξ the parameter of CHOOSE$_\xi$ executed at time $t-1$, run the job selected by CHOOSE$_\eta$, for $\eta = \delta(v/u, \xi)$.

Theorem 4.6.6 *Let us consider the problem of online scheduling of* 2*–uniform instances of unit jobs in the deterministic model with the standard profit model. Above we described algorithm* SWITCH*.*

The algorithm SWITCH *solves this problem and it is* $Q \approx 1.377$*–competitive, where* $Q \approx 1.377$ *is the largest root of* $Q^3 + Q^2 - 4Q + 1 = 0$*.*

Proof. The analysis of the 2–uniform case is based on the potential function argument. We define below a potential function Φ that maps all possible configurations into real numbers, and we prove a bound on the amortized cost of the algorithm in each step.

We fix an adversary schedule to be a canonical optimal schedule. Thus if the adversary schedules both jobs released at the same time, then the heavier one is scheduled first.

The configuration at time t is specified by the parameter ξ of CHOOSE$_\xi$ used at time $t-1$, the jobs $u \geq v$ released at time $t-1$, and the pending jobs $x, y \in \{u, v\}$ of the algorithm and the adversary, respectively, at time t. (For $t = 0$ we set $u = v = x = y = 0$ and $\xi = \alpha$.) We denote the potential at time t by $\Phi_{xy}(u, v, \xi)$, and define it as follows:

$$\begin{aligned} E &= 2 - Q^2 = \alpha(Q-1) \approx 0.104, \\ G &= 2(Q-1) - 1/Q \approx 0.028, \\ \Phi_{vy}(u,v,\xi) &= y - Qv, \\ \Phi_{uu}(u,v,\xi) &= E \cdot \lambda(\xi)(u+v), \\ \Phi_{uv}(u,v,\xi) &= (Q-1)v - (G \cdot \lambda(\xi) + 1/Q)u. \end{aligned}$$

We now consider a single step at time t. Let b and c be the jobs released at time t, where $b \geq c$. Let η denote the parameter of CHOOSE used at time t, and let $x', y' \in \{b, c\}$ denote the pending jobs of the algorithm and the adversary at $t+1$, respectively. The potential at time $t+1$ is $\Phi_{x'y'}(b, c, \eta)$; we refer to it as the 'new potential' as opposed to the 'old potential' $\Phi_{xy}(u, v, \xi)$

at time t. The rest of this section is devoted to the proof of the following inequality:

$$\Gamma \;=\; \Phi_{xy}(u,v,\xi) + Q \cdot \Delta profit_{\text{SWITCH}} - \Delta adv - \Phi_{x'y'}(b,c,\eta) \;\geq\; 0, \quad (4.4)$$

where $\Delta profit_{\text{SWITCH}}$ and Δadv are the profits of the algorithm and the adversary at time t. It is sufficient to prove inequality (4.4), because the Q–competitiveness of the algorithm follows immediately from (4.4) by summation over all times t and observing that $\Phi_{xy}(0,0,\xi) = 0$, i.e., the potential is zero on configurations with no old pending jobs, which includes the initial and final configurations.

We are now ready to prove (4.4) by a case analysis. During the proof, we need to verify number of relations between the constants we have defined so far. In most cases a rough calculation based on the numerical values given above is sufficient due to some slack; we explicitly mention the cases when the relations are tight.

Case 1: Suppose that v is pending for SWITCH at time t. Thus the old potential is $\Phi_{vy}(u,v,\xi) = y - Qv$, and SWITCH applies CHOOSE$_\alpha$ at time t. This also means that $\lambda(\eta) = \lambda(\alpha) = 0$.

Case 1.1: If $v < \alpha b + (1-\alpha)c$ then SWITCH schedules b and has $x' = c$ pending at time $t+1$.

If the adversary schedules y, it has $y' = b$ pending at time $t+1$, the new potential is $\Phi_{cb}(b,c,\alpha) = b - Qc$, and we get

$$\Gamma \;=\; (y - Qv) + Qb - y - (b - Qc) \;=\; (Q-1)b + Qc - Qv \;\geq\; 0,$$

where the last inequality follows from the case condition, after substituting $\alpha = (Q-1)/Q$.

Otherwise, the adversary schedules b, it has $y' = c$ pending at time $t+1$, the new potential is $\Phi_{cc}(b,c,\alpha) = (1-Q)c$, and using $y \geq v$ we get

$$\Gamma \;\geq\; (v - Qv) + Qb - b - (1-Q)c \;=\; (Q-1)(b+c) - (Q-1)v \;\geq\; 0,$$

where the last inequality follows from the case condition, after substituting $\alpha = (Q-1)/Q$.

Case 1.2: If $v \geq \alpha b + (1-\alpha)c$ then CHOOSE$_\alpha$ schedules v and has $x' = b$ pending.

If the adversary schedules y, it has $y' = b$ pending at time $t+1$, the new potential is $\Phi_{bb}(b,c,\alpha) = 0$, and we get

$$\Gamma \;=\; (y - Qv) + Qv - y \;=\; 0.$$

Otherwise, the adversary schedules b, it has $y' = c$ pending at time $t+1$, the new potential is $\Phi_{bc}(b, c, \alpha) = (Q-1)c - b/Q$, and using $y \geq v$ we get

$$\Gamma \;\geq\; (v - Qv) + Qv - b - ((Q-1)c - b/Q) \;=\; v - \alpha b - (Q-1)c \;\geq\; 0,$$

where the last inequality follows from $Q - 1 < 1 - \alpha$ and the case condition.

Case 2: In this case $u > 0$ is the pending job for SWITCH at time t, and at time t SWITCH applies CHOOSE$_\eta$, where $\eta = \delta(v/u, \xi)$. The old potential is $\Phi_{uy}(u, v, \xi)$. To reduce the number of subcases, we first note that

$$\Phi_{uu}(u, v, \xi) \;\geq\; \Phi_{uv}(u, v, \xi). \tag{4.5}$$

Indeed, after substituting, this is equivalent to $((E+G)\lambda(\xi) + 1/Q)u \geq (Q - 1 - E \cdot \lambda(\xi))v$. Since $\lambda(\xi) \geq 0$ and $u \geq v$, it is sufficient to verify that $1/Q > Q - 1$, which is true.

Since Γ is a linear function of job weights and, when the job weights are rescaled, η as well as other coefficients do not change, we may assume that $u = 1$. It is also convenient to define $l = \lambda(\xi)$. Recapitulating the definitions, we have

$$\begin{aligned} l &= \lambda(\xi) \;=\; \min\{1, (\xi - \alpha)/(\beta - \alpha)\}, \\ \eta &= \eta(v, l) \;=\; v\alpha + (1 - v)(\beta + (\gamma - \beta)l) \;=\; \delta(v, \xi). \end{aligned}$$

The function $\lambda(\xi)$ increases from 0 to 1 as ξ increases from α to β, and is equal to 1 for $\xi \geq \beta$. Thus $0 \leq l \leq 1$. The function $\eta(v, l)$ is non–increasing in v and non–decreasing in l in the whole domain of v and l. Furthermore, $\alpha \leq \eta(v, l) \leq \gamma$, $\eta(1, l) = \alpha$, and $\eta(0, l) \geq \beta$.

In each case of the analysis we need to minimize a linear function of b and c subject to $0 \leq c \leq b$ and $\eta b + (1 - \eta)c = 1$. Since the feasible domain is a line segment, the minimum must be attained at one of the endpoints which are $b = c = 1$ and $b = 1/\eta$, $c = 0$. Thus the minimum can be found by comparing these two values of the investigated linear function.

Case 2.1: $1 < \eta b + (1 - \eta)c$ and thus SWITCH schedules b.

Case 2.1.1: The adversary schedules $y = u = 1$. Using $v \geq 0$, we bound the old potential as $\Phi_{uy}(u, v, \xi) = \Phi_{uu}(u, v, \xi) \geq E \cdot l$. The new potential is $\Phi_{cb}(b, c, \eta) = b - Qc$ and we get

$$\begin{aligned} \Gamma &\geq E \cdot l + Qb - 1 - (b - Qc) \\ &= E \cdot l - 1 + (Q-1)b + Qc \qquad (4.6) \\ &\geq E \cdot l - 1 + (Q-1)/\eta \qquad (4.7) \end{aligned}$$

The last inequality is justified as follows: expression (4.6) is a linear function of b, c, increasing in both b and c. From the case condition and $b \geq c \geq 0$ it follows that (4.6) decreases when b and/or c are decreased until $\eta b + (1-\eta)c = 1$, and under this constraint, $(Q-1)b + Qc$ is minimized for $b = 1/\eta$, $c = 0$. At the other endpoint of the feasible region, i.e., at $b = c = 1$, the value is larger, since $2Q - 1 > (Q-1)/\alpha \geq (Q-1)/\eta$.

Now (4.7) is minimized for $v = 0$, because $\eta = \delta(v, \xi)$ is decreasing in v. We substitute $v = 0$ in (4.7), multiply by $\eta(0, l)$, and substitute the definition of $\eta(0, l)$ to obtain

$$\begin{aligned} \eta(0,l) \cdot \Gamma &\geq E \cdot l(\beta + (\gamma - \beta)l) - \beta - (\gamma - \beta)l + Q - 1 \\ &\geq E \cdot l\beta - \beta - (\gamma - \beta)l + Q - 1 \\ &\geq 0. \end{aligned}$$

The final inequality holds since $\beta = Q-1$ and $E \cdot \beta > \gamma - \beta$. This holds, since $E \cdot \beta > 0.037$ and $\gamma - \beta < 0.02$, including the rounding errors. Alternatively, substituting the definitions of E, β, and γ, the inequality reduces to a degree–4 polynomial inequality in Q which, using the definition of Q, can be reduced to degree–2 inequality $7Q^2 - 6Q - 5 > 0$, that can be verified using again the definition of Q.

Case 2.1.2: The adversary schedules $y = v$. Using $l \leq 1$ and the definition of G, the old potential is bounded by $\Phi_{uy}(u, v, \xi) = \Phi_{uv}(u, v, \xi) = (Q-1)v - G \cdot l - 1/Q \geq (Q-1)v - 2(Q-1)$. The new potential is $\Phi_{cb}(b, c, \eta) = b - Qc$. We bound the linear function of b and c exactly as in the previous case to obtain

$$\begin{aligned} \Gamma &= (Q-1)v - 2(Q-1) + Qb - v - (b - Qc) \\ &= -(2-Q)v - 2(Q-1) + (Q-1)b + Qc \\ &\geq -(2-Q)v - 2(Q-1) + (Q-1)/\eta. \end{aligned}$$

We now notice that $\eta = \eta(v, l)$ is increasing with l, thus it is sufficient to substitute the value of η for $l = 1$. In this case we get after multiplying by $\eta(v, 1)$ and substituting its value

$$\eta(v,1) \cdot \Gamma \geq -(\alpha v + (1-v)\gamma)((2-Q)v + 2(Q-1)) + (Q-1).$$

For $v = 1$, the right–hand side is equal to 0. To conclude that $\Gamma \geq 0$, it is sufficient to show that the right–hand side decreases for $v \in [0, 1]$. It is a convex quadratic function (as $\gamma > \alpha$), thus it is sufficient to verify that its derivative at $v = 1$ is at most 0. The derivative is $(\gamma - \alpha)((2-Q)v + 2(Q-1)) - (\alpha v + (1-v)\gamma)(2-Q)$, which at $v = 1$ equals $\gamma Q - 2\alpha < 0$.

Case 2.1.3: The adversary schedules b. Using (4.5), $v \geq 0$, $l \leq 1$, and the definition of G the old potential is bounded by $\Phi_{uy}(u, v, \xi) \geq \Phi_{uv}(u, v, \xi) \geq -G - 1/Q = -2(Q-1)$. The new potential is $\Phi_{cc}(b, c, \eta) = (1-Q)c$. We have

$$\begin{aligned} \Gamma &\geq -2(Q-1) + Qb - b - (1-Q)c \\ &= (Q-1)(b+c-2) \\ &\geq 0. \end{aligned}$$

To justify the last inequality, note that, given the case constraint, $b + c$ is minimized at $b = c = 1$, as the value at $b = 1/\eta$, $c = 0$ is larger, since $2 < 1/\gamma \leq 1/\eta$.

Case 2.2: Suppose that $1 \geq \eta b + (1-\eta)c$. Then SWITCH executes $u = 1$.

Case 2.2.1: The adversary schedules $y = u = 1$. The old potential is bounded by $\Phi_{uy}(u, v, \xi) = \Phi_{uu}(u, v, \xi) \geq 0$ and the new potential is $\Phi_{bb}(b, c, \eta) = E \cdot \lambda(\eta)(b + c)$. We get

$$\begin{aligned} \Gamma &\geq Q - 1 - E \cdot \lambda(\eta)(b+c) \\ &\geq Q - 1 - E \cdot \lambda(\eta)/\eta. \end{aligned}$$

The last inequality follows since $b+c$ is maximized when $\eta b + (1-\eta)c = 1$, using the case condition. Under this restriction, it is maximized when $b = 1/\eta$, $c = 0$, as the value for $b = c = 1$ is smaller because $1/\eta \geq 1/\gamma > 2$.

Using the definition of λ, the value of $\lambda(\eta)/\eta$ is maximized for $\eta = \beta$, where $\lambda(\eta)/\eta = 1/\beta$. Thus $\Gamma \geq Q - 1 - E/\beta > 0$.

Case 2.2.2: The adversary schedules $y = v$. Using $l \leq 1$ and the definition of G, the old potential is bounded by $\Phi_{uy}(u, v, \xi) = \Phi_{uv}(u, v, \xi) = (Q-1)v - G \cdot l - 1/Q \geq (Q-1)v - 2(Q-1)$. The new potential is $\Phi_{bb}(b, c, \eta) = E \cdot \lambda(\eta)(b+c)$. We bound the linear function of b and c exactly as in the previous case to obtain

$$\begin{aligned} \Gamma &\geq (Q-1)v - 2(Q-1) + Q - v - E\lambda(\eta)(b+c) \\ &\geq (2-Q)(1-v) - \frac{E \cdot \lambda(\eta)}{\eta} \\ &\geq (2-Q)(1-v) - \frac{E(\eta - \alpha)}{\alpha(\beta - \alpha)}, \end{aligned} \tag{4.8}$$

where the last inequality follows from $\eta \geq \alpha$ and the definition of λ. The right–hand side of (4.8) is linear in v, since $\eta = \eta(v, l)$ is a linear function of v. Thus it is sufficient to verify that (4.8) is non–negative for $v \in \{0, 1\}$. For

$v = 1$, it is equal to 0. For $v = 0$, we use $\eta \leq \gamma$ and the whole expression is at least

$$2 - Q - \frac{E(\gamma - \alpha)}{\alpha(\beta - \alpha)} > 0.$$

To verify the last inequality numerically, note that $E/\alpha = Q - 1 < 0.4$, so it is sufficient to check that $(\gamma - \alpha)/(\beta - \alpha) < 1.5$. Alternatively, the inequality can be verified by substituting the definitions of the parameters in terms of Q and reducing it to $5Q^2 - 12Q + 7 < 0$, which holds by the definition of Q.

Case 2.2.3: The adversary schedules b. Using inequality (4.5), the old potential is bounded by $\Phi_{uy}(u, v, \xi) \geq \Phi_{uv}(u, v, \xi) = (Q - 1)v - G \cdot l - 1/Q$. The new potential is $\Phi_{bc}(b, c, \eta) = (Q - 1)c - (G \cdot \lambda(\eta) + 1/Q)b$. We have

$$\Gamma \;\geq\; (Q - 1)v - G \cdot l - 1/Q + Q - (1 - G \cdot \lambda(\eta) - 1/Q)b - (Q - 1)c.$$

The expression on the right–hand side is a linear function of b and c. We claim that it is minimized at $b = 1/\eta$, $c = 0$. Indeed, subtracting its value at $b = 1/\eta$, $c = 0$, from the value at $b = c = 1$, we get

$$(1-G\cdot\lambda(\eta)-1/Q)(1/\eta-1)-(Q-1) \;\geq\; (1-G-1/Q)(1/\gamma-1)-(Q-1) \;=\; 0,$$

where we use $\lambda(\eta) \leq 1$ and $\eta \leq \gamma$ in the inequality, and the last equality follows from the definitions of G and γ. Thus, after substituting $b = 1/\eta$ and $c = 0$, and multiplying by η,

$$\eta \cdot \Gamma \;\geq\; \eta \cdot ((Q - 1)v - G \cdot l - 1/Q + Q) - 1 + G \cdot \lambda(\eta) + 1/Q. \quad (4.9)$$

We want to show that the right–hand side of (4.9) is non–negative.

If $\eta \geq \beta$, then $\lambda(\eta) = 1$, and by the definitions of G and η, the right–hand side of (4.9) is equal to

$$[v\alpha + (1 - v)(\beta + (\gamma - \beta)l)]((Q - 1)v - G \cdot l - 1/Q + Q) - (3 - 2Q).$$

This is a concave quadratic function in v, so it is sufficient to verify that it is non–negative for $v \in \{0, 1\}$. For $v = 1$, the function is decreasing with l, so it is minimized at $l = 1$ and the value is $\alpha - (3 - 2Q) > 0$. For $v = 0$, the function is a concave quadratic function in l, so it is sufficient to verify that it is non–negative for $l \in \{0, 1\}$. For $l = 0$ the value is $\beta(-1/Q + Q) - (3 - 2Q)$, and for $l = 1$ it is $\gamma(2 - Q) - (3 - 2Q)$. Both values are equal to 0, by the definitions of β, γ, and Q.

It remains to verify that (4.9) is non–negative when $\eta \leq \beta$. In this case, the right–hand side of (4.9) is equal to

$$\eta \cdot ((Q - 1)v - G \cdot l - 1/Q + Q) - 1 + 1/Q + G \cdot (\eta - \alpha)/(\beta - \alpha). \quad (4.10)$$

For each l, $\eta(v,l)$ is continuous and decreasing in v from $\eta(0,l) \geq \beta$ to $\eta(1,l) = \alpha < \beta$, so there is unique $v = v_l$ for which $\eta(v,l) = \beta$. The expression (4.10) is again a concave quadratic function in v, and we know that it is non–negative at $v = v_l$ from the analysis of the previous case. As in this sub–case we have $v_l \leq v \leq 1$, it remains to verify that (4.10) is non–negative for $v = 1$. In this case its value is $\alpha(2Q-1-G\cdot l-1/Q)-1+1/Q \geq 0$, using $l \leq 1$ and the definition of G.

We have now examined all cases, completing the proof of inequality (4.4), and thus also the proof of Theorem 4.6.6. □

4.7 Bounded jobs scheduling

We consider the following online scheduling problem of unit jobs. Each job j is specified by (r_j, d_j, w_j), release time r_j is integer, deadline d_j is integer, weight w_j is non–negative real, unit processing time $p_j = 1$. Deadline $d_j \leq r_j + s$ for the s–bounded jobs.

In this section we present results related to the problem of scheduling of s–bounded jobs in deterministic and randomized model. These results are product of our joint research with other co–authors.

4.7.1 2–Bounded instances in randomized model

In this section we give a randomized algorithm for 2–bounded instances with competitive ratio 1.25. This matches the lower bound from [20], and thus completely resolves the 2–bounded case. In addition, our algorithm is memoryless.

For $a, b \geq 0$, define

$$p_{ab} = \begin{cases} 1 & \text{if } a \geq b \\ \frac{4a}{5b} & \text{otherwise} \end{cases}$$

Also, let $q_{ab} = 1 - p_{ab}$. Note that p_{ab} satisfies the following properties for any $a, b \geq 0$:

$$5p_{ab}a \geq 4a - b \tag{4.11}$$
$$5(p_{ab}a + q_{ab}b) \geq 4b \tag{4.12}$$
$$5p_{ab}a + 2q_{ab}b \geq 4a \tag{4.13}$$
$$5p_{ab}a + 2q_{ab}b \geq b \tag{4.14}$$

Algorithm R2B**.** Let a and b denote the heaviest jobs of span 1 and span 2, respectively, released at this time step, and c the heaviest pending job (of span 2) issued in the previous step. Let $u = \max(c, a)$. Execute u with probability p_{ub} and b with probability q_{ub}.

Theorem 4.7.1 *Let us consider the problem of online scheduling of* 2*–bounded instances of unit jobs in the randomized model with the standard profit model. Above we described algorithm* R2B*.*

The algorithm R2B *solves this problem and it is* 1.25*–competitive.*

Proof: Without loss of generality, we can assume that at each step exactly one job of span 1 is released. All jobs of span 1 except the heaviest one can be simply ignored, and if no job is released, we can introduce a job of weight 0. Similarly, we can assume that at each step (except last) exactly one job of span 2 is released. This can be justified as follows: If, at a given time t, the optimal schedule contains a job of span 2 released at t, we can assume that it is the heaviest such job. A similar statement holds for Algorithm R2B, since its decision at each step depends only on the heaviest job of span 2. Thus all the other jobs of span 2 can be ignored in this step, and treated as if they are released with span 1 in the following time step.

At a given step, the state of R2B is given by a pair $\langle x, \sigma \rangle$, where x is the job of span 2 released in the previous step, and σ is the probability that x was executed in the previous step. Denote by $\bar{\sigma} = 1 - \sigma$ the probability that x is pending in the current step. In other words, the value of c in the algorithm is 0 with probability σ and x with probability $\bar{\sigma}$. To describe the state of the adversary, let $z \in \{0, x\}$ be a variable such that $z = x$ if the adversary has not scheduled x (i.e., x is pending in the adversary schedule) and $z = 0$ if the adversary has no pending job.

We define the potential function for each configuration described by a triple $\langle x, \sigma, z \rangle$. Note that this slightly deviates from the use of the potential method in the randomized case, as discussed after Lemma 4.4.1. In our case, the potential is a function of the distribution of R2B's current state, and is not a random variable. Nevertheless, Lemma 4.4.1 still applies.

Let $\Phi_{x\sigma z}$ denote the potential function in the configuration $\langle x, \sigma, z \rangle$. We put $\Phi_{x\sigma z} = 0$ if $z = 0$ and $\Phi_{x\sigma z} = \frac{1}{4} x \cdot \max(5\sigma - 1, 3\sigma)$ if $z = x$.

Consider one step, where the configuration is $\langle x, \sigma, z \rangle$, for $z \in \{0, x\}$, and two jobs a, b are released, of span 1 and span 2, respectively. The new configuration is $\langle b, \sigma', z' \rangle$, where $\sigma' = \sigma q_{ab} + \bar{\sigma} q_{vb}$, for $v = \max(a, x)$, and $z' \in \{0, b\}$. Using Lemma 4.4.1, we need to show that for each adversary move:

$$R \cdot \mathbf{Exp}[\Delta \mathit{profit}_{\text{R2B}}] - \Phi_{b\sigma' z'} + \Phi_{x\sigma z} \;\geq\; \Delta \mathit{adv} \tag{4.15}$$

where $\Delta profit_{\text{R2B}}$ is the weight of the randomly chosen job scheduled by R2B and Δadv the weight of the job scheduled by the adversary.

Case 1: Adversary schedules b. Then $\Delta adv = b$ and $z' = 0$. For a fixed value of u in the algorithm, the expected profit of the algorithm is $p_{ub}u + q_{ub}b$ and (4.12) implies $\frac{5}{4}(p_{ub}u + q_{ub}b) \geq b$. By averaging over $u \in \{a, v\}$ we get $R \cdot \mathbf{Exp}[\Delta profit_{\text{R2B}}] \geq b$. This, together with $\Phi_{x\sigma z} \geq 0$ and $\Phi_{b\sigma' z'} = 0$, implies (4.15).

Case 2: The adversary does not schedule b. Then $z' = b$ and $\Phi_{b\sigma' z'} = \frac{1}{4}b \cdot \max(5\sigma' - 1, 3\sigma')$. The algorithm executes b with probability $\sigma q_{ab} + \bar{\sigma} q_{vb} = \sigma'$, a with probability σp_{ab}, and v with probability $\bar{\sigma} p_{vb}$, so $\mathbf{Exp}[\Delta profit_{\text{R2B}}] = \sigma' b + \sigma p_{ab} a + \bar{\sigma} p_{vb} v$. Substituting into (4.15), it is sufficient to prove that

$$\min(b, 2\sigma' b) + 5\sigma p_{ab} a + 5\bar{\sigma} p_{vb} v + 4 \cdot \Phi_{x\sigma z} \geq 4 \cdot \Delta adv \tag{4.16}$$

Case 2.1: The adversary schedules a. Then $\Delta adv = a \leq v$. Since $\Phi_{x\sigma z} \geq 0$, it is sufficient to show (4.16) with $\Phi_{x\sigma z}$ replaced by 0. For the first term of the minimum, we use (4.11) twice and get

$$\begin{aligned} b + 5\sigma p_{ab} a + 5\bar{\sigma} p_{vb} v &= \sigma(b + 5p_{ab} a) + \bar{\sigma}(b + 5p_{vb} v) \\ &\geq 4\sigma a + 4\bar{\sigma} v \;\geq\; 4a. \end{aligned}$$

For the second term of the minimum, we use (4.13) twice and get

$$\begin{aligned} 2\sigma' b + 5\sigma p_{ab} a + 5\bar{\sigma} p_{vb} v &= \sigma(5p_{ab} a + 2q_{ab} b) + \bar{\sigma}(5p_{vb} v + 2q_{vb} b) \\ &\geq 4\sigma a + 4\bar{\sigma} v \;\geq\; 4a. \end{aligned}$$

Case 2.2: $z = x$ and the adversary schedules z. It must be the case that $v = x \geq a$, as otherwise the adversary would prefer to schedule a. We have $\Delta adv = x$.

If $x \geq b$, then $p_{xb} = 1$. We use $4\Phi_{x\sigma z} = 4\Phi_{x\sigma x} \geq (5\sigma - 1)x$ and obtain

$$5\bar{\sigma} p_{xb} x + 4\Phi_{x\sigma z} \;\geq\; 5\bar{\sigma} x + 5\sigma x - x = 4x,$$

which implies (4.16).

It remains to consider the case $x < b$. Using (4.11), (4.14) and (4.13) we obtain

$$b + 5\bar{\sigma} p_{xb} x \;\geq\; b + \bar{\sigma}(4x - b) \;=\; \sigma b + 4\bar{\sigma} x$$

and

$$\begin{aligned} 2\sigma' b + 5\sigma p_{ab} a + 5\bar{\sigma} p_{xb} x &= \sigma(5p_{ab} a + 2q_{ab} b) + \bar{\sigma}(5p_{xb} x + 2q_{xb} b) \\ &\geq \sigma b + 4\bar{\sigma} x \end{aligned}$$

Together with $4\Phi_{x\sigma z} = 4\Phi_{x\sigma x} \geq 3\sigma x$ and $x < b$ this implies

$$\min(b, 2\sigma' b) + 5\sigma p_{ab} a + 5\bar{\sigma} p_{xb} x + 4\Psi_{x\sigma z} \;\geq\; \sigma b + 4\bar{\sigma} x + 3\sigma x \;\geq\; 4x,$$

and (4.16) follows. □

4.7.2 s–Bounded instances in deterministic model

The 2–bounded (deterministic) case is now well understood: there exists an online algorithm with competitive ratio ϕ, and no better ratio is possible [4, 20, 5]. In this section, we extend the upper bound of ϕ to 3–bounded instances by proving that Algorithm EDF$_{\phi-1}$ is ϕ–competitive.

Algorithm EDF$_\alpha$**:** Let h be the heaviest pending job and f be the earliest–deadline pending job such that $w_f \geq \alpha w_h$. Execute f.

Theorem 4.7.2 *Let us consider the problem of online scheduling of 3–bounded instances of unit jobs in the deterministic model with the standard profit model. Above we described algorithm* EDF$_\alpha$.

The algorithm EDF$_{\phi-1}$ *solves this problem and it is ϕ–competitive.*

Proof: We fix a canonical earliest–deadline adversary schedule A. Let E be the schedule computed by EDF$_{\phi-1}$. We use the following charging scheme: Suppose that j is the job scheduled by the adversary at time t. If j is executed in E before time t, charge j to its copy in E. Otherwise, charge j to the job in E scheduled at time t.

Fix some time step t. Let f and j be the jobs scheduled at time t in E and A, respectively. By the definition of EDF$_{\phi-1}$, let h be the heaviest pending job in E at time t, and let f be the earliest–deadline job that is pending at time t and satisfies $w_f \geq (\phi-1)w_h = w_h/\phi$.

Job f receives at most two charges: one from j and one from itself, if f is executed in A at some later time. Ideally, we would like to prove that the sum of the charges is at most ϕw_f. It turns out that in some cases this is not true, and, if so, we then show that for the job g scheduled by E in the next step, the total of all charges to f and g is at most $\phi(w_f + w_g)$. Summing over all such groups of one or two jobs, the ϕ–competitiveness of EDF$_{\phi-1}$ follows.

If f receives only one charge, it is at most ϕw_f: If this charge is from f, it is trivially at most w_f. If the charge is from j (not scheduled before t in E), then j is pending at t in E and thus $w_j \leq w_h \leq \phi w_f$, by the definition of EDF$_{\phi-1}$. In this case the group consist of a single job and we are done.

It remains to handle the case when f receives both charges. In this case, obviously, $j \neq f$ and j is pending in E at time t. Since in the canonical earliest–deadline schedule A job j is strictly before f, yet f is chosen by EDF$_{\phi-1}$, it follows that $w_j < (\phi-1)w_h$.

If $w_f = w_h$, then f is charged at most $w_f + w_j \leq (1+\phi-1)w_h = \phi w_f$, and we have a group with a single job again.

Otherwise, $w_f < w_h$ and by the rule of EDF$_{\phi-1}$, it follows that $d_h > d_f$. Furthermore, since the adversary does not schedule f at time t, we have $d_f \geq t+2$. The span is bounded by 3, and thus the only possible case is that $d_h = t+3$ and $d_f = t+2$. Thus the adversary schedules f at time $t+1$. The weight of the job g scheduled at time $t+1$ in E is $w_g \geq (\phi-1)w_h$, as $h \neq f$ is still pending in E. Furthermore, g gets only the charge from itself, as the adversary at time $t+1$ schedules f which is charged to itself. The total weight of the jobs charged to f and g is at most $w_j + w_f + w_g \leq (\phi-1)w_h + w_f + w_g \leq \frac{3}{2}(w_f + w_g)$, since both w_f and w_g are at least $(\phi-1)w_h$. In this last case we have a group of two jobs. □

A more careful analysis yields an upper bound of $2 - \Theta(1/s)$ on the competitive ratio of EDF$_\alpha$ on s–bounded instances, for an appropriately chosen α. More precisely, for each $s \geq 4$, let λ_s be the unique non–negative solution of the equation

$$(2-\lambda_s)(\lambda_s^2 + \left\lfloor \frac{s}{3} \right\rfloor \lambda_s + s - 2 - 2\left\lfloor \frac{s}{3} \right\rfloor) \;=\; \lambda_s^2 - \lambda_s.$$

Theorem 4.7.3 *Let us consider a fixed integer $s \geq 4$ and the problem of online scheduling of s–bounded instances of unit jobs in the deterministic model with the standard profit model. Above we described algorithm* EDF$_\alpha$. *Constant λ_s is defined above.*

Then the algorithm EDF$_{1/\lambda_s}$ *solves this problem and it is λ_s–competitive.*

Proof: Throughout the proof, we write λ instead of λ_s. For any time t, let M_t be the maximal weight of a job available to EDF$_{1/\lambda}$ at time t; define $M_t = 0$ if no job is available.

First we show that we can restrict ourselves to instances where all the jobs have weights λ^i for some integer i. For these instances, however, we assume that the algorithm has no control over how the ties are resolved, and given two jobs of equal weight, the adversary can dictate which one should be considered heavier. Still, the ties are resolved in a consistent manner for both algorithms. More precisely, a *valid run* for such instances is defined so that at each time step, we schedule the earliest deadline job (applying our usual tie–breaking convevntion) from the set of pending jobs that contains all jobs with weight strictly bigger than M_t/λ and an arbitrary subset of jobs (chosen by the adversary) with weight equal to M_t/λ.

Claim A: Without loss of generality, it is sufficient to prove the theorem for valid runs on instances where all the jobs have weights of the form λ^i, for some integer i.

Call a job *bad* if its weight is not equal to λ^i for an integral i. Now we show that any instance with some bad jobs can be converted to an instance with a smaller number of bad jobs and with the same or larger competitive ratio on some valid run. Express each weight of a bad job j as $w_j = a_j\lambda^{e_j}$ for integral e_j and $1 < a_j < \lambda$. Let $b_{\min} = \min_j(a_j - 1)$ and $b_{\max} = \min_j(\lambda - a_j)$ (the minima are taken over all bad jobs j). Now replace the weight of each bad job j by $(a_j + b)\lambda^{e_j}$, for some $b \in [-b_{\min}, b_{\max}]$. From the definition of $b_{\min}$ and $b_{\max}$, it follows that the order of the weights of jobs does not change, as well as the result of comparisons of one weight to another weight divided by λ, except possibly for creating new ties. Consequently, any valid run on the original instance is also a valid run on the new instance, and the set of jobs scheduled in the original optimal solution gives also an optimal solution of the new instance. As the total weights of jobs scheduled both in the valid run and the optimal schedule are linear in b, their ratio is monotone in $[-b_{\min}, b_{\max}]$ and thus it is maximized either for $b = -b_{\min}$ or $b = b_{\max}$. Choose the appropriate b of these two possibilities and the corresponding modified instance. By the definition of $b_{\min}$ and $b_{\max}$, the number of bad jobs has decreased. After repeating this process a sufficient number of times we will convert the initial instance into one without bad jobs, and the ratio between the weight of the optimal schedule and the weight of $\text{EDF}_{1/\lambda}$'s schedule will not decrease. This completes the proof of Claim A.

Fix a valid run of $\text{EDF}_{1/\lambda}$ on an instance with no bad jobs and denote the resulting schedule by E. At any time t, either a job of weight M_t or M_t/λ is scheduled. If a job of weight M_t/λ is scheduled, the job with weight M_t remains pending at time $t + 1$ (if its deadline were $t + 1$, the valid run would schedule such a job at time t); thus in this case we have $M_{t+1} \geq M_t$.

Fix an earliest–deadline adversary schedule A. We define the charging scheme as follows. For any integer time t, let j be the job scheduled at t in A and f be the job scheduled in E. If j is completed in E before time t and $w_j \geq M_t$, charge j to j in E. Otherwise, charge j to f in E.

By the charging scheme, each job f in E receives at most two charges. Denoting by t the time when f is scheduled in E, f can receive a charge from the job scheduled in A at time t, and also from itself, if f is scheduled in A at or after time t.

It remains to prove that the charging scheme works correctly. The idea is similar to the proof of Theorem 4.7.2. We partition E into segments such that in each segment the total of all charges to the jobs in the segment is at most λ times their total weight. Summing over all such segments, this will imply λ–competitiveness of $\text{EDF}_{1/\lambda}$. Therefore, to complete the upper bound proof, it is now sufficient to prove the following claim.

Claim B: Schedule E can be partitioned into disjoint contiguous segments of jobs such that in each segment the total of all charges to the jobs in the segment is at most λ times their total weight.

We now prove Claim B. Let f be a job scheduled in E at time t. We start by some general observations.

(I) If f receives only one charge, then this charge is at most λ times its weight. If this charge is from f in A, it is trivially at most w_f. Otherwise, this charge is from a job j scheduled at time t in A. If j is scheduled before t in E, the charge is at most $M_t \leq \lambda w_f$ by the definition of the charging scheme. If j is not scheduled before t in E, then j is pending at t in E and thus $w_j \leq M_t \leq \lambda w_f$, by the definition of $\text{EDF}_{1/\lambda}$.

(II) If f receives both charges, the charge from the job j scheduled in A at time t is at most M_t/λ. It could be more only if j is not scheduled before t in E and $w_j > M_t/\lambda$. In that case, however, j is pending for $\text{EDF}_{1/\lambda}$ and has sufficiently large weight. In A, both j and f are pending at t and the adversary selects j. Since the ties are broken consistently, EDF must also prefer j and cannot schedule f.

We split E into segments starting from the beginning. Suppose that the currently processed time is t and E schedules a job f at time t. If f receives a single charge or $w_f = M_t$, we create a segment with a single job f. By the observations above, this segment is charged at most λ times its weight: if $w_f = M_t$, then f is charged at most $(1 + 1/\lambda)w_f \leq \lambda w_f$, by (II) and the inequality $\lambda \geq \phi$.

It remains to handle the case when f receives two charges and $w_f = M_t/\lambda$. For $i \geq t$, let f_i be the job scheduled in E at time i, and let $m \geq t$ be the smallest index such that $w_{f_m} = M_m$. (Such m exists, as eventually a maximal job is scheduled.) Thus $w_{f_i} = M_i/\lambda$ for $t \leq i < m$ and $M_t \leq M_{t+1} \leq \cdots \leq M_m$. Let Z be the set of jobs $f_t, \ldots, f_{m-1}$. We create a segment of jobs $f_t, \ldots, f_m$ and prove that its charging ratio is at most λ.

Let $k \geq 0$ be such that $M_m = \lambda^{k+1}$. For $i = 1, \ldots, k$, let X_i be the set of all jobs in Z with weight M_m/λ^i that receive two charges and let $x_i = |X_i|$. Also, let $X = \bigcup X_i$ and $x = |X| = \sum_{i=1}^{k} x_i$.

By the definition of Z, $\text{EDF}_{1/\lambda}$ schedules first all jobs in X_k, then X_{k-1}, and so on, up to X_1 (with possibly some jobs in $Z - X$ scheduled in–between the jobs from X.) Since every f_r in X receives also its own charge, it is scheduled in A after time r. Furthermore, it cannot be scheduled at time r' such that $r < r' \leq m$, for otherwise we would have $w_{f_r} < M_r \leq M_{r'}$,

and by the definition of the charging scheme f_r is not charged to itself in such a case. Thus all jobs in X are scheduled at time $m+1$ or later in A.

We claim that for each i,

$$x_i + x + 2 \;\leq\; s. \tag{4.17}$$

For a fixed i, let $i' \geq i$ be such that the last job finished by A from $X_i \cup \cdots \cup X_k$ belongs to $X_{i'}$. Let j be a job of maximal weight available when $\text{EDF}_{1/\lambda}$ schedules the first job of $X_{i'}$. Since j is scheduled in E only after all jobs in $X_{i'}$ despite the fact that its weight is larger, it must have strictly larger deadline than all the jobs in $X_{i'}$. Between the start of the first job in $X_{i'}$ in E and time $m+1$, $\text{EDF}_{1/\lambda}$ schedules all jobs in $X_{i'} \cup X_{i'-1} \cup \cdots \cup X_1$ and f_m. Between time $m+1$ and the time A finishes the last job of $X_{i'}$, A schedules all jobs in $X_i \cup \cdots \cup X_k$. Job j is available at all times from the start of the first job in $X_{i'}$ in E, until at least one time step after A finishes the last job of $X_{i'}$. Therefore we have

$$s \;\geq\; x_{i'} + x_{i'-1} + \cdots + x_1 + 1 + x_i + \cdots + x_k \;\geq\; x_i + x + 2.$$

Using (I), each job in $Z - X$ is charged at most λ times its weight. Let

$$W \;=\; \sum_{i=1}^{k} \frac{x_i}{\lambda^i}, \tag{4.18}$$

i.e., $WM_m = \lambda^{k+1} W$ is the total weight of jobs in X. Using (II), the jobs in X and f_m are charged a total of at most $2WM_m + (1 + 1/\lambda)M_m$ and their weight is $WM_m + M_m$. To finish the proof of λ–competitiveness, it is sufficient to show that

$$\lambda \;\geq\; \frac{2W + 1 + \frac{1}{\lambda}}{W+1} \;=\; 2 - \frac{1 - \frac{1}{\lambda}}{W+1}. \tag{4.19}$$

The right–hand side increases with W. Thus, we need to determine the largest possible value of W.

Suppose that integers x_i satisfy (4.17) and maximize W. Then we claim that this optimal solution satisfies the following conditions:

(a) $x_i \geq x_{i+1}$ for any $i \geq 1$. Otherwise, we could switch the values of x_i and x_{i+1}, preserving inequality (4.17) and increasing W. Furthermore, $x_1 > 0$, as otherwise $W = 0$ but $x_1 = 1$, $x_2 = x_3 = \ldots = 0$ is a feasible solution with $W > 0$.

(b) $x_i = 0$ for any $i \geq 3$. Otherwise, we could decrease both x_{i-1} and x_i by 1 and increase x_1 by 1. Since $\lambda \geq \phi$, this increases W by at least $1/\lambda - 1/\lambda^2 - 1/\lambda^3 \geq 0$, and it preserves (4.17) and (a).

(c) $2x_1 + x_2 + 2 = s$. Otherwise (4.17) implies a strict inequality and we could modify x_1, x_2 as follows: If $x_2 = 0$, increase x_2 to 1. If $x_2 > 0$, increase x_1 by 1 and decrease x_2 by 1. This increases W, and it preserves (4.17), (a), and (b).

(d) $x_1 \leq x_2 + 2$. Otherwise, we could increase x_2 by 2 and decrease x_1 by 1. This increases W, and it preserves (4.17), (a), (b) and (c).

By (a), (c) and (d), we get $(s-2)/3 \leq x_1 \leq s/3$. For any $s \geq 4$, the only integer in this range is $x_1 = \lfloor s/3 \rfloor$. Thus $x_2 = s - 2 - 2\lfloor s/3 \rfloor$, and we have

$$\begin{aligned} 2 - \frac{1-\frac{1}{\lambda}}{W+1} &\leq 2 - \frac{1-\frac{1}{\lambda}}{1 + \frac{\lfloor \frac{s}{3} \rfloor}{\lambda} + \frac{s-2-2\lfloor \frac{s}{3} \rfloor}{\lambda^2}} \\ &= 2 - \frac{\lambda^2 - \lambda}{\lambda^2 + \lfloor \frac{s}{3} \rfloor \lambda + s - 2 - 2\lfloor \frac{s}{3} \rfloor} = \lambda, \end{aligned}$$

by the definition of λ. This completes the proof of Claim B and the the upper bound.

Claim C: The competitive ratio of $\text{EDF}_{1/\lambda}$ is no better than λ.

To prove Claim C we present instances on which the competitive ratio of $\text{EDF}_{1/\lambda}$ approaches λ. Intuitively, the bad instance consists of exactly one segment corresponding to the worst case from the proof of Claim B above. Let x_1 and x_2 be the optimal values as defined in that proof. Let $\epsilon > 0$ be arbitrarily small. The instance contains the following jobs, written as (r_j, d_j, w_j): x_2 jobs $(0, x_2, 1-\epsilon)$, x_1+1 jobs $(x_2, x_1+x_2+1, \lambda-\epsilon)$, x_2 jobs $(0, s, 1)$, 1 job $(0, s, \lambda)$, $x_1 - 1$ jobs $(x_2, x_2 + s, \lambda)$, and 1 job $(x_2, x_2 + s, \lambda^2)$. It is easy to check that the adversary schedules all the jobs in the given order, while $\text{EDF}_{1/\lambda}$ schedules only the jobs with weights 1, λ, and λ^2. The total weight obtained by $\text{EDF}_{1/\lambda}$ is $\lfloor s/3 \rfloor \lambda + (s - 2 - 2\lfloor s/3 \rfloor) + \lambda^2$ and the total weight obtained by the adversary approaches $(2\lfloor s/3 \rfloor + 1)\lambda + 2(s - 2 - \lfloor 2s/3 \rfloor) + \lambda^2$ for $\epsilon \to 0$. Hence the competitive ratio approaches λ. □

Theorem 4.7.4 *Let us consider the problem of online scheduling of* 4*–bounded instances of unit jobs in the deterministic model with the standard profit model. Above we described algorithm* EDF_α*. Constant* λ_4 *is defined above.*

Then the algorithm EDF_{1/λ_4} *solves this problem and it is* $\lambda_4 \approx 1.732$*–competitive.*

Proof: This theorem obviously follows from Theorem 4.7.3, because this is its special case for $s = 4$. □

Theorem 4.7.5 *Let us consider sufficiently large integer s tending to infinity and the problem of online scheduling of s–bounded instances of unit jobs in the deterministic model with the standard profit model. Above we described algorithm* EDF_α. *Constant λ_s is defined above.*

Then the algorithm EDF_{1/λ_s} *solves this problem and it is $\lambda_s = 2-2/s+o(1/s)$–competitive.*

Proof: For $s > 4$ the equation is cubic. It can be verified that $2 - 2/s \leq \lambda_s \leq 2-1/s$, and in the limit for $s \to \infty$, $\lambda_s = 2-2/s+o(1/s)$. Thus this theorem follows from Theorem 4.7.3 in the limit case for s tending to infinity. □

Chapter 5

Resource augmentation

When we study a non–trivial open problem from arbitrary area and we want to establish new results related to the studied problem then it is very important and useful for us and all other researches as well first to simplify studied problem, analyze and solve simplified problem before solving the studied problem.

In the context of the online scheduling problems and the competitive analysis we speak about the technique of resource augmentation. However the meaning of the resource augmentation in the context of competitive analysis is a little bit stronger—we develop a new online algorithm for a simplified problem and compare its performance with the optimal solution of the non–simplified problem.

We use the resource augmentation technique in the context of the competitive analysis because of two main reasons:

- The analysis of simplified problems is usually simpler and it can help us to get insight into the problem and solving the simplified problem gives us better chance to solve the general problem.
- When the problem is completely solved or at least partially solved under the competitive analysis but we are not satisfied with reached competitive ratio—for example the competitive ratio is unbounded, it means that there is no constant–competitive online algorithm. But when we are not so strict and we allow some small additional resources then can get constant competitive online algorithm, of course for some problems only.

With this method we can consider resources on which we can apply the resource augmentation method—like speed of a processor, number of processors, deadlines, weights, etc.

As we already mentioned in the introduction of this thesis already Graham in 1966 was working with such ideas and methods. Formally the technique of resource augmentation as a method of the competitive analysis was introduced in 1995 by Kalyanasundaram and Pruhs in [37]. They were focused on the problems with unbounded competitive ratio.

5.1 Introduction

The resource augmentation method has been applied to lots of various problems. We focused our research on the resource augmentation of two resources:

- **Deadlines**—we consider a fixed constant k and the deadline a job j expires at time d_j for the offline algorithm and at time $d_j + k$ for the online algorithm.
- **Speed of processor**—we consider a fixed constant s, the processor is running s–times faster for the online algorithm than for the offline algorithm.

We discuss results of our research on the problems with resource augmentation in the following sections.

5.2 Results overview

Our results

These results are products of our own research in the area of resource augmentation. Results for the problem k–relaxed online scheduling were not published. Results for the problem of scheduling in overloaded systems were published in [Pub–4].

We studied the problem of online k–relaxed scheduling in standard model. In this are we have shown several results. First we have shown that there is no 1–competitive online algorithm for this problem, see Theorem 5.3.2. Also we have shown some interesting lower bounds for the problem. We have shown in Theorem 5.3.3 that there is no 1–relaxed online algorithm with competitive ratio better ≈ 1.05099. Moreover we have shown general lower bound on the competitive ratio for the k–relaxed online algorithms, see Theorem 5.3.4.

Also we were working on the problem of scheduling in overloaded systems—the resource augmentation method using speed–up. We were

interested in bounds for speed–up. We have shown a lower bound on speed–up for the problem with tight jobs in the standard model. We have shown that there is no online 1–competitive algorithm with speed–up $s < 2$. We have shown input instance consisting of tight jobs only. This result is presented in Theorem 5.4.1.

Joint results

Here we present results which are product of joint research in the area of resource augmentation and overloaded systems. Presented results were published in [Pub–4].

First we have shown in Theorem 5.4.2 a $e/(e-1) \approx 1.58$–competitive online algorithm for the scheduling problem in the metered profit model. Also we have shown a lower bound ≈ 1.236 on the competitive ratio for the online algorithm in this problem, see Theorem 5.4.3.

Second we have shown in Theorem 5.4.4 that there is no 1–competitive online algorithm in metered profit model, with speed–up better than $\Omega(\log\log\xi)$ where ξ is the importance ratio.

Last result is regarding to resource augmentation in the number of processor. In Theorem 5.4.5 we have shown a lower bound on the competitive ratio $\Omega(\sqrt[m]{\xi}/m)$ where m is number of machines used by the online algorithm while the offline can use only one, the problem is considered in standard profit model.

5.3 k–relaxed algorithms

In this section we study online algorithms with relaxed deadlines. A k–relaxed algorithm is allowed to process a job up to k time units after its deadline, but the optimal schedule is not allowed.

Model: Online problem. We assume discrete integral time, each job is specified by integers of its release time and deadline and weight. The processing times of the jobs are equal to a constant. Preemptions are not allowed.

Definice 5.3.1 *k–relaxed algorithm is an algorithm which process each job j in the time interval $[r_j, d_j + k]$ or does not process j.*

We study the competitiveness of arbitrary deterministic algorithms for a fixed k. We show some lower bounds and also we show an algorithm—an upper bound on the competitive ratio.

5.3.1 Previous work

Albers and Schmidt were working on similar problem in [3]. They studied the behaviour of the greedy algorithm for the problem of packet buffering and they developed modified greedy algorithm. They were working on the standard version of the problem and on the version with additional resources—they considered resource augmentation of speed and memory, it means that they consider larger buffers or higher transmission rates.

Let us note that they study almost the same scheduling problem as us but in different terminology—the problem of packet buffering with m unbounded buffers is equivalent to the problem of scheduling of unit jobs on m processors.

The greedy algorithm is obviously 2–competitive. They consider the resource augmentation of the capacity of buffers—the buffers for the online algorithm are c–times larger than for the offline algorithm. In that case they shown that the greedy algorithm is $(c+2)/(c+1)$–competitive. They also consider the resource augmentation of the transmission rate—the online algorithm transmits k–times more packets at the same time than the offline algorithm. They shown that the greedy algorithm is $1+1/k$–competitive. for c–times larger capatity of buffers is the competitive ratio of the online

5.3.2 No 1–competitive algorithm

The algorithm is allowed to use more resources than the adversary in the optimal schedule. Obviously the competitive ratio cannot be smaller that 1 because of simple instances—e.g. a single job. But the natural and non–trivial question is whether the competitive ratio is strictly greater than 1. This question is answered by the next theorem.

Theorem 5.3.2 *Let us consider a fixed integer $k > 0$ and the problem of online k–relaxed scheduling of jobs in the deterministic model with the standard profit model.*

Then there is no 1–competitive algorithm solving this problem.

Proof. In fact this theorem proves a lower bound on the competitive ratio of considered algorithms for this scheduling problem. We prove this theorem using the method of contradiction. For a while we assume 1–competitive algorithm and we construct an input instance and we prove a contradiction. Let us consider a fixed k and denote $n = k+2$. We define an incremental sequence of input instances $I_0, \ldots, I_{n-1}$. The input instance I_i consists of blocks of jobs $B_0, \ldots B_i$. The block B_i:

- starts at time $t = i(k+2)$,
- at t releases a job of weight 2^i and span 1,
- at $t+1$ releases a job of weight 2^i and span 1,
- at $t+1, \ldots, t+k+1$ releases jobs of weight W and span 2,

where W is a sufficiently large number.

We assume there is an 1–competitive k–relaxed algorithm for contradiction and we distinguish two cases to analyze it:

- the algorithm processes all jobs of span 1 of the input sequence. Hence the algorithm processed $2n$ jobs of span 1. Deadline of the last job is $n(k+2)+1$. The algorithm is allowed to use k additional time slots, hence it can schedule at most $n(k+2)+1+k-2n = (n+1)k+1 = (k+3)k+1$ heavy jobs of weight W. The optimal schedule gains $n(k+1) = (k+2)(k+1)$ heavy jobs. The optimal gain is strictly greater than the gain of the algorithm.
- otherwise, let B_i be the first block containing a job of span 1 not processed by the algorithm; thus we finish the input instance by this block B_i. Then the optimal gain is $i(k+1)W + 1 + 2 + 4 + \cdots + 2^i + 2^i$, the gain of the algorithm is at most $i(k+1)W + 1 + 1 + 2 + 2 + \cdots + 2^{i-1} + 2^{i-1} + 2^i$. Optimal gain is greater than the gain of the algorithm.

Optimal gain is greater than the gain of the algorithm in both cases. We get a contradiction with the assumption that the assumed algorithm is 1–competitive. Therefore there is no 1–competitive k–relaxed algorithm for the problem. □

5.3.3 Lower bounds

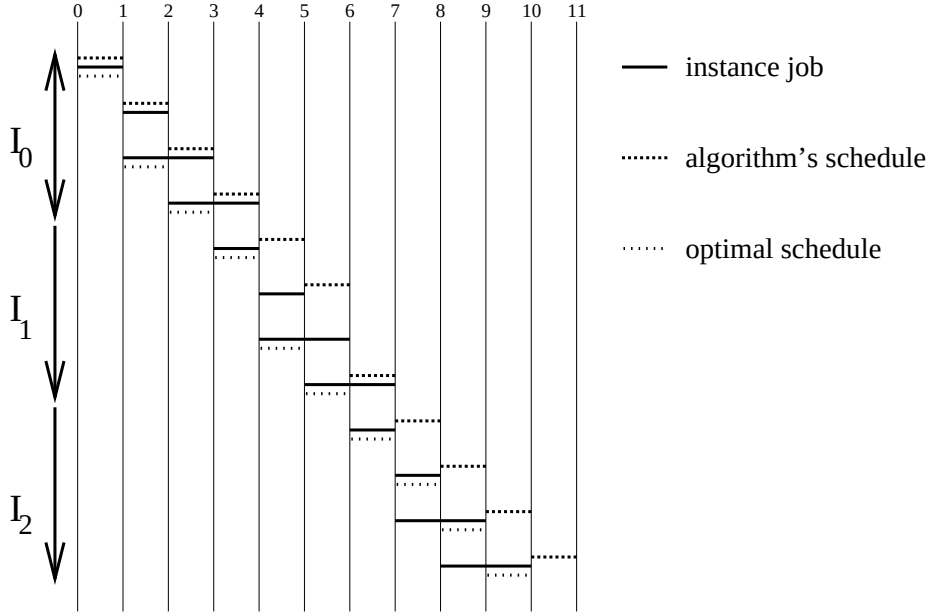
We slightly generalize the input instance as following. We define incremental sequence of instances $I_0, \ldots, I_{k+1}$ for a fixed constant $\alpha \geq 1$. The instance I_i consists of blocks $B_0, \ldots B_i$. The block B_i:

- starts at time $t = i(k+2)$,
- at t releases a job of weight α^i and span 1,
- at $t+1$ releases a job of weight α^i and span 1,
- at $t+1, \ldots, t+k+1$ releases jobs of weight α^{k+1} and span 2.

Theorem 5.3.3 *Let us consider the problem of online* 1*–relaxed scheduling of jobs in the deterministic model with the standard profit model.*

Then there is no algorithm solving this problem better than ≈ 1.05099*–competitive.*

Proof. We assume that there is a c–competitive 1–relaxed algorithm for the problem. We consider input instances I_0, I_1, I_2. First we consider that the algorithm loses a job of span 1 on an input instance I_i. See the instance and algorithm's and optimal schedule at the following picture.



Then the algorithm gains at most $2\alpha^2 + 1$ on I_0, $4\alpha^2 + \alpha + 2$ on I_1 and $6\alpha^2 + \alpha^2 + 2\alpha + 2$ on I_2. The adversary gains $2\alpha^2 + 2$ on I_0, $4\alpha^2 + 2\alpha + 1$ on I_1 or $6\alpha^2 + 2\alpha^2 + \alpha + 1$ on I_2. These facts and the c–competiveness of the algorithm force the following bounds:

$$c \geq \frac{2\alpha^2 + 2}{2\alpha^2 + 1} \text{for } I_0$$

$$c \geq \frac{4\alpha^2 + 2\alpha + 1}{4\alpha^2 + \alpha + 2} \text{for } I_1$$

$$c \geq \frac{8\alpha^2 + \alpha + 1}{7\alpha^2 + 2\alpha + 2} \text{for } I_2.$$

Otherwise the algorithm does not lose any of jobs with span 1, but the algorithm has 11 timeslots to process jobs, there are 6 jobs with span 1 and 6 jobs with span 2, hence the algorithm loses at least one of the heavy

jobs of span 2. Hence the algorithm gains at most $\alpha^5 + 2\alpha^2 + 2\alpha + 2$. The c–competitiveness forces the bound:

$$c \geq \frac{8\alpha^2 + \alpha + 1}{7\alpha^2 + 2\alpha + 2}.$$

Our analysis of the cases is complete, hence se at least one of these inequalities is satisfied. We optimize coefficient $\alpha = 2.42142$, exactly

$$\alpha = \frac{1}{6}\left(\sqrt[3]{161 + 3\sqrt{2118}} + \frac{19}{\sqrt[3]{161 + 3\sqrt{2118}}} + 5\right)$$

and we obtain the following lower bound

$$c > 1.05099.$$

□

Theorem 5.3.4 *Let us consider a fixed integer $k > 0$ and the problem of online k–relaxed scheduling of jobs in the deterministic model with the standard profit model.*

Then there is no algorithm solving this problem better than $1 + \frac{1}{2^{k+1}(k^2+3k+5)}$– competitive.

Proof. We assume that there is a c–competitive k–relaxed algorithm for the problem. We consider input instances $I_0, I_1, \ldots I_{k+1}$. Let us choose $\alpha = 2$. First we consider that the algorithm loses a job of span 1 on an input instance I_i. Then the algorithm on I_i gains

$$(i+1)(k+1)2^{k+1} + 2^i + 2\sum_{j=0}^{i-1} 2^j$$

while the optimum gains

$$(i+1)(k+1)2^{k+1} + 2^i + \sum_{j=0}^{i} 2^j.$$

Hence we get bounds for $i = 0, \ldots k+1$:

$$c \geq \frac{(i+1)(k+1)2^{k+1} + 2^i + \sum_{j=0}^{i} 2^j}{(i+1)(k+1)2^{k+1} + 2^i + 2\sum_{j=0}^{i-1} 2^j}.$$

Otherwise the algorithm does not lose any job of span 1, hence it has $(k+2)(k+2)+k+1$ time slots to process jobs of I_{k+1}. The algorithm processes all of $2(k+2)$ jobs with span 1, therefore it can process at most $(k+2)(k+2)+k+1-2(k+2) = k^2+3k+1$ jobs with span 2. The algorithm gains at most

$$(k^2+3k+1)2^{k+1} + 2\sum_{j=0}^{k+1} 2^j.$$

Observe that the algorithm has the same gain as in the case when the algorithm loses a job with span 1 on I_{k+1}.

We did analysis of all possible cases, hence at least one of these inequalities has to be satisfied. We eliminate the dependance on i from our inequalities using following operations:

$$\begin{aligned} c \;\geq\; & \frac{(i+1)(k+1)2^{k+1} + 2^i + \sum_{j=0}^{i} 2^j}{(i+1)(k+1)2^{k+1} + 2^i + 2\sum_{j=0}^{i-1} 2^j} \\ & = \frac{(i+1)(k+1)2^{k+1} + 2^i + 2^{i+1} - 1}{(i+1)(k+1)2^{k+1} + 2^i + 2^{i+1} - 2} \\ & \geq 1 + \frac{1}{(i+1)(k+1)2^{k+1} + 2^i + 2^{i+1} - 2} \\ & \geq 1 + \frac{1}{(k+2)(k+1)2^{k+1} + 2^{k+1} + 2^{k+2}} \\ & \geq 1 + \frac{2^{-k-1}}{k^2+3k+5}. \end{aligned}$$

□

5.4 Overloaded Systems

The following scheduling problem is studied: We are given a set of tasks with release times, deadlines, and profit rates. The objective is to determine a 1–processor preemptive schedule of the given tasks that maximizes the overall profit. In the standard model, each completed task brings profit, while non–completed tasks do not. In the metered model, a task brings profit proportional to the execution time even if not completed. For the metered task model, we present an efficient offline algorithm and improve both the lower and upper bounds on the competitive ratio of online algorithms.

5.4.1 Introduction

In most task scheduling problems the objective is to minimize some function related to the completion time. This approach is not useful in overloaded systems, where the number of tasks and their processing times exceed the capacity of the processor and not all tasks can be completed. In such systems, the goal is usually to maximize the number of executed tasks or, more generally, to maximize their value or profit.

The problem can be formalized as follows: we have a set of n tasks, each task j is specified by its release time r_j, deadline d_j, processing time p_j, and weight w_j representing its profit rate. Preemption is allowed, i.e., each task can be divided into any number of intervals, with arbitrary granularity. The objective is to determine a 1–processor preemptive schedule that maximizes the overall profit. The profit gained from processing task j can be defined in two ways. In the *standard model*, each completed task j brings profit $w_j p_j$, but non–completed tasks do not bring any profit. In the *metered model*, a task w_j executed for time $t \leq p_j$ brings profit $w_j t$ even if it is not completed.

In many real–world applications, algorithms for task scheduling are required to be *online*, i.e., to choose the task to process based only on the specification of the tasks that have already been released. An algorithm that approximates the optimal solution within a factor R is called *R–competitive*. Online algorithms are also studied in the framework called *resource augmentation*. The idea is to allow an online algorithm to use more resources (a faster processor or more processors) and then to compare its performance to the optimum solution (with no additional resources). For the scheduling problems, we then ask what competitive ratio can be achieved for a given speed–up factor s, or what speed–up is necessary to achieve 1–competitiveness. See [55, 15] for more information on competitive analysis.

The standard model. This problem has been extensively studied. Koren and Shasha [44] give a $(\sqrt{\xi}+1)^2$ - competitive algorithm, where $\xi = \max_j w_j / \min_j w_j$ is called the *importance factor*. This ratio is in fact optimal [14, 44]. Since no constant–competitive algorithms are possible in this model, it is natural to study this problem under the resource augmentation framework. Kalyanasundaram and Pruhs [37] present an online algorithm that uses a processor with speed 32 and achieves a constant competitive ratio. Lam and To [46] show an online algorithm with speed–up $O(\log \xi)$ and competitive ratio 1. One natural special case of this problem is when the tasks are *tight*, that is, for each j we have $d_j = r_j + p_j$. For this case, Koo *et al.*[43] give a 1–competitive algorithm with speed–up 14, and Lam *et*

al.[45] show that in order to achieve 1–competitiveness the speed–up must be at least $\phi \approx 1.618$.

The metered model. This version was introduced (in a different terminology) by Chang and Yap [17] in the context of thinwire visualization, where the profit represents overall *quality of service*. Metered preemptive tasks also provide a natural model for various decision making processes where an entity with limited resources needs to choose between engaging in several profitable activities. Chang and Yap proved that two online algorithms called FIRSTFIT and ENDFIT have competitive ratio 2. They also proved that no online algorithm can achieve a competitive ratio better than $2(2-\sqrt{2}) \approx 1.17$.

5.4.2 Lower bound on speed–up for tight jobs

In this section we mention our result for the problem of online scheduling with resource augmentation on speed–up of processor. We improve the lower bound from [45], by proving that, in order to achieve 1–competitiveness, an online algorithm needs speed–up at least 2.

This result is published in [Pub–4].

Theorem 5.4.1 *Let us consider the problem of online scheduling of tight jobs with allowed speed—up of processor, in the deterministic model with the standard profit model.*

Then there is no 1*–competitive algorithm solving this problem with speed–up* $s < 2$ *for scheduling of tight jobs.*

Proof: Let $\mathcal{A}$ be an online 1–competitive algorithm. We show an adversary strategy that, for any given n, forces $\mathcal{A}$ to run at speed $2 - 1/n$. The adversary chooses tasks from among $2n-1$ tasks defined as follows. Task 0 is $(0, n, n, 1)$. For $i = 1, \ldots, n-1$, task i is $(i-1, i, 1, 1)$ and task i' is $(i, n, n-i, n/(n-i))$.

The adversary strategy is this: issue tasks $0, 1, 2, ...$, as long as tasks $1, 2, \ldots, i$ are fully processed by $\mathcal{A}$ by time i. If $\mathcal{A}$ fails to fully process task i, the adversary issues task i' and halts. If this happens, the instance contains tasks $0, 1, \ldots, i, i'$ whose optimal profit is is $n + i$. To gain this profit $\mathcal{A}$ needs to process all tasks other than i. Their total length is $2n-1$, so $\mathcal{A}$'s speed must be at least $2 - 1/n$.

If $\mathcal{A}$ processes all tasks $1, ..., n-1$, the instance is $0, 1, \ldots, n-1$ and its maximum profit is n. To achieve this profit, $\mathcal{A}$ must also process task 0. Once again, this means that $\mathcal{A}$'s speed is at least $2 - 1/n$. □

5.4.3 Metered profit model

In this subsection we focus on the metered profit model in the online scheduling in the overloaded systems. These results have been published in the paper [Pub–4]. These results products of joint research with other co–authors. We provide a short overview of results published there without their proofs.

First we mention the upper bound—competitive online algorithm for metered tasks. We have shown an algorithm MIXED for the problem and we prove the following competitive ratio:

Theorem 5.4.2 *Let us consider the problem of online scheduling of jobs in the deterministic model with the metered profit model. Above we described algorithm* MIXED*.*

Then this algorithm MIXED *is* $e/(e-1) \approx 1.5820$*–competitive.*

We have shown a lower bound on the competitive ratio for the problem of online scheduling of metered tasks.

Theorem 5.4.3 *Let us consider the problem of online scheduling of jobs in the deterministic model with the metered profit model.*

Then there is no algorithm solving this problem better than $\sqrt{5}-1 \approx 1.236$*–competitive.*

These results improve both the lower and upper bounds from [17].

Next we study the resource augmentation version of this problem, and prove that no online algorithm with constant speed–up can be 1–competitive, neither in the metered profit model, nor in the standard model. In fact, we prove that the minimal speed–up needed to achieve 1–competitiveness is $\Omega(\log\log\xi)$. Thus we disprove a conjecture from [43] by showing that the problem with general deadlines is provably harder than the special case of tight deadlines.

Theorem 5.4.4 *Let us consider the problem of online scheduling of jobs with allowed speed–up of processor in the deterministic model with the metered profit model or alternatively with standard profit model. Let us denote the importance ratio as* ξ*.*

Then each 1*–competitive algorithm solving this problem has speed–up at least* $\Omega(\log\log\xi)$*. In particular, there is no constant speed–up* 1*–competitive algorithm.*

5.4.4 More processors

In this subsection we consider another way of resource augmentation. Here we consider online algorithms which are allowed to use more processors that the offline algorithms. We consider this in the standard profit model. These results have been published in the paper [Pub–4] and are products of joint research with other co–authors. We provide a short overview of results published there without their proofs.

Furthermore we focus the model where an online algorithm is allowed to use m processors of speed 1, rather than a single faster processor. For this case we prove that the competitive ratio is $\Omega(\sqrt[m]{\xi}/m)$, even if all tasks are restricted to be tight.

Theorem 5.4.5 *Let us consider the problem of online scheduling of tight jobs in the deterministic model with the standard profit model, the online algorithms are allowed to use m processors against a 1–processor adversary. Let us denote the importance ratio as ξ.*

Then no online algorithm allowed to use m processors is better than $\Omega(\sqrt[m]{\xi}/m)$–competitive against adversary using 1 processor.

Observe that for tight tasks constant speed–up is sufficient for 1–competitiveness, so the lower bound shows that increasing the speed of a single processor is more powerful than increasing the number of processors of speed 1.

Chapter 6

Online scheduling of equal–length jobs

Let us consider the following scheduling problem. The input is a set of jobs with equal processing times, where each job is specified by its release time and deadline. The goal is to determine a single–processor, non–preemptive schedule that maximizes the number of completed jobs. In the online version, each job arrives at its release time.

It is known that a simple greedy algorithm is 2–competitive for this problem and that this ratio is optimal for deterministic algorithms. We present ways how to improve the competitive ratio.

6.1 Results overview

Our results

Here we present results of our own research on the problem of online scheduling of equal–length jobs. These results were already published in [Pub–1].

We give in Theorem 6.5.2 a deterministic $\frac{3}{2}$–competitive algorithm in the *preemption–restart* model. In this model, an online algorithm is allowed to abort a job during execution, in order to start another job. The algorithm gets credit only for jobs that are executed contiguously from beginning to end. Aborted jobs can be restarted (from scratch) and completed later. Note that the final schedule produced by such an algorithm is *not* preemptive. Thus the distinction between non–preemptive and preemption–restart models makes sense only in the online case. (The optimal solutions are always the same.)

In addition to the algorithm, we give in Theorem 6.5.4 a matching lower bound, by showing that no deterministic online algorithm with restarts can be better than $\frac{3}{2}$–competitive.

We also show in Theorem 6.5.4 a lower bound of $\frac{6}{5}$ for randomized algorithms with restarts.

Joint results

Here we presents results of our joint research on the problem of online scheduling of equal–length jobs. These results were already published in [Pub–1].

We give in Theorem 6.6.1 a barely random $\frac{5}{3}$–competitive algorithm that uses only one random bit. We also show a lower bound of $\frac{3}{2}$ on the competitive ratio of barely random algorithms that randomly choose one of two deterministic algorithms, see Theorem 6.6.5. If the two algorithms are selected with equal probability, we can further improve the bound to $\frac{8}{5}$ in Theorem 6.6.6.

6.2 Previous work

The problem of scheduling of equal–length jobs to maximize the number of completed jobs has been well studied in the literature. In the offline case, an $O(n \log n)$–time algorithm for the feasibility problem (checking if *all* jobs can be completed) was given by Garey *et al.* [28] (see also [56, 16]). The maximization version can also be solved in polynomial time [21, 7], although the known algorithms are rather slow. (Carlier [16] claimed an $O(n^3 \log n)$ algorithm but, as pointed out in [21], his algorithm is not correct.)

As the first positive result on the online version, Baruah *et al.* [13, 12] show that a deterministic greedy algorithm is 2–competitive; in fact, they show that 2–competitiveness holds for any non–preemptive deterministic algorithm that is never idle at times when jobs are available for execution.

Goldman *et al.* [29] gave a lower bound of $\frac{4}{3}$ on the competitive ratio of randomized algorithms and the tight bound of 2 for deterministic algorithms. We briefly sketch these lower bounds, as they illustrate well what situations an online algorithm needs to avoid in order to achieve a small competitive ratio. Let $p \geq 2$. The jobs, written in the form $j = (r_j, d_j)$, are $1 = (0, 2p + 1)$, $2 = (1, p + 1)$, $3 = (p, 2p)$. The instance consists of jobs 1,2 or jobs 1,3; in both cases the optimum is 2. Figure 6.1 illustrates the input instance and the adversary strategy. (In this figure, and later throughout

the paper, the horizontal dimension corresponds to the time axis, each job j in the input instance is drawn as a line segment spanning the interval $[r_j, d_j]$, and jobs that appear in the schedules are represented by rectangles of length p positioned at the actual time of execution.) In the deterministic

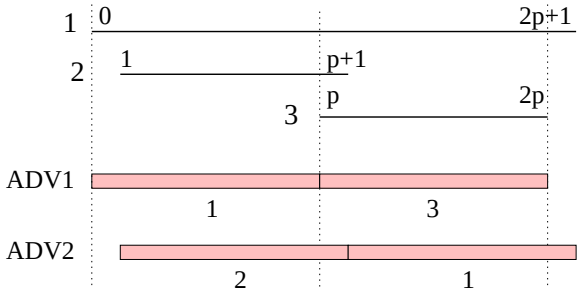


Figure 6.1: Jobs used in the lower bound proof.

case, release job 1; if at time 0 the online algorithm starts job 1, then release job 2, otherwise release job 3. The online algorithm completes only one job and the competitive ratio is no better than 2. In the randomized case, using Yao's principle, we choose each of the two instances with probability $\frac{1}{2}$. The expected number of completed jobs of any deterministic online algorithm is at most 1.5, as on one of the instances it completes only one job. Thus the competitive ratio is no better than $2/1.5 = \frac{4}{3}$.

Goldman *et al.* [29] show that the lower bound of 2 can be beaten if the jobs on input have sufficiently large "slack"; more specifically, they prove that a greedy algorithm is $\frac{3}{2}$–competitive for instances where $d_j - r_j \geq 2p$ for all jobs j. This is closely related to our algorithm with restarts: On such instances, our algorithm never uses restarts and becomes identical to the greedy algorithm. Thus in this special case our result constitutes an alternative proof of the result from [29]. Exploring further this direction, Goldwasser [30] obtained a parameterized extension of this result: if $d_j - r_j \geq \lambda p$ for all jobs j, where $\lambda \geq 1$ is an integer, then the competitive ratio is $1 + 1/\lambda$.

In our brief overview of the literature given above we focused on the case when jobs are of equal length and the objective function is the number of completed jobs. We need to stress though that, in addition to the work cited above, there is vast literature on real–time scheduling problems where a variety of other models is considered. Other or no restrictions can be placed on processing times, jobs may have different weights (profits), we can have multiple processors, and preemptions may be allowed. For example, once arbitrarily processing times and/or weights are introduced, no constant–competitive non–preemptive algorithms exist. Therefore it is

common in the literature to allow preemption with resume, where a job can be preempted and later started from where it was stopped.

The model with restarts was studied before by Hoogeveen *et al.* [34]. They present a 2–competitive deterministic algorithm with restarts for jobs with arbitrary processing times and objective to maximize the number of completed jobs. They also give a matching lower bound. Their algorithm does not use restarts on the instances with equal processing times, and thus it is no better than 2–competitive for our problem.

Real–time scheduling is an area where randomized algorithms have been found quite effective. Most randomized algorithms in the general scenarios use the classify–and–randomly–select technique by Lipton and Tomkins [49]. Typically, this method decreases the dependence of competitive ratio from linear to logarithmic in certain parameters (e.g., the maximum ratio between job weights), but it does not apply to the case of jobs with equal lengths and weights. Our randomized algorithm is based on entirely different ideas.

Barely random algorithms have been successfully applied in the past to a variety of online problems, including the list update problem [51], the k–server problem [9] and makespan scheduling [2, 25, 52]. In particular, the algorithm of Albers [2] involves two deterministic processes in which the second one keeps track of the first and corrects its potential "mistakes"—a coordination idea somewhat similar to ours, although in [2] the two processes are not symmetric.

The area of real–time scheduling is of course well motivated by multitudes of applied scenarios. In particular, the model of equal–length jobs—without or with limited preemption—is related to applications in packet switched networks. When different weights are considered, the problem has further connections to the "quality of service" issues (recently a fashionable phrase). Nevertheless, we shamelessly admit that this work has been partially driven by plain curiosity. It is quite intriguing, after all, that so little is known about the competitiveness of such a fundamental scheduling problem.

6.3 Preliminaries

The instance on input is a set of jobs $J = \{1, 2, \dots\}$. Each job j is given by its release time r_j and deadline d_j. All jobs have processing time p. (We assume that all numbers are positive integers and that $d_j \geq r_j + p$ for all j.) The *expiration time* of a job j is $x_j = d_j - p$, i.e., the last time when it can be started. A job j is called *admissible* at time t if $r_j \leq t \leq x_j$. A job j is called

tight if $x_j - r_j < p$.

A *non–preemptive schedule* $\mathcal{A}$ assigns to each completed job j an interval $[S_j^{\mathcal{A}}, C_j^{\mathcal{A}})$, with $r_j \leq S_j^{\mathcal{A}} \leq x_j$ and $C_j^{\mathcal{A}} = S_j^{\mathcal{A}} + p$, during which j is executed. These intervals are disjoint for distinct jobs. $S_j^{\mathcal{A}}$ and $C_j^{\mathcal{A}}$ are called the *start time* and *completion time* of job j. Without loss of generality, both are assumed to be integer. We adopt a convention that "job running (a schedule being idle, etc.) at time t" is an equivalent shortcut for "job running (a schedule being idle, etc.) in the interval $[t, t+1)$". Given a schedule $\mathcal{A}$, a job is *pending* at time t in $\mathcal{A}$ if it is admissible at t (that is, $r_j \leq t \leq x_j$) but not yet completed in $\mathcal{A}$. Note that according to this definition a job that is being executed at t may also be considered pending. When $\mathcal{A}$ is understood from context, we will typically use notation P_t to denote the set of jobs pending at time t.

For any set of jobs Q, we say that that Q is *feasible* at time t if there exists a schedule which completes all jobs in Q such that no job is started before t. Q is *flexible* at time t if it is feasible at time $t + p$.

Applying the Jackson rule [35], it is quite easy to determine whether a set P of pending jobs is feasible at t: Order the jobs in P in order of increasing deadlines, and schedule them at times t, $t + p$, $t + 2p$, etc. Then P is feasible if and only if all jobs in P meet their deadlines. Furthermore, if we want to compute the maximum–size feasible subset $P' \subseteq P$, we can start with $P' = \emptyset$, and then add jobs $j \in P - P'$ to P', one by one and in arbitrary order, as long as P' remains feasible. This means, in particular, that P' is a maximum–size feasible subset of P iff P' is a $\subseteq$–maximal feasible subset of P. All those properties can be proven by elementary exchange arguments, and the proofs are left to the reader.

We say that a job started by a schedule $\mathcal{A}$ at time t is *flexible in* $\mathcal{A}$ if the set of all jobs pending in $\mathcal{A}$ at t is flexible; otherwise the job is called *urgent*. Intuitively, a job is flexible if we could possibly postpone it and stay idle for time p, without losing any of the currently pending jobs; this could improve the schedule if a tight job arrives. On the other hand, postponing an urgent job can bring no advantage to the algorithm.

An *online algorithm* constructs a schedule incrementally, at each step t making decisions based only on the jobs released at or before t. Each job j, including its deadline, is revealed to the algorithm at its release time r_j. A *non–preemptive online algorithm* can start a job only when no job is running; thus, if a job is started at time t the algorithm has no choice but to let it run to completion at time $t + p$. An *online algorithm with restarts* can start a job at any time. If we start a job j when another job, say k, is running, then k is aborted and started from scratch when (and if) it is started again later. The unfinished portion of k is removed from the final schedule, which is

considered to be idle during this time interval. Thus the final schedule generated by an online algorithm with restarts is non–preemptive.

An online algorithm is called *c–competitive* if, for any set of jobs J and any schedule ADV for J, the schedule $\mathcal{A}$ generated by the algorithm on J satisfies $|\mathrm{ADV}| \leq c|\mathcal{A}|$. If the algorithm is randomized, the expression $|\mathcal{A}|$ is replaced by the expected (average) number of jobs completed on the given instance.

The definitions above assume the model—standard in the scheduling literature—with integer release times and deadlines, which implicitly makes the time discrete. Some papers on real–time scheduling work with continuous time. Both our algorithms can be modified to the continuous time model and unit processing time jobs without any changes in performance, at the cost of somewhat more technical presentation.

6.4 Properties of schedules

For every instance J, we fix a *canonical linear ordering* $\prec$ of J such that $j \prec k$ implies $d_j \leq d_k$. In other words, we order the jobs by their deadlines, breaking the ties arbitrarily but consistently for all applications of the deadline ordering. The term *earliest–deadline*, or briefly ED, now refers to the $\prec$–minimal job.

A schedule $\mathcal{A}$ is called *earliest–deadline–first* (or *EDF*) if, whenever it starts a job, it chooses the ED job of all the pending jobs that are later completed in $\mathcal{A}$. (Note that this may not be the overall ED pending job.)

A schedule $\mathcal{A}$ is *normal* if it satisfies the following two properties:

(n1) when $\mathcal{A}$ starts a job, it chooses the ED job from the set of all pending jobs;

(n2) if the set of all pending jobs in $\mathcal{A}$ at some time t is not flexible, then some job is running at t.

Obviously, any normal schedule is EDF, but the reverse is not true. All algorithms presented in this paper generate normal schedules. The properties (n1) and (n2) are reasonable, as the online algorithm cannot make a mistake by enforcing them. Formally, any online algorithm can be modified, using a standard exchange argument, to produce normal schedules without reducing the number of scheduled jobs. (We omit the proof, as we do not need this fact in the paper.)

The following property will be crucial in our proofs.

Lemma 6.4.1 *Suppose that a job j is urgent in a normal schedule $\mathcal{A}$. Then at any time t, $S_j^{\mathcal{A}} \leq t \leq x_j$, an urgent job is running in $\mathcal{A}$.*

Proof: Denote by P the set of jobs pending at time $S_j^{\mathcal{A}}$ (including j). By the assumption about j, P is not flexible at $S_j^{\mathcal{A}}$. Towards contradiction, suppose that $\mathcal{A}$ is idle or starts a flexible job at time t, where $S_j^{\mathcal{A}} \leq t \leq x_j$. Then the set Q of jobs pending at time t is flexible at t. Since j is the ED job from P (by the normality of $\mathcal{A}$) and $t \leq x_j$, all other jobs in P have not expired until t, and thus Q contains all the jobs from P that are not completed in $\mathcal{A}$ until time t.

Using the above properties, we can rearrange the schedule as follows. Since Q is flexible at t, we can schedule all jobs of Q at time $t + p$ or later, start j at t and schedule all jobs in $P - Q - \{j\}$ as in $\mathcal{A}$. But this shows that P is flexible at time $S_j^{\mathcal{A}}$—a contradiction. □

Lemma 6.4.2 *Let $\mathcal{X}$ be a normal schedule for a set of jobs J. Let $f : J \to J$ be a partial function such that if $f(j)$ is defined then j is scheduled as flexible in $\mathcal{X}$ and $r_{f(j)} \langle C_j^{\mathcal{X}} \leq x_{f(j)}$. Then there exists an EDF schedule $\mathcal{A}$ equivalent to $\mathcal{X}$ such that:*

(1) *All jobs $f(j)$ are completed in $\mathcal{A}$.*
(2) *Consider a time t when either $\mathcal{A}$ is idle or it starts a job and the set of all its pending jobs is feasible at t. Then all jobs pending at t are completed in $\mathcal{A}$. In particular, each job that is pending when $\mathcal{A}$ starts a flexible job is completed in $\mathcal{A}$.*
(3) *Let j be a job completed in $\mathcal{A}$, and let U be the set of all jobs j' with $r_{j'} \langle C_j^{\mathcal{A}}$ that are pending at time $C_j^{\mathcal{A}}$. (Note that the jobs released at time $C_j^{\mathcal{A}}$ are not included in U.) If U is feasible at time $C_j^{\mathcal{A}}$ then all the jobs in U are completed in $\mathcal{A}$.*

Furthermore, if $\mathcal{X}$ is constructed by an online algorithm and $f(j)$ can be determined online at time $C_j^{\mathcal{X}}$ for each flexible job j in $\mathcal{X}$, then $\mathcal{A}$ can be produced by an online algorithm.

Remarks: Property (1) is useful in our proofs, since it allows us to modify the schedule computed by the algorithm to resemble more the optimal schedule. Property (2) guarantees that any job planned to be scheduled is indeed scheduled in the future. Property (3) is a technical condition needed in the analysis of the algorithm with restarts.

Since $\mathcal{A}$ and $\mathcal{X}$ are equivalent, flexible jobs are the same and scheduled at the same times in $\mathcal{A}$ and $\mathcal{X}$. In particular, all the jobs j on which $f(j)$ is defined are scheduled at the same time both in $\mathcal{A}$ and $\mathcal{X}$—a property that will play an important role in our later arguments.

The basic idea of the construction of $\mathcal{A}$ is quite straightforward: Maintain a set Q_t of jobs that we plan to schedule. If the set of all pending jobs is feasible, we plan to schedule them all. In particular, if we start a flexible job

j at time t, the flexibility of j allows us to add to Q_{t+p} an extra job released during the execution of j; so if $f(j)$ is defined, we add $f(j)$. Similarly, to achieve (3), we can add all jobs released during the execution of j, as long as the resulting set remains feasible.

Proof: We construct $\mathcal{A}$ iteratively. Throughout the proof, t ranges over times when $\mathcal{X}$ is idle or starts a job. For each such t, let P_t and P'_t denote the set of jobs pending in $\mathcal{X}$ and $\mathcal{A}$, respectively.

We will maintain an auxiliary set of jobs Q_t that are pending at t in $\mathcal{X}$ and $\mathcal{A}$, that is $Q_t \subseteq P_t \cap P'_t$. Simultaneously with the construction, we prove inductively that, for all t, the following invariant holds:

$(*)$ Q_t is an $\subseteq$–maximal feasible subset of each of P_t and P'_t.

Before describing the construction, we make two observations. First, recall that condition $(*)$ implies that Q_t is also maximum with respect to size. Second, if any of sets P_t, P'_t, Q_t is flexible, then $Q_t = P_t = P'_t$, by the maximality of Q_t.

We now describe the construction. Initially, choose Q_0 as an arbitrary maximal feasible set of jobs released at time 0.

Assume we have already defined Q_t. If $\mathcal{X}$ is idle at t, we let $\mathcal{A}$ idle and choose an arbitrary $Q_{t+1} \supseteq Q_t$ by adding to Q_t the jobs released at $t+1$, as long as the set remains feasible. Since $\mathcal{X}$ is idle, P_t is flexible at t, and thus $Q_t = P_t = P'_t$ and Q_t is feasible at $t+1$. Thus $P_{t+1} = P'_{t+1}$ and $(*)$ holds at time $t+1$.

Now suppose that $\mathcal{X}$ starts a job j at time t. We consider two subcases, depending on whether j is flexible or urgent.

Case 1: j is flexible in $\mathcal{X}$. Then $\mathcal{A}$ starts j, too. In this case, $Q_t = P_t = P'_t$, and thus $P'_{t+1} = P_{t+1}$ as well. Clearly, j is flexible in $\mathcal{A}$ as well. Since Q_t is flexible, it is feasible at $t+p$. To construct Q_{t+p}, we start with $Q_{t+p} = Q_t - \{j\}$ and expand it by adding newly released jobs, one by one, as long as Q_{t+p} remains feasible. The jobs are added in the following order:

(i) First, add $f(j)$, if it is defined, pending, and not yet in Q_{t+p}.
(ii) Next, add all jobs j' with $t \langle r_{j'} \langle t+p$, in an arbitrary order.
(iii) Finally, add all jobs j' with $r_{j'} = t+p$, in an arbitrary order.

Since Q_t is feasible at $t+p$, $Q_t - \{j\} \cup \{f(j)\}$ is feasible at $t+p$ as well, so $f(j)$ can be added to Q_{t+p} in (i) without violating feasibility. By the construction, $(*)$ is satisfied at time $t+p$.

Case 2: j is urgent in $\mathcal{X}$. $\mathcal{A}$ starts the earliest–deadline (more precisely, $\prec$–minimal) job k from Q_t. Since Q_t is a maximal feasible set both for $\mathcal{X}$ and $\mathcal{A}$, it is non–empty whenever $\mathcal{X}$ starts a job. Furthermore, we know that Q_t is not flexible at t and thus k is urgent.

Let $T = Q_t - \{k\}$. We claim that:

(t1) T is a maximal subset of P_t (resp. P'_t) that is feasible at time $t+p$, and
(t2) $T \subseteq P_{t+p} \cap P'_{t+p}$.

That T is feasible at $t+p$ follows directly from the definition of T and the fact that k is the ED job in Q_t. For the same reason, all jobs in T are pending in $\mathcal{A}$ at time $t+p$. Since k is pending in $\mathcal{X}$ at t, and $\mathcal{X}$ schedules the ED pending job (as $\mathcal{X}$ is normal), we have $j \prec k$. Therefore all jobs in T are pending at time $t+p$ in $\mathcal{X}$ as well. We conclude that (t2) holds.

No job in $P'_t - Q_t$ can be feasibly added to T at time $t+p$, as otherwise it could be feasibly added to Q_t at time t, contradicting the maximality of Q_t for $\mathcal{A}$. The same argument applies to $\mathcal{X}$. Thus, T satisfies condition (t1) for both P_t and P'_t.

We construct Q_{t+p} similarly as in the previous case. We start with $Q_{t+p} = T$ and add newly released jobs, first the jobs released before $C_j^{\mathcal{X}} = t+p$, and then the jobs released at $C_j^{\mathcal{X}}$, one by one, as long as Q_{t+p} remains feasible. Again, the maximality of T and the construction implies that Q_{t+p} satisfies $(*)$ at time $t+p$.

This completes the construction. Obviously, $\mathcal{X}$ and $\mathcal{A}$ are equivalent. Also, $\mathcal{A}$ is EDF since whenever it schedules a job, it chooses the ED job of Q_t, and jobs $j' \in P'_t - Q_t$ are never added to Q_s for $s \rangle t$, so they will not be scheduled in $\mathcal{A}$.

By the construction, $\mathcal{A}$ schedules all the jobs that are in some plan Q_t. At any time t when $\mathcal{A}$ is idle or starts a flexible job, Q_t is flexible and thus $Q_t = P'_t$. This proves (2). Since $f(j) \in Q_{t+p}$ for $t = S_j^{\mathcal{X}}$, this also implies (1). Finally, to show (3), recall that when constructing Q_{t+p} for $t = S_j^{\mathcal{X}}$, we are first adding to Q_{t+p} all the jobs j' with $r_{j'} \langle t+p$ (by the assumption of the lemma, $j' = f(j)$ satisfies this); if they are all together feasible then they are all added and thus $U \subseteq Q_{t+p}$. □

6.5 Restarts help

Our algorithm with restarts is very natural. At any time, it greedily schedules the ED job. However, if a tight job arrives that would expire before the running job is completed, we consider a preemption. If all pending jobs can be scheduled, the preemption occurs. If not, it means that some pending job is necessarily lost and the preemption would be useless, so we continue running the current job and let the tight job expire.

We need an auxiliary definition. Suppose that a job j is started at time s by the algorithm. We call a job l a *preemption candidate* for j if $s \langle r_l \leq x_l \langle s+p$.

Algorithm TIGHTRESTART**.** At time t:

(TR1) If no job is running, start the ED pending job, providing there is at least one pending job, otherwise stay idle until some job is released.

(TR2) Otherwise, let j be the running job. If j was started as urgent or no preemption candidate is released at t, continue running j.

(TR3) Otherwise, let P_t^* be the set of all jobs pending at time t, including j but excluding any preemption candidates. If P_t^* is flexible at t, preempt j and start (at time t) a preemption candidate; choose the ED preemption candidate, if more are admissible at time t. Otherwise continue running j.

Let $\mathcal{X}$ be the final schedule generated by TIGHTRESTART, after removing the preempted parts of jobs. For any time t, as before, by P_t we denote the set of jobs that are pending in $\mathcal{X}$ at time t. We stress that we distinguish between $\mathcal{X}$ being idle and TIGHTRESTART being idle: at some time steps TIGHTRESTART can process a job that will be preempted later, in which case $\mathcal{X}$ is considered idle at these steps but TIGHTRESTART is not.

Lemma 6.5.1 *Schedule $\mathcal{X}$ is normal.*

Proof: By rules (TR1) and (TR3), TIGHTRESTART always starts the ED pending job; in (TR3) note that, by definition, any preemption candidate is tight and thus it has earlier deadline than any job in the flexible set P_t^* of the remaining pending jobs. The property (n1) of normal schedules follows.

If TIGHTRESTART is idle then there is no pending job. Thus, to show the property (n2), it remains to verify that $P_{t'}$ is flexible at any time t' when $\mathcal{X}$ is idle but TIGHTRESTART is not. This means that TIGHTRESTART is running a job which is later preempted.

Suppose TIGHTRESTART starts a flexible job j at time s and preempts it at time t. Let t' be any time such that $s \langle t' \langle t$. Since j is flexible at s and it is the ED job in P_s, no job in P_s expires before $s + p \rangle t$. Thus we have $P_s \subseteq P_{t'}^* \subseteq P_t^*$, by the definition of $P_{t'}^*$ and P_t^* in (TR3). As TIGHTRESTART preempts at time t, P_t^* is flexible at t. Consequently, $P_{t'}^* \subseteq P_t^*$ is flexible at t and also at $t' \langle t$. Using this for all t', we conclude that the first preemption candidate for j is released at t, as otherwise j would be preempted earlier. Thus no preemption preemption candidate is admissible at any t', $s \langle t' \langle t$, and $P_{t'} = P_{t'}^*$ which we have shown is flexible at t'. Thus (n2) holds and $\mathcal{X}$ is normal. □

Theorem 6.5.2 *Let us consider the problem of online scheduling of equal–length instances of jobs with allowed restarts, in the deterministic model with the standard profit model. Above we described algorithm* TIGHTRESTART*.*

Then the algorithm TIGHTRESTART *solves this problem and it is $\frac{3}{2}$–competitive.*

Proof: As usual, let ADV denote an optimal schedule.

Let us start by giving some intuition behind the charging scheme. Suppose that a job j is started at time t in ADV. If a job k is running at t in $\mathcal{X}$, we want to charge j to k. If $\mathcal{X}$ is running a job k which is later preempted by l, we charge $\frac{1}{2}$ to l and $\frac{1}{2}$ to k (using Lemma 6.4.2 to guarantee that the modified schedule completes k). The main problem is to handle the case when $\mathcal{X}$ is idle when j starts in ADV; we call such a j a *free* job. In this case, TIGHTRESTART was "tricked" into scheduling of j too early. We would like to charge j to itself. However, it may happen that then j would be charged twice, so we need to split this charge and find another job that we can charge $\frac{1}{2}$. The definition of $f(j)$ below chooses such a job and Lemma 6.4.2 again guarantees that the modified schedule completes $f(j)$.

The difficulty that arises in the above scheme is that, due to preemptions in TIGHTRESTART's schedule and to idle times in ADV, the jobs can become misaligned. To deal with this problem, we define a matching M between the jobs in $\mathcal{X}$ and ADV. Typically, a job k in $\mathcal{X}$ is matched to the first unmatched job in ADV that starts later than k. In some situations we match a free job k to itself.

We now proceed with the formal proof. First, we define a partial function $f : J \to J$. For any job j scheduled as flexible in $\mathcal{X}$, we define $f(j)$ as follows.

(f1) If at some time t, $S_j^{\mathcal{X}} \leq t \langle C_j^{\mathcal{X}}$, ADV starts a job k which is not a preemption candidate then let $f(j) = k$.

(f2) Otherwise, if there exists a job k with $S_j^{\mathcal{X}} \langle r_k \langle C_j^{\mathcal{X}} \leq x_k$ such that ADV does not complete k, then let $f(j) = k$ (choose arbitrarily if there are more such k's).

(f3) Otherwise, $f(j)$ is undefined.

Notice that $f(\cdot)$ is one–to–one, for the first two cases are disjoint, and in each case j is uniquely determined by $k = f(j)$: If $k = f(j)$ and the first case applied to j, then j is the job that is being executed by $\mathcal{X}$ when ADV starts k. If the second case applied to j, then j is the job being executed by $\mathcal{X}$ at the release time of k.

According to Lemma 6.5.1, $\mathcal{X}$ is a normal schedule. Let $\mathcal{A}$ be the schedule constructed in Lemma 6.4.2 from $\mathcal{X}$ and function $f(\cdot)$. Since $\mathcal{A}$ is equivalent to $\mathcal{X}$, it also satisfies Lemma 6.4.1.

Call a job j scheduled in ADV a *free* job if TIGHTRESTART is idle at time S_j^{ADV}. This condition implies that at time S_j^{ADV} no job is pending in $\mathcal{A}$; in particular, by Lemma 6.4.1, j is completed as a flexible job by time S_j^{ADV} in $\mathcal{A}$.

Now define a partial function $M : J \to J$ which is a matching of (some) occurrences of jobs in $\mathcal{A}$ to those in ADV. Process the jobs k scheduled in $\mathcal{A}$

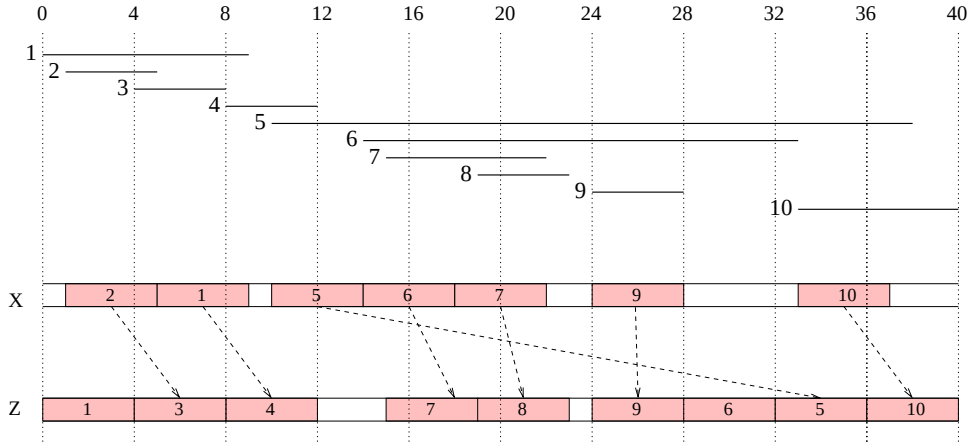


Figure 6.2: An example of an instance, the schedule $\mathcal{A}$ produced by TIGHTRESTART, and the construction of M (represented by directed edges). The processing time is $p = 4$. Jobs are identified by positive integers. Preempted pieces of jobs are not shown.

in the order of increasing $S_k^{\mathcal{A}}$. For a given k, let j be the first unmatched job in ADV started at or after $S_k^{\mathcal{A}}$, or, more specifically, a job with smallest S_j^{ADV} among those with $S_j^{\mathrm{ADV}} \geq S_k^{\mathcal{A}}$ and such that $j \neq M(k')$ for all k' in $\mathcal{A}$ with $S_{k'}^{\mathcal{A}} \langle S_k^{\mathcal{A}}$. If no such j exists, $M(k)$ is undefined. Else, if k is a free job, not in the current range of M, and $S_j^{\mathrm{ADV}} \geq C_k^{\mathcal{A}}$, then let $M(k) = k$. Otherwise, let $M(k) = j$. The definition implies that M is one–to–one. (See Figure 6.2.)

Lemma 6.5.3 *Let j be a job executed in* ADV.

(1) *If $\mathcal{A}$ executes some job when j starts in* ADV, *that is $S_k^{\mathcal{A}} \leq S_j^{\mathrm{ADV}} \langle C_k^{\mathcal{A}}$ for some k, then j is in the range of M.*

(2) *If j is free and $f(j)$ is undefined then j is in the range of M.*

Proof: Part (1) is simple: Suppose that $\mathcal{A}$ is executing some job k at S_j^{ADV}, and consider the step in the construction of M when we are about to define $M(k)$. If j is not in the range of M at this time, then we would define $M(k)$ as j.

We now prove (2). Let $s = S_j^{\mathcal{A}}$ be the start time of j in $\mathcal{A}$ and $s' = C_j^{\mathcal{A}} = s + p$ its completion time. Since j is free, it is completed in $\mathcal{A}$ before it is started in ADV, that is $S_j^{\mathrm{ADV}} \geq s'$.

Suppose for a contradiction that j is not in the range of M. By the definition of M, this implies that $M(j) = l$ for some job l with $s \langle S_l^{\mathrm{ADV}} \langle s'$. Otherwise, during the construction of M when we are about to define $M(j)$, we would set $M(j) = j$.

Since $f(j)$ is undefined, by condition (f1), l must be a preemption candidate for j, that is $s\langle r_l \leq x_l \langle s'$. Furthermore, as TIGHTRESTART does not preempt j when l is released, the set $P^*_{r_l}$ is not flexible.

Figure 6.3 illustrates the argument that follows. The idea is this: Since j is not preempted even though a preemption candidate l arrives, $\mathcal{A}$ must be nearly full between s' and d_j. So, intuitively, one of the jobs scheduled in this interval should overlap in time with the occurrence of j in ADV, and this job would end up being matched to j. The rigorous argument gets a bit technical because of possible gaps in the schedules.

Let $K = \{j' \mid s\langle r_{j'} \langle s' \leq x_{j'}\}$ be the set of all jobs released during the execution of j in $\mathcal{A}$, excluding preemption candidates. Since $f(j)$ is undefined, by condition (f2), all these jobs are completed in ADV, and obviously they cannot be completed before S_l^{ADV}. Thus K is feasible at C_l^{ADV} and also at $s' \leq C_l^{\mathrm{ADV}}$.

Let $U = \{j' \in P_{s'} \mid r_{j'} \langle s'\}$ be the set of jobs pending at s' that are released strictly before s'. Since $\mathcal{A}$ is an EDF schedule and j is flexible (because j is free), all jobs $j' \in P_s - \{j\}$ have $x_{j'} \geq x_j \geq s'$, so they are still pending at s'. Therefore $U = P_s \cup K - \{j\} = P^*_{r_l} \cup K$.

We claim that U is feasible at s'. Suppose, towards contradiction, that it's not. Since K is feasible at s' and all jobs $j' \in U - K$ have $x_{j'} \geq x_j$, TIGHTRESTART would then execute urgent jobs from s' until at least the time x_j. Thus $\mathcal{A}$ would not be idle at time $S_j^{\mathrm{ADV}} \leq x_j$, contradicting the assumption that j is free. We conclude that U is feasible at s', as claimed.

Now, by the feasibility of K at s' and Lemma 6.4.2(3) and by the flexibility of U at s and Lemma 6.4.2(2), $\mathcal{A}$ completes all jobs in U. Furthermore, all jobs in U are scheduled between s' and S_j^{ADV}, as TIGHTRESTART is idle at S_j^{ADV}.

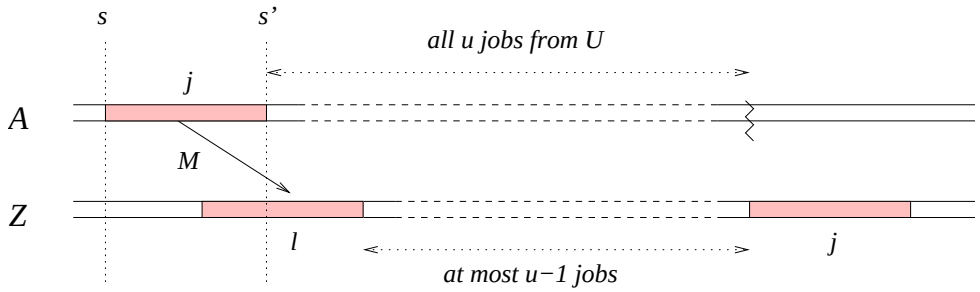


Figure 6.3: Illustration for the proof of Lemma 6.5.3(2).

Let $u = |U|$. Next we claim that
(i) $S_j^{\mathrm{ADV}} - C_l^{\mathrm{ADV}} \langle up$, and
(ii) ADV does not schedule any of the jobs in U after j.

If either of these claims is violated, $U \cup \{j\}$ would be feasible at C_l^{ADV}, for we can first schedule K, which is feasible at C_l^{ADV}, and then the remaining jobs from U. Indeed, if (i) is violated, we can complete all jobs in U by the time S_j^{ADV}, which is smaller than the deadlines in $P_{r_l}^* - K$, and start j at S_j^{ADV}. If (ii) is violated, let j' be the job in U scheduled after j in ADV. We know that $S_j^{\mathrm{ADV}} - C_l^{\mathrm{ADV}} \rangle S_j^{\mathrm{ADV}} - s' - p \geq (u-1)p$, thus we can complete all jobs in U by the time S_j^{ADV} and schedule j and j' as in ADV.

By the previous paragraph, if any of (i) or (ii) does not hold, then $P_{r_l}^* \cup \{j\} \subseteq U \cup \{j\}$ is feasible at $r_l + p \leq C_l^{\mathrm{ADV}}$, contradicting the assumption that l (which is a preemption candidate) did not cause preemption. We thus obtain that (i) and (ii) are true, as claimed.

Summarizing, $\mathcal{A}$ completes the u jobs in U between s' and S_j^{ADV}, and, by (ii), these jobs are not executed after S_j^{ADV} in ADV. Therefore, if j were not in the range of M, the jobs in U would have to be matched to the jobs in ADV between C_l^{ADV} and S_j^{ADV}, which is not possible, because there are at most $u-1$ such jobs, by (i). We can thus conclude that j is indeed in the range of M. □

Charging scheme. Let j be a job started at time $t = S_j^{\mathrm{ADV}}$ in ADV. We charge j to jobs in $\mathcal{A}$ according to the following cases.

Case (I): $j = M(k)$ for some k. Charge j to k. By Lemma 6.5.3(1), this case always applies when $\mathcal{A}$ is not idle at t, so in the remaining cases $\mathcal{A}$ is idle at t.

Case (II): Otherwise, if j is free, then charge $\frac{1}{2}$ of j to the occurrence of j in $\mathcal{A}$ and $\frac{1}{2}$ of j to the occurrence of $f(j)$ in $\mathcal{A}$. Note that, since (I) does not apply, Lemma 6.5.3(2) implies that $f(j)$ is defined, and then Lemma 6.4.2 implies that both j and $f(j)$ are completed in $\mathcal{A}$.

Case (III): Otherwise, $\mathcal{A}$ is idle at t, but TIGHTRESTART is running some job l at t which is later preempted by another job l'. Charge $\frac{1}{2}$ of j to j and $\frac{1}{2}$ to l'. By Lemma 6.4.2(2), j is completed in $\mathcal{A}$. Job l' is urgent and thus it is completed as well.

Analysis. We prove that each job scheduled in $\mathcal{A}$ is charged at most $\frac{3}{2}$. Each job is charged at most 1 in case (I), as M defines a matching.

We claim that each job receives at most one charge of $\frac{1}{2}$. For the rest of the proof, we will distinguish two types of charges of $\frac{1}{2}$: *self–charges*, when j is charged to itself, and *non–self–charges*, when j is charged to a different job.

Suppose first that k receives a self–charge. (Obviously, it can receive only one.) By the charging scheme, $\mathcal{A}$ is idle at time S_k^{ADV}. This implies two things. First, k is not tight, so it cannot receive a non–self–charge in Case (III). Second, k cannot be in the range of $f(\cdot)$, since each job $f(j)$ is either not in ADV or, if it is, $\mathcal{A}$ is executing some job at time $S_{f(j)}^{\mathrm{ADV}}$. Therefore k cannot receive a non–self–charge in Case (II).

Next, suppose that k does not receive a self–charge. Since $f(\cdot)$ is one–to–one, k can receive at most one non–self–charge in Case (II). If k receives a non–self–charge in Case (III) from a job j, then k is started in $\mathcal{A}$ while ADV is executing j, so k can receive only one such charge. Finally, if k receives a non–self–charge in Case (II) then, by the definition of $f(\cdot)$, k is not a preemption candidate, so it cannot receive a non–self–charge in Case (III).

We conclude that each job completed in $\mathcal{A}$ gets at most one charge of 1 and at most one charge of $\frac{1}{2}$, and thus is charged a total of at most $\frac{3}{2}$. Each job in ADV generates a charge of 1. Thus, by summation over all jobs in ADV, we have $|\mathrm{ADV}| \leq \frac{3}{2}|\mathcal{A}|$, completing the proof of the theorem. □

We now show that the competitive ratio of our algorithm is in fact optimal.

Theorem 6.5.4 *Let us consider the problem of online scheduling of equal–length instances of jobs with allowed restarts, in the deterministic or randomized model with the standard profit model.*

Then there is no deterministic algorithm solving this problem better than $\frac{3}{2}$–competitive. There is no randomized algorithm solving this problem better than $\frac{6}{5}$–competitive.

Proof: For $p \geq 2$, consider four jobs given in the form $j = (r_j, d_j)$: $1 = (0, 3p+1)$, $2 = (1, 3p)$, $3 = (p, 2p)$, $4 = (p+1, 2p+1)$. The instance consists of jobs $1, 2, 3$ or $1, 2, 4$.

There exist schedules that schedule three jobs $1, 3, 2$ or three jobs $2, 4, 1$, in this order. (See Figure 6.4.) Therefore the optimal solution consists of three jobs.

In the deterministic case, release jobs 1 and 2. If at time 1, job 2 is started in the online algorithm, release job 3, otherwise release job 4. The online algorithm completes only two jobs. As the optimal schedule has three jobs, the competitive ratio is no better than $\frac{3}{2}$.

Our proof for randomized algorithms is based on Yao's principle [61, 15]. We define a probability distribution on our two instances, as follows: Always release jobs 1 and 2, and then a randomly chosen one job from 3

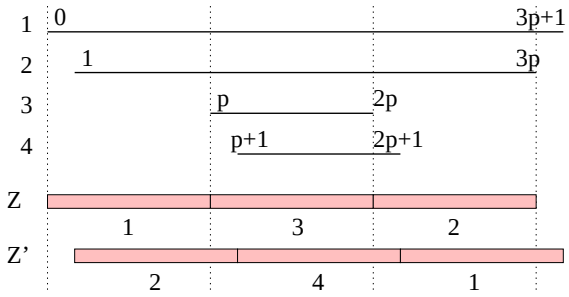


Figure 6.4: Jobs used in the lower bounds with restarts.

and 4, each with probability $\frac{1}{2}$. If $\mathcal{A}$ is any deterministic online algorithm, then the expected number of jobs completed by $\mathcal{A}$ is at most 2.5, as on one of the instances it completes only 2 jobs. Using Yao's principle, we conclude that no randomized algorithm can have competitive ratio smaller than $3/2.5 = \frac{6}{5}$. □

6.6 Barely random model

First, addressing an open question in [29, 30], we give a $\frac{5}{3}$–competitive randomized algorithm. Interestingly, our algorithm is *barely random*; it chooses with probability $\frac{1}{2}$ one of two deterministic algorithms, i.e., it uses only one random bit. These two algorithms are two *identical* copies of the same deterministic algorithm, that are run concurrently and use a shared lock to break the symmetry and coordinate their behaviors. We are not aware of previous work in the design of randomized online algorithms that uses such mechanism to coordinate identical algorithms—thus this technique may be of its own, independent interest.

In this section we present $\frac{5}{3}$–competitive barely random algorithm. This algorithm uses only one random bit; namely, at the beginning of computation it chooses with probability $\frac{1}{2}$ between two deterministic algorithms. We also show a lower bound for barely random algorithms: any randomized algorithm that randomly chooses between two schedules has ratio at least $\frac{3}{2}$. If these schedules (algorithms) are chosen with probability $\frac{1}{2}$ each, we improve the lower bound to $\frac{8}{5}$.

Algorithm RANDLOCK**.** We describe our algorithm in terms of two identical processes that are denoted by $\mathcal{X}$ and $\mathcal{Y}$. Each process is, in essence, a scheduling algorithm that receives its own copy of the input instance J and computes its own schedule for J. (This means that a given job can

be executed by both processes, at the same or different times.) We chose to use the term "process" rather than "algorithm", since $\mathcal{X}$ and $\mathcal{Y}$ are not fully independent; they both have access to a shared lock mechanism used to coordinate their behavior.

Each process $\mathcal{X}$ and $\mathcal{Y}$ is defined as follows:

(RL1) If there are no pending jobs, wait until some job is released.

(RL2) If the set of pending jobs is not flexible, execute the ED pending job.

(RL3) If the set of pending jobs is flexible and the lock is available, acquire the lock (ties broken arbitrarily), execute the ED pending job, and release the lock upon its completion.

(RL4) Otherwise, wait until the lock becomes available or the set of pending jobs becomes non–flexible (due to progress of time or new jobs being released).

Algorithm RANDLOCK selects initially one of the two processes $\mathcal{X}$ or $\mathcal{Y}$, each with probability $\frac{1}{2}$. Then it simulates the two processes on a given instance, outputting the schedule generated by the selected process.

Before we analyze the algorithm, we illustrate its behavior on the instance in Figure 6.5. Both processes schedule only three jobs, while the optimal schedule has five jobs. Thus RANDLOCK is not better than $\frac{5}{3}$–competitive.

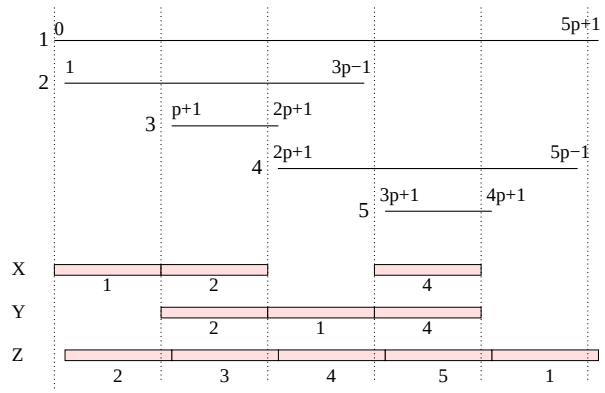


Figure 6.5: An instance on which RANDLOCK schedules three jobs out of five. Job 1 is executed as flexible by both processes; the other jobs are executed as urgent.

Theorem 6.6.1 *Let us consider the problem of online scheduling of equal–length instances of jobs in the randomized model with the standard profit model. Above we described algorithm* RANDLOCK.

Then the algorithm RANDLOCK *solves this problem and it is* $\frac{5}{3}$*–competitive.*

Proof: Overloading the notation, let $\mathcal{X}$ and $\mathcal{Y}$ denote the schedules generated by the corresponding processes on a given instance J. By rules (RL2), (RL3), both schedules are normal. Fix an arbitrary schedule ADV for the given instance J.

We start by modifying the schedules $\mathcal{X}$ and $\mathcal{Y}$ according to Lemma 6.4.2. Define a partial function $f^{\mathcal{A}} : J \to J$ as follows. Let $f^{\mathcal{A}}(k) = h$ if k is a flexible job completed in $\mathcal{X}$ and h is a job started in ADV during the execution of k in $\mathcal{X}$ and admissible at the completion of k in $\mathcal{X}$, i.e., $S_k^{\mathcal{X}} \leq S_h^{\mathrm{ADV}} < C_k^{\mathcal{X}} \leq x_h$. Otherwise (if k is urgent or no such h exists), $f^{\mathcal{A}}(k)$ is undefined. Note that if h exists, it is unique for a given k. Then we define $\mathcal{A}$ to be the schedule constructed from $\mathcal{X}$ in Lemma 6.4.2 using function $f^{\mathcal{A}}(\cdot)$. Analogously we define function $f^{\mathcal{B}}(\cdot)$, and we modify schedule $\mathcal{Y}$ to obtain schedule $\mathcal{B}$. We stress that these new schedules $\mathcal{A}$ and $\mathcal{B}$ cannot be constructed online as their definition depends on ADV; they only serve as tools for the analysis of RANDLOCK.

Since $\mathcal{A}$ (resp. $\mathcal{B}$) is equivalent to a normal schedule $\mathcal{X}$ (resp. $\mathcal{Y}$), Lemma 6.4.1 still applies to $\mathcal{A}$ (resp. $\mathcal{B}$) and the number of completed jobs remains the same as well.

Throughout the proof we use the convention that whenever $\mathcal{D}$ denotes one of the schedules $\mathcal{A}$ and $\mathcal{B}$, then $\bar{\mathcal{D}}$ denotes the other one.

Lemma 6.6.2 *Let $\mathcal{D} \in \{\mathcal{A}, \mathcal{B}\}$, and let $\bar{\mathcal{D}}$ be the other process of* RANDLOCK. *Suppose that at time t $\mathcal{D}$ is idle or is executing an urgent job and $\bar{\mathcal{D}}$ is idle. Then each job admissible at time t is completed in $\bar{\mathcal{D}}$ as a flexible job by time t.*

Proof: The lemma is a direct consequence of the lock mechanism. By the assumption, the lock is available at time t, yet the process corresponding to $\bar{\mathcal{D}}$ does not schedule any job. This is possible only if no job is pending. Consequently, any job k admissible at time t must have been completed in $\bar{\mathcal{D}}$ by time t. Furthermore, if k would be executed as urgent in $\mathcal{D}$ before time t then, since $S_k^{\mathcal{D}} \leq t \leq x_k$, Lemma 6.4.1 implies that $\mathcal{D}$ could not be idle at time t. This shows that k is completed as a flexible job. □

The charging scheme Our proof is based on a charging scheme. Each adversary job will generate a charge of 1. This charge will be distributed among the jobs in schedules $\mathcal{A}$ and $\mathcal{B}$, in such a way that each job in these schedules will receive a charge of at most $5/6$. This will imply the $5/3$ bound on the competitive ratio of RANDLOCK.

Let j be a job started in ADV at time $t = S_j^{\mathrm{ADV}}$. This job generates several charges of different weights to (the occurrences of) the jobs in schedules $\mathcal{A}$ and $\mathcal{B}$. Each charge is uniquely labelled as a *self–charge* or an *up–charge*.

Self–charges from j go to the occurrences of j in $\mathcal{A}$ or $\mathcal{B}$, and *up–charges* from j go to the jobs running at time t in $\mathcal{A}$ and $\mathcal{B}$. If one of the processes runs j at time t, then the charge to this job may be designated either as an up–charge or a self–charge; in Case (III) below such a j can even receive both a self–charge and an up–charge from j. The total of charges generated by j is always 1. The charges depend on the status of $\mathcal{A}$ and $\mathcal{B}$ at time t. (See Figure 6.6.)

Case (I): Both schedules $\mathcal{A}$ and $\mathcal{B}$ are idle. By Lemma 6.6.2, in both $\mathcal{A}$ and $\mathcal{B}$, j is completed as flexible by time t. We generate two self–charges of $1/2$ to the two occurrences of j in $\mathcal{A}$ and $\mathcal{B}$.

Case (II): One schedule $\mathcal{D} \in \{\mathcal{A}, \mathcal{B}\}$ is running an urgent job k and the other schedule $\bar{\mathcal{D}}$ is idle. By Lemma 6.6.2, in $\bar{\mathcal{D}}$, j is completed as flexible by time t. We generate a self–charge of $1/2$ to the occurrence of j in $\bar{\mathcal{D}}$ and an up–charge of $1/2$ to k in $\mathcal{D}$.

Case (III): One schedule $\mathcal{D} \in \{\mathcal{A}, \mathcal{B}\}$ is running a flexible job k and the other schedule $\bar{\mathcal{D}}$ is idle. We claim that j is completed in both $\mathcal{A}$ and $\mathcal{B}$. For $\bar{\mathcal{D}}$, this follows directly from Lemma 6.4.2(2). We now prove it for $\mathcal{D}$. If $r_j \leq S_k^{\mathcal{D}}$, then Lemma 6.4.2(2) applied to time $t' = S_k^{\mathcal{D}}$ implies that $\mathcal{D}$ completes j. If $x_j \geq C_k^{\mathcal{D}}$ then $f^{\mathcal{D}}(k) = j$, so $\mathcal{D}$ completes j by Lemma 6.4.2(1). The remaining case, namely $S_k^{\mathcal{D}} < r_j \leq t \leq x_j < C_k^{\mathcal{D}}$ cannot happen, for this condition implies that j is tight and thus the set of jobs pending at time t for $\bar{\mathcal{D}}$ is not flexible. So $\bar{\mathcal{D}}$ would not be idle at t, contradicting the case condition.

In this case we generate one up–charge of $1/3$ to k in $\mathcal{D}$ and two self–charges of $1/2$ and $1/6$ to the occurrences of j according to the two subcases below. Let $\mathcal{E} \in \{\mathcal{A}, \mathcal{B}\}$ be the schedule which starts j first (breaking ties arbitrarily).

Case (IIIa): If $\mathcal{E}$ schedules j as an urgent job and the other schedule $\bar{\mathcal{E}}$ is idle at some time t' satisfying $S_j^{\mathcal{E}} \leq t' \leq x_j$, then charge $1/6$ to the occurrence of j in $\mathcal{E}$ and $1/2$ to the occurrence of j in $\bar{\mathcal{E}}$.

We make here a few observations that will be useful later in the proof. Since in this case j is urgent in $\mathcal{E}$, and $\mathcal{E}$ is either idle or executes a flexible job at time t, Lemma 6.4.1 implies that j is executed in $\mathcal{E}$ after time t. It also implies that $\mathcal{E}$ runs urgent jobs between $S_j^{\mathcal{E}}$ and x_j. This means that $\mathcal{E}$ runs an urgent job at t'. Since $\bar{\mathcal{E}}$ is idle at time t' by the case condition, Lemma 6.6.2 implies that $\bar{\mathcal{E}}$ schedules j as flexible before time t'.

Case (IIIb): Otherwise charge $1/2$ to the occurrence of j in $\mathcal{E}$ and $1/6$ to the occurrence of j in $\bar{\mathcal{E}}$.

Case (IV): Both processes $\mathcal{A}$ and $\mathcal{B}$ are running jobs $k_{\mathcal{A}}$ and $k_{\mathcal{B}}$, respectively, at time t. We show below in Lemma 6.6.4 that in the previous cases one of $k_{\mathcal{A}}$, $k_{\mathcal{B}}$ receives a self–charge of at most $1/6$ from its occurrence in ADV. We generate an up–charge of $2/3$ from j to this job, and an up–charge of $1/3$ to the other one. No self–charge is generated in this case.

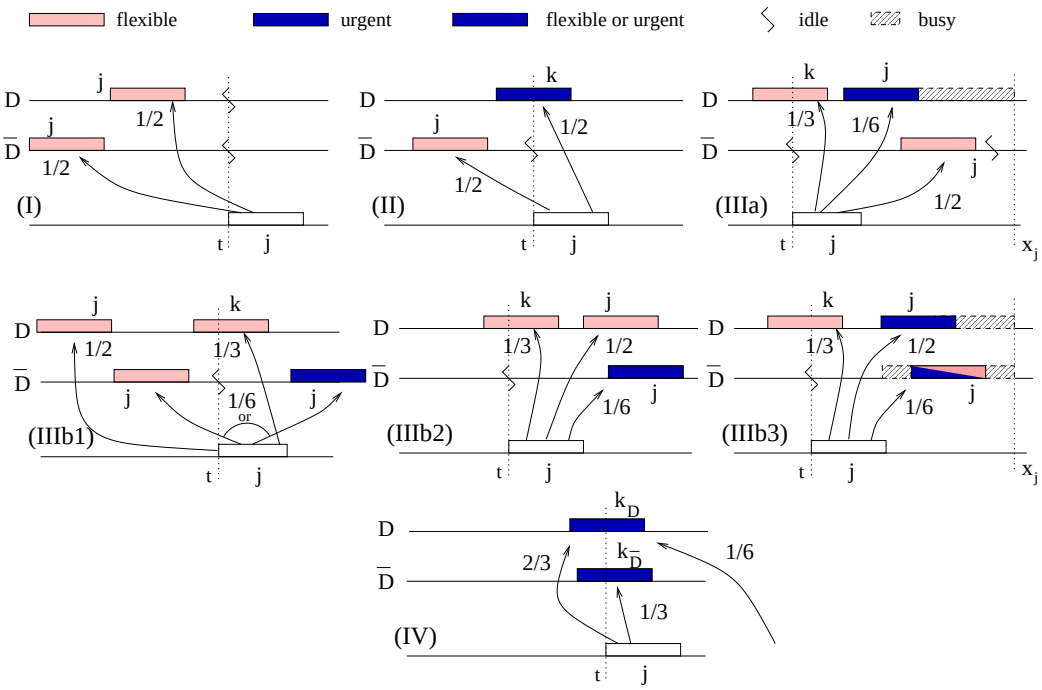


Figure 6.6: Illustration of the charging scheme in the analysis of Algorithm RANDLOCK. The figure gives examples of different types of charges. In Case (IIIb), there are several illustrations that cover possibilities playing a different role in the proof. (To reduce the number of cases, in the figures for Case (III) we assume that $j \neq k$ and j is executed in $\mathcal{D}$ before it is executed in $\bar{\mathcal{D}}$.)

This completes the description of the charging scheme. Before we resume the proof of the theorem, we prove two lemmas, the purpose of which is to justify the correctness of the charges in Case (IV).

Lemma 6.6.3 *Assume that Case (IV) applies to j. Suppose also that $k_{\mathcal{F}}$, for some $\mathcal{F} \in \{\mathcal{A}, \mathcal{B}\}$, is scheduled before j in* ADV *(that is, $S^{\mathrm{ADV}}_{k_{\mathcal{F}}} \leq t - p$), and that $k_{\mathcal{F}}$ in $\mathcal{F}$ receives a self–charge of $1/2$ generated in Case (IIIb) applied to $k_{\mathcal{F}}$. Then $k_{\bar{\mathcal{F}}} \prec k_{\mathcal{F}}$.*

Proof: Since $k_{\mathcal{F}}$ receives a charge of $1/2$ in (IIIb), the choice of $\mathcal{E}$ in Case (III) implies that $k_{\mathcal{F}}$ is executed in $\bar{\mathcal{F}}$ later than in $\mathcal{F}$, that is $S^{\bar{\mathcal{F}}}_{k_{\mathcal{F}}} \geq S^{\mathcal{F}}_{k_{\mathcal{F}}} > t-p$. On the other hand, $S^{\bar{\mathcal{F}}}_{k_{\bar{\mathcal{F}}}} \leq t$, so $k_{\mathcal{F}}$ must be executed in $\bar{\mathcal{F}}$ after $k_{\bar{\mathcal{F}}}$. Furthermore, $S^{\bar{\mathcal{F}}}_{k_{\bar{\mathcal{F}}}} > t - p \geq S^{\mathrm{ADV}}_{k_{\mathcal{F}}} \geq r_{k_{\mathcal{F}}}$ and thus $k_{\mathcal{F}}$ is pending in $\bar{\mathcal{F}}$ when $k_{\bar{\mathcal{F}}}$ is started. Since $\bar{\mathcal{F}}$ is EDF, we have $k_{\bar{\mathcal{F}}} \prec k_{\mathcal{F}}$, completing the proof. □

Lemma 6.6.4 *Assume that Case* (IV) *applies to j. Then for some $\mathcal{D} \in \{\mathcal{A}, \mathcal{B}\}$ the self–charge to $k_{\mathcal{D}}$ in $\mathcal{D}$ does not exceed $1/6$.*

Proof: Note that self–charges are generated only in Cases (I)–(III) and any self–charge has weight $1/2$ or $1/6$. Assume, towards contradiction, that both $k_{\mathcal{A}}$ and $k_{\mathcal{B}}$ receive a self–charge of $1/2$. At least one of $k_{\mathcal{A}}$ and $k_{\mathcal{B}}$ is scheduled as urgent in the corresponding schedule, due to the lock mechanism. Thus $k_{\mathcal{A}} \neq k_{\mathcal{B}}$, as (I) is the only case when two self–charges $1/2$ to the same job are generated and then both occurrences are flexible. Furthermore, if $j = k_{\mathcal{G}}$, for some $\mathcal{G} \in \{\mathcal{A}, \mathcal{B}\}$, then $k_{\mathcal{G}}$ would not receive any self–charge. Thus $k_{\mathcal{A}}$, $k_{\mathcal{B}}$, and j are three distinct jobs.

Choose $\mathcal{D}$ such that $k_{\mathcal{D}}$ is urgent in $\mathcal{D}$ (as noted above, such $\mathcal{D}$ exists). The only case when an urgent job receives a self–charge of $1/2$ is (IIIb). By Lemma 6.4.1, $\mathcal{D}$ executes urgent jobs at all times t', $t \leq t' \leq x_{k_{\mathcal{D}}}$, which, together with the condition for Case (III) applied to $k_{\mathcal{D}}$ (namely that $\mathcal{D}$ is either idle or executes a flexible job at $S^{\mathrm{ADV}}_{k_{\mathcal{D}}}$), implies that $S^{\mathrm{ADV}}_{k_{\mathcal{D}}} \leq t$. As $j \neq k_{\mathcal{D}}$, it follows that $S^{\mathrm{ADV}}_{k_{\mathcal{D}}} \leq t - p$. By Lemma 6.6.3, $k_{\bar{\mathcal{D}}} \prec k_{\mathcal{D}}$ and $x_{k_{\bar{\mathcal{D}}}} \leq x_{k_{\mathcal{D}}}$. Furthermore, since (IIIa) does not apply to $k_{\mathcal{D}}$, $\bar{\mathcal{D}}$ is also not idle at any time t', $t \leq t' \leq x_{k_{\mathcal{D}}}$.

We now show that the assumption of a self–charge of $1/2$ to $k_{\bar{\mathcal{D}}}$ in $\bar{\mathcal{D}}$ leads to a contradiction. The proof is by considering several cases. In most cases the contradiction is with the fact that, as shown in the previous paragraph, both processes are busy at all times between t and $x_{k_{\mathcal{D}}}$. (Keeping in mind that $x_{k_{\bar{\mathcal{D}}}} \leq x_{k_{\mathcal{D}}}$.)

If this charge is generated in Case (I) or (II) then, by the case conditions, $\bar{\mathcal{D}}$ would be idle at time $S^{\mathrm{ADV}}_{k_{\bar{\mathcal{D}}}}$, and we would have $S^{\mathrm{ADV}}_{k_{\bar{\mathcal{D}}}} \geq S^{\bar{\mathcal{D}}}_{k_{\bar{\mathcal{D}}}}$, and thus $t \leq S^{\mathrm{ADV}}_{k_{\bar{\mathcal{D}}}} \leq x_{k_{\bar{\mathcal{D}}}}$, which is a contradiction.

Suppose that this self–charge is generated in Case (III). Similarly as before, the condition of this case implies that one process is idle at time

$S^{\mathrm{ADV}}_{k_{\bar{\mathcal{D}}}}$, so we must have $S^{\mathrm{ADV}}_{k_{\bar{\mathcal{D}}}} \leq t$, for otherwise we would have again an idle time between t and $x_{k_{\bar{\mathcal{D}}}}$.

We have now two subcases. If the self–charge originated from Case (IIIa), the condition of this case implies that there is an idle time t' between $S^{\mathrm{ADV}}_{k_{\bar{\mathcal{D}}}}$ and $x_{k_{\bar{\mathcal{D}}}}$. As $t \leq t' \leq x_{k_{\bar{\mathcal{D}}}}$, this is again a contradiction.

The last possibility is that this self–charge originated from Case (IIIb). But then $S^{\mathrm{ADV}}_{k_{\bar{\mathcal{D}}}} \leq t - p$, as $j \neq k_{\bar{\mathcal{D}}}$, and Lemma 6.6.3 above applies to $k_{\bar{\mathcal{D}}}$. However, the conclusion that $k_{\mathcal{D}} \prec k_{\bar{\mathcal{D}}}$ contradicts the linearity of $\prec$ as $k_{\mathcal{D}} \neq k_{\bar{\mathcal{D}}}$ and we have already shown that $k_{\bar{\mathcal{D}}} \prec k_{\mathcal{D}}$.

Summarizing, we get a contradiction in all the cases, completing the proof of the lemma. □

Continuing the proof of the theorem, we now show that the total charge to each occurrence of a job in $\mathcal{A}$ or $\mathcal{B}$ is at most $5/6$. Suppose that k is executed in $\mathcal{D} \in \{\mathcal{A}, \mathcal{B}\}$. During the execution of k at most one job is started in ADV, thus k gets at most one up–charge in addition to a possible self–charge. If k does not receive any up–charge, it is self–charged $1/2$ or $1/6$, i.e., less than $5/6$.

If k receives an up–charge in (II), then k is an urgent job and, since $\bar{\mathcal{D}}$ is idle, it is already completed in $\bar{\mathcal{D}}$, so $S^{\bar{\mathcal{D}}}_k < S^{\mathcal{D}}_k$. The only case where the occurrence of k that is later in time is urgent and receives a self–charge is Case (IIIb), and in this case this self–charge is $1/6$. So the total charge would be at most $1/6 + 1/2 < 5/6$.

If a job receives an up–charge in (III), the up–charge is only $1/3$ and thus the total is at most $1/3 + 1/2 = 5/6$.

If a job receives an up–charge in (IV), Lemma 6.6.4 implies that the up–charges can be defined as claimed in the case description. The total charge is then bounded by $1/6 + 2/3 = 5/6$ and $1/2 + 1/3 = 5/6$, respectively.

The expected number of jobs completed by RANDLOCK is $(|\mathcal{A}| + |\mathcal{B}|)/2$. Since each job in $\mathcal{A}$ and $\mathcal{B}$ receives a charge of at most $5/6$, and all jobs in ADV generate a charge of 1, we have $(5/3)\cdot(|\mathcal{A}|+|\mathcal{B}|)/2 = (5/6)\cdot(|\mathcal{A}|+|\mathcal{B}|) \geq |\mathrm{ADV}|$. This implies that RANDLOCK is $5/3$–competitive. □

As discussed in the introduction, a lower bound of $4/3$ is known for randomized algorithms [29]. For barely random algorithms that choose between two deterministic algorithms, we can improve this bound to $3/2$. Assuming also that the two algorithms are selected with equal probability, we can further improve the bound to $8/5$.

Theorem 6.6.5 *Let us consider the problem of online scheduling of equal–length instances of jobs in the barely–randomized model with the standard profit model.*

Suppose that $\mathcal{R}$ s a barely–random algorithm randomly chooses one of two deterministic algorithms.

Then the algorithm $\mathcal{R}$ is not better than $\frac{3}{2}$–competitive.

Proof: Assume that $\mathcal{R}$ chooses randomly one of two deterministic algorithms, $\mathcal{A}$ and $\mathcal{B}$, with some arbitrary probabilities. Let $p \geq 3$ and write the jobs as $j = (r_j, d_j)$. We start with job $1 = (0, 4p)$. Let t be the first time when one of the algorithms, say $\mathcal{A}$, schedules job 1. If $\mathcal{B}$ schedules it at t as well, release a job $1' = (t+1, t+p+1)$; the optimum schedules both jobs while both $\mathcal{A}$ and $\mathcal{B}$ only one, so the competitive ratio is at least 2.

So we may assume that $\mathcal{B}$ is idle at t. Release job $2 = (t+1, t+2p+2)$. If $\mathcal{B}$ starts any job (1 or 2) at $t+1$, release job $3 = (t+2, t+p+2)$, otherwise release job $4 = (t+p+1, t+2p+1)$. $\mathcal{B}$ completes only one of the jobs 2, 3, 4. Since $\mathcal{A}$ is busy with job 1 until time $t+p$, it also completes only one of the jobs 2, 3, 4, as their deadlines are smaller than $t+3p$. So each of $\mathcal{A}$ and $\mathcal{B}$ completes at most two jobs.

The optimal schedule completes three jobs: If 3 is issued, schedule 3 and 2, back to back, starting at time $t+2$. If 4 is issued, schedule 2 and 4, back to back, starting at time $t+1$. In either case, two of jobs $2, 3, 4$ fit in the interval $[t+1, t+2p+2)$. If $t \geq p-1$, schedule job 1 at time 0, otherwise schedule job 1 at time $3p \geq t+2p+2$. Thus the competitive ratio of $\mathcal{R}$ is at least $3/2$. □

Theorem 6.6.6 *Let us consider the problem of online scheduling of equal–length instances of jobs in the barely–randomized model with the standard profit model. Suppose that $\mathcal{R}$ s a barely–random algorithm randomly chooses one of two deterministic algorithms each with probability $\frac{1}{2}$.*

Then the algorithm $\mathcal{R}$ is not better than $\frac{8}{5}$–competitive.

Proof: Assume that $\mathcal{R}$ chooses one of two deterministic algorithms, $\mathcal{A}$ and $\mathcal{B}$, each with probability $1/2$. Let $p \geq 3$ and write the jobs in the format $j = (r_j, d_j)$. We start with job $1 = (0, 6p)$. Let t be the first time when one of the algorithms, say $\mathcal{A}$, schedules job 1.

At time $t+1$ release job $2 = (t+1, t+p+1)$. If $\mathcal{B}$ does not start 2 at time $t+1$, then no more jobs will be released and the ratio is at least 2.

We may thus assume that $\mathcal{B}$ starts 2 at time $t+1$ and then starts 1 at some time $t' \geq t+p+1$. Release job $3 = (t'+1, t'+2p+2)$. If $\mathcal{A}$ starts job 3 at $t'+1$, release job $4 = (t'+2, t'+p+2)$, otherwise release $5 = (t'+p+1, t'+2p+1)$. By the choice of the last job, $\mathcal{A}$ can complete only one of the jobs $3, 4, 5$. Since $\mathcal{B}$ is busy with job 1 until time $t'+p \geq t'+3$, it also can complete only one of the jobs $3, 4, 5$, as their deadlines are strictly smaller than $t'+3p$. So $\mathcal{A}$ can complete 2 jobs only and $\mathcal{B}$ can complete 3 jobs.

The optimal schedule can complete all four released jobs. If 4 is issued, schedule $4, 3$, back to back, starting at time $t' + 2$. If 5 is issued, schedule $3, 5$, back to back, starting at time $t' + 1$. In either case, both jobs fit in the interval $[t' + 1, t' + 2p + 2)$. This interval is disjoint with the interval $[t + 1, t + p + 1)$ where 2 is scheduled. Finally, these two intervals occupy length $3p + 1$ of the interval $[0, 6p)$ and divide it into at most 3 contiguous pieces; thus one of the remaining pieces has length at least p and job 1 can be scheduled.

Summarizing, $\mathcal{R}$ completes at most $(2 + 3)/2 = 2.5$ jobs on average, while the optimal schedule completes 4 jobs. Therefore the competitive ratio is at least $4/2.5 = 8/5$, as claimed. □

Chapter 7

Conclusions

This thesis is based on our research interests in the area of approximation and online algorithms. We focused our effort on research of various online scheduling problems which is a stand–alone part of the considered area.

Because of the nature of methods of the competitive analysis the presented results concering the online scheduling problems are mostly lower bounds and upper bounds. Thus some of results improve previously known results and some of them are new results for studied restrictions of the scheduling problems. The improvement means that we significantly decreased the gap between the lower bound and upper bound of a studied problem or we matched lower bound and upper bound for the problem. As it is usual in this area the competition with the other researchers in the exact competitive ratios for the studied problems remains open until the upper bound and the lower bound are matched.

When we are talking about results concering closing the gap between the lower and upper bounds for the online scheduling problems then we developed two upper bounds and three lower bounds. We also developed two interesting negative results for the online scheduling problems with the resource augmentation.

All of the presented results with the exception of two results related to resource augmentation (one negative result and one lower bound) were already published or accepted in serious scientific journals.

Thus it remains to conclude our goals which we defined for this thesis have been satisfied and our research in the studied area was successful because interesting results were developed.

Bibliography

[1] S. Albers: *Better Bounds for Online Scheduling.* Proceedings of the 29th Annual ACM Symposium on the Theory of Computing, 130–139, 1997.

[2] S. Albers: *On Randomized Online Scheduling.* Proceedings of 34th Symposium on Theory of Computing (STOC), pages 134–143. ACM, 2002.

[3] S. Albers, M. Schmidt: *On the Performance of Greedy Algorithms in Packet Buffering..* SIAM, Journal of Compututing 35(2): 278–304, 2005.

[4] N. Andelman, Y. Mansour, A. Zhu: *Competitive Queuing Polices in QoS Switches.* Proceedings of 14th Symposium on Discrete Algorithms (SODA), pages 761–770. ACM/SIAM, 2003.

[5] B. Hajek: *On the competitiveness of online scheduling of unit–length packets with hard deadlines in slotted time.* Conference in Information Sciences and Systems, pages 434–438, 2001.

[6] N. Bansal, L. Fleischer, T. Kimbrel, M. Mahdian, B. Schieber, M. Sviridenko: *Further Improvements in Competitive Guarantees for QoS Buffering.* Proceedings of 31st International Colloquium on Automata, Languages and Programming (ICALP), volume 3142 of Lecture Notes in Computer Science, pages 196–207. Springer 2004.

[7] P. Baptiste: *Polynomial–Time Algorithms for Minimizing the Weighted Number of Late Jobs on a Single Machine with Equal Processing Times.* Journal of Scheduling, 2:245–252, 1999.

[8] Y. Bartal, F. Y. L. Chin, M. Chrobak, S. P. Y. Fung, W. Jawor, R. Lavi, J. Sgall, T. Tichý: *Online Competitive Algorithms for Maximizing Weighted Throughput of Unit Jobs.* Proceedings of the 21st Annual Symposium on Theoretical Aspects of Computer Science (STACS). Berlin, Springer–Verlag, pages 187–198, 2004.

[9] Y. Bartal, M. Chrobak, L. L. Larmore: *A Randomized Algorithm for Two Servers on the Line.* Information and Computation, 158:53–69, 2000.

[10] Y. Bartal, A. Fiat, H. Karloff, R. Vohra: *New Algorithms for an Ancient Scheduling Problem.* Proceedings of the 24th Annual ACM Symposium on the Theory of Computing, 51–58, 1992.

[11] Y. Bartal, H. Karloff, Y. Rabani: *A Better Lowerbound for Online Scheduling.* Information Processing Letters 50,3:113–116, 1994.

[12] S. K. Baruah, J. Haritsa, N. Sharma: *Online Scheduling to Maximize Task Completions.* Journal of Combinatorial Mathematics and Combinatorial Computing, 39:65–78, 2001.

[13] S. K. Baruah, J. Haritsa, N. Sharma: *Online Scheduling to Maximize Task Completions.* Proceedings of Real–Time Systems Symposium (RTSS), pages 228–236, 1994.

[14] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, F. Wang: *On the competitiveness of on–line real–time task scheduling..* Real–Time Systems, 4:125—144, 1992.

[15] A. Borodin, R. El–Yaniv: *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[16] J. Carlier: *Problèmes D'ordonnancement à Durées Égales.* QUESTIO, 5(4):219–228, 1981.

[17] E. C. Chang, C. Yap: *Competitive online scheduling with level of service.* Proceedings of 7th Annual International Computing and Combinatorics Conference, volume 2108 of Lecture Notes in Compoter Science, pages 453—462, 2001.

[18] B. Chen, A. van Vliet, G. Woeginger: *A Better Lowerbound for Randomized Online Scheduling Algorithms.* Information Processing Letters 51,5:219–222, 1994.

[19] B. Chen, A. van Vliet, G. Woeginger: *New Lower and Upper Bounds for Online Scheduling.* Operations Research Letters 16,4,221–230, 1994.

[20] F. Y. L. Chin, S. P. Y. Fung: *Online Scheduling for Partial Job Values: Does Timesharing or Randomization Help?.* Algorithmica, 37:149–164, 2003.

[21] M. Chrobak, C. Dürr, W. Jawor, L. Kowalik, M. Kurowski: *A Note on Scheduling Equal–Length Jobs to Maximize Throughput.* Manuscript, 2004.

[22] M. Chrobak, W. Jawor, J. Sgall, T. Tichý: *Online Scheduling of Equal–Length Jobs: Randomization and Restarts Help.* Proceedings of 31st International Colloquium on Automata, Languages, and Programming (ICALP), volume 3142 of Lecture Notes in Computer Science, pages 358–370. Springer, 2004.

[23] M. Englert, M. Westerman: *Considering suppressed packets improves buffer management in QoS switches..* Proceedings of 18th Symposium on Discrete Algorithms (SODA), ACM/SIAM, 209–218, 2007.

[24] M. Englert, M. Westermann: *Considering Suppressed Packets Improves Buffer Management in QoS Switches.* Proceedings of 18th Symposium on Discrete Algorithms (SODA), ACM/SIAM, 2007.

[25] L. Epstein, J. Noga, S. S. Seiden, J. Sgall, G. J. Woeginger: *Randomized Online Scheduling for Two Related Machines.* Journal of Scheduling, 4:71–92, 2001.

[26] L. Epstein, J. Sgall: *A Lower Bound for Online Scheduling on Uniformly Related Machines.* 1999.

[27] A. Fiat, G. J. Woeginger: *Online Algorithms—The State of the Art.* Springer, Berlin 1998.

[28] M. Garey, D. Johnson, B. Simons, R. Tarjan: *Scheduling Unit–Time Tasks with Arbitrary Release Times and Deadlines.* SIAM, Journal of Computing, 10(2):256–269, 1981.

[29] S. A. Goldman, J. Parwatikar, S. Suri: *Online Scheduling with Hard Deadlines.* Journal of Algorithms, 34:370–389, 2000.

[30] M. H. Goldwasser: *Patience Is a Virtue: The Effect of Slack on the Competitiveness for for admission control.* Journal of Scheduling, 6:183–211, 2003.

[31] M. H. Goldwasser, B. Kerbikov: *Admission Control with Immediate Notification.* Journal of Scheduling, 6:269–285, 2003.

[32] R. L. Graham: *Bounds for Certain Multiprocessing Anomalies.* Bell System Technical Journal 45,1563–1581, 1966.

[33] L. Grygarova: *Úvod do lineárního programování.* SPN, Praha 1975.

[34] H. Hoogeveen, C. N. Potts, G. J. Woeginger: *Online Scheduling on a Single Machine: Maximizing the Number of Early Jobs.* Operation Research Letters, 27:193–196, 2000.

[35] J. Jackson: *Scheduling a Production Line to Minimize Maximum Tardiness.* Technical Report 43, Management Science Research Project, Univ. of California, Los Angeles, U.S.A., 1955.

[36] D. S. Johnson: *Near–Optimal Bin Packing Algorithms.* PhD. thesis, MIT, Cambridge, MA, 1973.

[37] B. Kalyanasundaram, K. Pruhs: *Speed Is as Powerful as Clairvoyance.* Journal of the ACM, 47(4):214—221, 2000.

[38] A. Kesselman, Z. Lotker, Y. Mansour, B. Patt–Shamir, B. Schieber, M. Sviridenko: *Buffer Overflow Management in QoS Switches.* Proceedings of 33rd Symposium on Theory of Computing (STOC), pages 520–529. ACM, 2001.

[39] A. Kesselman, Z. Lotker, Y. Mansour, B. Patt–Shamir, B. Schieber, M. Sviridenko: *Buffer Overflow Management in QoS Switches.* SIAM, Journal of Computing (SICOMP), pages 33:563–583. 2004.

[40] A. Kesselman, Y. Mansour, R. van Stee: *Improved Competitive Guarantees for QoS Buffering.* Proceedings of 11th Annual European Symposium on Algorithms (ESA), volume 2832 of Lecture Notes in Computer Science, pages 361–372. Springer 2003.

[41] H. A. Kierstead, W. T. Trotter: *An Extremal Problem in Recursive Combinatorics.* Congressus Numerantium, 33:143–153, 1981.

[42] D. E. Knuth: *The Art of Computer Programming.* Addison–Wesley, prvni vydani, druhy svazek, 1968.

[43] C. K. Koo, T. W. Lam, T. W. Ngan, K. K. To: *Online scheduling with tight deadlines.* Proceedings of 26th Mathematical Foundations of Computer Science, volume 2136 of Lecture Notes in Compoter Science, pages 464—473, 2001.

[44] G. Koren, D. Shasha: d^{over}*: an Optimal On–Line Scheduling Algorithm for Overloaded Uniprocessor Real–Time Systems.* SIAM, Journal on Computing, 24:318—339, 1995.

[45] T. W. Lam, T. W. Ngan, K. K To: *On the speed requirement for optimal deadline scheduling in overloaded systems..* Proceedings of 15th International Parallel and Distributed Processing Symposium, pages 202, 2001.

[46] T. W. Lam, K. K To: *Trade–offs between speed and processor in hard–deadline scheduling.* Proceedings of 10th Symposium on Discrete Algorithms (SODA), pages 755–764, 1999.

[47] F. Li, J. Sethuraman, C. Stein: *Better online buffer management..* Proceedings of 18th Symp. on Discrete Algorithms (SODA), ACM/SIAM, 199–208, 2007.

[48] F. Li, J. Sethuraman, C. Stein: *Better Online Buffer Management.* Proceedings of 18th Symposium on Discrete Algorithms (SODA), ACM/SIAM, 2007.

[49] R. J. Lipton, A. Tomkins: *Online Interval Scheduling.* Proceedings of 5th Symposium on Discrete Algorithms (SODA), pages 302–311. ACM/SIAM, 1994.

[50] M. Nelson, J. L. Gailly: *The Data Compression Book.* M&T Books, New York 1996.

[51] N. Reingold, J. Westbrook, D. D. Sleator: *Randomized Competitive Algorithms for the List Update Problem.* Algorithmica, 11:15–32, 1994.

[52] S. Seiden: *Barely Random Algorithms for Multiprocessor Scheduling.* Journal of Scheduling, 6:309–334, 2003.

[53] S. Seiden: *Randomized Algorithms for that Ancient Scheduling Problem.* Proceedings of the 5th Workshop on Algorithms and Data Structures, 210–223, 1997.

[54] J. Sgall: *A Lower Bound for Randomized Online Multiprocessor Scheduling.* Information Processing Letters 63–1:51–55.

[55] J. Sgall: *Online scheduling..* In *Online Algorithms: The State of Art*, pages 196–227. Springer–Verlag, 1998..

[56] B. Simons: *A Fast Algorithm for Single Processor Scheduling.* Proceedings of 19th Foundations of Computer Science Symposium (FOCS), pages 246–252. IEEE, 1978.

[57] D. D. Sleator, R. E. Tarjan: *Amortized Efficiency of List Update and Paging Rules.* Communications of the ACM, 28:202–208, 1985.

[58] D. D. Sleator, R. E. Tarjan: *Self–Adjusting Binary Search Trees.* Journal of the ACM, 32:652–686, 1985.

[59] D. R. Woodal: *The Bay Restaurant—a Linear Storage Problem.* American Mathematical Monthly, 81:240–246, 1974.

[60] A. C. C. Yao: *New Algorithms for Bin Packing.* Journal of the ACM, 27:207–227, 1980.

[61] A. C. C. Yao: *Probabilistic Computations: Towards a Unified Measure of Complexity.* Proceedings of 18th Foundations of Computer Science Symposium (FOCS), pages 222–227. IEEE, 1977.