Diss. ETH No. 20164

Advice and Randomization in Online Computation

A dissertation submitted to
ETH Zurich

for the degree of
Doctor of Sciences

presented by
Dennis Komm
Dipl.-Inform., RWTH Aachen University
born on May 2, 1982
citizen of Germany

accepted on the recommendation of
Prof. Dr. Juraj Hromkovič, examiner
Prof. Dr. Peter Widmayer, co-examiner
Prof. Dr. Georg Schnitger, co-examiner

2012

Abstract

Online computation is both of theoretical interest and practical relevance as numerous computational problems require a model in which algorithms do not know the whole input at every time step during runtime. The established measurement for the output quality of these *online algorithms* is the so-called *competitive analysis*, introduced by Sleator and Tarjan in 1985. Similar to the decrease in accuracy we have to accept when efficiently (i. e., in polynomial time) solving $\mathcal{NP}$-hard problems, the *competitive ratio* describes what we have to pay for not knowing the future.

In this thesis, we want to measure how much additional information is both necessary and sufficient to escape from this dilemma, i. e., we want to understand what causes this, for the majority of problems, vast amount of precision we lose due to facing an online scenario. More specifically, we want to study the *advice complexity* of online problems, which describes the amount of information online algorithms lack, causing them to fail (compared to hypothetical offline algorithms that know all yet unrevealed parts of the input from the start). In the model used throughout this thesis, we equip online algorithms with an additional *advice tape* onto which an *oracle*, which sees the whole input before the algorithm is executed, may write binary information. The algorithm can then use these *advice bits* during computation. We call the minimum number of advice bits needed to compute an optimal solution for some online problem the *information content* of this problem. This information is what needs to be extracted from the instance in order to overcome the drawback of not completely knowing it in advance. We know that there exist well-studied online problems for which any solution computed by a deterministic online algorithm is (asymptotically) half as good as the optimal solution, because it does not know future input parts. However, a single bit of advice suffices to perform optimally. For many other problems, measuring the information content proves to be a more complicated task. Moreover, generalizing this idea, we study the tradeoff between obtaining high-quality results (i. e., creating online algorithms with a reasonable competitive ratio) and the number of advice bits both necessary and sufficient for this.

We study five online problems within the framework described above, the *job shop scheduling problem* with two jobs and unit-length tasks, the *disjoint path allocation problem*, the *k-server problem*, the *set cover problem*, and the *knapsack problem*. It turns out that online problems may behave very differently in terms of advice complexity. There are, for instance, problems that achieve very good results with a constant number of advice bits and other ones that, if given less advice than linear in the input size, are doomed to fail. Specifically, we ask how many advice bits are necessary and sufficient to (i) be optimal, (ii) to improve over purely deterministic strategies, or (iii) to be on par with (or better than) randomized strategies.

Since we may look at computing with advice as supplying the best possible random string for any input, we are particularly interested in the last point and the further rela-

tion between advice and randomization. Obviously, lower bounds on the advice complexity carry over to randomization and upper bounds on randomization carry over to computing with advice. For some particular problems, we further show how to construct *barely random* algorithms (i. e., randomized algorithms that only use a constant number of random bits) from algorithms with advice. Also, we introduce an online problem for which we study the combination of advice and randomization, showing how every additional advice bit significantly improves the output quality.

Zusammenfassung

Online-Berechnungen sind sowohl von theoretischem Interesse als auch von grossem Nutzen für die Praxis, da viele Berechnungsprobleme eine Analyse erfordern, bei der die gesamte Eingabe dem entsprechenden Algorithmus zur Laufzeit nicht bekannt ist. Um die Güte von diesen *Online-Algorithmen* zu messen, wird meist die so genannte *Competitive Analysis* verwendet. So wie die Approximationsgüte von Algorithmen für $\mathcal{NP}$-schwere Probleme angibt, was wir zahlen, um ein Problem effizient (in polynomieller Laufzeit) zu lösen, zeigt die *Competitive Ratio* auf, welchen Preis wir dafür zahlen, die Zukunft nicht zu kennen.

In dieser Arbeit beschäftigen wir uns mit der Frage, *was* genau dieses Dilemma verursacht, was also der Grund dafür ist, dass wir für eine grosse Anzahl der bekannten Probleme in Kauf nehmen müssen, in Online-Szenarien Ergebnisse sehr schwacher Qualität zu erhalten. Zu diesem Zweck untersuchen wir die *Advice-Komplexität* von Online-Problemen, das heisst die Menge an Informationen, die Online-Algorithmen (verglichen mit hypothetischen Offline-Algorithmen, die die gesamte Eingabe von Anfang an kennen) für die entsprechenden Probleme fehlt, um bessere Ergebnisse zu erzielen. In dem Modell, das wir hier betrachten, verfügen die Online-Algorithmen über ein *Advice-Band*, auf welchem ein *Orakel*, dem die gesamte Eingabe bekannt ist, binäre Informationen über diese schreiben kann. Diese *Advice-Bits* können dann vom jeweiligen Algorithmus während der Laufzeit benutzt werden. Die Anzahl dieser Bits, die notwendig sind, um ein optimales Ergebnis zu berechnen, nennen wir den *Informationsgehalt* des entsprechenden Problems, denn eben genau diese Information über die jeweilige Eingabe muss bekannt sein, um den Nachteil zu überwinden, mit dem Online-Algorithmen konfrontiert werden, weil sie nicht die gesamte Eingabe kennen. Wir kennen Online-Probleme, für die deterministische Algorithmen Lösungen berechnen, die (asymptotisch) doppelt so schlecht sind wie die optimale Lösung. Allerdings reicht bereits ein einziges Bit an Information, um ein optimales Ergebnis zu erzielen. Für viele andere Probleme ist die Analyse jedoch weitaus komplizierter. Ferner interessiert uns eine Verallgemeinerung obiger Frage, nämlich, wie sich der Tradeoff zwischen der sowohl benötigten als auch hinreichenden Anzahl an Advice-Bits und einer zu erreichenden Competitive Ratio verhält.

Wir untersuchen hier fünf Online-Probleme in diesem Modell, das *Job-Shop-Scheduling-Problem* mit zwei Jobs und Aufgaben mit Einheitslänge, das *Disjoint-Path-Allocation-Problem*, das *k-Server*-Problem, das *Set-Cover-Problem* und das *Rucksack-Problem*. Es zeigt sich, dass sich diese Online-Probleme zum Teil völlig unterschiedlich verhalten. Beispielsweise existieren Probleme, die mit einer konstanten Anzahl von Advice-Bits sehr gute Resultate ermöglichen und solche, die weiterhin nur schlechte Ergebnisse zulassen, solange der entsprechende Algorithmus weniger Bits erhält als eine in der Eingabelänge lineare Anzahl. Im Wesentlichen interessiert uns, wie viele Advice-Bits nötig und ausreichend sind, um (i) optimal zu sein, (ii) besser zu sein als deterministische Strategien oder (iii) genau so gut (oder sogar besser) zu sein als randomisierte Algorithmen.

Da wir den Advice als die jeweils beste zufällige Wahl für jede Eingabe betrachten können, ist der letzte Punkt von besonderem Interesse. Es ist offensichtlich, dass untere Schranken für die Advice-Komplexität sich auf randomisierte Verfahren übertragen und umgekehrt, dass obere Schranken für letztere Strategien ebenfalls gültig für Algorithmen mit Advice sind. Für einige konkrete Probleme zeigen wir ferner, wie aus Advice-Algorithmen *Barely-Random*-Algorithmen (also randomisierte Algorithmen, die nur eine konstante Anzahl von Zufallsbits verwenden) konstruiert werden können. Desweiteren stellen wir ein Problem vor, für das wir eine Kombination aus zufallsgesteuerter Berechnung *und* Advice entwerfen, wobei jedes weitere Advice-Bit, das der Algorithmus erhält, die Qualität der Ausgabe wesentlich verbessert.

Acknowledgements

First, I want to deeply thank my advisor Juraj Hromkovič for the support while simultaneously allowing me to explore the vast field of theoretical computer science on my own, never pushing me, but giving helpful advice when needed. On a more personal note, he has probably saved my life in the mountains at least once. Second, I want to thank Hans-Joachim Böckenhauer. During the last three and a half years at ETH, he has supported me whenever possible without ever getting tired of my repeated questions. Moreover, I thank the whole group of *Information Technology and Education* for establishing the nice working atmosphere without any pressure. Everyone was friendly from day one.

I am very grateful to Richard Královič and Tobias Mömke for the numerous inspiring discussions that were not restricted to computer science, and the friendships that evolved from them. It always was a pleasure to do research or have a beer with them (not seldom, both at the same time).

To a great extent, Rastislav Královič established the research field this thesis is devoted to. Also, he made it possible to visit Slovakia and do very productive work together with him and Richard. Thank you. Similarly, Peter Rossmanith, who invited me to Aachen several times, was a great person to work with; he often came up with new ideas when I thought we had hit a dead end.

I want to express my deepest thanks to Marcel Schöngens and Nils Jansen, who, together with Hans-Joachim, Richard, and Tobias, agreed to proofread the manuscript of this thesis, and did so very carefully. I thank Stefan Wolf, who pointed me, as well as Rastislav, to the related work done in cryptography. I am very thankful to both Georg Schnitger and Peter Widmayer for agreeing to review this thesis.

Most importantly, I thank my mom, my dad, my two brothers David and Jan, my grandmother Hildegard, my uncle Karl-Heinz, and my Maria for their support and advice although we do not get together as often as I wish.

"I never think of the future, it comes soon enough."

Albert Einstein

Contents

Chapter 1

Introduction

Many computational problems work in so-called *online environments* as their formulation and analysis demand a model in which an algorithm that deals with such a problem only knows a part of its input at any specific point during runtime. Typical examples include various tasks of operating systems or problems that arise in the area of routing and scheduling. These problems are referred to as *online problems* and the respective algorithms are called *online algorithms* (we formally define these terms in Section 1.3). In such an online setting, an online algorithm, denoted by `A`, has a huge disadvantage compared to offline algorithms, i. e., algorithms that know the whole input at the beginning of their computation. `A` has to make decisions at any time step i without knowing what parts of input will be provided at step $i + 1$ and afterwards. Since `A` has to produce a part of the final output in every step, it cannot revoke decisions it has already made. Furthermore, these decisions can only be made by merely taking input chunks from time steps 1 to i into account and possibly by applying some randomization.

The output quality of such an online algorithm is usually studied by *competitive analysis* [38, 76, 88, 93], introduced by Sleator and Tarjan in 1985 [144]. Informally speaking, here, the performance of `A` is measured by comparing it to that of an optimal offline algorithm, although the existence of the latter one is purely notional.

On the downside, we are not really satisfied with investigating how much worse we perform when not knowing the future. We want to get a deeper understanding of *what* we are actually missing, and thus the central question of this thesis is the following one:

What is the crucial information that an online algorithm really lacks?

As we will see in Section 1.6, there exist well-studied problems for which only one additional bit of information is sufficient to enable the construction of an optimal online algorithm [63, 64]. However, without this single bit, no online algorithm can perform better than twice as bad as the optimal offline solution. It is immediate that this does not hold in general, and thus we are interested in a formal framework that allows us to classify online problems according to how much information about the yet unrevealed input parts is needed for solving them either optimally or with a specific competitive ratio. Considering the second point, we are particularly interested in the best competitive ratios that are known to be achievable with deterministic or randomized strategies without any additional information.

We want to study the amount of information needed for an online algorithm to be optimal or to produce high-quality output by using the *advice complexity*, which was proposed by Dobrev, Královič, and Pardubská in 2008 [63]. To put it informally, this

model can be seen as a cooperation of a deterministic online algorithm A and an *oracle* O, which may passively communicate with A.

In this thesis, the oracle O, which has unlimited computational power, sees the whole input in advance and writes binary information onto an *advice tape* before A starts processing the input. Then, A may access these *advice bits* from the tape in a sequential manner, just as a randomized algorithm would use its random tape (we study the similarities and differences of these two concepts in Chapter 7). The advice complexity of A on an input I is now defined as the number of advice bits A reads while processing this input (a formal definition is given in Section 1.6). We consider the advice complexity as a function of the input size n by taking the maximum value over all inputs of length at most n.

Besides asking for the amount of advice that is necessary and sufficient to compute a high-quality solution, we also deal with the question of whether some small advice might help to achieve a competitive ratio that is significantly better than the one of the best known deterministic or randomized strategy. A large part of this thesis is devoted to the study of some well-known online problems in the framework of advice complexity and to comparing the results to each other. As randomization is often used to build algorithms that produce high-quality output in expectation, we are particularly interested in relating these two concepts.

1.1 This Dissertation

This thesis is organized as follows: In the remaining part of this chapter, we quickly recall some mathematical foundations as well as the basic notations and established knowledge about online computation and, in particular, competitive analysis. Afterwards, we formally introduce the model we want to use throughout this dissertation and survey its history. The subsequent five chapters are devoted to five particular online problems, which are each introduced right before we give our results regarding their advice complexity. Chapter 2 deals with a well-known scheduling problem, the *job shop scheduling problem* with two jobs and unit-length tasks. This is followed by an investigation on a specific routing problem, the *disjoint path allocation problem*, in Chapter 3. Chapter 4 is devoted to a very generic problem, the so-called *k-server problem*, followed by the *online set cover problem* in Chapter 5. The last problem, examined in Chapter 6, is an online version of the *knapsack problem*. In Chapter 7, we want to relate the advice complexity to randomization. First, we observe some general connection between both concepts; next, using some of the techniques we have used to design algorithms with small advice, we build *barely random* algorithms. To combine both approaches, we then introduce the so-called *boxes problem* for which we build online algorithms with advice that also use randomness. We conclude by pointing out further directions to follow, related work, and open questions in Chapter 8.

Most of the results that are presented in this thesis have already been published. (i) The results on the job shop scheduling problem were published in [108, 109]. (ii) The advice complexity of the disjoint path allocation problem was studied in [35]. (iii) The investigations on the k-server problem were published in [34]. (iv) The set cover problem was examined in [106]. (v) The knapsack problem was studied in [32]. (vi) The results about the boxes problem were published in [26]. Technical reports can be found in [31, 33, 36, 107, 110]. Some of the results were obtained together with the co-authors

of the papers just mentioned; theorems and lemmata obtained by these or other authors are stated without any proof while citing the corresponding publication.

1.2 Mathematical Foundations

In this section, we want to fix some notation and briefly comprise some of the mathematical basics we need in the following. For further reading, we point the reader to the standard literature; a very sophisticated overview on combinatorics and discrete mathematics as we use it here and far beyond is given in [80], the concept of randomized computation is found in [88, 128].

As usual, by $\mathbb{N}$ we denote the set of natural numbers including zero, i. e., $\mathbb{N} = \{0, 1, 2, \ldots\}$. The set of integers is denoted by $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$, rational numbers are fractions (*ratios*) of integers, i. e., $\mathbb{Q} = \{a/b \mid a, b \in \mathbb{Z}, b \neq 0\}$. The set of real numbers is denoted by $\mathbb{R}$; elements from $\mathbb{R} \setminus \mathbb{Q}$ are called irrational, because they are characterized by the fact that they are *no ratios*. When restricting ourselves to positive elements, we denote this by, e. g., $\mathbb{Q}^+$ and $\mathbb{R}^+$. For any set A, we denote the *power set* of A by $\mathcal{P}(A)$, formally $\mathcal{P}(A) := \{B \mid B \subseteq A\}$. For every positive real number that is not an integer x, by $\lfloor x \rfloor$ we denote the integer part (the *floor*) of x, and, complementing, by $\lceil x \rceil := \lfloor x \rfloor + 1$ we denote the *ceiling* of x, i. e., the next larger natural number; if x is an integer, however, $\lfloor x \rfloor = \lceil x \rceil = x$. For every natural number n, we have

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2},$$

which was rediscovered by Gauss at the end of the 18th century.

Throughout this thesis, by e we denote the Eulerian number[1] defined as

$$\mathrm{e} := \lim_{n \to \infty} \left(1 + \frac{1}{n}\right)^n = 2.7182\ldots .$$

As is the case for $\pi = 3.1415\ldots$, e is irrational, and we come across it in many applications and phenomena in nature [121]. Our major interest for this constant is motivated by the following inequality.

For any natural number n, we use Stirling's approximation[2] to get both upper and lower bounds on the *factorial* of n, i. e., $n! = 1 \cdot \ldots \cdot n$, obtaining

$$\sqrt{2\pi n}\left(\frac{n}{\mathrm{e}}\right)^n < n! < \sqrt{2\pi n}\left(\frac{n}{\mathrm{e}}\right)^n \left(1 + \frac{1}{11n}\right). \tag{1.1}$$

Note that, for any $n \geq 2$, it directly follows that

$$\left(1 + \frac{1}{11n}\right)\sqrt{2\pi n}\left(\frac{n}{\mathrm{e}}\right)^n \leq 1.05 \cdot \sqrt{2\pi n}\left(\frac{n}{\mathrm{e}}\right)^n, \tag{1.2}$$

which we use to get a simpler upper bound on $n!$ from time to time.

Moreover, for some of the following proofs, we need some basic combinatorics. By

$$\binom{n}{k} := \frac{n!}{k!(n-k)!}$$

[1] Leonhard Euler, ⋆15.04.1707, †18.09.1783, Swiss mathematician.
[2] James Stirling, ⋆05.1692, †05.12.1770, Scottish mathematician.

we denote the number of possibilities to choose a k-element subset from a set of n elements. This number is called the *binomial coefficient*, and we observe that

$$\frac{n!}{k!(n-k)!} = \frac{\prod_{i=1}^{k}(n-i+1)}{k!} \leq \frac{n^k}{k!}. \tag{1.3}$$

For any fixed n, the binomial coefficient is maximal for $k = \lfloor n/2 \rfloor$; we call $\binom{n}{\lfloor n/2 \rfloor}$ the *central binomial coefficient*.

Suppose that n is even, i. e., $n = 2m$ for a natural number m. Using (1.1) and (1.3), we obtain

$$\binom{2m}{m} \leq \frac{(2m)^m}{m!} < \frac{(2m)^m}{\sqrt{2\pi m}\left(\frac{m}{\mathrm{e}}\right)^m} = \frac{\left(\frac{2\mathrm{e}m}{m}\right)^m}{\sqrt{2\pi m}} \leq \frac{6^m}{\sqrt{2\pi m}}.$$

Note that, employing (1.1), a more careful analysis even yields

$$\frac{4^m}{2\sqrt{\pi m}} < \binom{2m}{m} < \frac{4^m}{\sqrt{\pi m}}. \tag{1.4}$$

The central binomial coefficient has an important characteristic: Due to Sperner's theorem[3], for any set S of n elements and for any family $\mathcal{F}$ of subsets of S, the cardinality of $\mathcal{F}$ is at most $\binom{n}{\lfloor n/2 \rfloor}$ given that no member of $\mathcal{F}$ is included in any other member of $\mathcal{F}$ (i. e., $\mathcal{F}$ is *subset-free*) [145].

Consider two positive functions $f(n)$ and $g(n)$ in a variable n. To investigate the asymptotic behavior of these functions, we use the Landau symbols[4] $\mathcal{O}$, o, Ω, ω, Θ [86], e. g.,

$$f(n) \in \mathcal{O}(g(n)) \iff \exists\, n_0, c > 0 \text{ such that } \forall\, n > n_0 \colon f(n) \leq c \cdot g(n).$$

When neglecting logarithmic factors for upper bounds, we use $\widetilde{\mathcal{O}}$ instead of $\mathcal{O}$ (known as the *soft* $\mathcal{O}$ notation).

As already mentioned, for some of our ideas we allow *randomized computation*. Let $\mathcal{E}$ be the finite set of all unique *elementary events* that may occur during computation; subsets of $\mathcal{E}$ are called *events*. For any event $x \in \mathcal{P}(\mathcal{E})$, we denote the probability that x occurs by

$$\mathrm{Prob}(x) = \sum_{z \in x} \mathrm{Prob}(\{z\}).$$

For any elementary event y, we have $\mathrm{Prob}(\{y\}) \geq 0$ and

$$\sum_{y \in \mathcal{E}} \mathrm{Prob}(\{y\}) = 1.$$

A *random variable* X is a function that maps every elementary event to a value (e. g., a real number), i. e.,

$$X \colon \mathcal{E} \to \mathbb{R}.$$

For every $c \in \mathbb{R}$, $\mathrm{Prob}(X = c)$ denotes the probability that X takes the value c. Furthermore, let $\mathbb{R}_X := \{d \in \mathbb{R} \mid \exists\, y \in \mathcal{E} \text{ such that } X(y) = d\}$; we denote the *expected value* of X by

$$\mathbb{E}[X] := \sum_{c \in \mathbb{R}_X} c \cdot \mathrm{Prob}(X = c).$$

[3] Emanuel Sperner, ⋆09.12.1905, †31.01.1980, German mathematician.
[4] Edmund Landau, ⋆14.02.1877, †19.02.1938, German mathematician.

When analyzing randomized algorithms, we often use the following property of the expected value which is called *linearity of expectation*: For $a, b \in \mathbb{R}$ and any two random variables Y and Z, we have [88]

$$\mathbb{E}[aY + b] = a\mathbb{E}[Y] + b \quad \text{and} \quad \mathbb{E}[Y + Z] = \mathbb{E}[Y] + \mathbb{E}[Z]. \tag{1.5}$$

To model some of the environments used in this thesis, we need to formally introduce *graphs*. For more details, see the standard literature [61, 86, 151].

Definition 1.1 (Graphs). *A* graph G *is a tuple* (V, E)*, where* $V = \{v_1, \dots, v_n\}$ *is a non-empty set of* vertices*,* E *is a subset of* $\{(v_i, v_j) \mid v_i, v_j \in V\}$*, and* $e = (v_i, v_j) \in E$ *is called an* edge *from* v_i *to* v_j*, for* $v_i, v_j \in V$*. (i) If* $E = \{(v_i, v_j) \mid v_i, v_j \in V, v_i \neq v_j\}$*, we call* G complete*; (ii) furthermore, if, for all* $v_i, v_j \in V$*, from* $(v_i, v_j) \in E$ *it follows that* $(v_j, v_i) \in E$*, we say that* G *is* undirected *and denote the edge* (v_i, v_j) *by* $\{v_i, v_j\}$*. In the following, let* G *be undirected. (iii)* $G = (V, E, c)$ *is an* edge-weighted *(or simply weighted)* graph*, where* $c\colon E \to \mathbb{R}^+$ *is an edge-cost function. (iv) If we have that, for any* v_i, v_j*,* $c(\{v_i, v_j\}) = 0$ *is equivalent to* $v_i = v_j$ *(identity of indiscernibles), and, for* v_i, v_j*, and* v_k*,* $c(\{v_i, v_j\}) \leq c(\{v_i, v_k\}) + c(\{v_k, v_j\})$ *(triangle inequality), we call* c *a* metric cost function *and* G *a* metric graph*.*

In some of the subsequent chapters (especially in Chapter 8), we expect the reader to be familiar with computational complexity, the basic complexity classes, such as $\mathcal{P}$ and $\mathcal{NP}$, and the concept of $\mathcal{NP}$-hardness (introductions to algorithmics and computational complexity are given in, e. g., [58, 78, 82, 86, 87, 104, 131, 139, 143, 150]). Following Church's thesis[5], we consider *Turing machines*[6] [149] that halt on any input [87] to be the formal description of algorithms. Cobham [56] and Edmonds [67] were the first to consider computations that run in polynomial time *efficient*, and this point of view is nowadays widely accepted [86]. A *decision problem* is a computational problem that is answered by "yes" or "no", e. g., "does the graph G contain a Hamiltonian cycle[7]?". The class $\mathcal{NP}$ contains all decision problems that can be solved (answered correctly) by means of *nondeterministic Turing machines* in polynomial time, and $\mathcal{P}$ is the class of all such problems that can even be solved by deterministic Turing machines in polynomial time. A decision problem D is called $\mathcal{NP}$-hard if it is (with respect to runtime) at least *as hard* to solve as all other problems in $\mathcal{NP}$ [86]. If D is $\mathcal{NP}$-hard and $D \in \mathcal{NP}$, D is called $\mathcal{NP}$-complete (as introduced by Cook [57]). The probably most fundamental question in theoretical computer science is whether $\mathcal{NP} = \mathcal{P}$ or $\mathcal{P} \subsetneq \mathcal{NP}$. For an introduction on the relation between $\mathcal{P}$ and $\mathcal{NP}$, we refer the reader to the above literature.

Finally, throughout this thesis, for the ease of presentation, by $\log x$ we denote the logarithm of x with base 2 and by $\ln x$ we denote the logarithm of x with base e.

1.3 Online Algorithms and Competitive Analysis

The term *algorithm* is well-defined: It is the formal description of a strategy, which always has to produce a solution for a specific problem, formalized, as we just stated, by Turing machines that always halt; it is probably the most basic concept of computer science. In classical algorithmics (the "study of algorithms" [82]), we are interested in designing

[5] Alonzo Church, ⋆14.06.1903, †11.08.1995, American mathematician.
[6] Alan Turing, ⋆23.06.1912, †07.06.1954, English computer scientist.
[7] William R. Hamilton, ⋆04.08.1805, †02.09.1865, Irish mathematician.

fast algorithms that create high-quality solutions for a large set of instances of specific problems.

Conversely, as already pointed out, in numerous real-world applications, another challenge arises for the algorithm designer: In many situations, not the whole input is known in advance, but it arrives piecewise in consecutive time steps. After every such time step, a piece of output has to be created which must not be changed afterwards, i. e., the algorithm has to compute definite parts of the output without knowing the whole input. We call such situations *online scenarios* and the according strategies to cope with them *online algorithms*. Typical examples include (i) many components of operating systems that frequently interact with the user (the probably most prominent example in this class is the *paging problem* [38, 88], which we introduce later in this section), (ii) various scheduling problems, and (iii) routing problems (consider a network routing device with a limited buffer capacity that has to maintain some *quality of service*, see, e. g., [50, 52]).

Detailed introductions to online computation are found in [6, 38, 76, 88, 93]. Let us now formalize our ideas. In what follows, we focus on the goal to minimize some *cost function* that is specified by the concrete problem at hand. It is straightforward to obtain analogous definitions when studying maximization problems.

Definition 1.2 (Online Minimization Problem). *An* online minimization problem *consists of a set $\mathcal{I}$ of inputs and a cost function. Every input $I \in \mathcal{I}$ is a sequence of* requests $I = (x_1, \ldots, x_n)$. *Furthermore, a set of feasible outputs (or solutions) is associated with every I; every output is a sequence of* answers $O = (y_1, \ldots, y_n)$. *The* cost function *assigns a positive real value* $\mathrm{cost}(I, O)$ *to every input I and any feasible output O. For every input I, we call any feasible output O for I that has smallest possible cost (i. e., that minimizes the cost function) an* optimal solution *for I.*

For the sake of an easy notation, if the input is clear from the context, we omit I and denote the cost of the solution O for I by $\mathrm{cost}(O)$. It is crucial to note that the basic properties of algorithms, which separate them from arbitrary programs [87], are *not* violated: Online algorithms halt on any finite input (which might be a prefix of a potentially infinite instance) and they create output of some well-defined form.

The established measurement for the output quality of an online algorithm is the *competitive ratio* [38, 76, 88, 93], i. e., the quotient of the cost of the solution the online algorithm computes for a particular problem instance and the cost of an optimal (offline) solution for this instance.

Definition 1.3 (Online Algorithm, Competitive Ratio). *Consider an input I of an online minimization problem. An* online algorithm $\mathtt{A}$ *computes the output sequence* $\mathtt{A}(I) = (y_1, \ldots, y_n)$ *such that y_i is computed from $x_1, \ldots, x_i$ and $y_1, \ldots, y_{i-1}$. We denote the cost of the computed output by* $\mathrm{cost}(\mathtt{A}(I))$. $\mathtt{A}$ *is* c-competitive *if there exists a non-negative constant α such that, for every I, we have*

$$\mathrm{cost}(\mathtt{A}(I)) \leq c \cdot \mathrm{cost}(\textsc{Opt}(I)) + \alpha,$$

where $\textsc{Opt}$ *is an optimal offline algorithm for the problem. We also call c the* competitive ratio *of* $\mathtt{A}$. *If $\alpha = 0$, then* $\mathtt{A}$ *is called* strictly c-competitive*;* $\mathtt{A}$ *is* optimal *if it is strictly* 1*-competitive.*

We denote the competitive ratio of $\mathtt{A}$ on the instance I by $\mathrm{comp}(\mathtt{A}(I))$. Note that the constant α in Definition 1.3 prevents the construction of lower bounds by creating instances

for which the difference between the cost of an optimal algorithm and the online algorithm is constant. The algorithms we construct are all *strictly* competitive which we will therefore not mention in the subsequent chapters. However, for some of the lower bounds that we show, our analysis will be more careful. If we consider the c-competitiveness of maximization problems, we require that $\text{cost}(\texttt{A}(I)) \geq 1/c \cdot \text{cost}(\textsc{Opt}(I)) - \alpha$.

Again, to obtain a less complicated notation, if I is clear from the context, by $\mathcal{A} := \texttt{A}(I)$ [$\mathcal{O}pt := \textsc{Opt}(I)$], we denote the solution computed by A [Opt] on the instance I. Also, note that we use the terms algorithm and strategy interchangeably.

As usually done when studying online problems, we neglect the runtime of the constructed algorithms [38, 93], although the algorithms we design are ordinarily efficient ones. When talking about *competitive algorithms*, commonly algorithms that achieve a *constant* competitive ratio are meant. For a large number of online problems, however, different parameters might come into play on which the ratio might heavily depend (e. g., the memory size that is associated with some online algorithm as we see in Section 1.3). Classically, bounds with respect to these parameters are accepted, but it is not common to measure the performance depending on the number of requests; algorithms that provably do not obtain anything better than a competitive ratio that grows with the number of requests are called *not competitive* [38]. In this thesis, we want to allow a more fine-grained analysis, because, specifically in the framework we use, it makes a difference whether the competitive ratio grows linearly or logarithmically in the input size. Thus, e. g., in Chapter 3, we study two different variables the competitive ratio might depend on, one being a parameter that is known to the algorithm in advance (which is classically done for this problem [14]) and the other one being the number of requests. Additionally, for many online problems, the number of requests is upper-bounded by some of these parameters, as, e. g., is the case for the problems we study in Chapters 2 and 5.

The term *competitive analysis* is due to Karlin et al. [100]; as already mentioned, this concept was introduced by Sleator and Tarjan in 1985 [144], although it was probably used for the first time implicitly by Yao [153] in 1980 (see [38] for more details on the history). Since its introduction, competitive analysis has been used to analyze numerous online problems and algorithms designed to solve them. Let us again emphasize that Opt is an *offline* algorithm that knows the whole input before it starts its computations and therefore its existence is purely hypothetical. Consequently, this powerful knowledge is something A does not possess; this is exactly the dilemma we are facing in online computation. Informally speaking, we might therefore state the following:

The competitive ratio describes how much we pay for not knowing the future.

As already briefly mentioned before, in simple words, the following investigations aim at capturing *what* it actually is we *pay for*. Again, we will find out that the answer varies, depending on the problem we are looking at, from only missing a single bit of information to a necessary amount of information that rapidly grows with the input size.

The Adversary: Modeling Hard Instances

When dealing with online problems [38, 88], we try to study an online algorithm's performance in terms of its output quality by means of a notional *adversary* denoted by Adv; we suppose that Adv tries to construct input instances in a malicious way. To this end, we assume that Adv knows in every time step what A will do and that it uses this knowledge and A's inability to foresee the future to harm it as much as possible.

If we can prove the existence of an adversary ADV that, for some online problem P, creates an instance for which it can ensure that *every* online algorithm fails to be c-competitive, we can conclude that there does not exist any c-competitive algorithm for P.

There are basically three different types of adversaries considered in the literature [6, 19]: (i) the oblivious adversary, (ii) the adaptive online adversary, and (iii) the adaptive offline adversary. The difference between an oblivious adversary and the two adaptive types is the following: The oblivious adversary constructs an input instance *before* an online algorithm A starts its computation, whereas adaptive adversaries *react* to the decisions A makes during runtime, which means they know the outcome of random decisions that were made up to the corresponding time step. The adaptive online adversary then has to produce the output in an online fashion, whereas the adaptive offline adversary creates the output afterwards (thus acting optimally). While these three models are equally powerful when talking about deterministic online algorithms (here, at the beginning of the computation, A's behavior is already fully *determined*), there may be huge differences between them when we are dealing with randomized algorithms (see Section 1.4). The idea to consider the three types of adversaries is due to Ben-David et al., and it was shown that, when a randomized online algorithm plays against an adaptive offline adversary, random computation does not help at all [19].

When dealing with deterministic or randomized online algorithms, we consider oblivious adversaries only, which is justified by the following observation: Suppose we prove the existence of such an adversary ADV for some problem such that ADV can ensure that no deterministic strategy is better than c-competitive. This means that, for any deterministic online algorithm A, there exists at least one input for which it is *never* better than c-competitive. Then again, this input could be very natural and appear frequently in practice. We may think of ADV as knowing A's source code and thus being able to anticipate how A reacts for any piece of input it is given. If we now consider a randomized online algorithm R, ADV knows all its deterministic moves, when a decision based on randomness is made, and with which probability R takes some specific action. However, ADV does not know the outcome of this random decision. The key is to construct randomized algorithms that, for every instance, behave well on average. For these algorithms there is no particular bad input that always causes them to fail, but only for a few random decisions. This means that, in practice, there do not exist instances that *on average* cause R to produce bad output, but only occasionally.

When dealing with online algorithms with advice, we use a different type of adversary we describe in Section 1.6; this adversary is discussed in more detail in Chapter 8.

Let us now give a short example to consolidate our intuition by reviewing some known results on the well-studied paging problem [5, 38, 76, 88, 93].

The Paging Problem

As already mentioned, typical online scenarios are often met when dealing with operating systems due to the frequent interaction with a user, who performs actions no algorithm may foresee. One very important problem operating systems have to handle is the paging problem [38, 88], PAGING for short, which we describe in a very simplified way; a practical view of the problem can be found in the standard literature [148].

The physical memory of a computer can be arranged in a hierarchy that starts at the top with very fast but expensive kinds like the CPU's registers and its caches, has slow but cheap types of memory, such as the hard disc, at its bottom, and the *random access*

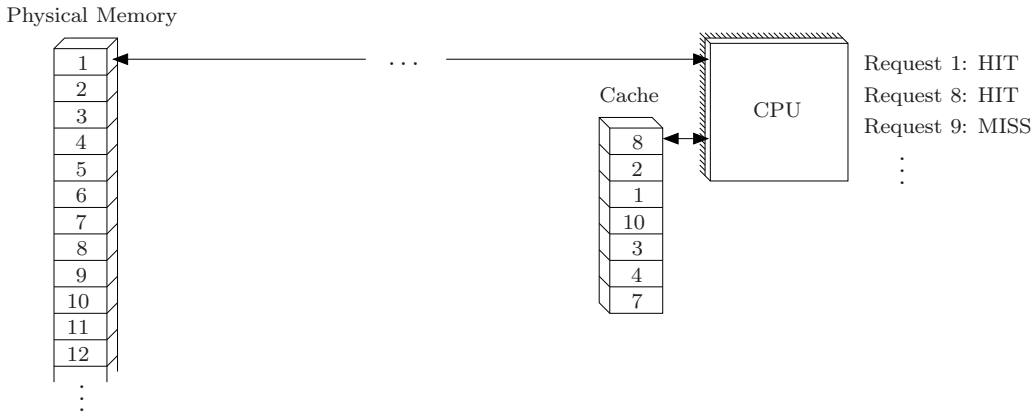


Figure 1.1: Schematic view of PAGING.

memory (RAM, often plainly called the *physical* or *main* memory) in between. In the following, we focus on a memory hierarchy that consists of only two types of memory: the physical memory and the much faster but therefore more expensive and thus smaller cache.

The discrepancy in speed between the physical memory and the CPU is known as the *von Neumann bottleneck*[8]. To use memory as efficiently as possible, the cache and the physical memory are subdivided into so-called *page frames* of a fixed size s. Any program, on the other hand, only works on a virtual memory that consists of *logical pages*, also each of size s. The operating system then maps the logical pages to the page frames. If a program wants to access a page that is currently not in the cache, but in the physical memory only, we call this a *page fault*. In this case, the operating system has to select some *victim* page frame from the cache, replace its content by the content of the requested page, and update the mapping of the virtual pages to the page frames. Obviously, the replaced logical page of the victim frame is no longer accessible in the cache. Selecting the victim frame is the main task of a paging algorithm. A schematic view of PAGING is shown in Figure 1.1. In what follows, we give a formal description.

Definition 1.4 (PAGING). *Consider a sequence of integers representing requests to logical pages $I = (x_1, \ldots, x_n)$, $x_i > 0$. An online algorithm* A *maintains a buffer (content of the cache) $B = \{b_1, \ldots, b_K\}$ of K integers, where K is a fixed constant known to* A*. Before processing the first request, the buffer gets initialized as $B = \{1, \ldots, K\}$. If* A *receives a request $x_i \in B$, it creates the partial output $y_i = 0$. However, if $x_i \notin B$, a* page fault *occurs, and* A *has to specify some* victim *b_j, i. e., $B := B \setminus \{b_j\} \cup \{x_i\}$ and $y_i = b_j$. The cost of the solution $\mathcal{A} = \mathtt{A}(I)$ is the number of page faults, i. e., $\mathrm{cost}(\mathcal{A}) = |\{y_i \mid y_i > 0\}|$.*

For this problem and for every online algorithm A, we can easily construct an adversary ADV that harms A in the worst possible way while merely using one more page than A has space to keep in the cache at one point in time. Consider Algorithm 1.1 which is due to Sleator and Tarjan [144]. It is obvious that any deterministic online algorithm playing against this adversary makes exactly one page fault per request, because it always has to load the one page into the cache that it just removed from it. It remains to show that there exists an optimal offline algorithm OPT that has a smaller cost: We observe that the first page fault is inevitable. However, after that, there are $K - 1$ more requests and

[8] John von Neumann, ⋆28.12.1903, †08.02.1957, Austrian-Hungarian mathematician.

Algorithm 1.1: Adversary for PAGING

```
output "Request page with index K + 1";
i := 1;
while i ≤ K − 1
    j := index of page removed by A;
    output "Request page with index j";
    i := i + 1;
end
```

$K+1$ pages in total. It follows that one of the initial pages, say page p, in the cache is not requested. If OPT removes p to load the page with index $K+1$ in the first time step, it makes no additional page fault.

It is straightforward to generalize this idea and create an adversary that constructs sequences of arbitrary length n such that any deterministic online algorithm makes a page fault with every request [144]. On the other hand, OPT (more specifically, the algorithm MIN, introduced by Belady, that implements the *offline* strategy *longest forward distance* [17]) only makes one page fault in every Kth step.

Theorem 1.5 (Sleator and Tarjan [144]). *For any deterministic online algorithm* A *for* PAGING*, there exists an input* I *such that*

$$\mathrm{comp}(\mathtt{A}(I)) = \frac{\mathrm{cost}(\mathtt{A}(I))}{\mathrm{cost}(\mathtt{OPT}(I))} \geq K,$$

where OPT *is an optimal offline algorithm.* ♢

We deal with PAGING in the framework of advice complexity in Section 1.7.

1.4 Randomized Algorithms

One of the main concepts among the strategies which are used to produce high-quality output (not only) in online computation is *randomization*. Here, we allow an online algorithm R to roll dice from time to time and base some of its calculations on the outcome. For the further study of randomized computation, we refer the reader to the standard literature [88, 104, 128, 131]. Formally, randomized online algorithms can be defined as follows; again, we focus on minimization problems.

Definition 1.6 (Randomized Online Algorithm). *Consider an input* I *of an online minimization problem. A* randomized online algorithm R *computes the output sequence* $\mathcal{R}^\phi := \mathtt{R}^\phi(I) = (y_1, \ldots, y_n)$ *such that* y_i *is computed from* $\phi, x_1, \ldots, x_i$*, where* ϕ *is the content of a* random tape*, i. e., an infinite binary sequence, where every bit is chosen uniformly at random and independently of all the others. By* $\mathrm{cost}(\mathtt{R}^\phi(I))$ *we denote the random variable expressing the cost of the solution computed by* R *on* I*.* R *is* c*-competitive in expectation if there exists a non-negative constant* α *such that, for every* I*,*

$$\mathbb{E}[\mathrm{cost}(\mathtt{R}^\phi(I))] \leq c \cdot \mathrm{cost}(\mathtt{OPT}(I)) + \alpha,$$

where, as above, OPT *is an optimal offline algorithm for the problem.*

For the ease of presentation, we omit ϕ if it is clear from the context. Later, we will point out some connections between randomized computation and computation with advice in online scenarios.

Fiat et al. showed that, for PAGING, every randomized online algorithm is at best H_K-competitive in expectation [74], where $H_K := \sum_{i=1}^{K} 1/i$ is called the Kth *harmonic number*. Moreover, there exist algorithms that achieve this ratio [1, 125]. Since H_K is roughly $\ln K$ [80] and we have seen that deterministic online algorithms are exponentially worse (see Section 1.3), randomization improves the performance of online algorithms for PAGING drastically (as it is the case for many of the problems considered in the literature [38]).

Barely Random Algorithms

Generating random numbers might be expensive. We therefore are particularly interested in so-called *barely random algorithms*, introduced in [136], that merely use a constant number of random bits, which is asymptotically minimal [38]. We construct such algorithms for the job shop scheduling problem with two jobs and unit-length tasks (see Chapter 2) and the simple online knapsack problem (see Chapter 6) in Section 7.1. For PAGING, we briefly explain how to use some insight we have gained from the construction of an online algorithm with advice [35] to construct a barely random algorithm, as shown in [109].

1.5 Alternative Models

As we have seen in Section 1.3, for PAGING, deterministic online algorithms are at a disadvantage in the game against an oblivious adversary ADV. A similar situation is faced when dealing with most of the online problems that were studied. It has been pointed out that the idea of competitive analysis is not fine-grained enough as it is, in general, too pessimistic [6, 18, 43, 64, 75, 93, 114]; in other words, many algorithms that perform very well in practice are considered to be very weak with respect to competitive analysis.

The most straightforward approach to weaken the requirements on online algorithms is to restrict the power of ADV. As for PAGING, Theorem 1.5 states that *no* deterministic online algorithm is better than K-competitive (a phenomenon which is referred to as the *triviality barrier* [75]). However, in practice, the strategy *least recently used* (LRU [148]) achieves good results compared to, e. g., a simple *first-in-first-out*-strategy (FIFO); strong evidence for this is given by Young [154], whereas it is not reflected at all by competitive analysis, see, e. g., [39, 114]. It is thus not surprising that especially PAGING was among the first problems for which alternative models were proposed. Another reason is that it seems reasonable to exploit one property of typical instances of this problem known as *locality of reference*: In practice, the set of pages that are likely to be requested after some particular request is rather small [142]; it is thus common to use *prefetching*. To formalize this behavior, Borodin et al. developed so-called *access graphs* for the analysis of PAGING [39]. Every vertex v_x of this graph corresponds to a page x that might be requested during runtime. Moreover, for every page x_2 that may be requested immediately after x_1, v_{x_1} and v_{x_2} must be adjacent, see also [94]. The authors conjectured that, with respect to this model, LRU is never worse than FIFO which was later proven by Chrobak and Noga [54].

Young introduced the concept of the *loose competitive ratio* for PAGING [154]; a revised version was then applied to a more general caching problem [156]. The concept is motivated by the following two facts: (i) The actual ratio of the cost of A's solution and the optimum can be neglected as long as the computed solution has a small cost in an absolute sense. (ii) For many online problems, the strength of ADV relies on the knowledge of some parameter of the concrete problem (e. g., the cache size K for PAGING) which means that hard instances might only be hard for some very specific choices of K. Now, for $\varepsilon, \delta > 0$,

an online algorithm A is said to be $(\varepsilon,\delta,K_{\max})$-loosely c-competitive if, for a $(1-\delta)$-fraction of all possible values of $K \in \{1,\ldots,K_{\max}\}$, the cost of A is either less than $\varepsilon \cdot n$ or less than $c \cdot \text{cost}(\mathcal{O}pt)$ with cache size K.

Raghavan devised the notion of a *statistical adversary* [135]. In this model, ADV may also create the input in an arbitrary fashion, but it must obey some conditions with respect to the mean and the standard deviation of the input. These values might be taken from a prior statistical analysis of the problem, thus providing a useful tradeoff between worst-case analysis and practice, see also [51]. Koutsoupias and Papadimitriou followed a similar approach when introducing the *diffuse adversary* [114] arguing that facing complete uncertainty about the input distribution is highly unlikely. In their model (which is a generalization of both the statistical adversary and the access graph model), ADV chooses a worst-case distribution from a class of possible distributions on the inputs that is known to A, see also [155].

Another approach to give an online algorithm A an advantage compared to the classical model is to enable A to have some *lookahead*, i. e., to allow it to look into the future for ℓ time steps. However, consider PAGING with lookahead ℓ. Since the adversary we use to model bad input instances knows A, it surely knows ℓ and may therefore proceed as given by Algorithm 1.2 (see, e. g., [5, 114] and also [18] where Ben-David and Borodin showed that lookahead does not help for the k-server problem which we will discuss in Chapter 4 and which is a generalization of PAGING). Clearly, A also fails to play against

Algorithm 1.2: Adversary for PAGING with lookahead ℓ

```
r := 1;
while r ≤ ℓ + 1
    output "Request page with index K + 1";
    r := r + 1;
while i < K − 1
    j := index of page removed by A;
    r := 1;
    while r ≤ ℓ + 1
        output "Request page with index j";
        r := r + 1;
    i := i + 1;
end
```

this adversary.

Albers followed a different approach by introducing and using a so-called *strong* lookahead that enables the algorithm to see ℓ *pairwise distinct* future requests [5]. This (more powerful) knowledge about the future does indeed help for PAGING: Let $\ell \leq K-2$. There exists a deterministic online algorithm (basically a variant of the above mentioned strategy LRU) with strong lookahead ℓ that is $(K-\ell)$-competitive, and this bound is tight; additionally, it was shown that there exists a randomized online algorithm with strong lookahead ℓ that achieves an expected competitive ratio of $H_{K-\ell}$ and that this bound is tight up to a factor of 2 [5]. Moreover, introducing the concept of *comparative analysis*, Koutsoupias and Papadimitriou studied the general power of algorithms with lookahead *compared* to plain online algorithms [114]. For PAGING, they showed that a lookahead of ℓ allows to improve the performance by a factor of $\min\{K, \ell+1\}$.

Another idea is to use what is known as *resource augmentation*, which was introduced in [97] (though used earlier implicitly [59]), see [59, 98, 133]. Here, A is compared to an

offline algorithm that has some disadvantage concerning the resources available. In [95], Iwama and Zhang used this relaxation of pure competitive analysis to study the online knapsack problem; for this problem, we combine resource augmentation and computing with advice in Chapter 6.

While resource augmentation allows the online algorithm to use more resources than the optimal offline algorithm, the *accommodating ratio*, introduced by Boyar and Larsen, restricts the allowed inputs in another way. Here, only those instances are permitted for which an optimal algorithm would not have an advantage from having more resources available than the online algorithm [42]. A relaxation leads to the *accommodating function*, established by Boyar, Larsen, and Nielsen [43]: Consider some online problem with m resources, e. g., the online bin packing problem with m bins [38]. An input x is called a γ-sequence if, with a multiple γ of the resources available (e. g., with $\gamma \cdot m$ bins), OPT can satisfy *all* requests (e. g., pack all items into bins). If, for any such input, A achieves a competitive ratio of c, we say that A is c-competitive on γ-sequences. Obviously, with γ tending to infinity, all instances are allowed and this measurement is the competitive ratio.

The term *semi-online* problem is widely used in the area of scheduling problems (an introduction is given by Brucker [46], see Chapter 2) for frameworks in which some specific property about the input is known in advance. Examples are the sum of all the processing times of all jobs that arrive online, simply the fact that these processing times are non-increasing, or the optimal makespan [66, 72, 84, 85, 102, 140].

A more detailed survey on the different refinements of competitive analysis that were proposed since its establishment is given in [75], which is Chapter 17 of [76], and in Chapter 3 of [65].

While also introducing an alternative model, our approach seems somewhat orthogonal to all of the above. We do not give the algorithms some specific advantage, like a lookahead or larger memory, but we are interested in the pure amount of information that is needed to increase performance. In this sense, the advice complexity is a generalization of some of the above concepts. Then again, to a great extent, these concepts aim at providing a more realistic way to compare known algorithms to each other; the most prominent examples are the aforementioned strategies LRU and FIFO for Paging and the large gap between their performances in practice. Conversely, our goal is to construct new algorithms that use additional information to perform well or to show that such algorithms do not exist for some specific amount of information. Hence, we try to give insight into the hardness of the *problem itself* under online considerations and not primarily into well-known strategies.

1.6 Advice Complexity

Before we elaborate and formalize the model used in this thesis, let us start with a straightforward example that has been first observed in [63, 64]. Consider the well-known ski rental problem [38, 93], SkiRental for short.

Definition 1.7 (SkiRental). *Suppose you want to go skiing, but you do not know in advance for how many days this is really possible due to, e. g., the weather conditions. You would like to maximize the number of days you ski; however, just in the morning of each day you get a reliable weather forecast. As you do not own any skiing gear, you may either, for every day, rent skis for unit cost* 1*, or buy the skis for cost* $k > 1$.

More on the problem can be found in [38]. The best-known randomized online algorithm is due to Karlin et al. and it achieves an expected competitive ratio of $e/(e-1)$, which is the best possible result against an oblivious adversary [99]; but let us focus on deterministic strategies.

Obviously, it makes sense to buy the skis if you ski more than k days, otherwise, you pay too much. Now we consider any deterministic online algorithm A and an adversary ADV that is able to control the weather, that acts according to Algorithm 1.3, and (as discussed in Section 1.3) that aims at boosting A's cost compared to the one of an optimal solution. Let us look at what happens if A plays against ADV: (i) If A decides to buy the skis on the

Algorithm 1.3: Adversary for SkiRental

```
output "Good skiing weather";
while A has not bought the skis
    output "Good skiing weather";
while true
    output "Bad skiing weather";
end
```

first day with good weather, ADV causes all other days to be bad for skiing; in this case, the competitive ratio of A is k. (ii) On the other hand, if A decides to buy the skis at day k, it pays $(k-1)+k$ in total. The competitive ratio is therefore $(2k-1)/k$, which tends to 2, for k tending to infinity. (iii) If A buys the skis at day $i < k$, the competitive ratio is $1+(k-1)/i \geq 2$. (iv) Finally, if A buys the skis at day $j > k$, its competitive ratio is $1+(j-1)/k \geq 2$. It follows that A cannot be better than 2-competitive; furthermore, the above analysis implies that this bound is tight, i. e., there exists a 2-competitive online algorithm, which is known as BreakEven, for SkiRental.

So what do deterministic strategies effectively lack that causes them to be twice as bad as the optimal solution? Obviously, if A knew the whole input sequence in advance, it could produce optimal output. However, this is not necessary. All that is needed is one single piece of information that is as small as it gets, i. e., one bit, telling A to "buy" or "rent" in advance. A then still has no clue about which days will have good or bad weather, but the *kernel* of the missing information is very small; we can say that this one bit is all the information we need to *extract* from the input.

In this example, the advice complexity of an optimal online algorithm (the *information content* [89] of SkiRental) is smallest possible; it is 1. Let us summarize: Deterministic online algorithms perform twice as bad as offline algorithms when dealing with SkiRental, and, astonishingly, this high price is paid for merely not knowing one single bit. In general, it can be a lot more involved to figure out what this crucial information is.

The First Model of Advice Complexity

The advice complexity of online problems was introduced by Dobrev, Královič, and Pardubská in 2008 as a new measurement for online algorithms addressing the problems of competitive analysis we described in Section 1.5 [63]. One of the motivations they had in mind when establishing the following model was the communication between a base station, which has powerful resources available, and a remotely controlled robot with very limited capabilities. As an example, we may think of Cape Canaveral and some Mars Rover. The authors were interested in bounding the relevant information that is needed

for the robot to perform some predefined tasks; in particular, they investigated PAGING and DIFFSERV (short for *differentiated services*, see, e. g., [118]).

Two different modes of operation were proposed and studied that allow different ways of communication between the base station, which was formalized by an oracle, and the algorithm (robot).

The helper model. Here, we think of an oracle O that oversees A's actions during runtime. O may interact with the algorithm by giving some bits of advice in every time step; this is done without a request for help by A. The crucial observation is that the advice may be empty, which can also carry some piece of information and thus may be exploited by the algorithm.

The answerer model. The second model is more restrictive in some sense as the algorithm A has to explicitly ask O for advice in some time step. O then has to respond with some advice string, i. e., it is not allowed to send an empty string, but still may encode some extra information into the length of its answer.

In both models, it is assumed that A knows the length n of the input in advance. A more detailed and formal description can be found in [64]. To get an understanding of how to work within this framework, let us use a similar attempt to construct advice for some online problem. We use the more powerful helper model and restrict ourselves to a specific class of online problems.

Suppose that the online problem we are dealing with is defined in a way such that A has to make at most one 0/1-decision in every time step (actually, this is a rather natural restriction as this is the case for many different problems, e. g., PAGING, as shown in [64]). A solution is thus determined by a *decision string* S of length n that represents the decisions made. Obviously, these decisions can be made correctly by an optimal offline algorithm OPT that knows the whole input in advance.

As in [64], we suppose that A and O work synchronized, i. e., they share a common clock and have agreed on a fixed length for the time steps (a very strong assumption, which is one of the reasons why we change the model in Section 1.6). Additionally [64], we also assume that A knows n in advance; moreover, the algorithms we construct are allowed to do some *bootstrapping*, i. e., before the first time step, some extra bits may be sent that are crucial for the further computations of A. However, we only allow this additional number of bits to be a small constant. It follows that A and O can exploit empty advices in a straightforward fashion: At the beginning, O sends one bit of advice to A indicating whether a fixed optimal decision string S contains more ones or more zeros. Without loss of generality, assume that there are more ones in S. O may then just send an arbitrary bit in time steps where the decision 0 is made by an optimal algorithm according to S, and nothing otherwise. It is clear that A will know exactly what to do and that at most $n/2+1$ advice bits will be sent in total (see Figure 1.2 (a)).

We observe that it does not matter to A whether a 1 or a 0 is communicated in some time step. We now show how to easily reduce the number of advice bits by simply using the bits sent to encode S backwards (see Figure 1.2 (b)). It follows that $(n-2)/3+3$ bits of advice are sufficient in general.

Theorem 1.8. *To enable an online algorithm* A *to produce an optimal solution, it suffices to send a total of* $(n-2)/3+3$ *bits, where at most one bit is sent in every time step, and at most three additional bits are sent before the input is processed.*

S:		1	1	0	1	0	1	0	0	1	1	0	1	0	1	1	0
Sent:	0			0		0		0	0			0		0			0

(a) An example for a decision string S and the corresponding advice sequence.

S:		1	1	0	1	0	1	0	0	1	1	0	1	0	1	1	0
Sent:	00			0		1		1	0			1					

(b) An example of how to further reduce the number of bits communicated.

Figure 1.2: An example communication exploiting empty strings.

Proof. Let n' denote the largest number smaller than n that is divisible by 3. The last $n - n' \leq 2$ bits of S are communicated to A at the beginning. Let S' denote the prefix of S of length n'. The oracle O then examines the first $2n'/3$ bits of S' and sends A the bit that appears least frequently; afterwards, O uses the positions of these bits to send a distinct suffix of S' in reverse order. Recall that A knows n and thus $n'/3$. We distinguish two cases.

Case 1. Suppose that there is exactly the same number of ones and zeros in the first two thirds of S'. O may then choose one of the two and communicate its choice to A using one bit. Since this bit appears exactly $n'/3$ times, it is possible to send exactly the last third of S'. Thus, A uses at most $(n - n') + 1 + n'/3 \leq (n - 2)/3 + 3$ bits of advice in this case.

Case 2. Without loss of generality, suppose that there are more ones than zeros in the first two thirds of S'. O tells A that a bit is sent in every time step in which the decision 0 has to be made according to S'. Since this number of zeros is strictly smaller than $n'/3$, the last bit is sent after more than $2n'/3$ time steps.

Consider time step $2n'/3+1$, and assume that the last $m < n'/3$ bits of S' are already known to A. A simply continues to decode the bits sent as in the first two thirds until it realizes that it now knows $n'/3$ bits. Let m' denote the number of zeros in the last third of S' that are not yet known to A. Clearly, we have $m' \leq n'/3 - m$ and therefore $m + m' \leq n'/3$. Hence, since A received m bits already and is given at most m' more bits, $(n - n') + 1 + n'/3 \leq (n - 2)/3 + 3$ advice bits are sufficient.

The claim follows. □

Up to this point, we have used some very simple ideas to show that it is sufficient to send one bit of advice in every third time step on average to communicate S, allowing A to be optimal. We now generalize this idea by discarding the constraint that at most one bit is permitted to be sent in every time step.

Theorem 1.9. *To enable an online algorithm* A *to produce an optimal solution, it suffices to send a total of βn bits, for a constant $\beta := 0.30104$ and n tending to infinity, and a constant number of bits before the input is processed.*

Proof. Again, without loss of generality, let there be more ones than zeros among the first $(1 - \alpha)n$ bits of S, for a constant $\alpha := 0.519$, and let z denote the number of these zeros. Moreover, let $\beta' := 0.302 > \beta > \beta'' := 0.301$. We distinguish two cases depending on the size of z; A is told which case to follow by one bit of advice at the beginning.

Case 1. Suppose that $z < \frac{\beta' n - 1}{2}$. In the following, we show how to send no more than $\beta'' n$ bits in total. By our assumptions, O has sent exactly z bits (which is at most

$0.151n - \frac{1}{2}$) when arriving at time step $(1-\alpha)n = 0.481n$. Note that, at this point, A already knows the last z bits, and thus $n - (1-\alpha)n - z = 0.519n - z$ bits are still required to be communicated. Let S' be the corresponding string that is yet unknown to A. Following Theorem 1.8, we know that $\frac{1}{3}(0.519n - z - 2) + 3$ bits suffice to communicate S', and O is allowed to send another $0.301n - z$ bits. Observe that

$$0.301n - z > \frac{1}{3}(0.519n - z - 2) + 3 \iff z < 0.192n - \frac{7}{2},$$

which is satisfied by $z < \frac{\beta' n - 1}{2} = 0.151n - \frac{1}{2} < 0.192n - \frac{7}{2}$, for every $n \geq 74$.

Case 2. Suppose that $z \geq \frac{\beta' n - 1}{2}$ and thus $z \geq \frac{\beta n - 1}{2}$. Moreover, by the definition of z, we have that $z \leq \frac{(1-\alpha)n}{2}$. For every $n \geq 6$,

$$\frac{\beta n - 1}{2} < \frac{0.302n}{2} - \frac{1}{2} < \frac{0.481n}{2} - 1 = \frac{(1-\alpha)n}{2} - 1 < \frac{(1-\alpha)n}{2}. \tag{1.6}$$

Next, let us estimate how many ways there are to send βn bits during these z time steps. First of all, there are exactly $2^{\beta n}$ different binary strings of length βn. Furthermore, for each of these strings, we may choose a different partitioning to distribute them among z slots (i. e., time steps in which a 0 appears). However, we are not allowed to send nothing in such a time step, because A still has to be told that a 0 appears in S in this step as well. There are $\beta n - 1$ many positions at which we may cut the string of length βn, and we have to cut it exactly $z - 1$ times to distribute the bits among the $\frac{(1-\alpha)n}{2}$ time steps in which a 0 occurs. Obviously, there are exactly

$$\binom{\beta n - 1}{z - 1}$$

possibilities to do this. Note that, since $z \leq \frac{(1-\alpha)n}{2} < \beta n$, the above term is well-defined. Our goal is to encode a bit string of length αn, or, in other words, $2^{\alpha n}$ many possibilities, so we have to ensure that

$$2^{\beta n}\binom{\beta n - 1}{z - 1} \geq 2^{\alpha n}. \tag{1.7}$$

Informally speaking, this means that there have to be sufficiently many possibilities in the first $(1-\alpha)n$ time steps to encode all binary strings of length αn using βn bits. We can exploit that A can distinguish different cases due to different distributions of the same βn-bit string over the time steps.

It remains to show that (1.7) holds for any z, for which $\frac{\beta n - 1}{2} \leq z \leq \frac{(1-\alpha)n}{2}$. Recall our discussion about the central binomial coefficient in Section 1.2: We have that $\binom{m}{k_2} < \binom{m}{k_1} < \binom{m}{k}$, for $k_2 > k_1 > k = m/2$. Thus,

$$\binom{\beta n - 1}{\frac{\beta n - 1}{2}} \geq \binom{\beta n - 1}{z - 1} \geq \binom{\beta n - 1}{\frac{(1-\alpha)n}{2} - 1}. \tag{1.8}$$

Hence, the left-hand side of (1.7) has to be a lower bound on the number of possibilities to send the βn bits which is a direct consequence of (1.6). We observe that, if z decreases from $\frac{(1-\alpha)n}{2}$ towards $\frac{\beta n - 1}{2}$, the left-hand side of (1.7) increases; a further decrease is covered by case 1. We obtain

$$2^{\beta n}\binom{\beta n - 1}{\frac{(1-\alpha)n}{2} - 1} \geq 2^{\alpha n} \iff \frac{1-\alpha}{2\beta} \cdot 2^{\beta n}\binom{\beta n}{\frac{(1-\alpha)n}{2}} \geq 2^{\alpha n}$$

and, by the definition of the binomial coefficient,

$$\frac{1-\alpha}{2\beta}\cdot 2^{\beta n}\cdot\frac{(\beta n)!}{\left(\beta n-\frac{(1-\alpha)n}{2}\right)!\left(\frac{(1-\alpha)n}{2}\right)!}\geq 2^{\alpha n},$$

which is, employing Stirling's approximation (see (1.1) and (1.2)), implied by

$$\frac{\frac{1-\alpha}{2\beta}\cdot\sqrt{\beta n}\cdot\beta^{\beta n}}{\sqrt{\left(\beta-\frac{1-\alpha}{2}\right)n}\left(\beta-\frac{1-\alpha}{2}\right)^{\left(\beta-\frac{1-\alpha}{2}\right)n}\sqrt{\left(\frac{1-\alpha}{2}\right)n}\left(\frac{1-\alpha}{2}\right)^{\frac{(1-\alpha)n}{2}}c}\geq 2^{(\alpha-\beta)n},$$

for a constant $c=1.05^2\cdot\sqrt{2\pi}$. Taking the logarithm (we easily verify that both sides of the above inequality are strictly positive), this is equivalent to

$$\begin{aligned}(\alpha-\beta)n\leq\ &\log(1-\alpha)-\log(2\beta)+\frac{1}{2}\log(\beta n)+\beta n\log\beta-\frac{1}{2}\log\left(\frac{1-\alpha}{2}n\right)\\&-\frac{1-\alpha}{2}n\log\left(\frac{1-\alpha}{2}\right)-\frac{1}{2}\log\left(\left(\beta-\frac{1-\alpha}{2}\right)n\right)\\&-\left(\beta-\frac{1-\alpha}{2}\right)n\log\left(\beta-\frac{1-\alpha}{2}\right)-c',\end{aligned}$$

for some constant $c'=\log c$. Hence,

$$\begin{aligned}(\alpha-\beta)n\leq\ &\log(1-\alpha)-\log 2-\frac{1}{2}\log\beta-\frac{1}{2}\log n+\beta n\log\beta\\&-\frac{1}{2}\log\left(\frac{1-\alpha}{2}\right)-\frac{1-\alpha}{2}n\log\left(\frac{1-\alpha}{2}\right)-\frac{1}{2}\log\left(\beta-\frac{1-\alpha}{2}\right)\\&-\left(\beta-\frac{1-\alpha}{2}\right)n\log\left(\beta-\frac{1-\alpha}{2}\right)-c'\end{aligned}$$

or, equivalently,

$$\begin{aligned}0\leq\ &\left(\beta\log\beta-\frac{1-\alpha}{2}\log\left(\frac{1-\alpha}{2}\right)-\left(\beta-\frac{1-\alpha}{2}\right)\log\left(\beta-\frac{1-\alpha}{2}\right)\right)n\\&-\alpha n+\beta n-\frac{1}{2}\log n-1+\log(1-\alpha)-\frac{1}{2}\log\beta-\frac{1}{2}\log\left(\frac{1-\alpha}{2}\right)\\&-\frac{1}{2}\log\left(\beta-\frac{1-\alpha}{2}\right)-c'\end{aligned}$$

is a sufficient condition for (1.7) to hold. Next, let

$$\gamma(\alpha,\beta):=\beta\log\beta-\frac{1-\alpha}{2}\log\left(\frac{1-\alpha}{2}\right)-\left(\beta-\frac{1-\alpha}{2}\right)\log\left(\beta-\frac{1-\alpha}{2}\right)-\alpha+\beta.$$

We obtain the condition

$$\begin{aligned}0\leq\ &\gamma(\alpha,\beta)n+\log(1-\alpha)\\&-\frac{1}{2}\left(\log(\beta n)+\log\left(\frac{1-\alpha}{2}\right)+\log\left(\beta-\frac{1-\alpha}{2}\right)\right)-c'',\end{aligned}\tag{1.9}$$

for a constant $c''=c'+1=\log\left(1.05^2\cdot\sqrt{2\pi}\right)+1$. Since n is the leading term in this expression, (1.9) holds for n tending to infinity if $\gamma(\alpha,\beta)>0$. An easy calculation

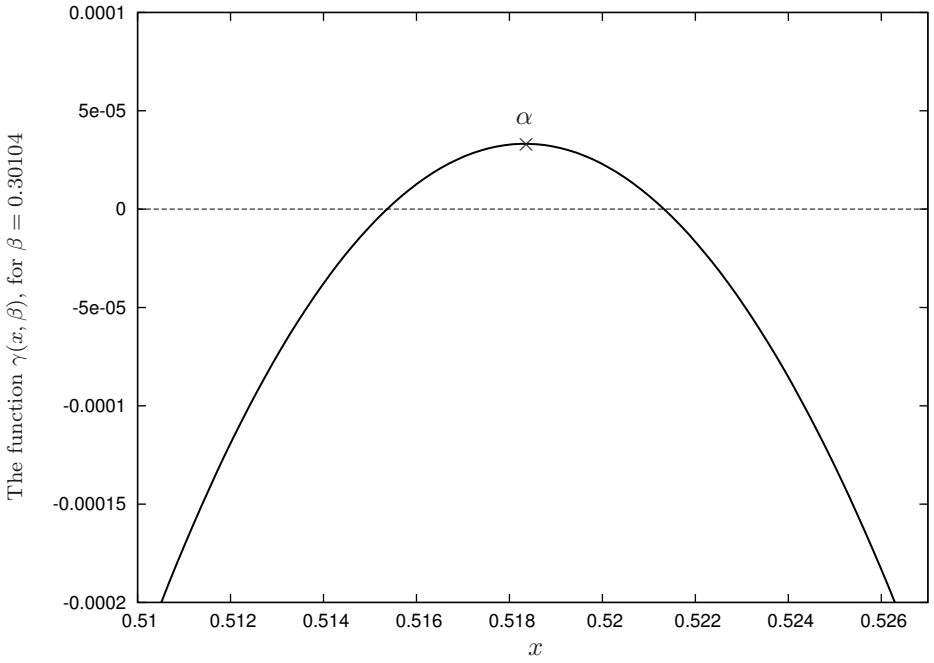


Figure 1.3: The function $\gamma(x, \beta)$.

shows that this is true. To this end, let us treat γ as a function of possible values x for α. Numerically solving gives that $\gamma(x) := \gamma(x, \beta)$, where β is fixed as 0.30104 (as given by the lemma), has roots at $x_1 \geq 0.51537$ and $x_2 \leq 0.52132$, and is positive between these values (see Figure 1.3); the actual maximum is obtained for $x_{\max} \approx 0.51834$. Thus, $\gamma(\alpha, \beta)$ is indeed strictly positive, which implies that it is sufficient to send 0.30104 bits per time step in this case.

This finishes the proof. □

We conclude that the model we used here allows for very powerful techniques reducing the number of sufficient advice bits βn by almost 70 percent compared to the straightforward communication of the n-bit string S. Note that, in [64], following another approach, it was even shown that roughly 0.2056 bits per time step are sufficient. However, to achieve this, $\mathcal{O}(\log n)$ bits need to be communicated before the first time step which was not allowed here.

The New Model of Advice Complexity

Consider the two modes of operation from [64] as introduced in the last subsection. In the helper model, the oracle actively communicates with `A`, sending an advice string whenever necessary. As mentioned before, this way `A` also learns something about the input if nothing is sent at all during some time step. This is not possible in the answerer model; however, in both modes of operation, information may be encoded into the *length* of the advice strings that are communicated by `O`. More specifically, as already observed in [35, 68], we are dealing with advice over an alphabet $\{0, 1, \$\}$, where \$ marks the end of an advice string supplied in one time step. Therefore, the two strings 0\$10\$ and 010\$ may be interpreted differently by `A`, but, when estimating the number of bits, they are only

counted *once* as the binary string 010. In the helper model, we even allow strings of the form \$\$0\$. Moreover, in both models, the size n of the input is known to A in advance.

We argue that these facts bias the real information needed to compute a solution, and we want to prevent such ways of encoding and measure the pure binary advice. Also, the lack of knowing n often contributes to the hardness of online problems. It thus seems to be inappropriate to neglect this fact if we want to understand the essence of what makes these problems hard. Basically, there are two approaches to overcome this drawback that were proposed independently in 2009.

Emek et al. used a model in which the online algorithm is restricted to reading only a fixed number of advice bits in every time step [70], introduced in [68]. This means that A gets a fixed number of d advice bits together with every request. The advice complexity is then defined as $d \cdot n$ for inputs of length n. This model was applied to the k-server problem (which, as already mentioned, we study in Chapter 4) and *metrical task systems* (a famous further generalization of the latter problem, introduced by Borodin, Linial, and Saks in [40], see also [38, 41, 93]).

In this thesis, we follow a different approach, introduced in [35]: The oracle sees the whole input in advance and knows the online algorithm at hand. It then writes all information necessary to an *advice tape* that the algorithm uses as an additional resource during runtime and that it may access sequentially. The advice complexity is then the total number of bits accessed.

The following observation implies that our model is more general and more powerful in this sense.

Observation 1.10. *Algorithms studied within the model from [70] admit algorithms of at least the same quality in our model (i. e., upper bounds carry over). On the other hand, lower bounds shown in our model imply the same lower bounds in the former model.*

We can consider online computation as a *zero-sum game* [146] between A and Adv [38]. When computing with advice, we introduce a third player O to this game that collaborates with A. Adv knows both its opponents, constructs an input accordingly, after which O inspects this input, prepares the advice tape, and finally A outputs a solution based on Adv's input and O's advice. Let us define the model formally.

Definition 1.11 (Online Algorithm with Advice). *Consider an input I of an online minimization problem. An* online algorithm A with advice *computes the output sequence $\mathcal{A}^\phi := \mathtt{A}^\phi(I) = (y_1, \ldots, y_n)$ such that y_i is computed from $\phi, x_1, \ldots, x_i$, where ϕ is the content of the advice tape, i. e., an infinite binary sequence.* A *is* c-competitive with advice complexity $b(n)$ *if there exists a non-negative constant α such that, for every n and for any input sequence I of length at most n, there exists some ϕ such that*

$$\mathrm{cost}(\mathtt{A}^\phi(I)) \leq c \cdot \mathrm{cost}(\mathtt{Opt}(I)) + \alpha$$

and at most the first $b(n)$ bits of ϕ have been accessed during the computation of the solution $\mathtt{A}^\phi(I)$.

Although the advice complexity $b(n)$ is a function of n, we will abbreviate the number of advice bits by b to get an easier notation. We define the terms strict competitiveness and optimal online algorithms with advice analogously to Definition 1.3. Moreover, online algorithms with advice for maximization problems and their competitive ratio are defined in the same way as for the deterministic (or randomized) case. Similar to the case of

randomized online algorithms, we write $\mathtt{A}(I)$ $[\mathcal{A}]$ instead of $\mathtt{A}^{\phi}(I)$ $[\mathcal{A}^{\phi}]$ if ϕ is clear from the context. If $\mathtt{A}$ accesses b bits of the advice tape during some computation, we say that b *advice bits are communicated* to $\mathtt{A}$ or that $\mathtt{A}$ *uses* b *bits of advice*. The advice complexity of $\mathtt{A}$ gives an upper bound on the number of communicated advice bits, depending on the size n of the input. The game between the three parties in our model can be described as follows.

1. Adv knows both the online algorithm A and the oracle O. Moreover, Adv knows the number of advice bits b (which depends on the size of the input n) that A uses.
2. Using this knowledge, Adv constructs an input I for A.
3. O inspects I and prepares the advice tape with content ϕ.
4. A produces the output depending on both the input I and the advice ϕ, while accessing at most the first b bits of ϕ.

A schematic view of the model is depicted in Figure 1.4. Note that we can think of an online algorithm A with advice as an algorithm that chooses, depending on the advice it is given, from a set of deterministic strategies we denote by $\mathrm{Alg}(\mathtt{A})$.

Observation 1.12. *Let* A *be an online algorithm that uses* b *bits of advice. Then* Adv *can treat* A *as a set of* 2^b *different deterministic online algorithms without advice. In particular,* Adv *knows each of the* 2^b *algorithms.*

This observation allows us to use two different techniques to show lower bounds on the number of advice bits required to obtain a certain output quality: (i) The hardness can be shown by a combinatorial argument. Suppose that there is an online algorithm A that uses b bits of advice. Thus, there must be an advice tape such that reading the first b bits enables A to distinguish a class of relevant inputs such that A is able to achieve a desired competitive ratio for all of them. The goal is then to show that there are different inputs that cannot be distinguished sufficiently. Since Adv knows O, it also knows these inputs. (ii) Using Observation 1.12, we can also do a direct proof. Here, we aim at designing an adversary Adv that creates an input instance such that any deterministic algorithm from $\mathrm{Alg}(\mathtt{A})$ fails to meet some required properties. This way, Adv basically deals with all possible deterministic strategies at once. Note that, although it may seem counterintuitive, these two points of view give Adv the same power (see Chapter 8).

For many of our techniques, we need to encode a natural number x that is at most n on the advice tape. Clearly, we can bound the number of bits necessary to do this from above by $\lceil \log(n+1) \rceil$. However, for us, x is usually non-zero, and we may therefore save one bit by the following observation, which we employ implicitly in such cases.

Observation 1.13. *Let* $x \in \{1, \ldots, n\}$. O *writes* $x-1$ *onto the advice tape using at most* $\lceil \log n \rceil$ *bits;* A *then decodes the number read, adding* 1 *to the result.*

When bounding numbers from below (i. e., when giving lower bounds), we usually omit the ceilings for the sake of a clearer notation. Furthermore, we often need to encode some number $x \in \{1, \ldots, n\}$ in a self-delimiting way; the following observation enables us to do this in a straightforward fashion.

Observation 1.14. *Let* $x \in \{1, \ldots, n\}$, *which means that we need* $m := \lceil \log x \rceil$ *bits to communicate* x *by encoding* $x-1$ *on the advice tape (see Observation 1.13). Moreover,*

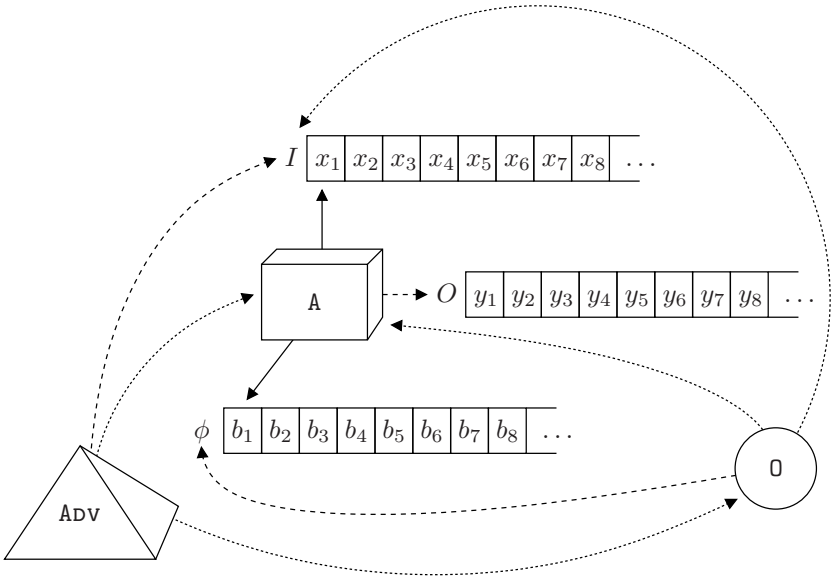


Figure 1.4: Schematic view of the model used in this thesis. For lines pointing from X to Y, a dotted line means that X knows the algorithm or sequence Y and a dashed line indicates that X creates the sequence Y. The solid lines pointing from A to the sequences I and ϕ mean that A uses them during runtime to create the output.

suppose that A *needs to know the end of the binary representation of* $x-1$ *on the advice tape for its further computations. We distinguish two cases depending on the size of* x*.*

Case 1. *Suppose that* $x-1=0$ *or* $x-1=1$*. In this case, the first bit of the advice tape is* 0 *and the second bit encodes* $x-1$*. Thus, two bits are read by* A *in total.*

Case 2. *Suppose that* $x-1\geq 2$ *and thus* $m\geq 2$*. Note that, since* m *is again strictly positive,* $\lceil \log m \rceil$ *bits are sufficient to communicate* m *by encoding* $m-1$*. Now, at first, the length of* $m-1$ *is written onto the advice tape using* $2\lceil \log m \rceil$ *bits in the following way: The bits belonging to the binary representation of* $m-1$ *are written on odd positions of the tape (the first bit on the advice tape is thus always* 1*), while the even positions are* 0 *as long as the previous odd bit still belongs to the binary representation of* $m-1$*. Thus, a* 1 *at an even position marks the end of the binary representation of* m*.*

Obviously, A *knows the length of* $m=\lceil \log x \rceil \leq \lceil \log n \rceil$ *afterwards and therefore* x*. To sum up, at most*

$$\max\{2, 2\lceil \log \lceil \log n \rceil \rceil + \lceil \log n \rceil\}$$

advice bits are sufficient to be communicated to A *in total. As we are usually interested in large values of* x*, we only consider the second argument of the* max*-function (as the first one is only valid for* $x=1$ *and* $x=2$*).*

To clarify this idea, see Table 1.1; suppose that the number $x=65536$ has to be encoded on the advice tape in a self-delimiting way and that A knows that any potential number to be encoded is strictly positive. As stated by Observation 1.14, O writes 65535 onto the advice tape, using $\lceil \log 65536 \rceil = 16$ bits. We assume that A does not know the length of x in binary, but it does know that this length, again, is non-zero. Therefore, O first writes the number 15 onto the advice tape using $\lceil \log 16 \rceil = 4$ bits; it needs another 4 bits to

x	$x-1$	$m-1$	$\lceil \log m \rceil$	advice string
1	0	–	–	0 0
2	1	–	–	0 1
3	2	1	1	1 1 1 0
4	3	1	1	1 1 1 1
5	4	2	2	1 0 0 1 1 0 0
6	5	2	2	1 0 0 1 1 0 1
7	6	2	2	1 0 0 1 1 1 0
8	7	2	2	1 0 0 1 1 1 1
9	8	3	2	1 0 1 1 1 0 0 0
10	9	3	2	1 0 1 1 1 0 0 1
⋮	⋮	⋮	⋮	⋮
65536	65535	15	4	1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
				$2\lceil \log \lceil \log x \rceil \rceil$ $\lceil \log x \rceil$
⋮	⋮	⋮	⋮	⋮

Table 1.1: Encoding a number x in a self-delimiting way; binary numbers that are used by A to compute the length of the binary representation of $m-1$ are highlighted in gray.

make this first part of the advice string self-delimiting; the concrete string is also shown in Table 1.1. Note that Observation 1.14 can be improved by iterating the same idea [117]. However, the bound given by Observation 1.14 is sufficient for our investigations.

When proving upper bounds on the advice complexity, we sometimes use the following idea. Suppose the oracle needs to communicate some n-bit string to the algorithm, but this string always contains only a few ones or only a few zeros; the next lemma describes an efficient encoding of such strings.

Lemma 1.15. *Consider an online algorithm* A *that, for every input instance, achieves a competitive ratio of c while using some n-bit advice string that contains at most n/t zeros or at least $n - n/t$ zeros, where $t > 2$ is a fixed constant. Then, it is possible to design an improved online algorithm* B *achieving the same competitive ratio c that knows the parameter t and has an advice complexity of at most*

$$\min\left\{ n \log\left(\frac{t}{(t-1)^{\frac{t-1}{t}}} \right), \frac{n \log n}{t} \right\} + 3 \log n + \mathcal{O}(1).$$

Proof. Let m be the number of all n-bit strings with at most n/t or at least $n - n/t$ zeros. To communicate any such string ϕ to the algorithm B, the oracle writes a number from $\{0, \ldots, m-1\}$ onto the advice tape that indicates the position of ϕ in the lexicographic ordering of all such possible strings. To do so, $\lceil \log m \rceil$ bits are sufficient. The algorithm, however, needs to know n and t to be able to decode the advice; since t is always known to the algorithm, it is sufficient to encode the value of n on the advice tape in a self-delimiting way. Observation 1.14 states that at most $2\lceil \log \lceil \log n \rceil \rceil + \lceil \log n \rceil$ bits suffice for this. Hence (note that $\log n > 2\lceil \log \lceil \log n \rceil \rceil$, for $n > 64$),

$$2\lceil \log \lceil \log n \rceil \rceil + \lceil \log n \rceil + \lceil \log m \rceil = \log m + 2 \log n + \mathcal{O}(1)$$

bits of advice are sufficient to achieve c-competitiveness.

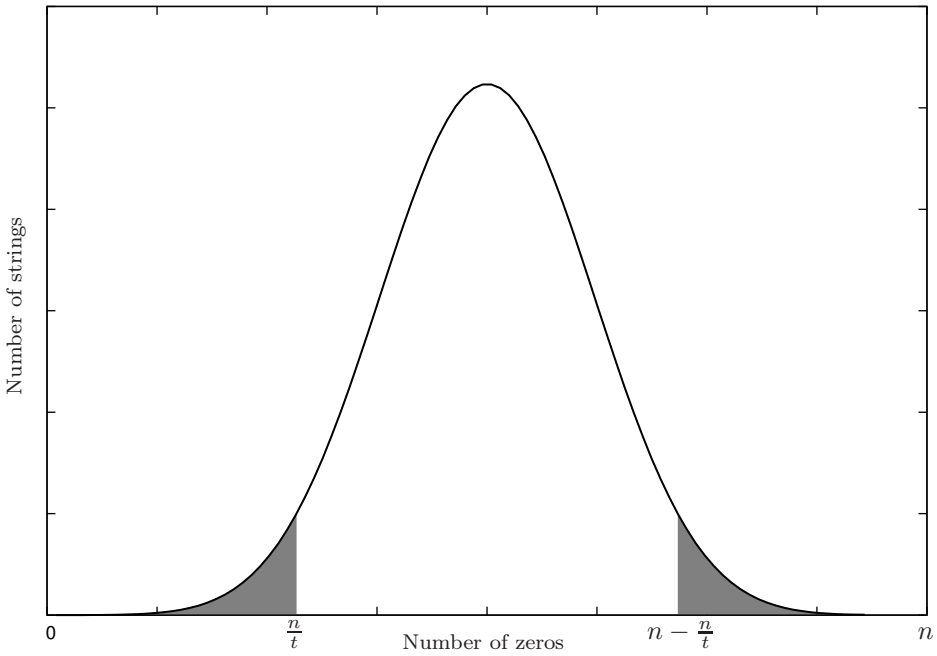


Figure 1.5: Encoding strings with at most n/t or at least $n - n/t$ zeros.

Now let us estimate the number m of possible strings (see Figure 1.5). Since $(t-1)n/t = n - n/t$ and $t > 2$ (again, see Section 1.2 about the central binomial coefficient), we get

$$m \leq 2\sum_{i=0}^{\lfloor \frac{n}{t} \rfloor} \binom{n}{i} = 2\binom{n}{0} + \sum_{i=1}^{\lfloor \frac{n}{t} \rfloor} \binom{n}{i} \leq 2 + 2\frac{n}{t}\binom{n}{\lfloor \frac{n}{t} \rfloor}$$

and subsequently

$$\log m \leq \log\left(\binom{n}{\lfloor \frac{n}{t} \rfloor}\right) + \log n + \mathcal{O}(1).$$

Substituting $q := \lfloor n/t \rfloor$ and using Stirling's approximation (we assume that $q \geq 1$, otherwise the claim is trivial), we get

$$\begin{aligned}
\log\left(\binom{n}{\lfloor \frac{n}{t} \rfloor}\right) = \log\left(\binom{n}{q}\right) &= \log\left(\frac{n!}{q!(n-q)!}\right) \\
&\leq \log\left(\frac{\sqrt{n} \cdot n^n}{\sqrt{2\pi q(n-q)} q^q (n-q)^{n-q}}\right) + \log 1.05 \\
&\leq \log\left(\frac{\sqrt{n}}{\sqrt{2\pi q(n-q)}} \cdot \frac{n^n}{q^q(n-q)^{n-q}}\right) + \mathcal{O}(1) \\
&= \frac{1}{2}\underbrace{\log\left(\frac{n}{2\pi q(n-q)}\right)}_{\leq 0} + \log\left(\frac{n^n}{q^q(n-q)^{n-q}}\right) + \mathcal{O}(1) \\
&\leq \log\left(\frac{n^n}{q^q(n-q)^{n-q}}\right) + \mathcal{O}(1).
\end{aligned}$$

It is straightforward to verify that $q^q(n-q)^{n-q}$ is decreasing in q, for $q \in (0, n/2)$, so we have

$$\begin{aligned}
\log\left(\binom{n}{\lfloor \frac{n}{t} \rfloor}\right) &\leq \log\left(\frac{n^n}{\left(\frac{n}{t}\right)^{\frac{n}{t}}\left(n-\frac{n}{t}\right)^{n-\frac{n}{t}}}\right) + \mathcal{O}(1) \\
&= n\log\left(\frac{n}{\left(\frac{n}{t}\right)^{\frac{1}{t}}\left(n-\frac{n}{t}\right)^{1-\frac{1}{t}}}\right) + \mathcal{O}(1) \\
&= n\log\left(\frac{n}{\left(\frac{n}{t}\right)^{\frac{1}{t}}\left(\frac{n}{t}(t-1)\right)^{\frac{t-1}{t}}}\right) + \mathcal{O}(1) \\
&= n\log\left(\frac{n}{(t-1)^{\frac{t-1}{t}}\left(\frac{n}{t}\right)^{\frac{1}{t}}\left(\frac{n}{t}\right)^{\frac{t-1}{t}}}\right) + \mathcal{O}(1) \\
&= n\log\left(\frac{t}{(t-1)^{\frac{t-1}{t}}}\right) + \mathcal{O}(1),
\end{aligned}$$

which concludes the proof of the first claim of the lemma. To prove the second one, recall that $\binom{n}{q} \leq n^q$ (see (1.3)), and thus

$$\log m \leq \mathcal{O}(1) + \log n + \log(n^q) \leq \frac{n}{t}\log n + \log n + \mathcal{O}(1).$$

The lemma follows. □

We are now ready to talk about the advice complexity of a concrete online problem. Again, as a starting point, we choose Paging.

1.7 Advice Complexity of the Paging Problem

In [35], Paging was one of the first problems that were studied within the framework of advice complexity as we use it here (i. e., the model described in Section 1.6). In what follows, let K denote the size of the cache as in Definition 1.4.

At first, using Lemma 1.15, it was shown that, to achieve a constant competitive ratio, it suffices to use a number of advice bits that is linear in the input size n as already indicated in the previous section.

Theorem 1.16 (Böckenhauer et al. [35]). *There exists a c-competitive online algorithm with advice for* Paging *with advice complexity*

$$n\log\left(\frac{c+1}{c^{\frac{c}{c+1}}}\right) + 3\log n + \mathcal{O}(1),$$

for every constant $c \geq 1$. ♢

As a direct consequence, we obtain that, to achieve a constant competitive ratio c, it suffices to read a constant number (smaller than 1) of advice bits on average with every request (see Figure 1.6). More precisely, with n tending to infinity, we have that, amortized, it is sufficient to read

$$\log\left(\frac{c+1}{c^{\frac{c}{c+1}}}\right)$$

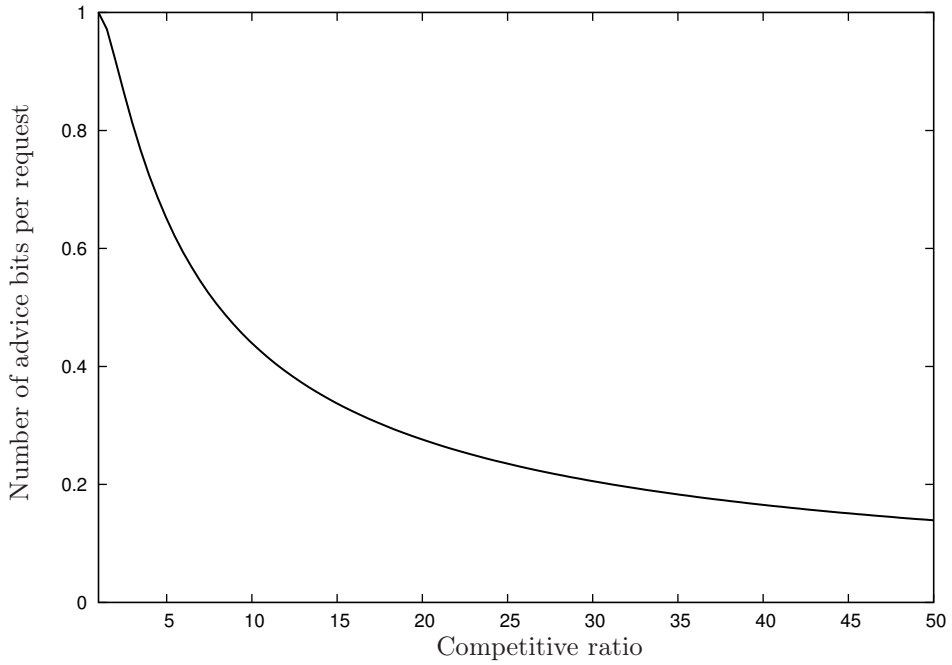


Figure 1.6: The competitive ratio and the number of advice bits sufficient per request.

advice bits per request (see Corollary 1 in [35]). Next, a lower bound on achieving a competitive ratio that is close to 1 was given.

Theorem 1.17 (Böckenhauer et al. [35]). *Let c be any constant such that $1 \leq c \leq 1.25$. For any online algorithm* A *with advice for* Paging *with a competitive ratio of c, at least*

$$\frac{1}{2K-2}\left(1+\log(3-2c)-(2c-2)\log\left(\frac{1}{2c-2}-1\right)\right)-\mathcal{O}\left(\frac{1}{n}\right)$$

advice bits are required per time step on average. The constant of the term $\mathcal{O}(1/n)$ depends on K and the parameters c and α of A *(as specified in Definition 1.3).* ♢

With c tending to 1 and K tending to infinity, we see that the above bound tends to 1/2. To get a better lower bound on the number of bits necessary to obtain an optimal result, the following result was shown.

Theorem 1.18 (Böckenhauer et al. [35]). *Any optimal online algorithm with advice for* Paging *needs to access at least*

$$\frac{1}{2K-2}\log\left(\binom{2K-2}{K-1}\right)-\mathcal{O}\left(\frac{1}{n}\right)\geq 1-\frac{\log(K-1)+d}{4(K-1)}-\mathcal{O}\left(\frac{1}{n}\right)$$

advice bits per time step on average, for some constant d not depending on K; the constant of the term $\mathcal{O}(1/n)$ depends on K. ♢

Note that the bound given in Theorem 1.18 tends to 1 with K and n tending to infinity and is therefore tight (there is a trivial upper bound, as already mentioned). It was also studied what can be done with small advice, i. e., which competitive ratio is reachable for a constant number of advice bits used overall. The following result might be considered rather surprising.

Theorem 1.19 (Böckenhauer et al. [35]). *Let $d < K$ be a power of 2. There exists an online algorithm with advice for* Paging *that uses $\log d$ advice bits and that achieves a competitive ratio of*

$$3 \log d + \frac{2(K+1)}{d}.$$

Furthermore, if only instances are considered that consist of $K+1$ pages, the bound improves by a factor of 2. ♢

Finally, the upper bound on inputs with $K+1$ different pages from Theorem 1.19 was complemented by the following theorem.

Theorem 1.20 (Böckenhauer et al. [35]). *Consider the class of inputs with $K+1$ possible pages and let d be a power of 2. Any online algorithm with advice for* Paging *that uses at most $\log d$ advice bits for these inputs has a competitive ratio of at least $K/d - \mathcal{O}(1/n)$.* ♢

Let us interpret the above results. To achieve optimality, linear advice is sufficient, and this bound is tight for large K. As we have seen in Section 1.3, deterministic strategies cannot be better than K-competitive. Interestingly, Theorem 1.19 states that merely 2 bits of advice supplied in total suffice to strictly improve the competitive ratio to $K/2 + \mathcal{O}(1)$. Furthermore, as we have pointed out in Section 1.3, randomized online algorithms can be at most $\ln K$-competitive. Theorem 1.19 yields that we are asymptotically on par with using $\log K$ bits of advice, which allows for a competitive ratio of $3 \log K + \mathcal{O}(1)$.

Chapter 2

Job Shop Scheduling

The first problem we are dealing with in terms of advice complexity is called *job shop scheduling*; a more detailed introduction and description can be found in [46, 78]. For this class of problems, we are given a number of so-called *jobs* that each need to use some of a set of given *machines* in some predefined order for some amount of time. The goal is to schedule all jobs on the machines while minimizing the running time of the machine with the highest load (called the *makespan*) by parallelizing as much of the work as possible.

In this chapter, we consider the following variant of job shop scheduling [35, 45, 88, 90, 126, 127], denoted by JSS. Let there be two jobs A and B, each of which consists of m *tasks*. These tasks must be executed in sequential order, and each task needs to be processed on a specific machine. There are exactly m such machines identified by their indices $1, \ldots, m$, and each job has exactly one task for every machine. Processing a task takes exactly one time unit, and, since both jobs need every machine exactly once, we may represent them as permutations π_A and π_B of $\{1, \ldots, m\}$. The meaning of such a permutation is that the tasks must be performed in the order specified by it and that, for every machine, the kth task must be finished before we may start with task $k+1$. If, in one time step, both jobs A and B ask for the same machine, one of them has to be delayed. The cost of a solution is given by the total time needed by both jobs to finish all tasks. As mentioned above, the goal is to minimize this time (the makespan). Let us define the problem formally.

Definition 2.1 (JSS). *Given two permutations $\pi_A = (p_1, \ldots, p_m)$ and $\pi_B = (q_1, \ldots, q_m)$, where $p_i, q_j \in \{1, \ldots, m\}$, for every $i, j \in \{1, \ldots, m\}$, an algorithm for* JSS *outputs two injective functions f_A and f_B such that $f_A, f_B \colon \{1, \ldots, m\} \to \{1, \ldots, 2m\}$, $f_A(p_i) < f_A(p_j)$ [$f_B(q_i) < f_B(q_j)$] if and only if $i < j$, and $f_A(p_i) \neq f_B(q_j)$ if $p_i = q_j$. The permutations π_A and π_B arrive successively, i. e., only p_1 and q_1 are known at the beginning and p_{i+1} [q_{j+1}] is revealed after p_i [q_j] has been processed; the aim is to minimize the makespan* $\max\{f_A(p_m), f_B(q_m)\}$.

While the general offline job shop scheduling problem is well known to be $\mathcal{NP}$-hard (shown by Garey, Johnson, and Sethi [79]), it is obvious that the considered special case is efficiently solvable in an offline scenario; as we see in the following paragraph, we simply need to calculate a shortest path on a sparse, directed, acyclic graph which can be done in linear time [58].

We use the following graphical representation [45, 81, 88, 147], which was used for the first time by Akers [4]. Consider an $(m \times m)$-grid where we label the x-axis with π_A and the

y-axis with π_B. The cell (p_i, q_j) models that, in the corresponding time step, A processes a task on machine p_i while B processes a task on q_j. A feasible schedule for the induced instance of JSS is a path that starts at the upper left-hand corner of the grid and leads to the lower right-hand corner. It may use diagonal steps whenever $p_i \neq q_j$. However, if $p_i = q_j$, both A and B ask for the same machine at the same time, and therefore one of them has to be delayed. In this case, we say that A and B collide, and we call the corresponding cells in the grid *obstacles* (see Figure 2.1). If the algorithm has to delay a job, we say that it *hits an obstacle* and may therefore not make a diagonal step, but either a horizontal or a vertical one. In the first case, B gets delayed, in the second case, A gets delayed.

Observation 2.2. *The following facts are well known [35, 88, 90].*

(i) Since π_A and π_B are permutations, there is exactly one obstacle per row and exactly one obstacle per column, for every instance.

(ii) It follows that there are exactly m obstacles overall, for every instance.

(iii) Every optimal solution has a cost of at least m, and thus every online algorithm is 2-competitive or better.

(iv) Every feasible solution makes exactly as many horizontal steps as it makes vertical ones. We call the number of horizontal [vertical] steps the delay *of the solution.*

(v) The cost of a solution is equal to m plus the delay of the solution.

(vi) Hitting an obstacle causes an additional cost of at most 1 (in certain situations even none) since one diagonal step can be simulated by exactly one horizontal and one vertical step.

Let diag_0 denote the main diagonal (from $(1,1)$ to (m,m)) in the grid. The diagonal that has a distance of i from diag_0 and lies below [above] it is denoted by diag_{-i} [diag_i]. Similar to [90], for any odd d, we consider a certain set of *diagonal strategies*

$$\mathcal{D}_d := \left\{ D_i \;\middle|\; i \in \left\{ -\frac{d-1}{2}, \ldots, \frac{d-1}{2} \right\} \right\},$$

where D_j is the strategy to move to the starting point of diag_j with j steps, to follow diag_j when possible, and to avoid any obstacle by making a horizontal step directly followed by a vertical one (thus returning to diag_j). Note that it is crucial for our analysis that the algorithm *returns* to the diagonal even though there might be situations where it is an advantage not to take the vertical step after the horizontal one.

In the following two sections, we give lower and upper bounds on the number of advice bits needed to achieve a certain output quality. Doing so, we improve and generalize some of the results obtained in [35].

2.1 Optimality

First, let us discuss the amount of information both sufficient and necessary for an online algorithm to produce an optimal solution for JSS. Hromkovič et al. have proven the following lemma [90].

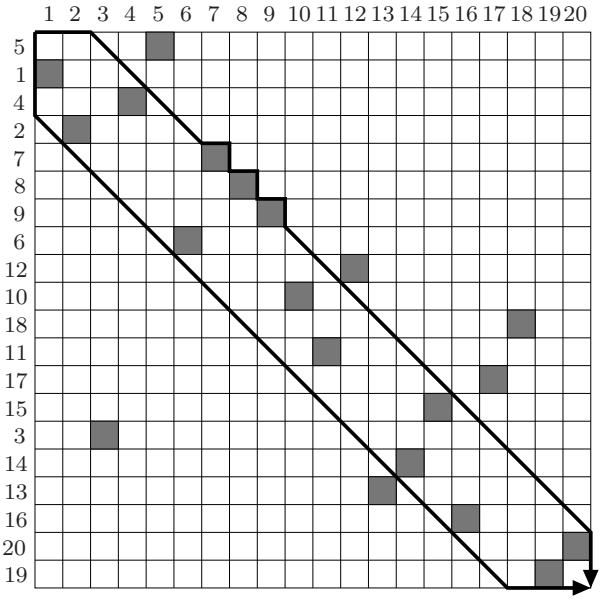


Figure 2.1: An example input with two jobs each of size 20 and the strategies D_{-3} and D_2; obstacles are marked by filled cells.

Lemma 2.3 (Hromkovič et al. [90]). *For every instance of* JSS*, there exists an optimal solution that has a cost of at most* $m + \lceil \sqrt{m} \rceil$. ♢

Using this result, the following theorem was shown in [35].

Theorem 2.4 (Böckenhauer et al. [35]). *For* JSS*, there is an optimal online algorithm with advice that uses at most* $2\lceil \sqrt{m} \rceil$ *advice bits for any instance.* ♢

The strategy is to get one bit of advice for every obstacle that is hit indicating whether to move horizontally or vertically to bypass it. Since we know that there always is an algorithm which makes at most $\lceil \sqrt{m} \rceil$ vertical and $\lceil \sqrt{m} \rceil$ horizontal steps while hitting at most $2\lceil \sqrt{m} \rceil$ obstacles, the claim follows. We now improve this upper bound by compressing the advice strings.

Theorem 2.5. *For* JSS*, there is an optimal online algorithm* A *with advice that uses at most* $2\lceil \sqrt{m} \rceil - \frac{1}{4} \log m$ *advice bits for any instance.*

Proof. Indeed, there are $2^{2\lceil \sqrt{m} \rceil}$ possible strings of length $2\lceil \sqrt{m} \rceil$ out of which the oracle provides one to the online algorithm thus using $2\lceil \sqrt{m} \rceil$ bits in the proof mentioned above. If the solution can be represented by a shorter string, we just append zeros to obtain a string of length $2\lceil \sqrt{m} \rceil$. Recall that A knows m and therefore $\lceil \sqrt{m} \rceil$. The crucial part is that all of these strings have a very nice structural property: due to Observation 2.2 (iv), they contain as many ones as they contain zeros. From (1.4), it immediately follows that, for a fixed m, there exist

$$\binom{2\lceil \sqrt{m} \rceil}{\lceil \sqrt{m} \rceil} < \frac{4^{\lceil \sqrt{m} \rceil}}{\sqrt{\pi \lceil \sqrt{m} \rceil}}$$

such strings. Enumerating all possible strings in canonical order and then merely communicating the index of the specific string gives that it suffices to use

$$\log\left(\frac{4^{\lceil\sqrt{m}\rceil}}{\sqrt{\pi\lceil\sqrt{m}\rceil}}\right) = \lceil\sqrt{m}\rceil \cdot \log 4 - \log\left(\sqrt{\pi\lceil\sqrt{m}\rceil}\right) \leq 2\lceil\sqrt{m}\rceil - \frac{1}{4}\log m$$

bits of advice. □

Next, we give a lower bound on optimality.

Theorem 2.6 (Böckenhauer et al. [35]). *For any $\varepsilon > 0$, at least*

$$\left\lfloor \frac{\sqrt{16m+9}-11}{8} \right\rfloor = \frac{\sqrt{m}}{2} - \varepsilon$$

advice bits are necessary to produce optimal output for any online algorithm with advice for JSS. ♢

Using a similar technique as in the proof of this theorem, we improve the bound by a factor of $\sqrt{2}$.

Theorem 2.7. *Let $\varepsilon > 0$. Any online algorithm with advice for* JSS *needs to use at least $\sqrt{m/2} - \varepsilon$ advice bits to be optimal.*

Proof. Let k be even and let m be a multiple of $2k + 13$. Consider the instance shown in Figure 2.2 that consists of three *levels* that are of sizes $k + 1$, 11, and k. Additionally, there is one row and one column we need to place spare obstacles in what follows. Suppose that exactly one out of the two light gray obstacles b_1 and b_2 is missing.

Obviously, there is an optimal solution that starts at the upper left-hand corner and follows the main diagonal until it hits the first obstacle c that is in its way. Depending on which one of the next two obstacles b_1 and b_2 is not present, it makes a horizontal or a vertical step to avoid the present one. It then follows diag_1 [diag_{-1}] for exactly 5 steps after which it hits another obstacle, which it bypasses by returning to the main diagonal. Thereafter, it does not hit any other obstacle until it reaches the lower right-hand corner. In total, the number of non-diagonal steps is exactly 2 (one horizontal and one vertical step, see Figure 2.2).

We now construct an instance I that consists of

$$s := \frac{m}{2k+13}$$

such sub-instances, which we call *widgets* in what follows. All these widgets are placed consecutively on the main diagonal (i. e., for every widget, its main diagonal is a part of the main diagonal of I) such that they do not overlap. For now, suppose that no optimal solution can diverge from the main diagonal by more than k (which means it cannot leave the gray field in Figure 2.2, which we call *active zone*). Obviously, an optimal solution for I enters every widget at its upper left-hand corner and leaves it at its lower right-hand corner, acting as described above in between. It follows that every optimal solution has a delay of exactly s.

Consider a feasible solution $\mathcal{B}$ that acts optimally in the first $i \geq 0$ widgets after which, in widget $i + 1$, it hits, without loss of generality, b_1. This means that $\mathcal{B}$ has to make strictly more than two non-diagonal steps in this widget. Returning to the main diagonal

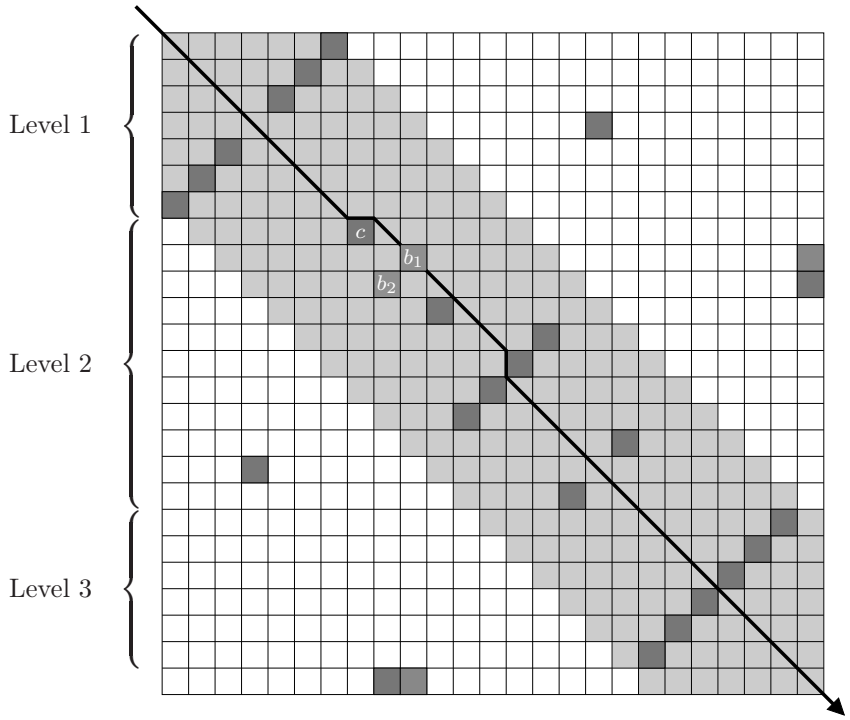


Figure 2.2: The widget of size $(2k + 13) \times (2k + 13)$ used to prove Theorem 2.7 and an optimal solution making one horizontal and one vertical step.

immediately (i. e., making a vertical step) results in being forced to make at least four non-diagonal steps. Since $\mathcal{B}$ has to make 2 non-diagonal steps also in all following widgets, it clearly is not optimal. The same holds if c is just bypassed by a horizontal step directly followed by a vertical step.

Recall that $\mathcal{B}$ may not leave the active zone. The only remaining strategy is to get further away from the main diagonal (again implying four non-diagonal steps in widget $i + 1$) and to enter the following widgets at a diagonal different from, without loss of generality above, diag_0. Suppose that $\mathcal{B}$ moves to diag_2 after hitting b_1 which also causes a total of four non-diagonal steps within widget $i+1$. After that, $\mathcal{B}$ may enter widget $i+2$ at diag_2, diag_4, or diag_6. However, at least two non-diagonal steps are necessary within it, and this obviously holds for all of the following widgets as well. The same observation can be made for any other even diagonal with a distance of less than k from the main diagonal. If $\mathcal{B}$ leaves the widget at an odd diagonal, it was forced to make one additional non-diagonal step. Afterwards, it is able to pass level 1 in the next widget without making a non-diagonal step. However, if it only makes one non-diagonal step at level 3, it leaves widget $i + 2$ at an even diagonal, resulting in at least two non-diagonal steps in widget $i + 3$.

It remains to choose k such that, if any solution leaves the active zone, it cannot be optimal. We conclude

$$\frac{m}{2k + 13} + 1 \leq k \iff m \leq 2k^2 + 11k - 13,$$

which is ensured if

$$k \geq \sqrt{\frac{m}{2} + \frac{225}{16}} - \frac{11}{4}.$$

Since k must be a natural number, we can safely set

$$k := \left\lceil \sqrt{\frac{m}{2} + \frac{225}{16}} - \frac{11}{4} \right\rceil.$$

Thus, we get

$$s = \frac{m}{2\left(\left\lceil \sqrt{\frac{m}{2} + \frac{225}{16}} - \frac{11}{4} \right\rceil\right) + 13} \geq \frac{m}{\sqrt{2m + \frac{225}{4}} + \frac{19}{2}} = \sqrt{\frac{m}{2}} - \varepsilon.$$

Until now, we did not talk about algorithms with advice. To this end, suppose that we consider the class $\mathcal{I}$ of all possible inputs as constructed in the above way. Since, for every widget, there are two possibilities (either b_1 or b_2 is missing), we have that $|\mathcal{I}| = 2^s$. Furthermore, for every $I \in \mathcal{I}$, there is *one unique* optimal solution, and for every two different $I, I' \in \mathcal{I}$, the corresponding optimal solutions are never the same.

Therefore, as a direct consequence of the pigeonhole principle, an optimal online algorithm needs to read s bits of advice at least. □

2.2 Small Competitive Ratio

Intuitively speaking, we now show that there always exists a cheap solution close to the main diagonal diag_0, which implies that only a few bits of advice are necessary to achieve a good result. In what follows, d is always a small odd constant which is independent of the input size. Furthermore, let $\gamma := d^2/4 - d$; note that $\gamma > -1$.

Recall that we call the number of horizontal [vertical] steps of a solution its delay; the cost of the solution is always equal to m plus its delay (see Observation 2.2 (v)). More specifically, the delay of a diagonal strategy D_i is $|i|$ plus the number of obstacles on diag_i. If $i \geq 0$, the strategy D_i makes i horizontal steps to reach diag_i and then a single horizontal step for every obstacle on diag_i. The argument for $i \leq 0$ is similar, but using vertical steps. Before we continue, we need the following lemma.

Lemma 2.8. *There is a diagonal strategy in $\mathcal{D}_d$ that has a delay of at most $\lceil \frac{\gamma+m}{d} \rceil$.*

Proof. As we have seen in Observation 2.2 (ii), there are exactly m obstacles in the whole grid that represents the instance at hand. Towards contradiction, suppose that the claim is wrong. Therefore, each of the considered strategies has a cost of at least $m + \lceil \frac{\gamma+m}{d} \rceil + 1$. This means that at least $\lceil \frac{\gamma+m}{d} \rceil + 1$ obstacles are on the main diagonal, at least $\lceil \frac{\gamma+m}{d} \rceil$ obstacles are on diag_{-1} and diag_1, in general at least

$$\left\lceil \frac{\gamma + m}{d} \right\rceil + 1 - i$$

obstacles have to be on diag_{-i} and diag_i, and finally

$$\left\lceil \frac{\gamma + m}{d} \right\rceil - \frac{d-3}{2}$$

obstacles are on $\text{diag}_{-(d-1)/2}$ and $\text{diag}_{(d-1)/2}$. Hence, we get a total of

$$\begin{aligned}
&\left\lceil \frac{\gamma+m}{d} \right\rceil + 1 + 2 \sum_{i=1}^{(d-1)/2} \left(\left\lceil \frac{\gamma+m}{d} \right\rceil + 1 - i \right) \\
&\geq \frac{\gamma+m}{d} + 1 + \left(\frac{\gamma+m}{d} + 1 \right)(d-1) - 2 \sum_{i=1}^{(d-1)/2} i \\
&= \left(\frac{\gamma+m}{d} + 1 \right) d - \frac{d^2-1}{4} = m + d + \frac{d^2}{4} - d - \frac{d^2-1}{4}
\end{aligned}$$

obstacles, which is strictly more than m and thus contradicts our assumption. □

We are now ready to prove the following theorem.

Theorem 2.9. *For every d, there exists an online algorithm $\mathtt{A}_d$ with advice for* JSS *that reads $\lceil \log d \rceil$ advice bits and that achieves a competitive ratio of*

$$1 + \frac{1}{d} + \frac{d}{4(m+1)} - \frac{d+1}{d(m+1)}.$$

Proof. Let $\mathtt{A}_d$ know d and read $\lceil \log d \rceil$ bits in total that tell the algorithm which out of the diagonal strategies from $\mathcal{D}_d$ to follow. As we have shown in Lemma 2.8, one out of these strategies has a delay of at most $\lceil \frac{\gamma+m}{d} \rceil$. Note that, if the optimal solution has cost m, this solution must take the main diagonal. But in this case, $\mathtt{A}_d$ is always optimal, because there are no obstacles on diag_0 and the corresponding delay is therefore 0. Hence, without loss of generality, we may assume a lower bound of $m+1$ on the cost of the optimal solution. To conclude from the above, we get a competitive ratio of $\mathtt{A}_d$ of at most

$$\frac{m + \left\lceil \frac{d^2/4-d+m}{d} \right\rceil}{m+1} \leq \frac{m + \frac{d^2/4-d+m}{d} + 1}{m+1} = \frac{m + \frac{m}{d} + \frac{d}{4}}{m+1} = 1 + \frac{1}{d} + \frac{d}{4(m+1)} - \frac{d+1}{d(m+1)}$$

as we claimed. □

Figure 2.3 shows how the competitive ratio of $\mathtt{A}_d$ behaves depending on the number of advice bits. In [90], it was shown that, for any $\varepsilon > 0$, any deterministic online algorithm without advice cannot be better than $(1 + 1/3 - \varepsilon)$-competitive. On the other hand, the competitive ratio of $\mathtt{A}_d$ tends to $1 + 1/7$ (recall that d is odd) with only 3 bits of advice, for m tending to infinity. Hence, we can beat deterministic strategies with only very little additional information. Recall that a similar result was shown for Paging in [35], as we mentioned in Section 1.7.

In Theorem 2.9, we did not care about the uniformity of $\mathtt{A}_d$ for different values of d. It is, however, not difficult to avoid the non-uniformity, i. e., to define a single algorithm $\mathtt{A}$ that reaches a competitive ratio tending to $1 + 1/d$, for any d. To do so, the oracle first encodes the number $\lceil \log d \rceil$ on the advice tape; this has to be done in a self-delimiting way. From Observation 1.14, it follows that at most $2\lceil \log \lceil \log d \rceil \rceil$ additional advice bits are sufficient to do so.

Corollary 2.10. *There exists an online algorithm $\mathtt{A}$ with advice for* JSS *that achieves a competitive ratio tending to $1 + 1/d$ with growing m with advice complexity $\lceil \log d \rceil + 2\lceil \log \lceil \log d \rceil \rceil$.* □

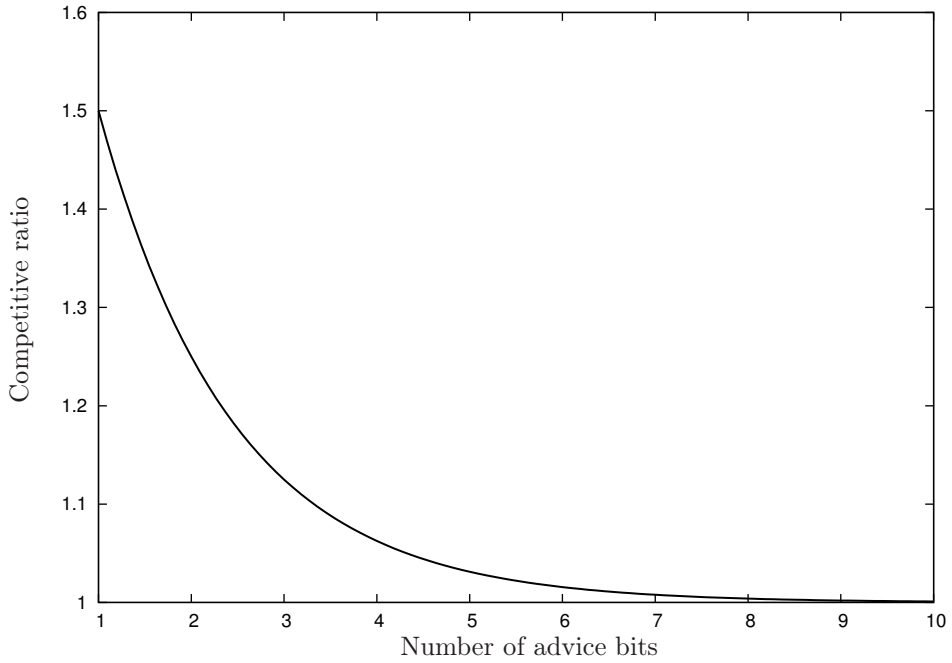


Figure 2.3: The competitive ratio of $\mathtt{A}_d$ depending on $\log d$, for m tending to infinity.

It is not difficult to see that the analysis of the algorithm $\mathtt{A}_d$ is almost tight for every d. To show this, we give a construction that blocks all diagonals the algorithm chooses from. Following any of the blocked diagonals causes the algorithm to have a cost of at least $m + m/d$, whereas an optimal solution has a cost of exactly $m + 1$.

Theorem 2.11. *For any d and any $\varepsilon > 0$, the competitive ratio of the algorithm $\mathtt{A}_d$ is not better than $1 + 1/d - \varepsilon$.*

Proof. Let m and l be even. We now describe how to sufficiently delay every possible diagonal strategy. Suppose we want to make sure that every strategy has a delay of at least l. At first, we place l obstacles in the center of the main diagonal, i. e., in the cells $(m/2-l/2+1, m/2-l/2+1)$ to $(m/2+l/2, m/2+l/2)$. For now, let us focus on the cells that are in the lower right-hand quadrant of the $(m \times m)$-grid. For each $i \in \{1, \ldots, (d-1)/2\}$, we create one block of obstacles; the block corresponding to i consists of $l - i$ obstacles. All of these obstacles are put on the ith diagonal above the main one, in consecutive rows, just below the rows used by block $i - 1$. In particular, the obstacles of block 1 are located in the cells

$$\left(\frac{m+l}{2} + 2, \frac{m+l}{2} + 1\right), \ldots, \left(\frac{m+l}{2} + l, \frac{m+l}{2} + l - 1\right),$$

the obstacles of block 2 are located in the cells

$$\left(\frac{m+l}{2} + l + 2, \frac{m+l}{2} + l\right), \ldots, \left(\frac{m+l}{2} + 2l - 1, \frac{m+l}{2} + 2l - 3\right),$$

etc. Hence, we need to use $l - i$ rows and $l - i + 1$ columns to build block i (the first column of the block is empty since block i is on a different diagonal than block $i - 1$).

To be able to successfully build all of the blocks, we need at least

$$\frac{l}{2} + 1 + (l-1) + 1 + (l-2) + \cdots + 1 + \left(l - \frac{d-1}{2}\right)$$

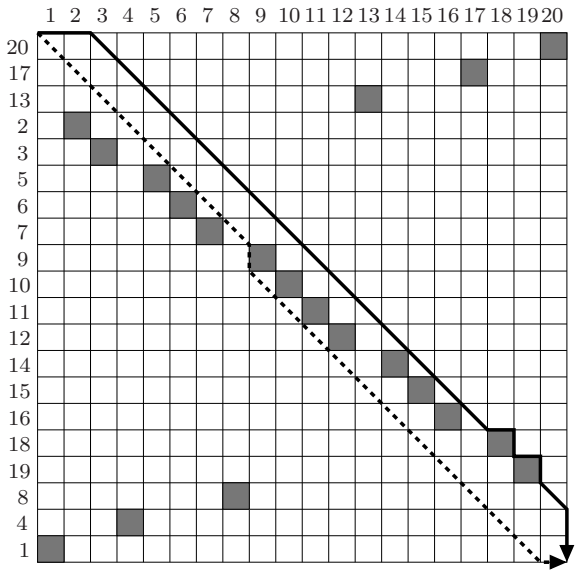


Figure 2.4: A hard instance for $\mathcal{D}_5$.

columns. Clearly, if there are enough columns available, there are enough rows available as well. Since we have exactly $m/2$ columns, we have to make sure that

$$\begin{aligned} \frac{m}{2} &\geq \frac{l}{2} + \sum_{i=1}^{(d-1)/2} (1 + l - i) \\ \iff \frac{m}{2} &\geq \frac{l}{2} + \frac{d-1}{2}(1+l) - \frac{d^2-1}{8} \\ \iff l &\leq \frac{m + \frac{d^2+3}{4} - d}{d}. \end{aligned}$$

We can ensure this by taking l to be the smallest even integer such that

$$l \geq \frac{m + \frac{d^2+3}{4} - d}{d} - 2.$$

The same construction can be performed in the upper left-hand quadrant in a symmetric way. In every block, there is one free column. It remains to use the rows not used by any block (nor by the obstacles in the main diagonal) to put a single obstacle in every such free column. To do so, we use the upper right-hand and lower left-hand quadrant. It is straightforward to observe that this is always possible, even without using any diagonal neighboring the main one.

An example of this construction for $m = 20$, $l = 4$, and $\mathcal{D}_5$ is shown in Figure 2.4. It is clear that any optimal solution has a cost of exactly $m+1$: An optimal solution follows the main diagonal until the first obstacle is hit. Afterwards, it makes one vertical step and follows the first diagonal below the main one (i. e., diag_{-1}).

$\mathtt{A}_d$ calculates a solution with a delay of at least l, i. e., with a cost of at least

$$m + l \geq m + \frac{m + \frac{d^2+3}{4} - d}{d} - 2 \geq (m+1)\left(1 + \frac{1}{d}\right) - 4 - \frac{1}{d} + \frac{d^2+3}{4d}.$$

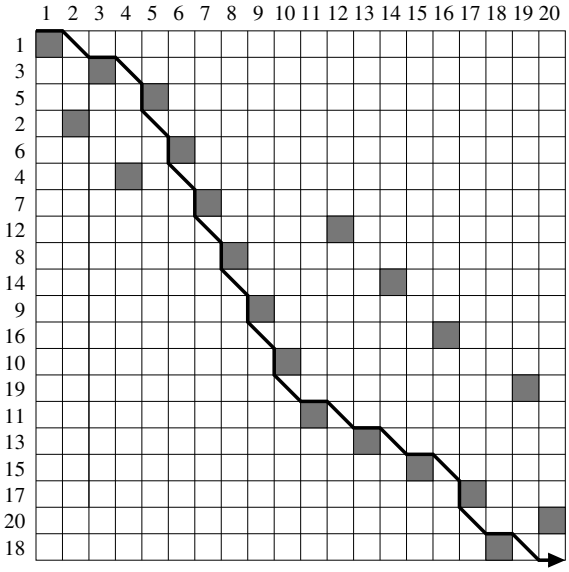


Figure 2.5: An example of how to place the obstacles in such a way that any deterministic algorithm cannot make two consecutive diagonal steps as presented in [90].

Therefore, the competitive ratio of $\mathtt{A}_d$ on this instance is at least

$$\frac{(m+1)\left(1+\frac{1}{d}\right)-4-\frac{1}{d}+\frac{d^2+3}{4d}}{m+1} = 1+\frac{1}{d}+\frac{d^2-16d-1}{4d(m+1)} \geq 1+\frac{1}{d}-\varepsilon \tag{2.1}$$

if m is large enough, which we can assume for infinitely many m. Finally, note that, if $d \geq 17$, (2.1) holds even if $\varepsilon = 0$. □

Up to this point, we have shown that, with a small constant number of advice bits, it is possible to perform very well. Additionally, in Theorem 2.7, we have proven that, for any arbitrarily small $\varepsilon > 0$, $\sqrt{m/2}-\varepsilon$ advice bits are necessary to create optimal output.

This poses the question of whether we can be $(1+o(1))$-competitive with reading a constant number of advice bits, i. e., whether it suffices to use a constant number of bits to get arbitrarily close to the optimal solution. In the following, we disprove this.

Theorem 2.12. *For any $\varepsilon > 0$, any online algorithm with advice for* JSS *that reads b advice bits cannot be better than*

$$\left(1+\frac{1}{3\cdot 2^b}-\varepsilon\right)\text{-competitive.}$$

Proof. Recall that it is known that any deterministic online algorithm $\mathtt{B}$ for JSS has a competitive ratio of at least $4/3-\varepsilon$, for any $\varepsilon > 0$ [90]. There exists an adversary that can make sure that, while $\mathtt{B}$ has not yet hit a border, every second step of $\mathtt{B}$ is not a diagonal one (which then, after some further estimations, results in a delay of at least $m/3$): The intuitive idea is that, after every diagonal step of $\mathtt{B}$, the algorithm reaches a column and a row in which the adversary has not yet placed any obstacle; this idea is shown in Figure 2.5. Furthermore, we already know that there always exists an optimal solution with a cost of at most $m+\lceil\sqrt{m}\,\rceil$ as stated by Lemma 2.3.

For any online algorithm that reads b bits of advice and any $\varepsilon > 0$, we find some (arbitrarily large) m and construct an input instance of size $m \times m$ such that the algorithm

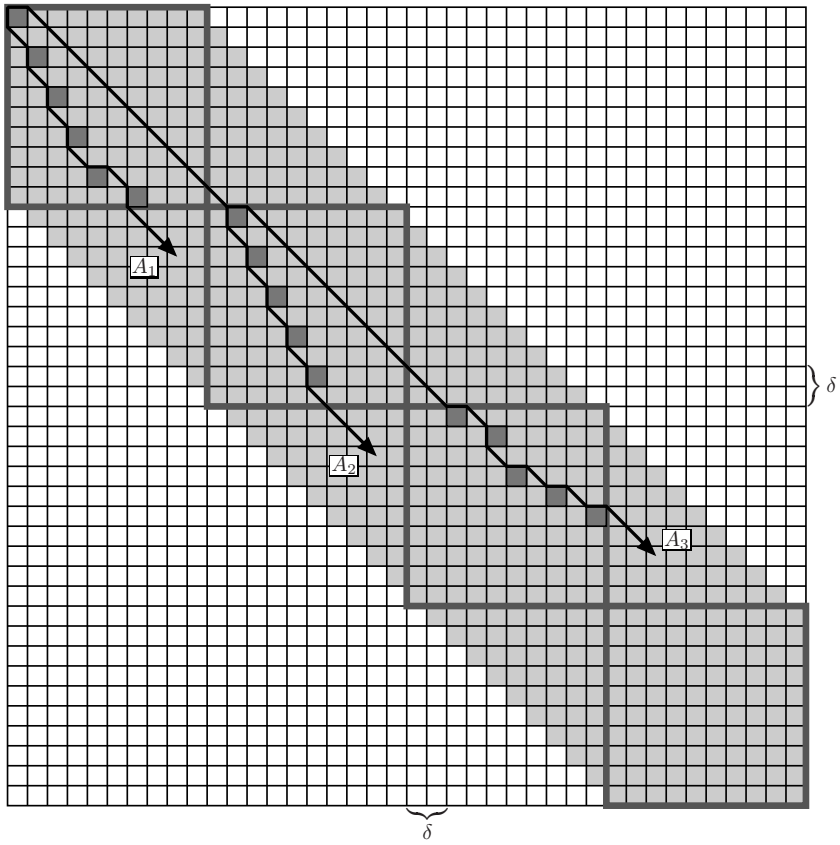


Figure 2.6: A hard instance for A as used in the proof of Theorem 2.12, which uses the construction of [90] (see Figure 2.5) 2^b times.

has a delay of at least $\frac{m}{3\cdot 2^b}$. Hence, the makespan of this algorithm is at least $m\big(1+\frac{1}{3\cdot 2^b}\big)$, and, since the optimal solution has a makespan of at most $m+\lceil\sqrt{m}\,\rceil$, the competitive ratio of the algorithm cannot be better than

$$\frac{m\big(1+\frac{1}{3\cdot 2^b}\big)}{m+\lceil\sqrt{m}\,\rceil} \geq 1+\frac{1}{3\cdot 2^b}-\varepsilon,$$

for any sufficiently large m.

In the following, let m be a multiple of 2^b. Suppose that we are now dealing with any online algorithm A with advice that reads b advice bits while processing an input of size m. We impose another virtual grid on the $(m\times m)$-grid, where each virtual cell consists of $m'\times m'$ original cells and $m' := m/2^b$. Let us now consider the 2^b virtual cells on the main diagonal (as shown in Figure 2.6). We call these cells *blocks* and label them $S_1,\ldots,S_{2^b}$.

Furthermore, similar to the proof of Theorem 2.7, we call all original cells that have a deviation of at most m' from the main diagonal the *active zone* (marked gray in Figure 2.6). Any algorithm that leaves this zone at any point makes at least m' horizontal [vertical] steps and thus has a delay of at least $m' > \frac{m}{3\cdot 2^b}$. We may therefore assume that the given algorithm never leaves the active zone.

Following Observation 1.12, we may think of A as a set $\mathrm{Alg}(\mathtt{A})$ of 2^b deterministic algorithms $A_1,\ldots,A_{2^b}$ we have to deal with. Without loss of generality, we may assume that each of these algorithms makes a diagonal step whenever possible [35]. We assign each deterministic algorithm $A_i\in\mathrm{Alg}(\mathtt{A})$ to exactly one block S_i. Now we construct the input

instance sequentially in a way such that all obstacles are located in some block S_i. Note that S_i spans the rows and the columns $m'i+1, \ldots, m'i+m'$. Recall that p_i denotes the ith task of the first job and q_i denotes the ith task of the second job. We thus construct the input such that $p_{m'i+1}, \ldots, p_{m'i+m'}$, as well as $q_{m'i+1}, \ldots, q_{m'i+m'}$, are permutations of the numbers $m'i+1, \ldots, m'i+m'$.

Assume that, so far, we have constructed $S_1, \ldots, S_{i-1}$. Next, we define S_i such that A_i has a delay of at least $\frac{m}{3 \cdot 2^b}$, regardless of the content of any S_j, for $j > i$. Without loss of generality, assume that A_i reaches the right border of S_{i-1} at distance δ above the main diagonal; the case that A_i reaches the bottom border of S_{i-1} is analogous. Moreover, if $i = 1$, we define $\delta := 0$, since the first algorithm A_1 starts at the top-left point of the main diagonal. In the following, we show how to ensure that A_i has a delay that is sufficiently large in block S_i while ignoring A_i's performance in all the other blocks.

Since there are no obstacles outside the blocks and we assume that A_i makes a diagonal step whenever possible, A_i makes δ diagonal steps after leaving S_{i-1} until it reaches the top border of S_i, i. e., the upper left-hand corner of the cell $(m'i+\delta+1, m'i+1)$. We assign the first δ tasks to the first job sequentially, i. e., $p_{m'i+j} := m'i + j$, for all $j \in \{1, \ldots, \delta\}$.

After A_i reaches the cell $(m'i+x, m'i+y)$, the first $m'i+x$ tasks of the first job and the first $m'i+y$ tasks of the second job must be assigned. In the sequel, we maintain the invariant that, in such a situation, only numbers up to $m'i + \max\{x, y\}$ are used for both jobs. This invariant holds before A_i reaches the cell $(m'i+\delta+1, m'i+1)$.

When A_i reaches S_i, we employ the strategy of [90] to ensure that every second step of A_i is non-diagonal: At first, we assign

$$p_{m'i+\delta+1} = q_{m'i+1} := m'i + \delta + 1,$$

thus creating an obstacle; therefore, the next step of A_i will be a non-diagonal one. Whenever A_i makes a horizontal [vertical] step, we assign the smallest possible task as the next task of the first [second] job. When A_i makes a diagonal step in cell $(m'i+x, m'i+y)$, thus reaching the upper left-hand corner of cell $(m'i+x+1, m'i+y+1)$, we assign

$$p_{m'i+x+1} = q_{m'i+y+1} := m'i + \max\{x, y\} + 1.$$

Hence, we create an obstacle and force A_i to make another non-diagonal step. It is easy to verify that we can always follow this strategy due to the validity of the invariant and that the invariant is never violated.

We use this strategy until A_i reaches the right or bottom border of S_i. Assume that A_i makes h horizontal steps, v vertical steps, and d diagonal steps in this part of the computation (i. e., in block S_i). Since every diagonal step is followed by a non-diagonal one and the first step is non-diagonal, we have $h + v \geq d$. We now give a lower bound on the total delay D of A_i on the constructed instance. Even though we have not yet constructed S_j, for $j > i$, we can proceed, because our bound will not depend on them (as we have mentioned before, we only consider the delay caused in block S_i). Recall that the total number of horizontal and vertical steps of A_i over the whole input must be equal, and D is defined as exactly this number, see Observation 2.2 (iv). We distinguish two cases depending on the relation between h and v.

Case 1. Suppose that $h \geq v$. In this case, A_i reaches the right border of S_i. Since A_i entered S_i in column $m'i+\delta+1$, there were $m' - \delta$ non-vertical steps, hence

$$m' - \delta = h + d \leq 2h + v \leq 3h.$$

Therefore, $h \geq (m'-\delta)/3$. Since A_i leaves S_{i-1} at distance δ above the main diagonal, it made at least δ horizontal steps before it entered S_i. Thus, we can bound the total number of horizontal steps of A_i, which is equal to D, by

$$\delta + h \geq \frac{m' + 2\delta}{3} \geq \frac{m'}{3}.$$

Case 2. Suppose that $h < v$. Assume that A_i leaves S_i at distance δ' above the bottom border of S_i; if A_i reaches the bottom border, $\delta' = 0$, otherwise $\delta' > 0$. Since A_i made $m' - \delta'$ non-horizontal steps in S_i, we have

$$m' - \delta' = v + d \leq 2v + h \leq 3v$$

and $v \geq (m' - \delta')/3$. After leaving S_i, A_i must make at least δ' vertical steps to end up at the main diagonal. Hence, the total number of vertical steps of A_i, which is equal to D, can be bounded by

$$\delta' + v \geq \frac{m' + 2\delta'}{3} \geq \frac{m'}{3}.$$

In both cases, A_i has a delay of at least $m'/3$. Therefore, after constructing all blocks S_i in the described way, we obtain an instance, for which every $A_j \in \mathrm{Alg}(\mathtt{A})$ has a makespan of at least $m + \frac{m}{3 \cdot 2^b}$ as we claimed. □

Using this result, an easy calculation shows that the bound from Theorem 2.9 is tight up to a multiplicative constant of

$$\frac{3 \cdot 2^b + 3}{3 \cdot 2^b + 1},$$

which tends to 1 for an increasing b.

In Section 7.1, we construct a barely random algorithm for JSS that basically uses the same ideas as the algorithm $\mathtt{A}_d$ with advice we have designed in this chapter. Moreover, in Section 7.2, we show that there exists a randomized online algorithm for JSS that does not only perform well in expectation, but almost always.

Chapter 3

Disjoint Path Allocation

In this chapter, we consider a special type of network topology where the entities are connected by one shared cable (i. e., we are dealing with a bus network as depicted in Figure 3.1). In every time step, two entities request to establish a permanent connection between each other. If this request is granted, all entities between these two are busy maintaining the connection and are thus unable to be involved in any other connection.

Formally, the *disjoint path allocation problem* (DPA for short) is defined on *paths*; a path of length m is a special graph (see Definition 1.1) $P = (V, E)$ with $m + 1$ vertices and m edges such that $E = \{\{v_i, v_{i+1}\} \mid i \in \{1, \ldots, m\}\}$. For the ease of presentation, we denote P by (v_1, v_m); moreover, a subpath of (v_1, v_m) from the vertex v_i to the vertex v_j is denoted by (v_i, v_j). For $L + 1$ entities, the above network is a path $P = (v_1, v_{L+1})$ of length L. All connections in P have a capacity of 1. For DPA (as described in [38], also referred to as *the call admission problem on a path network*), additionally a set of subpaths of P is given. Each subpath (v_i, v_j) is a so-called *call request*, i. e., a request to establish a permanent connection between the two endpoints v_i and v_j. If such a request is satisfied, no inner entity of the path may be part of any other call. Therefore, a *disjoint path allocation* is simply a set of edge-disjoint subpaths of P.

Definition 3.1 (DPA). *Given a path $P = (V, E)$, where $V = \{v_0, \ldots, v_L\}$ is a set of* entities*, and a set $\mathcal{P}$ of subpaths of P, where $|\mathcal{P}| = n$,* DPA *is the problem of finding a maximum set $\mathcal{P}' \subseteq \mathcal{P}$ of edge-disjoint subpaths of P. The subpaths $P_1, \ldots, P_n \in \mathcal{P}$ arrive in an online fashion.*

We assume that L is known to the online algorithm in advance, but that this is not the case for n. Recall that, since DPA is a maximization problem, an online algorithm $\mathtt{A}$ solving this problem is c-competitive if $\text{cost}(\textsc{Opt}(I)) \leq c \cdot \text{cost}(\mathtt{A}(I)) + \alpha$, for some constant

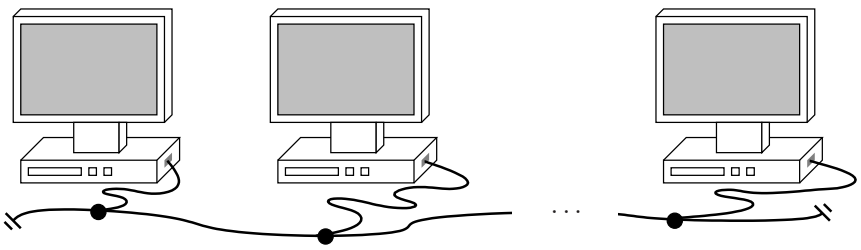


Figure 3.1: Schematic view of DPA.

α and every input I; A is strictly c-competitive if $\alpha = 0$ (see Definition 1.3). In the following, we use the terms *cost* and *gain* interchangeably.

3.1 Deterministic Algorithms

By some simple observations, it is clear that any deterministic online algorithm A is not better than strictly L-competitive [38]: An adversary ADV controlling $\mathcal{P}$ simply sends $P_1 = (v_0, v_L)$ as the first part of the input. Discarding P_1 results in ADV not sending any other subpath, and therefore A always has to satisfy the first request. ADV now sends the consecutive requests $P_2 = (v_0, v_1), P_3 = (v_1, v_2), \ldots, P_{L+1} = (v_{L-1}, v_L)$, none of which can be satisfied. Obviously, an optimal offline algorithm would discard P_1 in this case and satisfy all L requests P_2 to P_{L+1}. Note that there are exactly $L + 1 = n$ requests for this input, and A is therefore also not better than strictly $(n-1)$-competitive. This lower bound is tight: Consider a simple greedy algorithm G that always satisfies the first request given and then any other request possible.

Lemma 3.2. *The online algorithm* G *is both* $(n-1)$*-competitive and* L*-competitive.*

Proof. Since G always satisfies at least one request, we merely need to consider cases in which an optimal offline algorithm grants strictly more than $n-1$ requests to prove $(n-1)$-competitiveness. However, this is only the case if no subpath of the input intersects with any other subpath, but in this case G is optimal.

To prove that G is L-competitive, note that G satisfies at least one request, but no solution can grant more than L requests. □

Up to this point, we have only talked about strict competitiveness. Indeed, the above argument for the lower bounds does not directly carry over to general competitiveness (recall our discussion about α from Section 1.3). Suppose that, following the above strategy, the adversary sends the first request, A does not satisfy it, and no second request is therefore sent. Since we allow an additive constant α, setting $c = 1$ and $\alpha = 1$ gives

$$1 = \text{cost}(\text{OPT}(I)) \leq c \cdot \text{cost}(\text{A}(I)) + \alpha = 1 \cdot 0 + 1,$$

which means that, by this definition, A might still be 1-competitive. However, we can easily extend the above idea and prove the following lemma.

Lemma 3.3. *No deterministic online algorithm for* DPA *can be better than* $(n - \mathcal{O}(1))$*-competitive.*

Proof. Consider any c-competitive deterministic online algorithm A for DPA. By definition, there exists a fixed constant α such that $\text{cost}(\text{OPT}(I)) \leq c \cdot \text{cost}(\text{A}(I)) + \alpha$.

Let $\beta := \alpha + 1$. Moreover, for any n, let $L = \beta \cdot n$ (recall that, in this case, the graph has $\beta \cdot n + 1$ vertices). We consider a set of inputs that consist of two phases. In phase 1, there are β consecutive edge-disjoint requests

$$P_1 = (v_0, v_n), P_2 = (v_n, v_{2n}), \ldots, P_\beta = (v_{(\beta-1)n}, v_L)$$

of length $n = L/\beta$ each. If A does not satisfy the request P_i, then P_{i+1} is requested, and in the following, no requests intersecting with P_i are made. It directly follows that, in

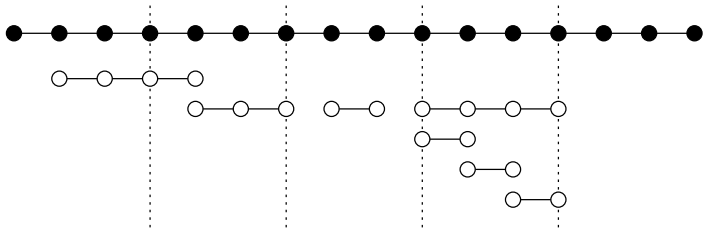


Figure 3.2: An example input for the algorithm B.

this case, Opt satisfies P_i. Therefore, if A does not satisfy any of the requests above, Opt satisfies them all. Observe that

$$\text{cost}(\texttt{Opt}(I)) \leq c \cdot \text{cost}(\texttt{A}(I)) + \alpha$$

now gives $\beta \leq c \cdot 0 + \alpha$ which is a direct contradiction. A therefore has to satisfy at least one of the first β requests. After A satisfies this request (let this be the ith one), we immediately start with phase 2. By the same idea as above, in this case, we extend the input by $n - i$ consecutive requests of length 1, all of which cannot be granted by A, because they are covered by the path A has just chosen to satisfy. It follows that Opt satisfies $n - 1$ requests, whereas A merely satisfies one request. As a result, we have

$$n - 1 = \text{cost}(\texttt{Opt}(I)) \leq c \cdot \text{cost}(\texttt{A}(I)) + \alpha = c + \alpha$$

yielding

$$c \geq n - (1 + \alpha) = n - \mathcal{O}(1)$$

as we claimed. □

To finish the discussion about comparing the concept of general competitiveness to the one of strict competitiveness, we now describe a simple online algorithm that, for every α, achieves a competitive ratio of $\lceil \frac{L}{\alpha+1} \rceil$, whereas we have already seen that, when talking about *strict* competitiveness, there is a tight lower bound of L. Consider an algorithm B that acts according to Algorithm 3.1.

Algorithm 3.1: Algorithm B for DPA

divide the input into $\alpha + 1$ segments, each of length at most $\lceil \frac{L}{\alpha+1} \rceil$;
if any request intersects with more than 1 segment
    discard;
else
    satisfy greedily;
end

Lemma 3.4. *The online algorithm* B *is* $\lceil \frac{L}{\alpha+1} \rceil$*-competitive.*

Proof. Let I be any input instance of DPA. It is clear that there can be at most α disjoint requests that are in more than one segment and that are at the same time taken by an optimal algorithm Opt. Consider any of the remaining $\text{cost}(\texttt{Opt}(I)) - \alpha$ requests that are granted by Opt. Each of them is made in some segment, and thus there is some request satisfied in this segment by B. Since the length of one segment is at most $\lceil \frac{L}{\alpha+1} \rceil$, we can

account at most that many requests satisfied by OPT to a single request satisfied by B. Hence, we have

$$\text{cost}(\text{OPT}(I)) - \alpha \leq \left\lceil \frac{L}{\alpha+1} \right\rceil \text{cost}(\text{B}(I))$$

as we claimed. □

An example input and B's strategy on this input are shown in Figure 3.2: The black vertices mark the network. Suppose the requests arrive in the order given from left to right and row after row. Here, $L = 15$, $\alpha + 1 = 5$, and $\lceil \frac{L}{\alpha+1} \rceil = \lceil \frac{15}{5} \rceil = 3$. The algorithm B satisfies the requests $2, 3$, and 4 and none else, whereas OPT satisfies the requests $1, 2, 3, 5, 6$, and 7.

Next, we want to study both lower and upper bounds on the number of advice bits needed by online algorithms to obtain a specific competitive ratio. To do so, we first have to introduce a special class of input instances on which we base some of our proofs.

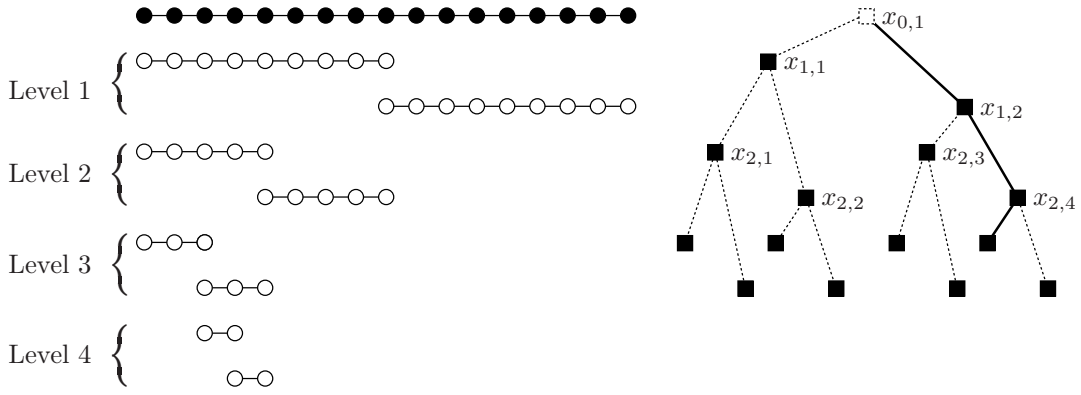
3.2 The Class $\mathcal{I}$ of Inputs

For some of the following proofs in this chapter, we consider input instances as depicted in Figure 3.3 (a). For every natural number h, we define the class $\mathcal{I}_h$ ($\mathcal{I} := \bigcup_{h \in \mathbb{N}} \mathcal{I}_h$) in the following way. Every element of $\mathcal{I}_h$ consists of $h + 1$ so-called *levels*. Level 1 consists of two edge-disjoint consecutive requests that split the line network into two parts of the same size. In general, two disjoint consecutive requests on level $i + 1$ do the same with *one* of the intervals from level i. This is iterated until two requests of size 1 appear on level $h + 1$. Figure 3.3 (a), for instance, shows an input from $\mathcal{I}_3$. It is obvious that any optimal algorithm satisfies exactly one interval on the first h levels, allowing it to satisfy both on level $h + 1$. In this example, an optimal strategy is to satisfy the second request on level 1 and level 2, the first request on level 3 and therefore being able to satisfy the two requests on level 4. Note that, if an algorithm diverges from this strategy at some point, it is not able to satisfy any more requests for this instance afterwards.

We may represent an optimal solution for any input sequence as described above by a path from the root to a leaf in a complete binary tree $\mathcal{T}$ of height h with its root on a notional level 0 (see Figure 3.3 (b)). The 2^h leaves of $\mathcal{T}$ represent the 2^h different inputs of the class $\mathcal{I}_h$. Let $\mathcal{O}pt$ denote such an optimal path for some input instance from $\mathcal{I}_h$; we say that an optimal algorithm OPT makes moves *according to this path*. For an arbitrary online algorithm A that satisfies one request on level i, we say that A makes the *correct* decision on this level if it also acts according to $\mathcal{O}pt$. Conversely, if A satisfies one interval not according to $\mathcal{O}pt$ or satisfies both requests, we say that A makes the *wrong* decision on level i; in this case, we say that A is *out* (after level i). Furthermore, for every i and an input instance $I_h \in \mathcal{I}_h$, let $\text{cost}_i(\text{A}(I_h))$ denote the overall number of requests satisfied by A up to level i. If A is out after level i, this means that, for every $j \geq i$, $\text{cost}_j(\text{A}(I_h)) = \text{cost}_i(\text{A}(I_h))$, and obviously we have $\text{cost}(\text{A}(I_h)) = \text{cost}_i(\text{A}(I_h))$. Let correct(A) [wrong(A)] denote the set of time steps in which A makes the correct [wrong] decision. Since making the wrong decision can only happen once (because A is out afterwards), the overall gain of A is

$$\text{cost}(\text{A}(I_h)) \leq |\text{correct}(\text{A})| + 2 \cdot |\text{wrong}(\text{A})| \leq |\text{correct}(\text{A})| + 2, \tag{3.1}$$

which directly implies that, for an optimal algorithm OPT, $\text{cost}(\text{OPT}(I_h)) = h + 2$.



(a) An example input from the class $\mathcal{I}_3$.

(b) $\mathcal{T}$ and an optimal solution.

Figure 3.3: An example input of DPA.

3.3 Bounds with Respect to the Number of Requests

First, it was shown that DPA is a hard online problem even for randomized algorithms [35]: Let R be any randomized online algorithm and let $\mathbb{E}[\text{cost}(\mathtt{R}(I))]$ denote the overall number of requests satisfied by R in expectation (see Definition 1.6). The following theorem states that R is at least $(n/4 - \mathcal{O}(1))$-competitive.

Theorem 3.5 (Böckenhauer et al. [35]). *For every randomized online algorithm* R *for* DPA*, there exists an input* I *such that* $\mathbb{E}[\text{cost}(\mathtt{R}(I))] \leq 2$*, whereas an optimal solution satisfies at least* $n/2$ *requests.* ♢

The proof uses the class $\mathcal{I}$ of inputs introduced in the previous section. Note that the above theorem was also proven independently by Caragiannis et al. in a different setup dealing with computing independent sets in disc graphs in an online fashion [47].

Let us now present the main result of this chapter by showing a lower bound on the number of advice bits needed to achieve a particular competitive ratio.

Theorem 3.6. *For any online algorithm with advice for* DPA*, at least*

$$\frac{n+2}{2c} - 2$$

advice bits are required to achieve a strict competitive ratio of c*.*

Proof. As described in the previous section, let $I_h \in \mathcal{I}_h$ be an input instance of DPA and let $\mathcal{T}$ denote the corresponding binary tree. Furthermore, let A be an online algorithm with advice that reads $b < h$ bits from the advice tape for any input I_h.

Obviously, b bits of advice allow A to distinguish between 2^b different inputs whereas there are actually 2^h inputs in $\mathcal{I}_h$. Applying the pigeonhole principle gives that some advice string ϕ is used for at least 2^{h-b} different instances. For the ease of presentation, let us call the corresponding leaves in $\mathcal{T}$ *sinks*. Let $x_{i,k}$ (where $k \in \{1, \ldots, 2^i\}$ for $i \in \{0, \ldots, h\}$) denote the vertices of $\mathcal{T}$ on level i and let $\xi_{i,k}$ denote the number of sinks reachable from $x_{i,k}$. A makes a unique decision on every level $i+1$ that depends only on the advice string ϕ and the input at hand, which is uniquely described by $x_{i,k}$. In the sequel, we show that there exists a sink whose corresponding input is hard for A.

Assume that we have already constructed some prefix of the input that corresponds to some vertex $x_{i,k}$. Consider the behavior of $\mathtt{A}$ for the constructed input prefix $x_{i,k}$ and advice ϕ. We distinguish the following cases depending on the behavior of $\mathtt{A}$ on level $i+1$.

Case 1. Suppose that $\mathtt{A}$ satisfies one request on level $i+1$. In this case, we have $\mathrm{cost}_{i+1}(\mathtt{A}(I_h)) = \mathrm{cost}_i(\mathtt{A}(I_h)) + 1$. Without loss of generality, let $\mathtt{A}$ satisfy the left request. Note that $x_{i+1,2k-1}$ is the left child of $x_{i,k}$ and $x_{i+1,2k}$ is the right child of $x_{i,k}$. We distinguish two subcases.

1. If $\xi_{i+1,2k} = 0$, we are forced to extend the input to $x_{i+1,2k-1}$ since the advice string ϕ is already fixed.
2. If $\xi_{i+1,2k} \geq 1$, we extend the input to any sink in the subtree of $x_{i+1,2k}$, i. e., $\mathtt{A}$ satisfies the left request on level $i+1$, but we have just chosen the input such that the optimal algorithm satisfies the right request. As a result, $\mathtt{A}$ is out after processing the request on level $i+1$.

Case 2. Suppose that $\mathtt{A}$ satisfies both requests on level $i+1$. It then follows that $\mathrm{cost}_{i+1}(\mathtt{A}(I_h)) = \mathrm{cost}_i(\mathtt{A}(I_h)) + 2$, but $\mathtt{A}$ is out after that since it is not able to satisfy anymore requests henceforth; so we extend the input to any sink.

Case 3. Suppose that $\mathtt{A}$ satisfies no request on level $i+1$. It immediately follows that $\mathrm{cost}_{i+1}(\mathtt{A}(I_h)) = \mathrm{cost}_i(\mathtt{A}(I_h))$, but on the other hand, $\mathtt{A}$ is able to satisfy any request on the next level. We extend the constructed input to $x_{i+1,k'}$ such that

$$\xi_{i+1,k'} = \max\{\xi_{i+1,2k-1}, \xi_{i+1,2k}\},$$

i. e., we choose the subtree of $x_{i,k}$ that contains the larger number of sinks.

Suppose that case 1.1 occurs d times and that $f \in \{0, \ldots, h\}$ is the largest number such that $\mathtt{A}$ is not out after processing the request on level f. Note that, while processing the first f requests, only the cases 1.1 and 3 occurred, because otherwise, $\mathtt{A}$ is already out after processing level f. We have that $\xi_{0,1} \geq 2^{h-b}$, and case 3 occurred exactly $f-d$ times on the first f levels. Whenever case 3 occurred, the number of sinks in the subtree of the processed input prefix decreased by at most one half. Whenever case 1.1 occurred, the number of sinks did not decrease at all. Hence, after processing level f, there are at least

$$2^{h-b} \cdot \left(\frac{1}{2}\right)^{f-d} = 2^{h-f+d-b}$$

and at most 2^{h-f} sinks. This gives

$$b \geq d, \tag{3.2}$$

which formally proves the intuitive idea that at least one advice bit is needed for every decision that increases the number of granted requests and also allows $\mathtt{A}$ not to be declared out.

Observe that an optimal algorithm is able to satisfy exactly $n/2+1$ requests (one on every level up to level h and both on level $h+1$). However, $\mathtt{A}$ makes exactly d correct decisions, so, due to (3.1), it satisfies at most $d+2$ requests. We can now give a lower bound on the strict competitive ratio c that depends on the number of advice bits b. By the definition of the strict competitive ratio, we get that

$$\mathrm{comp}(\mathtt{A}(I_h)) = c \geq \frac{\frac{n+2}{2}}{d+2}$$

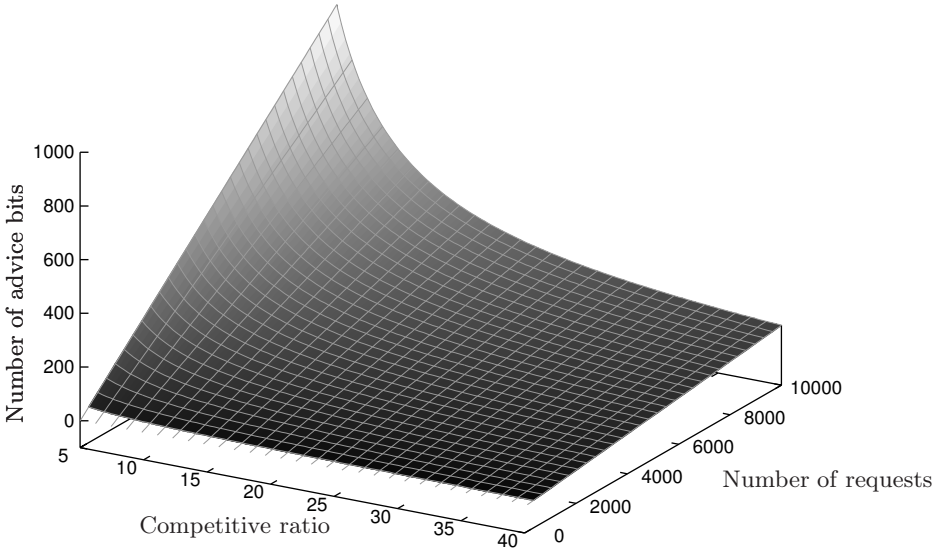


Figure 3.4: The number of advice bits necessary depending on c and n.

and thus

$$d + 2 \geq \frac{n+2}{2c}. \tag{3.3}$$

Finally, (3.2) and (3.3) easily imply

$$b + 2 \geq \frac{n+2}{2c},$$

which means that at least $(n+2)/(2c) - 2$ bits of advice are necessary to be strictly c-competitive. □

The relation between the strict competitive ratio achievable and the number of advice bits that are at least required is shown in Figure 3.4. Note that, by setting $c = 1$, we immediately get a lower bound on the advice complexity for achieving an optimal solution.

Corollary 3.7. *For any online algorithm with advice, at least*

$$b_{\text{opt}} = \frac{n-2}{2}$$

advice bits are required to compute an optimal solution for DPA. □

Clearly, an upper bound on the number of advice bits sufficient to produce an optimal solution is n, i. e., one bit is given for every request indicating whether it is part of the solution or not. Next, we provide an upper bound on the advice complexity sufficient to obtain a given competitive ratio c. We do so by presenting an online algorithm with advice for DPA, whose advice complexity is only a factor of $\log n$ away from the lower bound of Theorem 3.6 for large c; moreover, the bound is asymptotically tight for any constant c.

Theorem 3.8. *For every c, there exists a c-competitive online algorithm with advice for* DPA *that reads at most*

$$\min\left\{ n \log\left(\frac{c}{(c-1)^{\frac{c-1}{c}}} \right), \frac{n \log n}{c} \right\} + 3 \log n + \mathcal{O}(1)$$

advice bits.

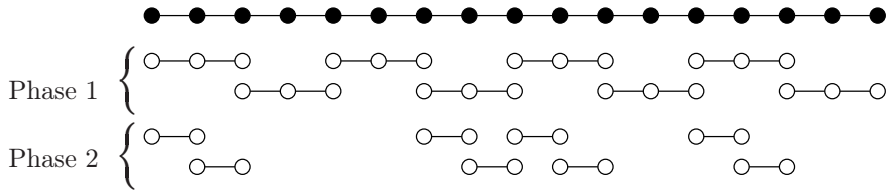


Figure 3.5: An example input of DPA.

Proof. Consider a very simple online algorithm A with advice that reads one advice bit per request and that satisfies this request if and only if this bit is 1. Since the optimal solution for any instance of length n has a cost of at most n, communicating an advice string with at most n/c ones (i. e., at least $n - n/c$ zeros) is sufficient to achieve a competitive ratio of c. Hence, using Lemma 1.15 for $t := c$ implies the claim of the theorem. □

The previous theorem also shows that $\mathcal{O}\big((\log n)^2\big)$ advice bits are sufficient to obtain a competitive ratio asymptotically better than the one any randomized algorithm is able to achieve. Indeed, to get a competitive ratio of $c = n/\log n$, $\mathcal{O}\big((\log n)^2\big)$ advice bits suffice.

3.4 Bounds with Respect to the Size of the Line Network

Instead of considering the number of requests, we can also take the size of the input line network as a basis for measuring the advice complexity; as we have stated in Section 1.3, this is what is classically done. In the following, we show that, also with respect to this measure, a linear number of advice bits is required for computing an optimal solution for DPA.

Theorem 3.9. *Any online algorithm with advice needs to read at least $L/2$ advice bits to achieve optimality.*

Proof. Let L be even. Consider the following set of inputs which basically consist of two phases similar to the proof of Lemma 3.3. In the first $L/2$ time steps (phase 1), the paths $P_1 = (v_0, v_2)$, $P_2 = (v_2, v_4)$, …, $P_{L/2} = (v_{L-2}, v_L)$ are requested. Phase 2 consists of $L/2$ rounds. Every round j may either be empty (no request is made) or it contains two consecutive requests P_{j_1} and P_{j_2} such that P_{j_1} and P_{j_2} are edge-disjoint subpaths of P_j. For $L = 16$, an example is shown in Figure 3.5.

Obviously, for any j, the requests P_{j_1} and P_{j_2} can only be granted if P_j has been discarded before. Since phase 1 is always the same for all inputs, different inputs can be represented by a bit vector $(b_1, \ldots, b_{L/2})$ of length $L/2$, where b_j indicates whether P_{j_1} and P_{j_2} are requested or not for this input. Clearly, any online algorithm A has to behave differently for every string to achieve optimality in any case. The pigeonhole principle then directly implies that at least $L/2$ advice bits are necessary to distinguish all cases. □

As we have seen in Theorem 3.5, randomization does not help for DPA with respect to the number of requests n. However, a randomized online algorithm R with an expected competitive ratio of $\lceil \log L \rceil$ was given by Awerbuch et al. [14]. It is easy to see that it is sufficient to encode all random decisions made by R as advice to enable an online algorithm A with advice to be at least as good; in such a case, A makes exactly the same decisions as R (see Chapter 7). The idea of R is to cluster the line network into $\lceil \log L \rceil$ different groups of possible requests and only accept calls that are within a certain group (this principle

is called *classify and randomly select* [38]). Since the only time randomness is employed is when the group is chosen, it suffices to communicate a number between 1 and $\lceil \log L \rceil$ to be as good as R. We conclude this chapter with the following straightforward remark (recall that L is known to any online algorithm).

Corollary 3.10. *There exists a* $\lceil \log L \rceil$*-competitive online algorithm* $\mathtt{A}_L$ *with advice for* DPA*, which uses at most* $\lceil \log \lceil \log L \rceil \rceil$ *advice bits.* □

Finally, let us point out that the network topology considered in this chapter is very simple, and thus the lower bounds given carry over to more complex ones that are generalizations of paths, i. e., trees. Moreover, it is obvious that we can employ the same ideas when talking about ring-networks (i. e., when we identify the first and the last node of the line).

Chapter 4

The k-Server Problem

The *k-server problem* (k-Server) is certainly one of the most generic and famous online problems. It has been introduced by Manasse, McGeoch, and Sleator in 1988 [120]. We are given a graph G with m vertices and k so-called *servers* that are able to move along the edges of G. In every time step, a vertex is requested, and this request must be answered by moving some server to this vertex, incurring some cost that is specified by a cost function associated with G. As already pointed out, k-Server is a generalization of Paging [38]: For any instance of Paging with cache size k and m potential pages, we can construct a complete graph with m vertices and set all edge costs to 1. The positions of the k servers then correspond to the content of the cache in every time step.

k-Server has been thoroughly studied; for a survey we refer to [6, 38, 93], an in-depth introduction is given by Koutsoupias [111]. In [120], it has been conjectured that there exists a k-competitive deterministic online algorithm for k-Server, which continues to be one of the most famous open problems in theoretical computer science (known as the *k-server conjecture*). So far, the best known deterministic algorithm is due to Koutsoupias and Papadimitriou and it achieves a competitive ratio of $2k - 1$ [112] (and a strict competitive ratio of $4k - 2$ [69]). Moreover, if the input graph is a tree, there is a k-competitive algorithm [53]. As for employing randomization, it is conjectured that there exists a $\Theta(\log k)$-competitive randomized online algorithm (known as the *randomized k-server conjecture* [111], RKSC for short). Very recently, it was shown by Bansal et al. that there is a randomized online algorithm that is $\widetilde{\mathcal{O}}\big((\log m)^3(\log k)^2\big)$-competitive in expectation [16]. This algorithm improves over the one from Koutsoupias and Papadimitriou if

$$m \in o\left(2^{(2k)^{\frac{1}{3+\varepsilon}}}\right)$$

and its competitive ratio is polylogarithmic in k if m is polynomial in k. While this does not hold for large graphs with respect to k, we may still consider the RKSC *almost* proven. Intriguingly, before that, there was no randomized online algorithm known that is better than the deterministic one from [112]. Some connections between the RKSC and the advice complexity are discussed in Chapter 7.

Let us now formally define the problem studied in this chapter.

Definition 4.1 (k-Server). *Let $G = (V, E, d)$ be a complete undirected metric weighted graph as in Definition 1.1, where V is a (not necessarily finite) set of vertices, E is a set of edges, and $d\colon E \to \mathbb{R}$ is a metric cost function. Furthermore, we are given a set of k* servers*, which are located at some of the vertices of G. Let $C_i \subseteq V$ be the multiset of*

vertices occupied by servers in time step i; a vertex occupied by j servers occurs j times in C_i. We also call C_i the configuration *at time step i. Then, a vertex v_i is requested and some servers may be moved yielding a new configuration C_{i+1}. The* request v_i *is* satisfied *if, after this movement of servers, some server is located at v_i, i. e., if $v_i \in C_{i+1}$. The* distance *between two configurations C_1 and C_2 is given by the unique cost of a minimum-weight matching between C_1 and C_2.*

*k-*Server *is the problem to satisfy all requests $v_1, \ldots, v_n$ while minimizing the sum of the distances between all pairs of consecutive configurations.*

Although Definition 4.1 allows to place several servers at the same point, it is easy to see that this is not necessary; we can modify any online algorithm for k-Server such that it never moves more than one server to one vertex in a way such that this modification does not increase the cost of any solution computed by this algorithm.

A solution for an instance of k-Server is a sequence of configurations, and, between two configurations, an arbitrary number of servers can be moved. However, sometimes it is convenient to restrict ourselves to so-called *lazy algorithms* [38] that move at most one server in response to each request. Due to the triangle inequality, this can be done without loss of generality, as any algorithm for k-Server can be transformed into a lazy one without increasing the cost of any solution it produces [38]. It is easy to see that, for the case of lazy algorithms, the solutions can be uniquely described as a sequence of servers used to satisfy individual requests. Throughout this chapter, we assume that all algorithms we deal with are lazy. However, in Section 4.3, we construct an algorithm that is consistent with the above definition of laziness, but that may move the unique server used in one time step back to its original position afterwards.

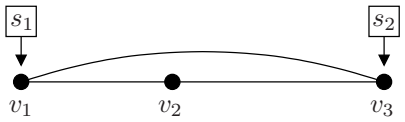


Figure 4.1: A hard instance for `G`; the squares mark the starting positions of the servers.

Note that a simple greedy algorithm `G` is arbitrarily bad already for 2-Server; there exists a well-known hard instance [38]. Suppose there are three vertices v_1, v_2, and v_3 such that $d(\{v_1, v_2\}) < d(\{v_2, v_3\})$ and $d(\{v_1, v_3\})$ is as large as possible; the server s_1 is initially placed at v_1 and s_2 is placed at v_3 (see Figure 4.1). The first request is v_2. `G` moves s_1 to satisfy it, whereas $\mathcal{O}pt$ uses the server s_2. `ADV` then requests v_1 and v_2 alternatingly. While $\mathcal{O}pt$ does not move any other server for all of these requests, `G` has to use s_1, which leads to paying $d(\{v_1, v_2\})$ in every time step. Nevertheless, in Section 4.3, we design an algorithm that partially follows the greedy approach and, making use of the advice supplied, is able to restrict the harm done by greedy moves.

4.1 Lower Bound on Optimality

At first, we again focus on the number of advice bits needed to obtain an optimal solution. More specifically, we show that, if an online algorithm `A` with advice is optimal, there exist instances for which `A` needs to read large advice together with every request. First, we give a bound for inputs with k requests, which we generalize for instances of arbitrary length afterwards. For the following construction, note that any graph with a cost function d that

maps edges to values 1 and 2 only trivially obeys the triangle inequality and is therefore metric.

Lemma 4.2. *For any $k \in \mathbb{N}$, there exists an instance of k-Server with k requests in total, for which any online algorithm* A *with advice needs to read at least $k(\log k - c)$ advice bits to be optimal, for some constant $c < 1.443$.*

Proof. Let $k \in \mathbb{N}$ and let $G = (U \cup W, E, d)$ be a complete bipartite graph with a metric cost function $d\colon E \to \{1, 2\}$, where $U = \{u_1, \ldots, u_k\}$ and $W = \{w_1, \ldots, w_{2^k}\}$. Since $|W| = 2^k$, we can define a bijective function $\mathrm{Set}\colon W \to \mathcal{P}(U)$ that maps every vertex from W to a unique subset of vertices from U. We define the edge costs as follows: for $u \in U$ and $w \in W$, let

$$d(\{u, w\}) := \begin{cases} 2 & \text{if } u \in \mathrm{Set}(w) \\ 1 & \text{otherwise.} \end{cases}$$

Additionally, since formally any instance of k-Server has to contain a complete weighted graph, we define the costs of all edges from $(U \times U) \cup (W \times W)$ to be 2. We call edges of cost 1 *cheap* and edges of cost 2 *expensive*. A schematic view of the constructed graph for $k = 4$ is shown in Figure 4.2. Let $G_i \subseteq W$ denote the vertices from W that correspond to subsets of U with exactly i elements, i. e., $G_i = \{w \in W \mid |\mathrm{Set}(w)| = i\}$. It follows that

$$|G_i| = \binom{k}{i}.$$

We construct a class of instances $\mathcal{I}'$ in the following way. An instance $I \in \mathcal{I}'$ consists of a graph G as above, where every vertex of U is covered by a single server, and a sequence $(x_0, \ldots, x_{k-1})$ of requests such that, for $j \in \{0, \ldots, k-1\}$,

1. $x_j \in G_j$ and
2. $\mathrm{Set}(x_j) \subseteq \mathrm{Set}(x_{j+1})$.

Intuitively speaking, the first requested vertex is the unique vertex from W with only cheap edges to U. Every following request has exactly one more expensive edge than the one before; the requests are chosen in such a way that the set of expensively connected vertices from U is extended by one vertex in every time step.

In what follows, to get an easier notation, let us identify the vertices from U with their indices. We may represent I as a permutation π_I of $\{1, \ldots, k\}$ in the following way:

$$\pi_I(j) := \mathrm{Set}(x_j) \setminus \mathrm{Set}(x_{j-1}), \text{ for } j \in \{1, \ldots, k-1\}, \text{ i. e., } \pi_I(j) = y_j \in U,$$
$$\pi_I(k) := U \setminus \{\pi_I(j) \mid j \in \{1, \ldots, k-1\}\}.$$

In other words, $\pi_I(j)$ denotes that vertex from U that is connected to the requested vertex via an expensive edge from request x_j on. The unique optimal solution $\mathcal{O}pt$ for I with a cost of exactly k can also be described by π_I in the following way. For every j, $\mathcal{O}pt$ satisfies the jth request x_{j-1} by moving one server from some vertex from U, in particular from the vertex $\pi_I(j)$, to the requested vertex from W, via a cheap edge. It is easy to see that there is no solution with a cost of less than k, since all the servers start in U, all requests are different vertices from W, and every edge has cost of at least 1. To see why $\mathcal{O}pt$ is indeed the *unique* optimal solution, consider an offline environment where an optimal offline algorithm Opt receives the whole input at once and may satisfy the requests

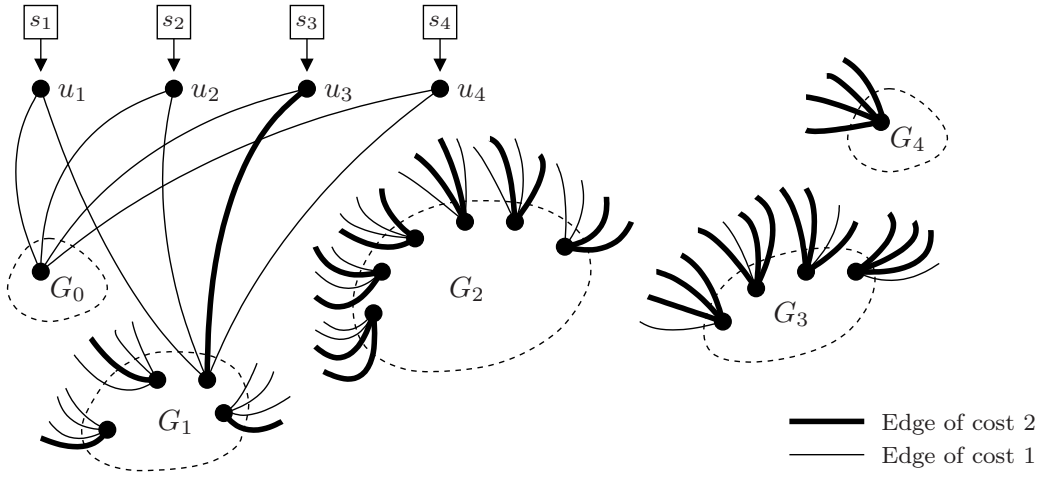


Figure 4.2: An example input of 4-Server as used in the proof of Lemma 4.2. Note that, for the ease of presentation, not all edges are shown completely.

in an arbitrary order. It does so in the opposite order the requests are made: The last vertex requested is from the group G_{k-1} and there is one unique vertex $\pi_I(k)$ connected to it with a cheap edge. The vertex that was requested before is from G_{k-2}. Due to our construction, it also has a cheap edge to $\pi_I(k)$ and to a second vertex $\pi_I(k-1)$, so that Opt now uses this second edge. Following this strategy, it is immediately clear that Opt uses exactly k edges of cost 1 and that its strategy is the only one being not more expensive than k.

Since we may represent any instance from $\mathcal{I}'$ by a unique permutation of $\{1,\dots,k\}$, we need to distinguish $k!$ different cases. It remains to show that we also need a unique advice string for every input to be solved optimally by any online algorithm A with advice. Towards contradiction, let I_1 and I_2 be two different inputs from $\mathcal{I}'$, and suppose that A is optimal for both of them. However, for the same advice string ϕ, the algorithm A behaves deterministically. Let us take the algorithm's point of view: In time step 1, the only vertex from G_0 is requested, and A uses some server to satisfy this request. Then, in time step 2, it is revealed whether this was a good choice, i. e., whether the server at $\pi_I(1)$ was used to serve the first request optimally. After that, the algorithm chooses a second server to move and again, in time step 3, it is revealed whether this was a good choice, and so on. Suppose that the corresponding permutations of I_1 and I_2 differ at position j for the first time. This means that, in time step $j-1$, the algorithm has to make two different choices for the different inputs. But since it reads the same prefix of the input up to this point and furthermore uses the same advice string, it has to behave in the same way. This directly implies that A cannot be optimal for both I_1 and I_2. We conclude that we need a different advice string for every instance and therefore $\log(k!)$ advice bits. Using Stirling's approximation (see (1.1)), we get

$$\log(k!) \geq \log\left(\sqrt{2\pi k}\left(\frac{k}{\mathrm{e}}\right)^k\right) = \frac{1}{2}(\log(2\pi)+\log k)+k(\log k-\log \mathrm{e}) \geq k(\log k - c),$$

where $c = \log \mathrm{e} < 1.443$, which concludes our proof. □

Now we generalize this statement in the following theorem, where we deal with an unbounded number of requests. The idea is to use two graphs as in the proof of Lemma 4.2

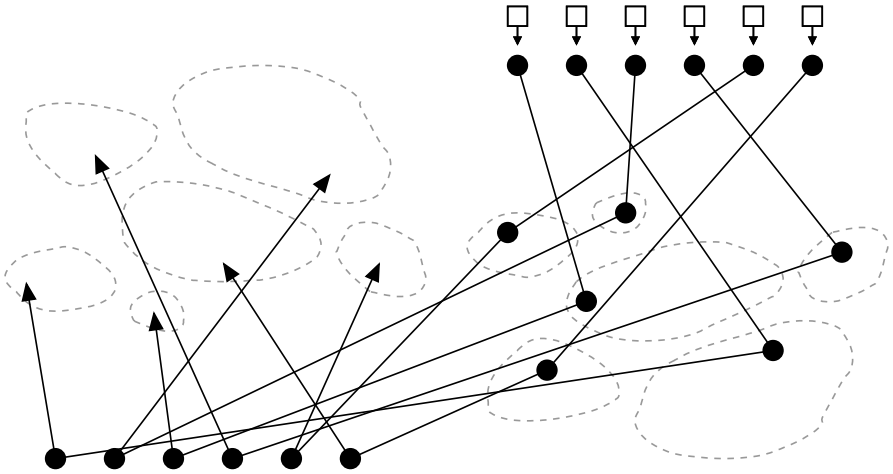


Figure 4.3: The graph used in the proof of Theorem 4.3; the lines mark the optimal trajectories of the servers.

and to connect them in a special manner. The optimal solution is forced to act in a unique way on each of these two graphs alternatingly, and any wrong step within any of the two cannot be compensated later.

Theorem 4.3. *The number of advice bits necessary to allow any online algorithm* A *with advice to be optimal on every instance of* k-Server *can be bounded from beneath by* $n\left(\frac{1}{2}\log k - \frac{1}{2}c\right)$*, for some constant* $c < 1.443$.

Proof. We create a class of instances $\mathcal{I}$ as follows (see Figure 4.3). Suppose we take two disjoint graphs $(U_1 \cup W_1, E_1, d_1)$ and $(U_2 \cup W_2, E_2, d_2)$ as used for constructing instances from $\mathcal{I}'$ in the proof of Lemma 4.2; the groups $G_{1,i}$ and $G_{2,i}$ are thus defined as above. We then connect all vertices from W_1 to the vertices from $U_2 = \{u_{2,1}, \ldots, u_{2,k}\}$ such that, for $j \in \{0, \ldots, k-1\}$, the edges from $G_{1,j}$ to $u_{2,j+1}$ have cost 1 and all other edges have cost 2. The vertices from W_2 are connected to the vertices from $U_1 = \{u_{1,1}, \ldots, u_{1,k}\}$ in the same way. All other newly added edges are assigned cost 2. Let there be n requests in total and let n be a multiple of $4k$. At the beginning, the servers are located at the vertices from U_1. For any instance from $\mathcal{I}$, the first k requests are vertices from W_1 that correspond to a permutation π_1 as in the proof of Lemma 4.2. After that, there are k consecutive requests from U_2, where each of the vertices is requested exactly once. Next, k vertices from W_2 are requested according to some permutation π_2 followed by k requests of the k vertices from U_1. We continue in this fashion until we have made n requests in total. Each such input can be written as

$$\pi_1, u_{2,1}, \ldots, u_{2,k}, \pi_2, u_{1,1}, \ldots, u_{1,k}, \pi_3, u_{2,1}, \ldots, u_{2,k}, \pi_4, \ldots, \pi_{n/(2k)}, u_{1,1}, \ldots, u_{1,k},$$

where π_{2i-1} is a subset of the vertices from W_1 and π_{2i} is a subset of the vertices from W_2, for $i \in \{1, \ldots, n/(2k)\}$. We call the $4k$ consecutive requests from W_1, U_2, W_2, and U_1 a *round*. These instances have an optimal solution $\mathcal{O}pt$ that, whenever moving servers from U_1 $[U_2]$ to W_1 $[W_2]$, acts according to the corresponding permutation and that moves the unique server that is located in $G_{1,j}$ $[G_{2,j}]$ to $u_{2,j+1}$ $[u_{1,j+1}]$ when moving servers from W_1 $[W_2]$ to U_2 $[U_1]$.

We now show that $\mathcal{O}pt$ is indeed the unique optimal solution. Towards contradiction, suppose that there exists a solution $\mathcal{A}$ that differs from $\mathcal{O}pt$ while not being worse than

$\mathcal{O}pt$. Clearly, $\mathcal{O}pt$ has cost 1 in every time step. Let us first show that there are no time steps in which $\mathcal{A}$ may have cost 0, i. e., in which it does not move any server. By the construction of the inputs, the same vertex is not requested twice within the same round. This means that, if in some round a server already resides at some vertex that is now requested, this server was not moved since the last round (recall that we only consider lazy algorithms). Without loss of generality, suppose that this server is positioned at some vertex from W_1. We know that $\mathcal{O}pt$ used this server to satisfy a request from U_2, W_2, and U_1 afterwards, which each causes cost 1. Instead, $\mathcal{A}$ uses a server that is already located at some vertex from U_2 to satisfy some request from U_2 yielding cost 2 in this time step, because all edges from U_2 to U_2 have cost 2. The same applies to at least one request from W_2 and U_1, and therefore $\mathcal{A}$ saved cost 1, but paid 6 instead of 3 after one round is over. For $\mathcal{A}$, it thus follows that it also moves exactly one server per time step, which causes it to pay cost 1. Now let i denote the first time step in which $\mathcal{A}$ deviates from $\mathcal{O}pt$.

For the ease of presentation, we only consider the following two cases. All other cases are handled analogously.

Case 1. Suppose that $\mathcal{O}pt$ moves a server from U_1 to W_1 in time step i. If $\mathcal{A}$ uses a server from W_1 to satisfy this request, $\mathcal{A}$ pays 2 instead of 1 which, as we just showed, cannot be compensated afterwards. Due to Lemma 4.2, we already know that, if $\mathcal{A}$ chooses a server from U_1 other than the one taken by $\mathcal{O}pt$, this also causes cost 2 eventually. Since i is the first time step in which the two solutions differ, it follows that no servers are located in U_2 or W_2. Thus, this case leads to a contradiction to the optimality of $\mathcal{A}$ and hence cannot occur.

Case 2. Suppose that $\mathcal{O}pt$ moves a server from W_1 to U_2 in time step i. Using a server located in U_2 again causes cost 2. Moreover, observe that there is only one unique group $G_{1,j}$ in W_1 that is connected to $u_{2,j+1}$ with edges of cost 1. Since $\mathcal{A}$ and $\mathcal{O}pt$ acted identically up to this point, before the first request from U_2 arrived in this round, they both positioned exactly one server at each group $G_{1,k}$. It directly follows that, if $\mathcal{A}$ uses a different server than $\mathcal{O}pt$, it again pays 2 instead of 1. Similar to case 1, no servers may be located in U_1 or W_2.

Since there is one unique optimal solution that needs to act according to the permutations $\pi_1, \dots, \pi_{n/(2k)}$, by the same argumentation as in the proof of Lemma 4.2, it directly follows that at least

$$\frac{n}{2k} \log(k!) \geq \frac{n}{2}(\log k - c)$$

bits of advice are necessary in total to be optimal, for $c = \log \mathrm{e}$; this implies that, amortized, at least

$$\frac{1}{2} \log k - \frac{1}{2} c$$

advice bits are necessary in every time step. □

Note that, if we allow the graph to have an unbounded size, it is easy to construct a lower bound of $n(\log k - \log \mathrm{e})$ by branching the graph infinitely often: If `A` served the first k requests according to the corresponding permutation π_1, all k servers are located at unique vertices that we use as the starting positions for the next k requests corresponding to a permutation π_2 and so on. To prevent `A` from being able to anticipate any requests by inspecting the graph, we need to do this construction for *any* possible set of starting

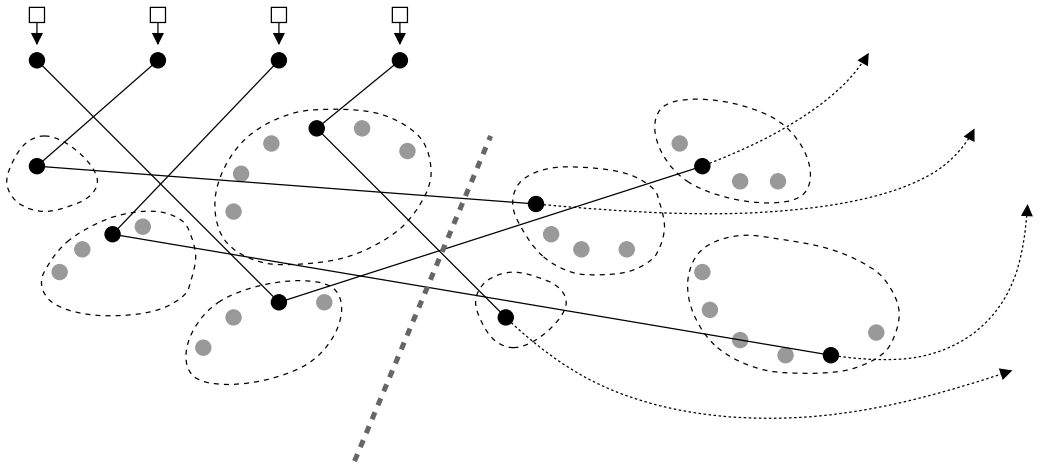


Figure 4.4: A generalization of Theorem 4.3 for infinite graphs.

positions, which causes an exponential branching with every additional k requests. The first two rounds (without branches that are not used) are shown in Figure 4.4.

The lower bounds presented in this section directly carry over to the model of [70], where the number of advice bits used per time step is fixed (see Section 1.6). Furthermore, [70] contains an upper bound of $\log k$ bits per request. Hence, the lower bound from Theorem 4.3 is essentially tight up to a factor of 2.

4.2 Upper Bound for the Euclidean Plane

In this section, we consider the subproblem of k-Server where the underlying metric space is the two-dimensional Euclidean plane, i. e., the distance between any two vertices $p = (p_x, p_y)$ and $q = (q_x, q_y)$ is given by

$$d(\{p, q\}) := \|p - q\| = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}.$$

For this case, we propose a simple algorithm A with advice that achieves a constant competitive ratio while using a linear number of advice bits; in particular, the algorithm reads a *constant* number of bits with every request.

Let us fix a parameter b such that the algorithm uses b bits of advice per request. A works as follows: If the requested point is $r = (r_x, r_y)$, it divides the plane into 2^b disjoint segments $S_1, \ldots, S_{2^b}$ with their origin in r and with an angle of $2\pi/2^b$ each. Then A reads b bits of advice that identify the segment S_i in which the server used by an optimal solution for this time step is located; it serves the request greedily with the closest server from S_i. To show that A achieves a constant competitive ratio, we first prove the following technical lemma we need in the analysis.

Lemma 4.4. *For $0 < \alpha \leq \pi/4$, let $\mathcal{C}(\alpha, R)$ be the region of the Euclidean plane $\mathcal{C}(\alpha, R) := \{(r\cos\varphi, r\sin\varphi) \mid 0 < \varphi \leq \alpha, 0 < r \leq R\}$. For any point $p \in \mathcal{C}(\alpha, R)$, let*

$$f(p) := \frac{\|p\|}{R - \|[R, 0] - p\|},$$

where $\|v\|$ is the Euclidean length of v. Then $f(p)$ is maximized over the region $\mathcal{C}(\alpha, R)$ for $p_{\max} = (R\cos\alpha, R\sin\alpha)$ and $f(p_{\max}) = \frac{1}{1-2\sin\left(\frac{\alpha}{2}\right)}$.

Proof. The situation described by the lemma is illustrated in Figure 4.5 (a). Consider any point $p \in \mathcal{C}(\alpha, R)$ such that $p = (r\cos\varphi, r\sin\varphi)$. Then

$$f(p) = \frac{r}{R - \sqrt{(R - r\cos\varphi)^2 + (r\sin\varphi)^2}} = \frac{r}{R - \sqrt{r^2 - 2Rr\cos\varphi + R^2}}.$$

For $0 < \alpha \leq \pi/4$, $f(p)$ is increasing in φ, and the maximum is attained for $\varphi = \alpha$. For the remainder of the proof, let us thus set $\varphi = \alpha$ and treat $f(p)$ as a function of r only. We immediately get that the roots of the denominator of $f(p)$ are $r_1 = 0$ and $r_2 = 2R\cos\alpha$. Note that $\cos(\pi/4) = 1/\sqrt{2} > 1/2$ and, since $0 < \alpha \leq \pi/4$, $f(p)$ is continuous for $r \in \langle 0, R\rangle$. Let us substitute $X := \sqrt{r^2 - 2Rr\cos\alpha + R^2}$. Taking the derivative, we get

$$\begin{aligned} f'(p) &= \frac{R - X + r\frac{1}{2}\left(r^2 - 2Rr\cos\alpha + R^2\right)^{-\frac{1}{2}}(2r - 2R\cos\alpha)}{(R - X)^2} \\ &= \frac{R - X + r\left(\sqrt{r^2 - 2Rr\cos\alpha + R^2}\right)^{-1}(r - R\cos\alpha)}{(R - X)^2} \\ &= \frac{R - X + X^{-1}(r^2 - Rr\cos\alpha)}{(R - X)^2} = \frac{X(R - X) + r^2 - Rr\cos\alpha}{X(R - X)^2} \\ &= \frac{XR - r^2 + 2Rr\cos\alpha - R^2 + r^2 - Rr\cos\alpha}{X(R - X)^2} = R\frac{X + r\cos\alpha - R}{X(R - X)^2}. \end{aligned} \tag{4.1}$$

Clearly, R and the denominator of (4.1) are always positive; therefore, it is sufficient to show that

$$X = \sqrt{r^2 - 2Rr\cos\alpha + R^2} > R - r\cos\alpha,$$

which is equivalent to (note that $R - r\cos\alpha$ is non-negative)

$$\sqrt{r^2 - r^2(\cos\alpha)^2} > 0. \tag{4.2}$$

Since $r^2(1 - (\cos\alpha)^2)$ is strictly positive, (4.2) is always true. As a result, we have that $f(p)$ is increasing in r as well. Finally, note that $\cos(2\beta) = 1 - 2(\sin\beta)^2$ [44]; thus, we get

$$\begin{aligned} f((R\cos\alpha, R\sin\alpha)) &= \frac{\|(R\cos\alpha, R\sin\alpha)\|}{R - \|(R, 0) - (R\cos\alpha, R\sin\alpha)\|} \\ &= \frac{R}{R - \sqrt{R^2(1 - \cos\alpha)^2 + R^2(\sin\alpha)^2}} \\ &= \frac{1}{1 - \sqrt{(\sin\alpha)^2 + (\cos\alpha)^2 + 1 - 2\cos\alpha}} \\ &= \frac{1}{1 - \sqrt{2(1 - \cos\alpha)}} \\ &= \frac{1}{1 - \sqrt{2\left(1 - 1 + 2\left(\sin\left(\frac{\alpha}{2}\right)\right)^2\right)}} \\ &= \frac{1}{1 - \sqrt{4\left(\sin\left(\frac{\alpha}{2}\right)\right)^2}} = \frac{1}{1 - 2\sin\left(\frac{\alpha}{2}\right)} \end{aligned}$$

as we claimed. □

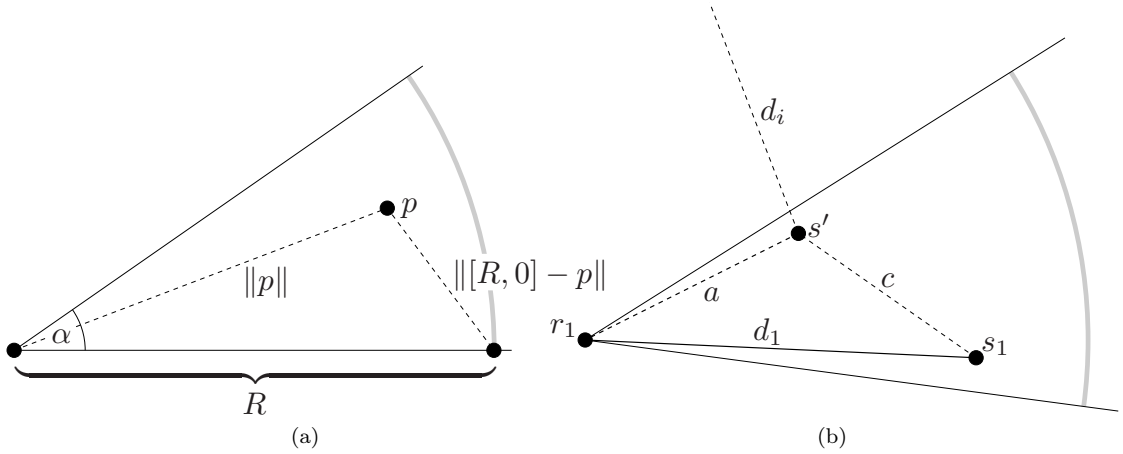


Figure 4.5: (a) Illustration of Lemma 4.4. (b) A segment as used by A.

Now we are ready to analyze the competitive ratio of A.

Theorem 4.5. *Let $b \geq 3$ and* A *be defined as above. For any instance $I = (C, r_1, \ldots, r_n)$ and any solution $\mathcal{B}$, the cost of* A *on I is at most $q \cdot \mathrm{cost}(\mathcal{B})$, where*

$$q := \frac{1}{1 - 2\sin\left(\frac{\pi}{2^b}\right)},$$

i. e., A *is q-competitive using b bits of advice per request.*

Proof. Let us adopt the following notation: As in Definition 4.1, a configuration C is a multiset of k points that are occupied by the servers. A configuration $C_{p_1 \mapsto p_2}$ is obtained from C by moving a server from $p_1 \in C$ to p_2. An instance of the problem is a configuration and a sequence of requests (points); the length of the instance is the number of requests. We restrict ourselves to lazy algorithms, therefore, as already noted, a solution of an instance is a sequence of servers. To describe a server that is used to satisfy a certain request, it is sufficient to specify the point occupied by this server. Thus, the solution can be described by a sequence of points as well.

Let $\mathcal{B}$ serve the ith request r_i by a server located at s_i, incurring a cost of $d_i = \|s_i - r_i\|$. For the first request, A uses b bits of advice to specify the segment of angle $\alpha = 2\pi/2^b$ around r_1 in which s_1 is located and moves the closest server in this segment, say s', to r_1, incurring a cost of $a \leq d_1$. Hence, after the first request, $\mathcal{B}$ leads to a configuration $C_{s_1 \mapsto r_1}$, whereas A is in a configuration $C_{s' \mapsto r_1}$; this situation is illustrated in Figure 4.5 (b).

The proof is done by induction on n. If $n = 1$, the cost of A is $a \leq d_1 = \mathrm{cost}(\mathcal{B})$. Let $n > 1$ and let r_i be the first request that is served by s' in $\mathcal{B}$. Consider the instance $I' = (C_{s' \mapsto r_1}, r_2, \ldots, r_n)$; the sequence $(s_2, \ldots, s_{i-1}, s_1, s_{i+1}, \ldots, s_n)$ is a solution for I' with a cost of at most $c + \sum_{i=2}^{n} d_i$, where c is the distance between s_1 and s' (see Figure 4.5 (b)). By induction, the cost of A on I' is at most $q \cdot (c + \sum_{i=2}^{n} d_i)$, and therefore the cost of A on I is at most

$$a + q \cdot \left(c + \sum_{i=2}^{n} d_i \right).$$

Due to Lemma 4.4, we have that $a \leq q(d_1 - c)$ and the claim follows. □

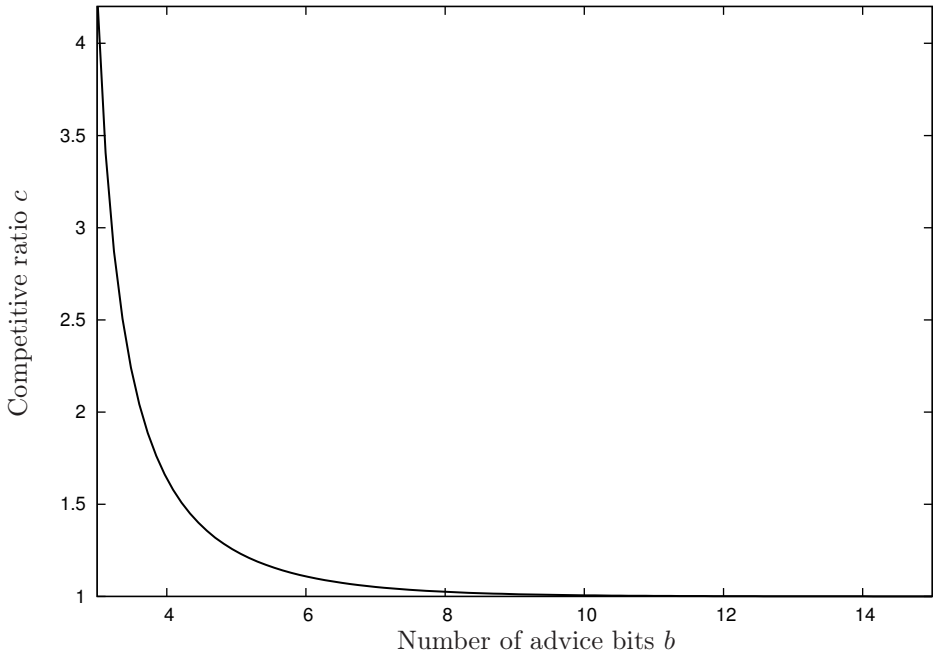


Figure 4.6: The competitive ratio depending on the number of advice bits b per request.

Note that, with b tending to infinity, the competitive ratio of A converges to 1, see Figure 4.6.

4.3 Upper Bound for the General Case

We now focus on the tradeoff between the number of advice bits and the competitive ratio achievable in general metric spaces. As we have seen at the beginning of this chapter, a simple greedy strategy is very bad for k-Server. However, in what follows, we construct an algorithm with advice that follows a greedy strategy for some of the requests and that is able to bound the harm done by these moves by using the advice tape from time to time.

Theorem 4.6. *For every $b \geq 2$, there exists an online algorithm* A *with advice for k-*Server *that uses $b \cdot n$ advice bits for inputs with n requests and that achieves a competitive ratio of*

$$2\left\lceil \frac{\lceil \log k \rceil}{b-1} \right\rceil \leq 2\left(1 + \frac{1}{b-1}\right) + \frac{2}{b-1} \log k.$$

Proof. At first, let us fix the algorithm A. With every request, A reads one bit of advice called a *control bit*. If this bit is 0, A satisfies the request greedily with the nearest server. Afterwards, this server is returned to its original position before the next request. If the control bit is 1, A reads the next $\lceil \log k \rceil$ bits. These bits specify which server should be used to satisfy the request. After the request is satisfied, the server is left at its new position.

We prove that, for every input instance, there exists an advice string such that A has a competitive ratio of

$$c := 2\left\lceil \frac{\lceil \log k \rceil}{b-1} \right\rceil.$$

Furthermore, A uses at most $b \cdot i$ advice bits while processing the first i requests. Let us give some intuition first: The greedy moves may be cheap at the very beginning of an instance, but they can cause some additional cost later that might propagate in the subsequent time steps (as we have seen at the beginning of this chapter). However, in our model, we may from time to time stop this propagation by specifying the server that is used by an optimal algorithm. We show that we can amortize the steps in which costs propagate in such a way that the total cost is not too high.

Consider any input instance I consisting of n requests $r_1, \ldots, r_n$. Let $\mathcal{S}^{(0)}$ be an optimal solution for I; assume that $\mathcal{S}^{(0)}$ satisfies the request r_i with cost c_i. We construct a sequence $\mathcal{S}^{(1)}, \ldots, \mathcal{S}^{(n)}$ of feasible solutions for I sequentially, together with a sequence of advice tapes $\phi^{(1)}, \ldots, \phi^{(n)}$, such that the following properties hold.

1. Let $j \leq i$. $\mathcal{S}^{(i)}$ satisfies the request r_j with cost $c_j^{(i)}$ which is at most twice the sum of some request costs of $\mathcal{S}^{(0)}$, i. e.,

$$c_j^{(i)} \leq 2 \sum_{l \in D_j^{(i)}} c_l,$$

   where $D_j^{(i)}$ is some subset of $\{1, \ldots, n\}$. We call the jth step of $\mathcal{S}^{(i)}$ a *dead step*. Similarly, we call the set $D_j^{(i)}$ *dead set* and the cost $c_j^{(i)}$ *dead cost*.

2. Let $j > i$. Similar to the previous case, $\mathcal{S}^{(i)}$ satisfies the request r_j with cost $c_j^{(i)}$ which is at most *once* the sum of some request costs of $\mathcal{S}^{(0)}$, i. e.,

$$c_j^{(i)} \leq \sum_{l \in D_j^{(i)}} c_l,$$

   where $D_j^{(i)}$ is some subset of $\{1, \ldots, n\}$. We call the jth step of $\mathcal{S}^{(i)}$ a *live step*. Similarly, we call the set $D_j^{(i)}$ *live set* and the cost $c_j^{(i)}$ *live cost*.

3. Let i be arbitrary but fixed. For every j, the request cost c_j contributes to at most $c/2$ costs $c_l^{(i)}$, i. e., j belongs to at most $c/2$ sets $D_l^{(i)}$.

4. Let i be arbitrary but fixed. For every j, the request cost c_j contributes to at most one live cost $c_l^{(i)}$, i. e., j belongs to at most one set $D_l^{(i)}$, where $l > i$.

5. Consider any live step j of $\mathcal{S}^{(i)}$ (i. e., $j > i$). The request r_j is satisfied by $\mathcal{S}^{(i)}$ by moving a single server to the vertex r_j without any other moves (i. e., the moved server is left at r_j).

6. For any $j \leq l$ and any i, the sets $D_j^{(i)}$ and $D_l^{(i)}$ are either disjoint or $D_j^{(i)} \subseteq D_l^{(i)}$. Note that the disjointness of live sets is already guaranteed by property 4.

7. Consider an arbitrary i. The algorithm A, given the advice $\phi^{(i)}$, processes the first i requests in the same way as $\mathcal{S}^{(i)}$.

It is not difficult to see that properties 1–3 ensure that, for any i, the total cost of $\mathcal{S}^{(i)}$ does not exceed the total cost of $\mathcal{S}^{(0)}$ multiplied by $2 \cdot c/2$. Property 7 ensures that A, given the advice $\phi^{(n)}$, processes all requests in the same way as $\mathcal{S}^{(n)}$, hence A achieves a

competitive ratio of c. In the sequel, we show how to construct the sequence of solutions $\mathcal{S}^{(1)}, \ldots, \mathcal{S}^{(n)}$, as well as the sequence of advice tapes $\phi^{(1)}, \ldots, \phi^{(n)}$. Afterwards, we analyze how many advice bits are necessary for this construction.

We define $\mathcal{S}^{(i)}$ and $\phi^{(i)}$ inductively. Each $\phi^{(i)}$ contains some finite defined prefix followed by arbitrary (undefined) bits. For $i = 0$, the defined prefix of $\phi^{(0)}$ is empty. It is easy to see that $\mathcal{S}^{(0)}$ and $\phi^{(0)}$ satisfy properties 1–7; it is sufficient to define $D_j^{(0)} := \{j\}$, for all j.

Assume that $\mathcal{S}^{(i-1)}$ and $\phi^{(i-1)}$ are defined. We obtain $\mathcal{S}^{(i)}$ by applying a few modifications to $\mathcal{S}^{(i-1)}$. In particular, the first $i-1$ steps of $\mathcal{S}^{(i)}$ are always identical to $\mathcal{S}^{(i-1)}$. The live ith step of $\mathcal{S}^{(i-1)}$ needs to be transformed into a dead ith step of $\mathcal{S}^{(i)}$. While doing so, some other live steps might be changed as well. The advice tape $\phi^{(i)}$ is obtained by taking the defined prefix of $\phi^{(i-1)}$ and appending the information necessary for `A` to perform the ith step in the same way as $\mathcal{S}^{(i)}$ does. We distinguish two cases.

Case 1. For any $l \in D_i^{(i-1)}$, there are less than $c/2$ sets $D_j^{(i-1)}$ containing l. Assume that, in $\mathcal{S}^{(i-1)}$, the request r_i is satisfied by moving a server from the vertex v to r_i (property 5 ensures that this is always the case). In $\mathcal{S}^{(i)}$, we replace this step by a greedy step, i. e., the request r_i will be satisfied by the closest server located at the vertex w, and the server will be returned to w afterwards. Since we took the closest server, $c_i^{(i)} \leq 2c_i^{(i-1)}$, hence it is sufficient to define $D_i^{(i)} := D_i^{(i-1)}$. The next steps of $\mathcal{S}^{(i)}$ can be defined to be identical to the steps of $\mathcal{S}^{(i-1)}$ until the solution $\mathcal{S}^{(i-1)}$ uses the server located at r_i to satisfy some request r_j. Since there is no server located at r_i in $\mathcal{S}^{(i)}$, we need to modify this step: In $\mathcal{S}^{(i)}$, the request r_j is satisfied by moving the server from v to r_i and then to r_j. This is possible, because the server at v was not used after the request r_i in $\mathcal{S}^{(i-1)}$. The cost $c_j^{(i)}$ of such a move is at most $c_i^{(i-1)} + c_j^{(i-1)}$, hence it is sufficient to define $D_j^{(i)} := D_i^{(i-1)} \cup D_j^{(i-1)}$ (note that property 4 applied for $\mathcal{S}^{(i-1)}$ ensures that this union is disjoint). All remaining steps of $\mathcal{S}^{(i)}$ are the same as in $\mathcal{S}^{(i-1)}$.

It is easy to see that properties 1, 2, and 5 hold for $\mathcal{S}^{(i)}$. After transforming $\mathcal{S}^{(i-1)}$ into $\mathcal{S}^{(i)}$, only the elements from $D_i^{(i-1)}$ occur in more sets. Therefore, due to the assumption of this case, property 3 holds. Since, due to property 4 applied for $\mathcal{S}^{(i-1)}$, no element of $D_i^{(i-1)}$ is in $D_l^{(i-1)}$, for any $l > i$, and since $D_i^{(i)}$ is not a live set, property 4 holds for $\mathcal{S}^{(i)}$ as well. Since property 4 holds for $\mathcal{S}^{(i)}$, property 6 could be violated only if $D_j^{(i)}$ was no superset of some $D_l^{(i)}$, for $l \leq i$. This, however, cannot happen, because $D_j^{(i)} \subseteq D_i^{(i)}$ and property 6 holds for $\mathcal{S}^{(i-1)}$.

To satisfy property 7, it is sufficient to extend the advice $\phi^{(i-1)}$ by a single bit: a control bit 0.

Case 2. Some element of $D_i^{(i-1)}$ is contained in $c/2$ sets $D_j^{(i-1)}$. In this case, the solution $\mathcal{S}^{(i)}$ is defined to be exactly the same as $\mathcal{S}^{(i-1)}$. Hence, properties 1–6 trivially hold. In order to satisfy property 7, we extend the advice $\phi^{(i-1)}$ by $1 + \lceil \log k \rceil$ bits: a control bit 1 followed by $\lceil \log k \rceil$ bits describing which server should be used to satisfy the request r_i. Note that, in this case, no element $l \in D_i^{(i)}$ occurs in any live set $D_j^{(i)}$. Furthermore, l cannot reappear later, i. e., l does not occur in any $D_j^{(i')}$, for $i' \geq i$ and $j > i$.

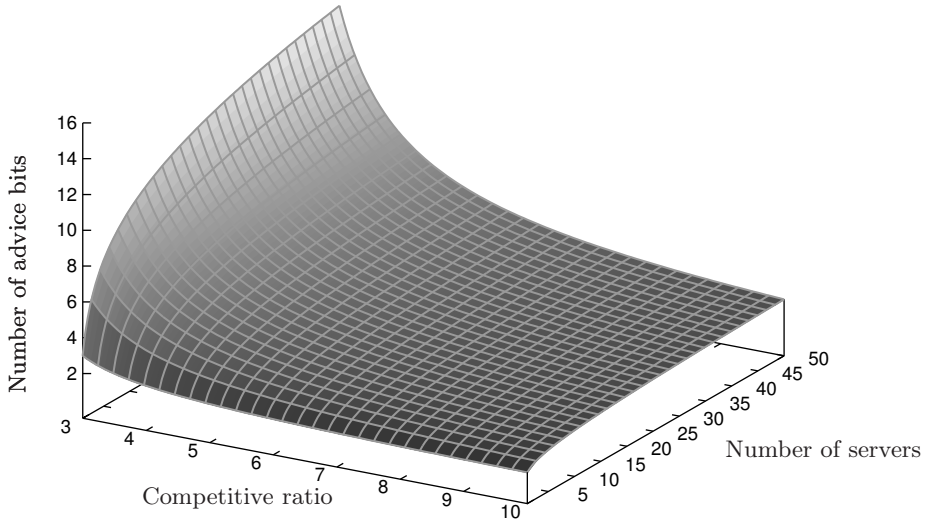


Figure 4.7: The number of advice bits per time step depending on c and k.

Properties 1–7 ensure that A, given the advice $\phi^{(n)}$, solves I correctly with a competitive ratio of c. It remains to analyze the length of the advice used by A.

The construction of the advice tapes easily implies that A, given the advice $\phi^{(i)}$, accesses only the defined prefix of $\phi^{(i)}$ during the first i steps of computation. Next, we show that the defined prefix of $\phi^{(i)}$ has a length of at most $i \cdot b$.

It is easy to see that $\phi^{(i)}$ contains exactly i control bits. Consider any block of $\lceil \log k \rceil$ consecutive non-control bits appearing in $\phi^{(i)}$. These bits were created in case 2 when performing the jth step of induction (i. e., when $\mathcal{S}^{(j)}$ and $\phi^{(j)}$ were created). This, however, means that some element of $D_j^{(j-1)}$ occurs in $c/2 - 1$ sets $D_l^{(j-1)}$ $(l < j)$. For any such l, case 1 occurred for the lth step of induction (as noted in case 2, if that case occurred, no element of $D_l^{(l)} = D_l^{(j-1)}$ would occur in $D_j^{(j-1)}$, a contradiction). Hence, to any block of $\lceil \log k \rceil$ consecutive non-control bits appearing in $\phi^{(i)}$, we can injectively assign $c/2 - 1$ induction steps where no non-control bits were created. The fact that the assignment is injective is guaranteed by property 6 and the fact that elements in a set $D_j^{(j-1)}$ handled in case 2 cannot reappear later. Equivalently, we can injectively assign $c/2$ control bits to every block of $\lceil \log k \rceil$ non-control bits. As a result, there are at most

$$i \cdot \frac{\lceil \log k \rceil}{\frac{c}{2}} = i \cdot \frac{2\lceil \log k \rceil}{2\left\lceil \frac{\lceil \log k \rceil}{b-1} \right\rceil} \leq i \cdot \frac{\lceil \log k \rceil}{\frac{\lceil \log k \rceil}{b-1}} = i(b-1)$$

non-control bits in $\phi^{(i)}$. To sum up, $\phi^{(i)}$ contains at most $i(b-1) + i = i \cdot b$ defined bits, and thus A uses at most $n \cdot b$ advice bits in total. □

Theorem 4.6 implies the following corollary about the sufficient number of advice bits to achieve some specific competitive ratio.

Corollary 4.7. *For every $c > 2$, there exists a c-competitive online algorithm* A *with advice for k-*Server *that uses at most*

$$1 + \frac{2(1 + \log k)}{c - 2}$$

advice bits per time step. □

Figure 4.7 shows how the sufficient number of advice bits behaves with respect to c and k. We have proven Theorem 4.6 in the model of an advice tape. Nevertheless, the result of this theorem can easily be adapted to the model with a fixed number of advice bits received with every request, as used in [70] (see Section 1.6): Consider the advice tape created in the proof of Theorem 4.6. Let us separate this tape into two bit sequences, one containing only control bits, the other one containing non-control bits only. Afterwards, we interleave these sequences, always taking $b-1$ non-control bits followed by one control bit to create a single b-bit advice message. $\mathtt{A}$ then reads $b-1$ non-control bits with every message and stores them into a FIFO data structure. Afterwards, it reads the control bit; if this bit is 1, $\lceil \log k \rceil$ bits are extracted from the FIFO queue. The proof of Theorem 4.6 ensures that there are always sufficiently many bits stored in the queue when a control bit 1 arrives.

Finally, note that Theorem 4.6 improves the result of [70] exponentially: With b advice bits per request, the feasibility of achieving a competitive ratio of $k^{\Theta\left(\frac{1}{b}\right)}$ was proven in [70], whereas Theorem 4.6 shows the existence of an online algorithm $\mathtt{A}$ with advice that achieves a competitive ratio of

$$\operatorname{comp}(\mathtt{A}(I)) \leq \log\left(k^{\Theta\left(\frac{1}{b}\right)}\right)$$

on any instance I. A further discussion of the consequences of Theorem 4.6 is given in Chapter 7.

Chapter 5

The Set Cover Problem

In the offline *set cover problem*, we are given a ground set X of n elements and a family $\mathcal{S}$ of m sets that each contain some of those elements. The objective is to find a smallest possible number of members of $\mathcal{S}$ such that their union covers all elements of X. In the weighted version of the problem, every member of $\mathcal{S}$ is associated with a positive weight; here, we aim at minimizing the sum of all weights of the members that are chosen. The set cover problem is a very old and famous optimization problem [86, 150]. The corresponding decision problem is among *Karp's 21 $\mathcal{NP}$-complete problems*[1]: As one of the first computational problems in the literature, it was proven to be $\mathcal{NP}$-complete by Karp in 1972 [101], which implies that the set cover problem is $\mathcal{NP}$-hard (for the relation between the two problems and the concept of *threshold languages*, see, e. g., [86]). Since then, the problem has been extensively studied; it is well known [55, 96, 119] that the straightforward greedy approach achieves an approximation ratio of H_n (the nth harmonic number, see Section 1.4), and, intriguingly, this bound is essentially tight as shown by Feige [73].

Alon et al. studied the set cover problem in an online environment [8]; this variant was introduced in [7]. Here, X and $\mathcal{S}$ are known to any online algorithm in advance, but only a subset of X is actually requested during runtime, and these requests arrive online. After each such request j, an online algorithm has to specify a member of $\mathcal{S}$ that contains j if it is not yet included in the sets that were chosen before.

In the following, we consider the online version of the unweighted set cover problem, SetCover for short, introduced and studied by the authors. As always, let us start with the formal definition.

Definition 5.1 (SetCover). *Given a* ground set X *of size* n*, a set of* requests $X' \subseteq X$*, and a family* $\mathcal{S} \subseteq \mathcal{P}(X)$ *of size* m*, a feasible solution for* SetCover *is any subset* $\{S_1, \ldots, S_k\}$ *of* $\mathcal{S}$ *such that*

$$\bigcup_{i=1}^{k} S_i \supseteq X'.$$

The aim is to minimize k*, i. e., to use as few sets as possible. The elements of* X' *arrive successively one by one in consecutive time steps. An online algorithm solves* SetCover *if, immediately after each yet uncovered request* j*, it specifies a set* S_i *such that* $j \in S_i$*.*

Note that Alon et al. constructed a deterministic $\mathcal{O}(\log m \log n)$-competitive online algorithm that even works for the weighted version of SetCover [8]. Also, the authors

[1] Richard M. Karp, ⋆03.01.1935, American computer scientist.

gave a lower bound that almost matches the upper one for most values of m and n. In the following, we give bounds on the advice complexity in both the size of the ground set X and the size of the family $\mathcal{S}$. Obviously, $n = |X|$ is an upper bound on the number of requests. Furthermore, note that we may assume that $\mathcal{S}$ does not contain any set that is a subset of any other set from $\mathcal{S}$. If this is not the case, sets that are contained in other sets may be removed from the input by a preprocessing step; since the sets are unweighted, it is never a disadvantage to prefer supersets. From this, it directly follows, due to Sperner's theorem (see Section 1.2), that we have

$$m = |\mathcal{S}| \leq \binom{n}{\lfloor \frac{n}{2} \rfloor} =: N(n),$$

which implies that m may be exponential in n. However, due to the hardness of the vertex cover problem [86], which is a sub-problem of SetCover, we know that also instances where $m \in \mathcal{O}(n)$ may be hard in the offline case. This motivates the study of advice depending on both n and m.

5.1 Bounds with Respect to the Size of the Ground Set

Let us again start by analyzing the advice complexity for calculating an optimal solution. In the following, by a very easy argument, we show that a number of advice bits that is linear in $|X|$ is sufficient to create optimal output.

Theorem 5.2. *There exists an optimal online algorithm* A *with advice for* SetCover *that uses* $n - 1$ *advice bits.*

Proof. The proof is straightforward. The oracle O simply writes the characteristic function of X' on the advice tape, i. e., a 1 at position i if and only if the ith element from X is contained in X'. However, A knows one requested element from X before it has to take any action. Thus, if the first element requested is j, the jth position is simply skipped on the tape, i. e., not written down by O. □

Although the above algorithm is trivial, it is interesting to note that we are able to complement this result with an almost matching lower bound. In other words, we now show that a linear number of advice bits is also necessary to be optimal.

Theorem 5.3. *At least* $\log(N(n-1)) > n - \frac{1}{2}\log(n-1) - 3$ *advice bits are necessary for any online algorithm* A *with advice for* SetCover *to achieve optimality.*

Proof. Let n be even. Consider the set family $\mathcal{S}$ that contains all subsets of X of size $n/2$; clearly, there are exactly $N(n)$ such sets. Now Adv requests $n/2$ items, starting with one fixed item x_1 (after which A has to start with choosing a member of $\mathcal{S}$). It is clear that one single member of $\mathcal{S}$ is sufficient to cover all requests. Since A knows that x_1 is included in the set it has to choose, there are

$$\binom{n-1}{n/2-1}$$

remaining candidate sets. Observe that, since n is even, it holds that $n/2 - 1 = \lfloor (n-1)/2 \rfloor$. Consequently, there are exactly $N(n-1)$ sets left out of which A has to select one. Therefore, A needs to distinguish between $N(n-1)$ different advices to distinguish this many sets. If A reads strictly less than $\log(N(n-1))$ advice bits, this merely enables it

to choose among strictly less than $N(n-1)$ strategies, which is equivalent to choosing among this many deterministic algorithms (see Observation 1.12).

By the pigeonhole principle, there exist two distinct sets S_1 and S_2 (i. e., inputs that both contain x_1, but differ in at least one element) for which A chooses the same deterministic strategy $A \in \mathrm{Alg}(\mathtt{A})$. Clearly, A cannot be optimal in both cases. Finally, using (1.4), we get

$$\begin{aligned}\log(N(n-1)) &= \log\left(\binom{n-1}{\lfloor\frac{n-1}{2}\rfloor}\right) \geq \log\left(\frac{4^{(n-1)/2}}{2\sqrt{(n-1)\pi/2}}\right)\\ &= (n-1) - \log\left(2\sqrt{(n-1)\pi/2}\right)\\ &> n - \log\left(\sqrt{8(n-1)}\right) - 1\\ &> n - \frac{1}{2}\log(n-1) - 3,\end{aligned}$$

which finishes the proof. □

The next question we are dealing with is how many bits of advice are necessary and sufficient to achieve (strict) c-competitiveness.

Theorem 5.4. *There exists a c-competitive online algorithm* A *with advice for* SetCover *that uses at most $n - ((c-1)k+1) + \lceil\log k\rceil + 2\lceil\log\lceil\log k\rceil\rceil$ advice bits, where k is the size of an optimal solution.*

Proof. Suppose that an optimal solution $\mathcal{O}pt$ covers all $|X'|$ requests with k members of $\mathcal{S}$. Following Observation 1.14, the number k can be communicated to A in a self-delimiting way using $\lceil\log k\rceil + 2\lceil\log\lceil\log k\rceil\rceil$ advice bits. Note that A may use at most $c\cdot\mathrm{cost}(\mathcal{O}pt) = \mathrm{cost}(\mathcal{O}pt) + (c-1)k$ sets to be c-competitive.

As in the proof of Theorem 5.2, O writes the characteristic function of X' onto the advice tape. Recall that A knows the first element from X' that is requested before it has to make any decision. Since A may use $(c-1)k$ additional sets in comparison to $\mathcal{O}pt$, it only needs to read the first $n-(c-1)k-1$ bits of advice, compute the optimal solution $\mathcal{O}pt^*$ for this sub-instance (which, of course, has a smaller cost than $\mathcal{O}pt$) and take one additional set for every requested element that is not already covered by $\mathcal{O}pt^*$. □

Note that, for any $k \geq 5$,

$$c \leq ck + 1 - k - \lceil\log k\rceil - 2\lceil\log\lceil\log k\rceil\rceil$$

follows from

$$c \geq \frac{k + \log k + 2\log\log k + 2}{k-1},$$

which is true for any $c \geq 3$. Together with Theorem 5.4, this implies the following corollary.

Corollary 5.5. *Let I be some instance of* SetCover *such that any optimal solution uses at least five sets to cover all requests. For any $c \geq 3$, there exists a c-competitive online algorithm* A *with advice for I that uses at most $n - c$ advice bits.* □

Next, generalizing the idea from the proof of Theorem 5.3, we show a lower bound on the number of advice bits required for any online algorithm that achieves strict c-competitiveness. To do so, we again consider an adversary Adv that plays against some

online algorithm A with advice, i. e., against 2^b deterministic algorithms at once (see Observation 1.12). Thus, similar to the proof of Theorem 2.12, Adv constructs one instance that is hard for *all* algorithms $A_1, \ldots, A_{2^b} \in \mathrm{Alg}(\mathtt{A})$.

Theorem 5.6. *Any online algorithm with advice for* SetCover *that reads at most*

$$\left(\frac{n-1}{c(c+1)} - \frac{c+2}{c+1}\right) \log\left(\frac{c+1}{c}\right)$$

advice bits is guaranteed to be worse than strictly c-competitive.

Proof. Let n be a multiple of $c+1$. In the following, for any given c, $\mathcal{S}$ consists of all possible sets of size $n/(c+1)$ in $\mathcal{P}(X)$. Similar to the previous discussion, an adversary Adv chooses exactly $n/(c+1)$ items such that there exists a unique optimal solution that consists of exactly one set. Thus, a strictly c-competitive algorithm is allowed to choose c sets at most.

Let A be an online algorithm that uses b bits of advice. We will determine a number b such that all algorithms in Alg(A) fail to be strictly c-competitive, i. e., are forced to choose at least $c+1$ sets. To this end, Adv proceeds in rounds, where, in each round, all algorithms from Alg(A) are forced to choose an additional set. Then again, as already mentioned, there is an optimal solution that consists of only one single set for the whole instance.

There are $c+1$ rounds numbered $0, \ldots, c$. At the end of round i, Adv has to ensure that all algorithms have chosen at least $i+1$ sets so far. In each time step, the algorithms in Alg(A) are partitioned into those that already chose an additional set in the current round (these algorithms are called *out for this round*) and those that did not yet choose one additional set.

Adv creates an input instance $I = (i_1, \ldots, i_{n/(c+1)})$ that corresponds to exactly one set in $\mathcal{S}$. As in the proof of Theorem 5.3, the item i_1 is fixed as x_1 (in round 0, i. e., before A produces any output) such that each algorithm in Alg(A) has to choose a set containing x_1. Subsequently, Adv enters round 1 and it chooses an item such that as many algorithms from Alg(A) as possible are out for this round, because they do not have this item covered yet. Adv continues in this fashion until, eventually, round 1 is finished since *all* algorithms are out for this round; the other rounds work analogously.

Let p_0 be 1 and let $p_1, \ldots, p_c$ be natural numbers such that Adv chooses p_i elements to finish round i, i. e., Adv chooses $p_i - p_{i-1}$ many items to force all algorithms in Alg(A) to be out for round i. At the beginning of round r, there are, in total, 2^b subsets of $\mathcal{S}$ that were determined by the algorithms from Alg(A) each of which contains the items of the earlier rounds. We can assume that there is no uncovered item; otherwise, Adv requests exactly this one thus finishing round r. Clearly, we do not have to consider any algorithms that already chose $r+1$ sets in previous rounds. Therefore, Adv only deals with algorithms that have chosen a total number of exactly r sets and thus at most $rn/(c+1)$ items. From these items, p_{r-1} are already fixed, because they were requested in previous rounds; hence, there are $rn/(c+1) - p_{r-1}$ items that may be requested to make one particular algorithm out for round r. Summing over all 2^b algorithms, the number of occurrences of all not yet requested items is at most

$$2^b \cdot \left(\frac{rn}{c+1} - p_{r-1}\right).$$

On the other hand, in the first time step of round r, ADV can choose from $n - p_{r-1}$ items. It follows that the average number of occurrences of one choosable item is at most

$$\frac{2^b\left(\frac{rn}{c+1} - p_{r-1}\right)}{n - p_{r-1}}.$$

Therefore, the least frequently covered item has already been covered by at most that number of different algorithms in $\mathrm{Alg}(\mathtt{A})$; this is exactly the item $i_{p_{r-1}+1}$ that ADV chooses in this time step. Since any algorithm is out for round r if it has chosen a family of sets not containing this item, in the subsequent steps, we only have to focus on the remaining algorithms in $\mathrm{Alg}(\mathtt{A})$.

More generally, ADV can reduce the number of algorithms that are not out for round r by a fraction of

$$\frac{\frac{rn}{c+1} - j}{n - j}$$

or more, for $j \geq p_{r-1}$, when requesting the $(j+1)$th element of the instance I. Next, let $s \leq rn/(c+1) + 1$ be a natural number such that

$$2^b \prod_{j=p_{r-1}}^{s-1} \frac{\frac{rn}{c+1} - j}{n - j} < 1. \tag{5.1}$$

Then, it follows that $p_r \leq s$. Note that, since $r < c+1$, we know that $n > rn/(c+1) \geq s-1$ holds in (5.1), thus the fraction is well-defined. Moreover, there exists s such that (5.1) holds, e. g., $s = rn/(c+1) + 1$. Therefore, the value of the left-hand side of (5.1) is at most

$$2^b \prod_{j=p_{r-1}}^{s-1} \frac{\frac{rn}{c+1}}{n} = 2^b \left(\frac{r}{c+1}\right)^{s-p_{r-1}}. \tag{5.2}$$

As a result, to satisfy (5.1), it is sufficient to choose s such that

$$\begin{aligned} & 2^b \left(\frac{r}{c+1}\right)^{s-p_{r-1}} < 1 \\ \iff & \log\left(\left(\frac{r}{c+1}\right)^{s-p_{r-1}}\right) < \log\left(2^{-b}\right) \\ \iff & b \cdot \frac{1}{\log(c+1) - \log r} < s - p_{r-1}. \end{aligned}$$

Following this, we can safely set

$$s := \left\lfloor \frac{b}{\log(c+1) - \log r} \right\rfloor + 1 + p_{r-1}. \tag{5.3}$$

Clearly, ADV succeeds if it uses not more than $n/(c+1)$ items in total, which means that

$$1 + \sum_{r=1}^{c} (p_r - p_{r-1}) \leq \frac{n}{c+1}. \tag{5.4}$$

Since $p_r \leq s$, (5.4) is guaranteed by

$$1 + \sum_{r=1}^{c} \left(\left\lfloor \frac{b}{\log\left(\frac{c+1}{r}\right)} \right\rfloor + 1 \right) \leq \frac{n}{c+1},$$

which is implied by

$$b \cdot \sum_{r=1}^{c} \frac{1}{\log\left(\frac{c+1}{r}\right)} \leq \frac{n}{c+1} - c - 1 \iff b \leq \frac{\frac{n}{c+1} - c - 1}{\sum_{r=1}^{c} \frac{1}{\log\left(\frac{c+1}{r}\right)}},$$

which again holds if

$$b \leq \frac{\frac{n-c^2-2c-1}{c+1}}{\frac{c}{\log\left(\frac{c+1}{c}\right)}} = (n - c^2 - 2c - 1)\frac{\log\left(\frac{c+1}{c}\right)}{c^2 + c} = \left(\frac{n-1}{c(c+1)} - \frac{c+2}{c+1}\right)\log\left(\frac{c+1}{c}\right).$$

It follows that, if b is smaller than claimed in the statement of the theorem, ADV can make sure that any algorithm that uses b advice bits is worse than strictly c-competitive. □

Note that, for the instances we constructed in the proof of Theorem 5.6, there exists a very simple *deterministic* online algorithm that achieves a competitive ratio of $c + 1$. Clearly, it is sufficient to use $c + 1$ disjoint sets from $\mathcal{S}$ to cover all items from X. This means that, for such instances, we observe a very interesting threshold: for being a little better, we have to pay with using a linear number of advice bits instead of no advice at all.

Due to the fact that

$$\left(\frac{n-1}{c(c+1)} - \frac{c+2}{c+1}\right)\log\left(\frac{c+1}{c}\right) \in \Omega(n),$$

for any constant c, we immediately obtain the following statement.

Corollary 5.7. *Any online algorithm with advice that achieves a constant strict competitive ratio c is required to use a number of advice bits that is linear in n.* □

Therefore, the upper bound of Theorem 5.4 is tight up to a constant factor.

5.2 Bounds with Respect to the Size of the Set Family

In the previous section, we measured the advice complexity in the number $n = |X|$ of elements contained in the ground set. Now we look at SETCOVER from a different perspective by measuring the advice in the number $m = |\mathcal{S}|$ of given sets. As we discuss at the end of this section, we obtain bounds that are not comparable with those presented before.

First, let us consider the advice needed to create an optimal output. Encoding the characteristic function of the sets that are used by an optimal solution immediately gives the following result.

Theorem 5.8. *There exists an optimal online algorithm* A *with advice for* SETCOVER *that uses m bits of advice.* □

As in the previous section, this very naive approach is asymptotically the best we can hope for.

Theorem 5.9. *Any online algorithm* A *with advice for* SETCOVER *needs to read at least*

$$\frac{m \log 3}{3} - 2$$

advice bits to be optimal.

Proof. For any m, let m' denote the largest number that is smaller than or equal to m and that is a multiple of 3, i. e., $m' \geq m-2$. Moreover, let $|X| = n := 4m'/3$. ADV chooses X such that there are $n/4$ items $y_1, \ldots, y_{n/4}$ and $3n/4$ items $x_{i,j}$, where $i \in \{1,2,3\}$, $j \in \{1, \ldots, n/4\}$. After that, ADV defines $\mathcal{S}$ to contain exactly the following sets. For $k \in \{1, \ldots, n/4\}$, there are three sets $\{y_k, x_{1,k}\}$, $\{y_k, x_{2,k}\}$, and $\{y_k, x_{3,k}\}$. Since $\mathcal{S}$ has to be of size m, ADV adds $m - m'$ many *dummy* sets to $\mathcal{S}$ that each contain one unique item y_j only, for $j \in \{1, \ldots, m-m'\}$; these sets are never considered by any optimal solution (which A may assume, because they are subsets of other sets).

Next, ADV requests all items y_i; hence, since each request has to be satisfied, A has to fix $n/4$ sets to cover all items. After that, for every i, ADV requests one of the three items $x_{1,i}$, $x_{2,i}$, and $x_{3,i}$. Note that any combination leads to a distinct optimal solution that consists of exactly $m'/3$ sets. This way, A has to correctly choose one out of $3^{m'/3}$ possible families as answers for the first $m'/3$ requests.

If, however, A uses less than $\log(3^{m'/3})$ advice bits, there must be two identical advices for two different sequences of correct answers (which again follows from the pigeonhole principle). Thus, ADV can pick the one not chosen by the algorithm. Consequently, a lower bound on the advice complexity is

$$\log\left(3^{m'/3}\right) \geq \frac{m-2}{3} \log 3 > \frac{m \log 3}{3} - 2,$$

which is larger than $m/2$, for m tending to infinity. □

Let us now consider algorithms that aim at achieving c-competitiveness.

Theorem 5.10. *For any $c \leq \log m + 1$, there exists a c-competitive online algorithm* A *with advice for* SETCOVER *that uses at most*

$$m - \frac{(c-1)m}{\lceil \log m \rceil + c - 1} + 1$$

advice bits.

Proof. Suppose that $\mathcal{O}pt$ solves an arbitrary instance using exactly k sets. In the following, let $t := m/(\lceil \log m \rceil + c - 1)$. First, we observe that, if $k \leq t$, it clearly suffices to communicate

$$\frac{m}{\lceil \log m \rceil + c - 1} \cdot \lceil \log m \rceil = m - \frac{(c-1)m}{\lceil \log m \rceil + c - 1}$$

bits to A to be optimal by writing the indices of all sets taken by $\mathcal{O}pt$ onto the advice tape. Clearly, we need one additional bit to indicate that this is the case. Note that we do not need to encode k into the advice, but we can pad the advice string by duplicating certain sets instead.

We may therefore assume the other case, i. e., $k > t$. Suppose that, in this case, O again writes the characteristic function of $\mathcal{S}$ onto the advice tape. As in the proof of Theorem 5.4, A may use $(c-1)k \geq (c-1)t$ more sets than $\mathcal{O}pt$ to guarantee c-competitiveness. This means that, if A reads the first $m - (c-1)t$ bits of the advice tape, it can act optimally up to this point and may safely take one set for each remaining uncovered request. We immediately verify that

$$m - (c-1)t = m - \frac{(c-1)m}{\lceil \log m \rceil + c - 1}.$$

Finally, as above, the number k does not need to be communicated to A. Since m and c are known to A by construction, no further advice is necessary. □

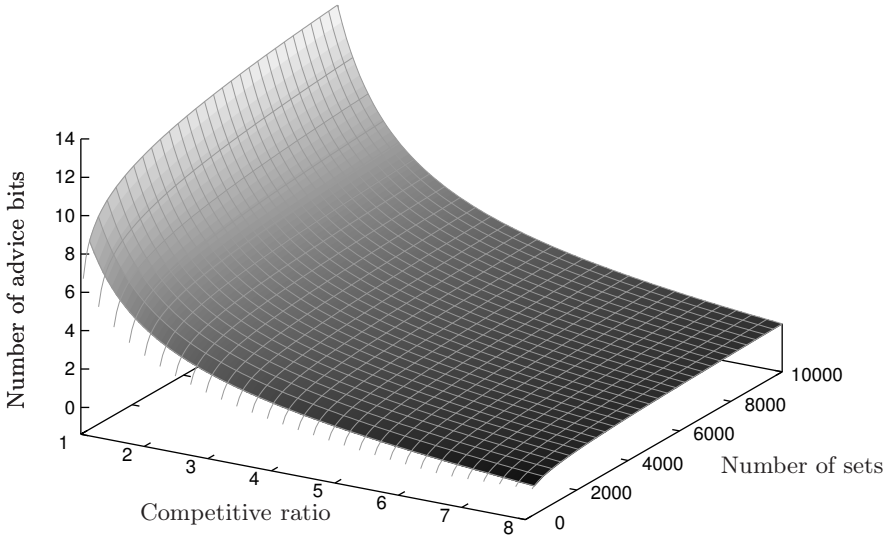


Figure 5.1: The number of advice bits required depending on c and m.

We now give a lower bound on yielding strict c-competitiveness measured in m. This way, we obtain a slightly better lower bound than the one that is directly implied by the lower bound measured in n from the previous section.

Theorem 5.11. *Any online algorithm* A *with advice for* SetCover *needs to read at least* $(\log m)/c - 1$ *advice bits to be strictly* c*-competitive.*

Proof. Let A use b bits of advice and let $\gamma := 2^b$. Adv constructs a ground set X that consists of all words of lengths up to c (including the empty word λ) over an $(\gamma + 1)$-ary alphabet. Every element S from $\mathcal{S}$ is uniquely defined by a word $w(S)$ of length c. Moreover, S contains *all* prefixes of $w(S)$. Note that, by definition, λ is included in every set.

Without loss of generality, we assume that, in what follows, no algorithm chooses more than one set per request. Now, at first, Adv requests the element λ which causes every algorithm from Alg(A) to choose one set from $\mathcal{S}$. Every of these sets contains one element of length 1. However, the (at most) 2^b different choices left at least one such element x' uncovered. Adv now requests an element of length 2 that has the prefix x'. It immediately follows that every algorithm from Alg(A) has to choose a second set. Adv now inspects the words of length 2 within these sets. There is at least one word x'' (which has the prefix x') not covered and Adv uses it as a prefix to choose a word of length 3 which it requests in what follows, etc.

In general, after i requests, all algorithms from Alg(A) used at least i sets, and thus Adv can ensure that A used c sets in total after c requests, where all requested words are increasing prefixes of a distinct word w of length c.

However, by construction, there again exists *one* set in $\mathcal{S}$ that covers all requests, which is the unique optimal solution. We conclude that, in order to be strictly c-competitive, $b = \log \gamma$ has to be chosen such that

$$m < (\gamma + 1)^c \iff m^{1/c} - 1 < 2^b,$$

which can be ensured by choosing

$$b > \frac{\log m}{c} - 1.$$

Hence, we may assume that b is at least $(\log m)/c - 1$ as we claimed. □

The lower bound on the tradeoff between c and b is shown in Figure 5.1. Compare these results to the ones measured in n from the previous section. As for optimality, in Theorem 5.9, we had $4m/3 \geq n \geq 4(m-2)/3$ and thus $m \leq (3n+8)/4$ yielding a lower bound of less than $((n+3)\log 3)/4$, which is worse than the one of Theorem 5.3. On the other hand, in the proof of Theorem 5.3, we had $m = N(n)$, which means that it merely gives a logarithmic lower bound in m, whereas Theorem 5.9 proves a linear lower bound. If we look at the tradeoff between the advice bits and the competitive ratio c, in Theorem 5.11, we clearly have $m = (\gamma+1)^c$ and

$$n = \sum_{i=0}^{c} (\gamma+1)^i = \frac{(\gamma+1)^{c+1} - 1}{\gamma}.$$

Thus, Theorem 5.11 yields a lower bound that is logarithmic in n and which is therefore worse than the one of Theorem 5.6. Furthermore, inspecting the proof of this theorem, we observe that here

$$m = \binom{n}{n/(c+1)} \geq (c+1)^{n/(c+1)}.$$

Hence, the bound of Theorem 5.11 is better with respect to m and c.

Chapter 6

The Knapsack Problem

In the previous chapter, we studied SetCover which, in its offline version, is well-known to be $\mathcal{NP}$-hard. Another such problem is the *knapsack problem* [88, 103], whose decision version is also among Karp's 21 $\mathcal{NP}$-complete problems [101]. Here, we are given a knapsack of some fixed capacity and a number of items that can be packed into this knapsack. Every such item has both a weight and a value associated with it; in its simple version, the value and the weight of each item are identified. The objective is to maximize the sum of all values of items packed into the knapsack while not exceeding its capacity. There is a *pseudo-polynomial-time algorithm* known that uses dynamic programming. Ibarra and Kim used this approach to design a *fully polynomial-time approximation scheme* [88] for the knapsack problem in [91]; thus, considering the hardness with respect to approximation, the knapsack problem is easier than the set cover problem, which is, as we already know, not even approximable by a constant factor.

In the online version of the problem, denoted by Knapsack, the items arrive one after another in consecutive time steps; after each item is offered to the algorithm, it has to decide whether it packs the item into the knapsack or not. Formally, Knapsack is the following maximization problem.

Definition 6.1 (Knapsack, SimpleKnapsack). *Let I denote a sequence of n items that are pairs of weights and values, i. e., $I = (s_1, \ldots, s_n)$, $s_i = (w_i, c_i)$, where $0 < w_i \leq 1$ and $c_i > 0$, for $i \in \{1, \ldots, n\}$. A feasible solution for* Knapsack *is given by any set of indices $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} w_i \leq 1$; the goal is to maximize $\sum_{i \in S} c_i$. The items are given in an online fashion. For each item, an online algorithm* A *must specify whether it is part of the solution or not, as soon as it is offered.*

In the simple *version of* Knapsack*, denoted by* SimpleKnapsack*, each item has a value smaller than* 1 *that is equal to its weight.*

Since the value of an optimal solution for any instance of SimpleKnapsack is bounded by the constant capacity 1 of the knapsack, we only consider strict competitiveness for this problem (see Definition 1.3). Moreover, note that we use the terms weight and size interchangeably.

6.1 The Unweighted Case

At first, we consider the unweighted version SimpleKnapsack of the problem as presented in Definition 6.1. Let us start by discussing deterministic online algorithms. The following fact is well-known.

Theorem 6.2 (Marchetti-Spaccamela and Vercellis [122]). *Any deterministic online algorithm for* SimpleKnapsack *(and thus for* Knapsack*) without advice fails to be competitive.* ♢

Let us now consider an online algorithm `G` that implements a straightforward greedy approach. This means that `G` takes any item while there is space left for it in the knapsack. Of course, this strategy also fails in general (as the last theorem implies), but for a particular set of instances it works quite well as stated by the following observation.

Observation 6.3. *Let I denote any instance of* SimpleKnapsack *where every item has a weight of at most β. Then* `G` *achieves a gain of at least $1 - \beta$ or it is optimal.*

Indeed, if the sum of all weights is less than 1, `G` is optimal. Then again, if this is not the case, the space in the knapsack that is not covered by `G`'s solution cannot be larger than β.

Online Algorithms with Advice

Next, we study the number of advice bits both sufficient and necessary to produce optimal output. As with the majority of the other problems that we have considered so far, there is a linear upper bound; again, we use the characteristic function.

Theorem 6.4. *For* SimpleKnapsack*, there exists an optimal online algorithm* `A` *with advice that uses n advice bits.*

Proof. For each of the n items, one bit of advice tells `A` whether this item is part of an arbitrary, but fixed, optimal solution or not. □

This bound is tight as the next theorem shows.

Theorem 6.5. *Any online algorithm with advice for* SimpleKnapsack *needs to read at least $n - 1$ advice bits to be optimal.*

Proof. For any n, consider the input $1/2, 1/4, \ldots, 1/2^{n-1}, s$, where the item s is defined as

$$s := 1 - \sum_{i=1}^{n-1} b_i 2^{-i},$$

for some binary vector $b = (b_1, \ldots, b_{n-1})$. Consider the first $n-1$ items of the input. Any two different subsets of these items have a different weight. From this, it directly follows that, for any distinct value of b, there exists a *unique* optimal solution with gain 1. In other words, when s is offered, there was one optimal choice for the algorithm.

If any online algorithm uses strictly less than $n - 1$ advice bits, it cannot distinguish between all 2^{n-1} different inputs due to the pigeonhole principle. As a result, it computes the same output for the first $n - 1$ items for two different instances and thus produces a sub-optimal solution for at least one of them. □

Next, let $\mathtt{A}_1$ be an online algorithm with advice that reads one advice bit. This bit indicates whether there exists an item s within the input that has size $> 1/2$. If this bit is 0, $\mathtt{A}_1$ acts greedily; if it is 1, $\mathtt{A}_1$ takes nothing until an item of size $> 1/2$ appears (and anything else greedily afterwards).

Theorem 6.6. *The online algorithm* $\mathtt{A}_1$ *for* SimpleKnapsack *is* 2*-competitive.*

Proof. Suppose that there is no item of size $> 1/2$. In this case, the claim is implied by Observation 6.3. However, if there exists an item of size $> 1/2$, the claim follows immediately. □

This result seems counterintuitive. With merely one bit of advice and a straightforward approach we jump from an unbounded output quality (see Theorem 6.2) to 2-competitiveness. However, any further increase of the number of advice bits does not help until a logarithmic number is reached. $\mathtt{A}_1$ therefore reaches the best competitive ratio we can hope for when dealing with any constant number of advice bits.

Theorem 6.7. *Let* $b < \lfloor \log(n-1) \rfloor$ *and* $\varepsilon > 0$*. No online algorithm* $\mathtt{A}$ *with advice for* SimpleKnapsack *that uses* b *advice bits is better than strictly* $(2-\varepsilon)$*-competitive.*

Proof. Let $\delta := \varepsilon/(4-2\varepsilon)$ and let $\mathtt{A}$ read b bits of advice. Consider the class $\mathcal{I}$ of inputs I_j, for $j \in \{1, \ldots, n-1\}$, of the form

$$\frac{1}{2}+\delta, \frac{1}{2}+\delta^2, \ldots, \frac{1}{2}+\delta^j, \frac{1}{2}-\delta^j, \frac{1}{2}+\delta, \ldots, \frac{1}{2}+\delta,$$

where the item of weight $1/2+\delta$ appears $n-j-1$ times at the end of the instance, for $j \in \{1, \ldots, n-1\}$. Obviously, since $|\mathcal{I}| > 2^b$, there are more inputs than strategies to choose from, and thus there are two different inputs for one advice string. In order to be optimal, $\mathtt{A}$ needs to take the jth and $(j+1)$th item for the instance I_j, and hence this choice is unique for every input from $\mathcal{I}$. When dealing with I_j, for any other choice of items, $\mathtt{A}$ achieves a gain of at most $1/2+\delta$, leading to a strict competitive ratio of

$$\frac{1}{\frac{1}{2}+\delta} = 2-\varepsilon$$

as we claimed. □

The competitive ratio that is achievable with respect to the number of used advice bits now makes a second jump as stated by the following theorem.

Theorem 6.8. *Let* $\varepsilon > 0$*. There exists an online algorithm* $\mathtt{A}$ *with advice for* Simple-Knapsack *that achieves a competitive ratio of* $1+\varepsilon$ *and that reads at most*

$$\left\lceil \frac{3\varepsilon+3}{\varepsilon} \right\rceil \cdot \lceil \log n \rceil + 2 \cdot \left\lceil \log\left(\left\lceil \frac{3\varepsilon+3}{\varepsilon} \right\rceil + 1 \right) \right\rceil + 2 \cdot \lceil \log \lceil \log n \rceil \rceil + 1$$

advice bits.

Proof. Let $\delta := \varepsilon/(3+3\varepsilon)$. Suppose that there does not exist any item within the input of size larger than δ which can be indicated using one bit at the beginning of the advice tape. Then, by Observation 6.3, $\mathtt{A}$ may safely take sets greedily which leads to a competitive ratio of

$$\frac{1}{1-\delta} = 1 + \frac{\delta}{1-\delta} = 1 + \frac{\varepsilon}{3+2\varepsilon} \leq 1+\varepsilon.$$

Now assume the contrary, i. e., there exist some items of size $> \delta$. The oracle inspects the optimal solution which consists of two disjoint sets of items S_1 and S_2, where S_1 denotes the set of i *heavy* items of size $> \delta$ and S_2 contains j *light* items of size $\leq \delta$. Let s_1 [s_2] be the sum of all weights of the items in S_1 [S_2]. The indices of all heavy items are written onto the advice tape using $i \cdot \lceil \log n \rceil$ bits. Moreover, we need to communicate i which can be done using another $\lceil \log \lceil 1/\delta \rceil \rceil$ bits. Since the sum of all weights of any solution does not exceed 1, we clearly have $i \leq 1/\delta$, i. e., i is constant with respect to n. For being able to decode the advice string, additionally the length $\lceil \log n \rceil$ of such an index has to be included in the advice in a self-delimiting form using $2\lceil \log \lceil \log n \rceil \rceil$ bits (see Observation 1.14).

Moreover, let the oracle encode a number k onto the advice tape such that

$$k\delta \leq s_2 < (k+1)\delta.$$

Since A knows ε and therefore δ, it computes $k\delta$ and thus obtains a lower bound on s_2, i. e., the weight of the part of the solution that is due to the light items. Every such light item is taken as long as their sum is below $k\delta$. It is clear that $k \leq 1/\delta$ due to $s_2 \leq 1$. According to Observation 6.3, A packs at least as many items from S_2 into the knapsack such that their sum is not smaller than $k\delta - \delta \geq s_2 - 2\delta$. Observe that, if there do not exist any light items (i. e., S_2 is empty), A is clearly optimal, because it takes all heavy items. Thus, we may assume that there exists at least one light item and the optimal solution takes it. Furthermore, if, under this assumption, the optimal solution would be smaller than $1 - \delta$, it follows that it takes all light items. In this case, we set $k := \lceil 1/\delta \rceil$ which again results in an optimal algorithm. We therefore assume the contrary, i. e., that $\text{cost}(\mathcal{O}pt) \geq 1 - \delta$. Consequently, we get a competitive ratio of

$$\frac{s_1 + s_2}{s_1 + s_2 - 2\delta} \leq \frac{1}{1 - 3\delta} = 1 + \frac{3\delta}{1 - 3\delta} = 1 + \varepsilon.$$

Since k is an integer from $\{0, \ldots, \lceil 1/\delta \rceil\}$, it can be encoded using $\lceil \log(\lceil 1/\delta \rceil + 1) \rceil$ bits (recall that, in Observation 1.13, we assumed that the number that is encoded is non-zero). Note that the lengths of i and k do not need to be encoded in a self-delimiting way since δ is known to A. The total number of advice bits used by the algorithm is thus at most

$$\begin{aligned} &1 + i \cdot \lceil \log n \rceil + 2 \cdot \left\lceil \log\left(\left\lceil \frac{1}{\delta} \right\rceil + 1 \right) \right\rceil + 2 \cdot \lceil \log \lceil \log n \rceil \rceil \\ &\leq 1 + \left\lceil \frac{3\varepsilon + 3}{\varepsilon} \right\rceil \cdot \lceil \log n \rceil + 2 \cdot \left\lceil \log\left(\left\lceil \frac{3\varepsilon + 3}{\varepsilon} \right\rceil + 1 \right) \right\rceil + 2 \cdot \lceil \log \lceil \log n \rceil \rceil, \end{aligned}$$

which finishes the proof. □

Randomized Online Algorithms

As we have just seen (see Theorem 6.8), logarithmic advice helps a lot. We now show that randomization is less powerful.

Theorem 6.9. *No randomized online algorithm for* SimpleKnapsack *can be better than strictly* 2*-competitive in expectation (independent of the number of random bits the computation is based on).*

Proof. Let $\varepsilon > 0$. Consider the following class of inputs. At first, an item of weight ε is offered. After that, either nothing else is offered or an additional item of weight 1.

Now consider any randomized online algorithm R that decides to use the first item with non-zero probability p, because else its gain is obviously 0 on the instance consisting of the item of weight ε only. If R takes the item, of course, it cannot use the second one if it is offered. On the other hand, if R does not take the first item (with probability $1-p$), it does not have any gain if there is no second item. Now suppose that the second item is offered; R then has a strict competitive ratio of

$$\frac{1}{p \cdot \varepsilon + (1-p) \cdot 1}.$$

Then again, if the second item is not offered, it has a strict competitive ratio of

$$\frac{\varepsilon}{p \cdot \varepsilon}.$$

By equalizing the ratios [146], we get

$$\frac{1}{(\varepsilon - 1) \cdot p + 1} = \frac{1}{p} \iff p = \frac{1}{2 - \varepsilon}.$$

Thus, R is not better than strictly $(2-\varepsilon)$-competitive in expectation. □

Surprisingly, in Section 7.1, we show that there is a barely random algorithm for SimpleKnapsack that achieves an expected competitive ratio of 2 and that only uses one random bit.

Resource Augmentation

In Section 1.5, we have, among others, described the idea of *resource augmentation* to overcome some of the drawbacks online algorithms face. This model was used for Knapsack [95] as well as for many other online problems, see, e. g., [59, 98, 133]. Here, we want to combine this concept with the one of using advice to investigate how much an online algorithm benefits from having more resources available concerning the amount of information sufficient to produce almost optimal output. More precisely, we now allow an online algorithm A with advice to overpack the knapsack by some $\delta > 0$, whereas the optimal solution is merely allowed to fill it up to 1.

Theorem 6.10. *Let $1/4 > \delta > 0$. There exists an online algorithm A with advice for* SimpleKnapsack *that achieves a competitive ratio of $1 + 3 \cdot \delta/(1-4\delta)$ in the δ-resource-augmented model, while using at most*

$$\left\lceil 2 \log \left\lceil \frac{1}{\delta} \right\rceil + \frac{1}{\delta} \cdot \log \left\lceil \frac{1}{\delta^2} \right\rceil \right\rceil + 1$$

advice bits.

Proof. Consider any instance I and let $\mathcal{O}pt = \textsc{Opt}(I)$ denote an optimal solution computed by an optimal offline algorithm Opt. Suppose that $\text{cost}(\mathcal{O}pt) \leq 1/2$. In this case, there is obviously no item of size $> 1/2$, and a simple greedy strategy enables A to be optimal. We fix the first advice bit to indicate whether $\mathcal{O}pt$ has size $1/2$ or smaller and, in the further analysis, assume the contrary.

Similar to the proof of Theorem 6.8, let

$$\mathcal{O}pt := \{x_1, \ldots, x_k\} \dot{\cup} \{y_1, \ldots, y_m\}$$

denote an optimal solution, where the *heavy* items x_i have weights $\geq \delta$ and the *light* items y_j have weights $< \delta$; clearly, we have $k \leq 1/\delta$. A knows δ and is designed such that it reads all the approximate sizes (computed by rounding down to the next multiple of δ^2) of all heavy items and the fraction of the knapsack that is filled using light ones from the advice tape. Light items are taken greedily.

First, we show how the heavy items are encoded. To this end, let

$$\overline{x}_i := j \text{ such that } j \cdot \delta^2 \leq x_i < (j+1) \cdot \delta^2,$$

for every heavy item x_i. All $\overline{x}_i$s are sequentially written onto the advice tape and read by A right after the first item is offered. Thus, if A is offered any item x', it checks whether the corresponding $\overline{x}_i$ is part of the advice, i. e., if there exists $\overline{x}_i$ such that $\delta^2 \cdot \overline{x}_i \leq x' < \delta^2 \cdot (\overline{x}_i + 1)$. If so, x' is packed into the knapsack as an element x'_i corresponding to x_i; else it is discarded. In the former case, there exists x_i that is part of $\mathcal{O}pt$ and $x_i - \delta^2 < x'_i < x_i + \delta^2$. Clearly, there are at most k different $\overline{x}_i$s (as many as there are corresponding heavy items) and each is at most of size $1/\delta^2$. Hence, to communicate all of these values, we need no more than

$$k \cdot \log \left\lceil \frac{1}{\delta^2} \right\rceil \leq \left\lceil \frac{1}{\delta} \cdot \log \left\lceil \frac{1}{\delta^2} \right\rceil \right\rceil$$

advice bits, which is constant with respect to n. However, to be able to decode the advice, A needs to know k beforehand. The value of k can be written onto the advice tape in a self-delimiting form using another $2\lceil \log k \rceil \leq 2\lceil \log \lceil 1/\delta \rceil \rceil$ additional bits. The number of bits needed to encode the elements $\overline{x}_i$ can be calculated by A without any further knowledge.

Note that we have

$$-\delta + \sum_{i=1}^{k} x_i \leq \sum_{i=1}^{k} x'_i \leq k \cdot \delta^2 + \sum_{i=1}^{k} x_i \leq \delta + \sum_{i=1}^{k} x_i,$$

which means that, for every heavy item, A chooses an item such that the algorithm uses at most δ^2 more space within the knapsack. Thus, the sum of the chosen heavy items uses at most δ more space than the heavy items in $\mathcal{O}pt$.

We distinguish two cases with respect to the size of an optimal solution.

Case 1. Suppose that $\text{cost}(\mathcal{O}pt) < 1 - \delta$. This directly implies that all light items are part of the optimal solution, because otherwise $\mathcal{O}pt$ would not be optimal. A uses at most δ more space for the heavy items than $\mathcal{O}pt$, hence it takes all light items as well. On the other hand, the sum of the weights of the heavy items chosen by A is at least $-\delta + \sum_{i=1}^{k} x_i$, thus A has a gain of at least $\text{cost}(\mathcal{O}pt) - \delta$. It follows that

$$\text{comp}(\mathtt{A}(I)) \leq \frac{\text{cost}(\mathcal{O}pt)}{\text{cost}(\mathcal{O}pt) - \delta} = 1 + \frac{\delta}{\text{cost}(\mathcal{O}pt) - \delta} \leq 1 + \frac{2\delta}{1 - 2\delta} \leq 1 + \frac{3\delta}{1 - 4\delta},$$

where the third inequality follows from the fact that $\text{cost}(\mathcal{O}pt) \geq 1/2$.

Case 2. Suppose that $1 - \delta \leq \text{cost}(\mathcal{O}pt) \leq 1$. Let us now consider the light items. To this end, let $g := 1 - \sum_{i=1}^{k} x_i$ denote the space $\mathcal{O}pt$ is left with after packing all

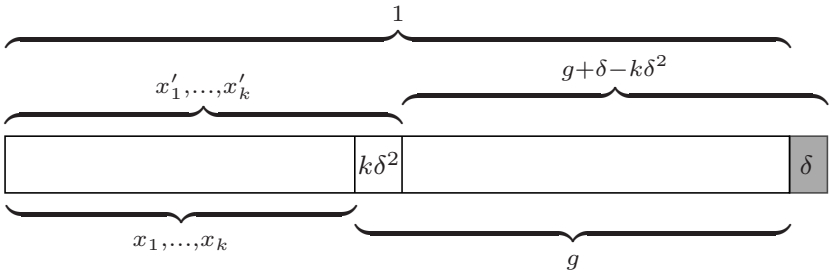


Figure 6.1: The gains of both $\mathcal{O}pt$ and A.

heavy items into the knapsack (see Figure 6.1). A does not know g, but calculates the approximate value

$$g' := 1 - \sum_{i=1}^{k} (\overline{x}_i + 1)\delta^2.$$

It follows that $g-\delta \leq g' \leq g$. Note that both A and $\mathcal{O}pt$ have all light items available. Now, for $z \in \{g, g'\}$, consider the instance $I(z)$ shrunk to a knapsack capacity of z and light items only, and let $\mathcal{O}pt(z)$ denote the corresponding optimal solution for this instance. Since A acts greedily on $I(g')$, by Observation 6.3, it follows that A is either optimal or has a gain of at least $g' - \delta \geq g - 2\delta \geq \text{cost}(\mathcal{O}pt(g)) - 2\delta$. On the other hand, OPT obtains a gain of exactly $\text{cost}(\mathcal{O}pt(g))$ on $I(g)$. Thus,

$$\begin{aligned}\text{comp}(\mathtt{A}(I)) = \frac{\text{cost}(\mathcal{O}pt)}{\text{cost}(\mathtt{A}(I))} &\leq \frac{\text{cost}(\mathcal{O}pt)}{\sum_{i=1}^{k} x'_i + g' - \delta} \leq \frac{\text{cost}(\mathcal{O}pt)}{\sum_{i=1}^{k} x_i + g - 2\delta} \\ &\leq \frac{\text{cost}(\mathcal{O}pt)}{\sum_{i=1}^{k} x_i + \text{cost}(\mathcal{O}pt(g)) - 3\delta} = \frac{\text{cost}(\mathcal{O}pt)}{\text{cost}(\mathcal{O}pt) - 3\delta} \leq 1 + \frac{3\delta}{1-4\delta},\end{aligned}$$

which proves the claim.

The theorem follows. □

6.2 The Weighted Case

We now consider the general online knapsack problem, KNAPSACK, from Definition 6.1 where every item has both a weight and a value. However, our results only hold if we restrict ourselves to instances where the values and weights can be represented within polynomial space. More formally, for any item x, let $w(x)$ be the weight of x, $c(x)$ be the value of x, and $r(x) := c(x)/w(x)$ be the ratio of its value and weight. We assume that, for every x, $c(x)$ and $w(x)$ are rational numbers, and their numerators and denominators are bounded by $2^{p(n)}$, for some fixed polynomial $p(n)$.

First of all, we note that the lower bounds for SIMPLEKNAPSACK from Section 6.1 carry over immediately, since we are now dealing with a generalization of the above problem. Second, Theorem 6.4 also applies for KNAPSACK. As a next step, we show that, as long as less than logarithmic advice is supplied, any online algorithm with advice fails to be competitive.

Theorem 6.11. *No online algorithm with advice for* KNAPSACK *that uses strictly less than* $\log n$ *advice bits is competitive.*

Proof. Suppose that A reads $k < \log n$ advice bits which allows it to distinguish at most 2^k different inputs. We construct a set $\mathcal{I}$ of n different instances as follows. Let $\gamma := 2^n$ and let I_s be the instance determined by the n items

$$(1, \gamma), (1, \gamma^2), \ldots, (1, \gamma^s), (1, 1), \ldots, (1, 1), (1, 1),$$

for $s \in \{1, \ldots, n\}$ and $\mathcal{I} = \{I_s \mid 1 \leq s \leq n\}$. Obviously, since $|\mathcal{I}| > 2^k$, there are more inputs than strategies to choose from, and therefore there are two different inputs for one advice string. Let these two instances be I_i and I_j and assume $i > j$. The unique optimal solution $\mathcal{O}pt$ for I_i [I_j] fills the knapsack with the ith [jth] item yielding a gain of γ^i [γ^j].

If A does not choose the jth item when it is given the instance I_j, its gain is at least a factor of γ away from $\text{cost}(\mathcal{O}pt)$. In the following, we thus assume the contrary. Since A cannot distinguish between I_i and I_j in the first j time steps (and it is given the same fixed advice string), it also takes the jth item when it is given I_i. This results in a competitive ratio of at least $\gamma^i/\gamma^j \geq \gamma$, which finishes the proof. □

In the following, we show how to solve Knapsack almost optimally while using logarithmic advice. It immediately follows that the bound from Theorem 6.11 is tight up to a constant factor.

Theorem 6.12. *Let $\varepsilon > 0$. There exists an online algorithm A with advice for* Knapsack *that achieves a competitive ratio of $1 + \varepsilon$ using at most $\mathcal{O}(\log n)$ advice bits.*

Proof. Let $\delta := (\sqrt{1+\varepsilon} - 1)/(2\sqrt{1+\varepsilon} + 1)$. Consider any optimal solution $\mathcal{O}pt$ and let

$$c' := (1+\delta)^{\lfloor \log_{1+\delta}(\text{cost}(\mathcal{O}pt)) \rfloor},$$

i. e., c' is an approximation of $\text{cost}(\mathcal{O}pt)$ such that

$$\frac{\text{cost}(\mathcal{O}pt)}{1+\delta} < c' \leq \text{cost}(\mathcal{O}pt).$$

Next, let $x_1, \ldots, x_k$ be all items in $\mathcal{O}pt$ with values of at least $\delta \cdot c'$. Since there are at most $\text{cost}(\mathcal{O}pt)/(\delta \cdot c')$ such items, we immediately get $k \leq (1+\delta)/\delta$.

Let $\mathcal{S}_1$ be an (offline) solution constructed as follows. At first, all *heavy* items $x_1, \ldots, x_k$ are taken; then, the rest of the knapsack is filled using items that have values less than $\delta \cdot c'$, which is done greedily by the ratio of their values and weights in descending order. Consider $\mathcal{S}_1$ plus the item x that is the first one that did not fit into the knapsack in the greedy phase of the construction of $\mathcal{S}_1$. Clearly, $\mathcal{S}_1 \cup \{x\}$ has a higher cost than $\mathcal{O}pt$. Since $c(x) \leq \delta \cdot c' \leq \delta \cdot \text{cost}(\mathcal{O}pt)$, we get that

$$\text{cost}(\mathcal{S}_1) \geq (1-\delta)\text{cost}(\mathcal{O}pt).$$

Let $y_1, \ldots, y_l$ denote the items of $\mathcal{S}_1$ that are added in the greedy phase. Without loss of generality, assume that $r(y_1) \geq r(y_2) \geq \cdots \geq r(y_l)$ and let

$$r' := (1+\delta)^{\lceil \log_{1+\delta}(r(y_l)) \rceil},$$

i. e., r' is an approximation of $r(y_l)$ such that

$$r(y_l) \leq r' < r(y_l) \cdot (1+\delta).$$

Let m be the largest number such that $r(y_m) \geq r'$, i. e., the items $y_1, \ldots, y_m$ have ratios of at least r' and all other items $y_{m+1}, \ldots, y_l$ have ratios between r' and $r'/(1+\delta)$. Let v denote the space that is not occupied by $x_1, \ldots, x_k, y_1, \ldots, y_m$ in $\mathcal{S}_1$, i. e.,

$$v := 1 - \sum_{i=1}^{k} w(x_i) - \sum_{i=1}^{m} w(y_i).$$

Intuitively speaking, if we consider the part of $\mathcal{S}_1$ that consists of the items y_i, for $i > m$, we see that this is a solution for an "almost-unweighted" knapsack instance with knapsack capacity v. Therefore, we can approximate it by a solution for SimpleKnapsack without doing much harm. To this end, let

$$v' := (1+\delta)^{\lfloor \log_{1+\delta} v \rfloor},$$

i. e., v' is an approximation of v such that

$$\frac{v}{1+\delta} < v' \leq v.$$

Furthermore, let

$$\{z_1, \ldots, z_j\} = S := \{y_i \mid y_i \in \{y_{m+1}, \ldots, y_l\}, w(y_i) \geq \delta \cdot v'\},$$

i. e., $z_1, \ldots, z_j$ are all items from $\mathcal{S}_1$ that have a ratio of roughly r' and whose weights are at least a δ-fraction of v'. Since $v' > v/(1+\delta)$, there are at most $(1+\delta)/\delta$ such items.

Let

$$u := v - \sum_{i=1}^{j} w(z_i)$$

denote the space not occupied by $x_1, \ldots, x_k, y_1, \ldots, y_m, z_1, \ldots, z_j$ and let

$$u' := (1+\delta)^{\lfloor \log_{1+\delta} u \rfloor},$$

i. e., u' is an approximation of u such that

$$\frac{u}{1+\delta} < u' \leq u.$$

Again, we consider an (offline) solution $\mathcal{S}_2$ that is constructed as follows. At first, all items

$$x_1, \ldots, x_k, y_1, \ldots, y_m, z_1, \ldots, z_j$$

are taken. After that, we use all remaining items of weight less than $\delta \cdot v'$ and a ratio of at least $r'/(1+\delta)$; each of these items is greedily added to $\mathcal{S}_2$ if it fits into a reserved space of size u'. We now show that

$$\text{cost}(\mathcal{S}_2) \geq \frac{1-2\delta}{(1+\delta)^2} \text{cost}(\mathcal{S}_1). \tag{6.1}$$

To this end, consider two cases. If the greedy construction of $\mathcal{S}_2$ takes all possible items, $\mathcal{S}_2$ contains all items included in $\mathcal{S}_1$ and (6.1) follows trivially. Therefore, we may assume the contrary.

Obviously, the cost of $\mathcal{S}_1$ is at most

$$\sum_{i=1}^{k} c(x_i) + \sum_{i=1}^{m} c(y_i) + v \cdot r' \leq \sum_{i=1}^{k} c(x_i) + \sum_{i=1}^{m} c(y_i) + v' \cdot (1+\delta) \cdot r'.$$

Algorithm 6.1: Algorithm A for KNAPSACK

```
for each item x
    if x = x_i, for some i, use;
    else if c(x) ≥ δ · c', discard;
        else if r(x) ≥ r' or x = z_i, for some i, use;
            else if r(x) < r'/(1 + δ) or w(x) ≥ δ · v', discard;
                else if total weight taken in this step ≤ u', use;
                    else discard;
end
```

On the other hand, the cost of $\mathcal{S}_2$ is at least

$$\sum_{i=1}^{k} c(x_i) + \sum_{i=1}^{m} c(y_i) + v' \cdot (1 - 2\delta) \cdot \frac{r'}{1+\delta},$$

because all items $z_1, \ldots, z_j$ (summing up to $v - u$) get packed and the greedy step fills the space that remains up to at least $u' - \delta \cdot v'$ (see Observation 6.3). Every such item has a ratio of at least $r'/(1+\delta)$. Note that

$$u' - \delta \cdot v' + v - u \geq (1-\delta)v' + u' - u \geq (1-\delta)v' - u'\delta \geq (1-2\delta)v'.$$

Thus, (6.1) is true.

Putting everything together, we finally get

$$\text{cost}(\mathcal{S}_2) \geq \frac{1-2\delta}{(1+\delta)^2}\text{cost}(\mathcal{S}_1) \geq \frac{(1-2\delta)^2}{(1+\delta)^2}\text{cost}(\mathcal{O}pt) = \frac{\text{cost}(\mathcal{O}pt)}{1+\varepsilon}.$$

Let us now look at the number of advice bits sufficient to be communicated to A. At first, O needs to encode n and k which can be done using not more than $2\lceil\log\lceil\log n\rceil\rceil + 2\lceil\log n\rceil$ bits. Furthermore, since A knows δ, it suffices to read at most

$$\left\lceil \log \left\lfloor \log_{1+\delta}\left(2^{p(n)}\right)\right\rfloor \right\rceil \leq \log\left(\frac{\log\left(2^{p(n)}\right)}{\log(1+\delta)}\right) + 1 \in \mathcal{O}\left(\log\left(n^d\right)\right)$$

advice bits to communicate c', where d is the degree of the polynomial $p(n)$. We immediately see that, to encode r', v', and u', we also need at most $\mathcal{O}\left(\log\left(n^d\right)\right)$ advice bits. The indices of the items x_i can be specified using $k \cdot \lceil\log n\rceil \leq (1+\delta)/\delta\lceil\log n\rceil$ additional bits. Similarly, the indices of the items z_i can be communicated using $j \cdot \lceil\log n\rceil \leq (1+\delta)/\delta\lceil\log n\rceil$ bits.

We conclude that at most $\mathcal{O}\left(\log\left(n^d\right)\right) = \mathcal{O}(\log n)$ advice bits are needed in total. The online algorithm A with advice works as shown in Algorithm 6.1 to construct $\mathcal{S}_2$ while using the advice as specified above. □

Finally, note that the $\mathcal{O}$ notation in Theorem 6.12 hides a larger constant than the one that we have for the unweighted case in Theorem 6.8.

Chapter 7

Advice and Randomization

In this chapter, we want to investigate some connections between computing with advice and randomized computation. To this end, consider randomized online algorithms as given in Definition 1.6. In both models, the algorithms we construct basically choose from a set of deterministic strategies (see Observation 1.12). When using randomness, we choose according to the output of some random source, when using advice, an oracle tells us which strategy to choose for any instance. In this sense, the advice may be seen as the *best* random string for every input. It is thus very simple to verify that, if there is a c-competitive randomized algorithm R solving some online problem P using b random bits, there also exists a c-competitive online algorithm A with advice that solves P with advice complexity b. This fact can be used to propagate the lower bounds on advice complexity to lower bounds on randomized algorithms that use a restricted number of random bits as stated by the following observation.

Observation 7.1. *If no online algorithm* A *with advice can achieve a competitive ratio of c using b bits of advice for some online problem P, there cannot exist any c-competitive randomized online algorithm* R *for P using b random bits. On the other hand, if there exists a randomized online algorithm* R *that is c-competitive in expectation and that uses b random bits, there also exists a c-competitive online algorithm using b bits of advice.*

Hence, online problems for which a large number of advice bits is provably necessary cannot be efficiently solved by randomized algorithms that use a small number of random bits. The opposite direction does not hold, i. e., it is not always possible to transform a well-performing online algorithm with advice into a well-performing randomized online algorithm. For example, consider a (very artificial) problem of guessing a single bit: The first request is always '?', the second request is either 0 or 1, and all remaining requests are '$*$'. The algorithm must answer the request '?' by 0 or 1 and all subsequent requests by '$*$'. The answer to '?' always induces cost 1. Furthermore, if the reply is equal to the second request, all subsequent requests induce cost 0, otherwise they induce cost 1. Easily, an optimal solution has cost 1 and it can be reached with a single bit of advice. On the other hand, any randomized algorithm, regardless of the number of random bits used, cannot gain an expected cost better than $n/2$ for inputs of n requests, which yields an expected competitive ratio of $n/2$. Thus, even problems that can be optimally solved with one single bit of advice might not be efficiently solvable by randomized online algorithms. Another example is SkiRental, although here the gap is not as large: As discussed in Section 1.6, no randomized online algorithm can be better than $\mathrm{e}/(\mathrm{e}-1)$-competitive

against an oblivious adversary [99], whereas there exists an optimal online algorithm with advice that only uses one advice bit.

Nevertheless, the ideas used to construct well-performing algorithms with advice may sometimes be adapted for the randomized setting as well, as we will see in the next section. Then again, supplying advice instead of random bits seems much more powerful, because instead of talking about a good performance of all strategies at hand *on average*, we only require the *existence* of one well-performing strategy; thus, it would not seem too surprising if we could save some bits when computing with advice instead of allowing randomization. In [34], the following theorem was proven, which basically confirms this intuition for minimization problems. Note that, subsequently, we do not require any bound on the number of random bits used by the original algorithm.

Theorem 7.2 (Böckenhauer et al. [34]). *Consider an online minimization problem P, where $\mathcal{I}(n)$ is the set of all possible inputs of length n and $|\mathcal{I}(n)| = I(n)$. Furthermore, suppose that there exists a randomized online algorithm* R *with a worst-case expected competitive ratio of at most E. Then, for any fixed $\varepsilon > 0$, it is possible to construct a deterministic online algorithm* A *with advice that uses at most*

$$\lceil \log n \rceil + 2\lceil \log \lceil \log n \rceil \rceil + \log\left(\frac{\log(I(n))}{\log(1+\varepsilon)}\right)$$

advice bits and that achieves a competitive ratio of $(1+\varepsilon)E$. ♢

The proof, using Lemma 1.15 and ideas from Yao's minmax principle[1] [152], shows how to select, from all possible random strings the algorithm R may use, a few strings that induce a good performance; these may then be supplied as advice. The resulting online algorithm, however, does not necessarily work in polynomial time. Also, note that this technique cannot be adapted to maximization problems in a straightforward fashion.

Anyway, the above result has an interesting implication for k-Server (see Chapter 4): Recall that the RKSC states that, for any metrical space, there is a randomized online algorithm with an expected competitive ratio of $\Theta(\log k)$. As we have pointed out, this bound is almost met for many reasonable instances since there exists a randomized online algorithm with an expected competitive ratio that is polylogarithmic in k, for graphs of size m, where m is polynomial in k [16]. For m vertices, there are obviously at most m^n different inputs of length n. (i) This implies that, according to Theorem 7.2, for such graphs, there is an online algorithm with advice complexity $\mathcal{O}(\log n + \log\log(m^n)) = \mathcal{O}(\log n + \log(n \log m)) = \mathcal{O}(\log n + \log(n \cdot k)) = \mathcal{O}(\log n + \log k)$ that also admits a competitive ratio that is polylogarithmic in k. Our results, however, grant a competitive ratio of $\mathcal{O}(\log k)$ only by using $\Omega(n)$ advice bits. Hence, our upper bound is still (almost) exponentially far away from the optimum. (ii) On the other hand, any lower bound on the number of advice bits of the form $\omega(\log n)$ on arbitrary (i. e., *large*) graphs would disprove the RKSC on general inputs.

Another question might be whether Theorem 7.2 also holds if we neglect the strict positiveness of ε, i. e., whether we can construct an online algorithm with advice that is *exactly as good* as the original randomized algorithm R. A negative answer to this question was given in [34] by introducing the following online problem.

Definition 7.3 (Winner-takes-all problem). *The input is a sequence of requests $x_1 := 0, x_2, \ldots, x_{n+1}$, where $x_i \in \{0,1\}$; a solution is a sequence $y_1, \ldots, y_n$, where $y_i \in \{0,1\}$.*

[1] Andrew Chi-Chih Yao, ⋆24.12.1946, Chinese computer scientist.

The cost of a given solution is 1 *if* $y_i = x_{i+1}$*, for all* $i \in \{1, \ldots, n\}$*, and* 2 *otherwise. Hence, the optimal solution has cost* 1 *and all other solutions pay an extra penalty of* 1.

We can now argue that the best possible randomized online algorithm guesses every bit with equal probability, incurring an expected cost of $2 - \frac{1}{2^n}$ [34]. On the other hand, any online algorithm with advice that uses less than n advice bits has a worst-case performance of 2, so n bits are needed to be on par with randomization. It follows that Observation 7.1 is tight.

Finally, we want to find some explanation for the behavior of algorithms with advice when it comes to optimal output. To this end, we need the following observation about randomized computation.

Observation 7.4. *If, for some online problem* P*, there does not exist an optimal deterministic algorithm, then there does not exist any optimal randomized algorithm for* P*, neither.*

Proof. Assume the contrary, i. e., that there exists some optimal randomized online algorithm R for P. For *every* random choice R makes, the output produced is thus optimal for every instance. But then we can just simulate R with some fixed random string which gives an optimal deterministic online algorithm for P. □

Therefore, for all problems that we might consider interesting, no optimal randomized algorithms exist. However, note that Observation 7.4 does not hold if we do not speak about optimality, but only about a competitive ratio that tends to 1 with growing input size.

This might explain why, for many of the problems that we have considered in this thesis, small advice merely helps up to a specific point. In particular, the advice complexity to create optimal output is usually linear in the input size. However, if these problems admit a well-performing randomized online algorithm, we can be as good using advice, and often the advice complexity to achieve this is rather low. For JSS, it has been shown that randomization is very powerful: There exists a randomized online algorithm that is $(1 + 1/\sqrt{m})$-competitive, which is *almost* optimal [88] (note that, in [90], a $(1 + 4/\sqrt{m})$-competitive randomized online algorithm was given by studying a more general setting); the construction of this algorithm implies that we need at most $\log m + 1$ advice bits to be equally good [35, 126]. However, to be optimal, we need to make an exponential jump to $\Omega(\sqrt{m})$ advice bits, as we have seen in Chapter 2. When dealing with DPA and measuring in the size of the line network, we also need a linear number of advice bits to be optimal, but using knowledge from a well-performing randomized algorithm, we can create good solutions with a small number of bits. An optimal online algorithm with advice for SimpleKnapsack uses at least $n - 1$ advice bits, but with merely one bit of advice, we can be 2-competitive (and one random bit also suffices to be 2-competitive in expectation, as we will see in the next section).

In fact, many of the proof techniques we used for proving lower bounds on the advice complexity for producing optimal output fail immediately, as soon as we relax the requirements on the performance of the online algorithms just a little bit. For instance, for k-Server, the technique we used in the proof of Theorem 4.3 does not work if we allow the online algorithm to deviate from optimality only by some $\varepsilon > 0$, i. e., to use a few edges of cost 2. Similarly, for JSS, we cannot assume anymore that the online algorithm has to know which obstacles are missing in the widgets used in the proof of Theorem 2.7

if it is allowed to use an additional constant number of steps compared to the optimal solution.

7.1 Barely Random Algorithms

As we have already pointed out in Section 1.4, it might be expensive to generate random numbers. Hence, we are interested in designing good randomized algorithms that use as few random bits as possible. As we have seen, it is possible to measure the amount of random bits needed by a randomized algorithm as a function of the input length, in a similar way as time complexity, space complexity, or advice complexity is measured. Randomized algorithms that use only a constant number of random bits, regardless of the input size, are called *barely random algorithms* [38], introduced in [136]. The number of random bits used by these algorithms is asymptotically minimal, thus they can be considered the best algorithms with respect to the amount of randomness employed.

As we have seen in the last section, we cannot hope for a general technique to construct barely random algorithms from algorithms with advice that use a constant number of advice bits. Nevertheless, the proofs used to construct well-performing algorithms with advice may sometimes be adapted to randomized computation. In this way, we obtain some interesting results about barely random algorithms.

Job Shop Scheduling

As in Chapter 2, we consider the class $\mathcal{D}_d$ of diagonal strategies, for some odd constant $d \geq 1$. Consider a barely random algorithm $\mathtt{R}_d$ that randomly chooses a strategy from this class using at most $\lceil \log d \rceil$ random bits. Our results from Section 2.2, together with Observation 7.1, imply that we cannot obtain anything better than the competitive ratio achieved by the online algorithm $\mathtt{A}_d$ with advice as described in Section 2.2. However, we can get very close to $\mathtt{A}_d$ as the following theorem shows.

Theorem 7.5. *The algorithm $\mathtt{R}_d$ achieves an expected competitive ratio of*

$$1 + \frac{1}{d} + \frac{d^2 - 1}{4dm}.$$

Proof. For every odd d, consider the following random variables $X_1, X_2, X, Y \colon \mathcal{D}_d \to \mathbb{R}$, where $X_1(D_i)$ is the delay caused by the initial horizontal [vertical] steps made by the strategy D_i, $X_2(D_i)$ is the delay caused by D_i hitting obstacles, $X(D_i) := X_1(D_i) + X_2(D_i)$ is D_i's overall delay, and $Y(D_i) := m + X(D_i)$ is D_i's overall cost. Recall that D_{-j} and D_j make the same amount of non-diagonal (vertical or horizontal, respectively) steps at the beginning. Since there are exactly m obstacles in total for every instance, we get

$$\mathbb{E}[X_2] = \frac{1}{d}\left(X_2(D_0) + 2\sum_{i=1}^{(d-1)/2} X_2(D_i)\right) \leq \frac{m}{d},$$

and since $X_1(D_0) = 0$, we get

$$\mathbb{E}[X_1] = \frac{1}{d}\left(X_1(D_0) + 2\sum_{i=1}^{(d-1)/2} X_1(D_i)\right) = \frac{2}{d}\sum_{i=1}^{(d-1)/2} i = \frac{d^2 - 1}{4d}.$$

Due to the linearity of expectation, it follows that

$$\mathbb{E}[Y] = m + \mathbb{E}[X] = m + \mathbb{E}[X_2] + \mathbb{E}[X_1] \leq m + \frac{m}{d} + \frac{d^2-1}{4d}.$$

As a result, the expected competitive ratio of $\mathtt{R}_d$ is at most

$$\frac{\frac{(d+1)m}{d} + \frac{d^2-1}{4d}}{m} = 1 + \frac{1}{d} + \frac{d^2-1}{4dm},$$

as we claimed. □

Note that, if m increases, the expected competitive ratio of $\mathtt{R}_d$ tends to $1 + 1/d$. The bound of Theorem 7.5 is actually very close to the one shown for the corresponding online algorithm $\mathtt{A}_d$ with advice in Theorem 2.9, but slightly worse. In fact, we can use the proof of Theorem 7.5 as a probabilistic proof of an upper bound on the advice complexity of JSS. On the other hand, we can apply Theorem 2.12 and Observation 7.1 to obtain that, for any $\varepsilon > 0$, no randomized online algorithm that uses at most b random bits can obtain a competitive ratio better than $1 + 1/(3 \cdot 2^b) - \varepsilon$ in expectation. Hence, barely random algorithms for JSS cannot have a competitive ratio that tends to 1 with growing m. This means they perform worse than the randomized algorithm from [88] (using an unrestricted number of random bits), which can reach a competitive ratio that tends to 1 with increasing m.

The Simple Knapsack Problem

In Chapter 6, we have discussed the simple (i. e., unweighted) knapsack problem SIMPLE-KNAPSACK. We have given an online algorithm $\mathtt{A}_1$ with advice that is 2-competitive and only uses one advice bit (see Theorem 6.6). At first, suppose we use the same algorithm, but this time guess the advice bit. Obviously, this barely random algorithm, which we call $\mathtt{R}'_1$, is 2-competitive with probability 1/2 and not competitive with the same probability, i. e., 4-competitive in expectation. Considering $\mathtt{R}'_1$, this bound is tight as the next theorem shows.

Theorem 7.6. *The barely random algorithm* $\mathtt{R}'_1$ *for* SIMPLEKNAPSACK *cannot be better than strictly* 4*-competitive in expectation.*

Proof. Let $1/6 > \varepsilon > 0$. Consider three items of sizes

$$\frac{1}{2} - \varepsilon, 3 \cdot \varepsilon, \frac{1}{2} - \varepsilon.$$

A greedy approach takes the first two items and therefore obtains a gain of $1/2 + 2 \cdot \varepsilon$, whereas the algorithm that waits for an item of size $\geq 1/2$ gains nothing. Thus, $\mathtt{R}'_1$ is strictly c-competitive only for

$$c \geq \frac{1 - 2 \cdot \varepsilon}{\frac{1}{2} \cdot \left(\frac{1}{2} + 2 \cdot \varepsilon\right) + \frac{1}{2} \cdot 0} = 4 \cdot \frac{1 - 2 \cdot \varepsilon}{1 + 4 \cdot \varepsilon}.$$

Since ε can be arbitrarily small, no strict competitive ratio better than 4 can be reached in expectation. □

Considering the results of the last subsection, we now know better, but still, intuitively it does not seem too surprising that, when it comes to such a small piece of information, randomization (the average over good and bad) is twice as bad as computing with advice (*always* good). However, while this is right for this specific strategy, we get the following: Remarkably, randomization and advice admit the same performance for SimpleKnapsack when dealing with a small amount of either random or advice bits.

Theorem 7.7. *There exists a barely random algorithm* $\mathtt{R}_1$ *for* SimpleKnapsack *that achieves an expected competitive ratio of* 2 *and that only uses one random bit.*

Proof. Consider the following deterministic online algorithms A_1 and A_2. A_1 is the straightforward greedy algorithm for SimpleKnapsack. A_2 locally simulates A_1 and does not take any item until it realizes that an item just offered would not fit into A_1's solution anymore. A_2 then acts greedily starting from here. If the input consists of items that, in the sum, have a weight less than the knapsack's capacity, A_1 is obviously optimal, while A_2 has a gain of 0. If, however, this is not the case, the gain of A_1 plus the gain of A_2 is at least 1.

Let $\mathtt{R}_1$ choose between A_1 and A_2 uniformly at random. Obviously, one random bit suffices to do that. We then immediately get that the expected gain of $\mathtt{R}_1$ is at least $1/2$, and the expected competitive ratio of $\mathtt{R}_1$ is thus at most 2. □

Note that the lower bound of 2 on the competitive ratio for online algorithms with advice (see Theorem 6.7) also applies to the randomized case. Therefore, Theorem 7.7 is tight. Let us relate Theorem 7.7 to our results from Chapter 6: With either one random bit or one advice bit, we can (in expectation) achieve a competitive ratio of 2, and this bound is tight. However, considering randomness, any additional random bit does *not help at all* (see Theorem 6.9). As for computing with advice, there also is no improvement until we get a logarithmic number of advice bits leading to an online algorithm that gets arbitrarily close to an optimal solution. The above results imply that randomization and advice are equally powerful when we consider a sub-logarithmic number of bits.

The Paging Problem

The third problem we focus on has already been introduced and discussed in Section 1.3. Furthermore, in Section 1.7, we have briefly summarized the results from [35, 36] on the advice complexity of Paging. One of the most intriguing results is given by Theorem 1.19, which states that there exists an online algorithm $\mathtt{A}$ with advice that reads $\log b$ advice bits and that has a competitive ratio of at most

$$3 \log b + \frac{2(K+1)}{b} + 1,$$

where K is the buffer size of $\mathtt{A}$ (see Definition 1.4).

In [109], it was shown that the proof technique can be adapted to construct a barely random algorithm for Paging. In the original proof, the existence of a *good* advice string was shown by arguing that, on average, the advice strings are good for any instance; this allows a direct argumentation about the expectation, which was used to prove the following theorem.

Theorem 7.8 (Komm and Královič [109]). *Consider* Paging *with a buffer size of K and let $b < K$ be a power of 2. There exists a barely random algorithm for* Paging *that achieves a competitive ratio of at most*

$$3 \log b + \frac{2(K+1)}{b} + 1$$

and that uses $\log b$ random bits, regardless of the input size. ♢

As stated in [109] and to the best of our knowledge, all randomized online algorithms for Paging known so far that reach a competitive ratio of $\mathcal{O}(\log K)$ use $\Omega(n)$ random bits for inputs of length n, and no efficient barely random algorithm for Paging was known before. The previous theorem shows that there exists a barely random algorithm for Paging that only uses $\log K$ random bits and that reaches a competitive ratio of $\mathcal{O}(\log K)$, which is asymptotically equivalent to the best possible randomized algorithm [38].

We conclude that it seems worth searching for such randomized algorithms for problems where a constant number of advice bits proves to be powerful.

7.2 Bounds with High Probability

So far, we have analyzed the performance of the randomized online algorithms we constructed by relating the expected gain to that of an optimal offline algorithm [88]. In this section, we want to establish stronger statements of the following kind:

> *Not only are the random decisions made by some randomized online algorithm good on average, but almost always.*

In other words, we want to give bounds that are achieved *with high probability*, i. e., with a probability that tends to 1 with growing input size. Once more, let us consider JSS. More particularly, we want to revisit the randomized algorithm R from [88]. R chooses a strategy from the set $\mathcal{D}_{2\sqrt{m}+1} = \{D_{-\sqrt{m}}, \ldots, D_{\sqrt{m}}\}$ uniformly at random and, as mentioned, is $(1 + 1/\sqrt{m})$-competitive in expectation. We now show that R also computes a very good solution with high probability.

Theorem 7.9. *The randomized algorithm* R *is $(1 + f(m))$-competitive with high probability, where $f(m) \in \omega(1/\sqrt{m})$.*

Proof. Consider the randomized online algorithm R as described above. Recall that Observation 2.2 (iii) states that the optimal solution has a cost of at least m. Let l be the number of considered diagonals that cause R to have a cost of more than $m(1 + f(m))$. Then, the probability that the computed solution has a cost of more than $m(1 + f(m))$ is

$$p := \frac{l}{2\sqrt{m} + 1}.$$

Let us have a closer look at R. The delay of any solution it computes is caused by two things: the number i of horizontal [vertical] steps the algorithm makes at the beginning to reach a diagonal with distance i from the main diagonal and the number of obstacles hit. Since every obstacle that is hit is evaded by exactly one horizontal and one vertical move, this causes an additional cost of exactly 1. Note that i is at most $\sqrt{m}$ in any case.

Let l' denote the number of diagonals that contain more than $mf(m) - \sqrt{m}$ obstacles; we call such diagonals *expensive*. By the above observation, any diagonal that causes a

cost of more than $m+mf(m)$ must have more than $mf(m)-\sqrt{m}$ obstacles on it. It follows that $l \leq l'$ and hence $p \leq l'/(2\sqrt{m}+1)$. We can now bound l' from above as follows. Easily, if l is maximally large, all of the m obstacles are distributed on the expensive diagonals. Conversely, since there cannot be more than m obstacles for any instance, we have $m \geq l'(mf(m)-\sqrt{m})$ and therefore

$$l' \leq \frac{m}{mf(m)-\sqrt{m}}.$$

As a result, we get

$$p \leq \frac{\frac{m}{mf(m)-\sqrt{m}}}{2\sqrt{m}+1} = \frac{m}{(2\sqrt{m}+1)(mf(m)-\sqrt{m})} = \frac{1}{(2\sqrt{m}+1)\left(f(m)-\frac{1}{\sqrt{m}}\right)}.$$

Finally, since $f(m) \in \omega(1/\sqrt{m})$, it immediately follows that p tends to 0 if m tends to infinity, which finishes the proof. □

7.3 Amplification

When considering offline problems, we classify randomized algorithms according to their error probability and the nature of the errors [88], e. g., we distinguish between *Las Vegas*, *one-sided error Monte Carlo*, and *two-sided error Monte Carlo* algorithms. The main idea behind this classification is that, if a randomized algorithm has an error probability that is strictly less than 1/2, we may run this algorithm multiple times on the same input, decreasing the error probability with every additional run [88]; this concept is known as *amplification*.

It is crucial to note that the amplification method *cannot* be used in randomized online computation, because online algorithms have to answer requests immediately and may not change output they have already created. However, if we construct an online algorithm using both advice and a random source, these restrictions may change. More precisely, we can define two different types of oracles: (i) `O` does not know the output of any random decision made by a randomized online algorithm `R`. Still, the advice might be used to rule out some worst-case outputs for the concrete instance and thus improve `R`'s performance. We call these random bits *private random bits*. (ii) In the second model, `O` is more powerful and even anticipates the random decisions that are made by `R` during runtime. In this case, we talk about *public random bits*.

In the following, we use public random bits, and we show that, in this model, amplification is indeed possible. Consider JSS and the algorithm from [88] together with our results from the previous section.

Theorem 7.10. *Let k be constant. There is a randomized online algorithm `R` with advice and public random bits that uses $2k$ advice bits and that chooses an expensive diagonal with probability less than $1/m^k$.*

Proof. In the proof of Theorem 7.9, we have shown that an expensive diagonal is chosen with probability at most

$$\frac{1}{(2\sqrt{m}+1)\left(f(m)-\frac{1}{\sqrt{m}}\right)},$$

where $f(m) \in \omega(1/\sqrt{m})$. Suppose we use public random bits to randomly choose a diagonal from $\mathcal{D}_{2\sqrt{m}+1}$ (as defined in Section 7.2), and, after each such random choice, O tells us whether this choice was good or bad by writing either a 0 or a 1 onto the advice tape. If the decision was bad, we choose again; this is iterated at most d times, thus using at most d advice bits. We immediately get that

$$\left(\frac{1}{(2\sqrt{m}+1)\left(f(m)-\frac{1}{\sqrt{m}}\right)}\right)^d \leq \frac{1}{m^k}$$

$$\iff d\log\left(\frac{1}{(2\sqrt{m}+1)\left(f(m)-\frac{1}{\sqrt{m}}\right)}\right) \leq \log\left(\frac{1}{m^k}\right)$$

$$\iff \frac{k\log m}{\log\left((2\sqrt{m}+1)\left(f(m)-\frac{1}{\sqrt{m}}\right)\right)} \leq d,$$

which holds if

$$d \geq \frac{k\log m}{\log(\sqrt{m})} \iff d \geq 2k,$$

as we claimed. □

The above theorem implies the following statement.

Corollary 7.11. *There is a randomized online algorithm with advice that uses public random bits and four bits of advice and that chooses an expensive strategy with probability at most* $1/m^2$. □

A point of criticism of this model is that allowing public random bits is not adequate to measure the advice complexity, because O is too powerful. In the next section, we use the weaker model employing private random bits to study, at a concrete example, how much advice may help randomized online algorithms to increase their performance in a very generic setting.

7.4 The Boxes Problem

In this section, we want to study the tradeoff between randomization and advice for a specific online maximization problem. To this end, we first investigate the power of randomization by giving both (tight) lower and upper bounds and then boost the performance by allowing the online algorithms at hand to additionally use advice. Let A be a randomized online algorithm that achieves some expected gain g on inputs of length n while using b_1 bits of advice and b_2 private random bits. In simple words, in what follows we show that, if A uses $b_1' = b_1 + 1$ advice bits and b_2 private random bits, it gets an expected gain of g on inputs of length $2n$. It therefore turns out that, for the following problem, with any additional advice bit, A doubles its performance. Let us introduce the online problem, denoted by (n,k)-Boxes, we deal with in the following.

Definition 7.12 ((n,k)-Boxes). *There are n boxes $b_1,\ldots,b_n$ standing in a row, and we know that all are empty except for $k < \sqrt{n}$ boxes that stand next to each other and that contain some expensive item each. An online algorithm* A *is allowed to open exactly*

k boxes of its choice aiming at opening as many full ones as possible. After A *has opened k boxes, the (remaining) positions of the non-empty boxes are revealed, and* A*'s gain is the number of non-empty boxes it has opened.*

Clearly, the optimal solution of (n, k)-Boxes always has gain k. We call the position of the first full box the *starting position*; note that, for any instance of size n, there are exactly $n - k + 1$ possible starting positions. Adv tries to hide the full boxes in such a way that A's gain is (in expectation and/or for any advice string) as small as possible. At first, we can make the following straightforward observation about any deterministic online algorithm without advice for (n, k)-Boxes.

Theorem 7.13. *If $n \geq k(k+1)$,* Adv *can ensure that no deterministic online algorithm* A *has any gain.*

Proof. Adv knows A's deterministic strategy; however, for every box at some position i that A opens, the number of starting positions to hide the k items is decreased by at most k. The removed positions are those between i and $i-k+1$, where $i \geq k$ (see Figure 7.1); not all of them exist if $i < k$. Hence, if A chooses boxes such that these intervals are disjoint,

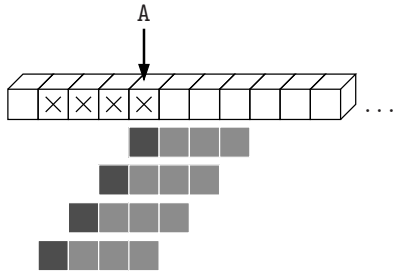


Figure 7.1: A inspecting box b_5 and the "forbidden" starting positions for Adv.

Adv is prevented from taking at most k^2 starting positions. Accordingly, if $n \geq k^2 + k$, Adv may hide the items in such a way that A is not able to find any full box at all. □

Thus, no deterministic algorithm is competitive if $n \geq k(k+1)$. In the next subsection, we show the power of randomization for (n, k)-Boxes.

Randomization

Since we are interested in values for k and n that are in the above order and the situation seems, as for the majority of problems we have studied in this thesis, desperate for any deterministic online algorithm, we now consider randomized online algorithms. Which box is chosen next may depend on which of the already opened boxes are full and on the randomness that is available to the algorithm. So far, the randomized algorithms we discussed used a random tape. As we have pointed out at the beginning of this chapter, it is then possible to measure the amount of randomness (i. e., the number of random bits used) as a function of the input size. Note that inputs for (n, k)-Boxes always have a constant size since n and k are fixed. As we are interested (without loss of generality) in randomized algorithms with a fixed upper bound on the running time, for any randomized algorithm R solving (n, k)-Boxes, there is an upper bound r on the number of random bits used by R.

Since we want to investigate this problem in detail, in the sequel, we aim for a very fine-grained analysis. As we know, if R is allowed to use b random bits, equivalently, it can use a random number between 1 and 2^b instead (see Observation 1.12). We generalize this model by considering algorithms that get one random number drawn uniformly from the set $\{1, \ldots, M\}$, where M is not restricted to be a power of two. Let $\mathcal{R}and(M)$ denote the set of randomized algorithms with randomness restricted to the uniformly random choice of a number from $\{1, \ldots, M\}$.

If we fix an algorithm R, then the random variable $B_s(\mathtt{R})$ denotes the number of full boxes opened by R for the starting position s; $\mathbb{E}[B_s(\mathtt{R})]$ then is the expected number of full boxes opened if Adv chooses the starting position s, and $\min_s \mathbb{E}[B_s(\mathtt{R})]$ is the worst-case expected gain of R. If R is clear from the context, we abbreviate $B_s(\mathtt{R})$ by B_s.

The following theorem gives a worst-case lower bound on the performance of any randomized online algorithm $\mathtt{R} \in \mathcal{R}and(M)$ solving (n,k)-Boxes.

Theorem 7.14. *For any randomized online algorithm* $\mathtt{R} \in \mathcal{R}and(M)$ *for* (n,k)-Boxes*, there exists a starting position* s *(i. e., an input instance of* (n,k)-Boxes*), such that*

1. *if* $M < \frac{n-k+1}{k^2}$*, then* $\mathbb{E}[B_s] = 0$*,*
2. *for any* M*,* $\mathbb{E}[B_s] \leq \frac{k^2(k+1)}{2(n-k+1)}$*, and*
3. *if* $M = c(n-k+1)/k^2$ *with* $c \in [1, 2k/(k+3)]$*, then*
   (a) $\mathbb{E}[B_s] \leq \frac{2(k+\frac{3}{2})}{M} - \frac{2(n-k+1)}{M^2 k}$ *and also*
   (b) $\mathbb{E}[B_s] \leq \frac{2(c-1)}{c^2} \cdot \frac{k^3}{n-k+1} + \frac{3k^2}{c(n-k+1)}$.

Proof. The algorithm R gets a random number from $\{1, \ldots, M\}$, so it can behave in M different ways, and each behavior occurs with probability $1/M$. This is equivalent to choosing one deterministic algorithm from a set $\{A_1, \ldots, A_M\} = \mathrm{Alg}(\mathtt{R})$ uniformly at random (which is a straightforward generalization of Observation 1.12). Recall that the k full boxes start at some starting position, i. e., a position between 1 and $n-k+1$.

1. Suppose that $M < (n-k+1)/k^2$. Basically, we now extend the idea from the proof of Theorem 7.13. The number of starting positions that allow Adv to place the obstacles in such a way that R does not find any item is $n-k+1-(Mk^2) > n-k+1-(n-k+1) = 0$, which implies the claim.

2. We can make a simple observation: Consider some specific (deterministic) algorithm $A_i \in \mathrm{Alg}(\mathtt{R})$. There are at most k starting positions such that A_i opens a full box in the first time step and gets at most k full boxes in total. Next, there are at most k starting positions such that A_i opens an empty box in the first time step and a full one in the second time step, and thus gets at most $k-1$ full boxes in total, and so on.

   Assume that the starting position is p. Let $\overline{O}_{i,p}$ denote the number of empty boxes opened by A_i until the first full box is found (or k if none) and let $O_{i,p} := k - \overline{O}_{i,p}$ and $O_p := O_{1,p} + \cdots + O_{M,p}$. Clearly, $O_{i,p}$ is an upper bound on the number of full boxes opened by A_i in total, and O_p/M is an upper bound on the expected number of full boxes opened by R. Hence, taking s such that O_s is minimal, the expected number of full boxes opened by R for the starting position s is

$$\mathbb{E}[B_s] = \frac{1}{M} O_s = \frac{1}{M} \min\{O_1, \ldots, O_{n-k+1}\}. \tag{7.1}$$

From the above observations, it follows that, for every deterministic strategy A_i, we have

$$O_{i,1} + \cdots + O_{i,n-k+1} \leq k^2 + k(k-1) + k(k-2) + \cdots + k = \frac{k^2(k+1)}{2},$$

and accordingly, using (7.1) and the fact that the minimum of a set of numbers can never be larger than its average, we get

$$\begin{aligned}
\mathbb{E}[B_s] &\leq \frac{1}{M} \cdot \frac{\sum_p O_p}{n-k+1} = \frac{1}{M(n-k+1)} \cdot \sum_p \sum_{i=1}^{M} O_{i,p} \\
&= \frac{1}{M(n-k+1)} \cdot \sum_{i=1}^{M} \sum_p O_{i,p} = \frac{1}{M(n-k+1)} \cdot \sum_{i=1}^{M} \frac{k^2(k+1)}{2} \\
&= \frac{1}{M} \cdot \frac{Mk^2(k+1)}{2(n-k+1)} = \frac{k^2(k+1)}{2(n-k+1)}. \qquad (7.2)
\end{aligned}$$

3. Now let us consider values of M such that $\frac{n-k+1}{k^2} \leq M \leq \frac{2(n-k+1)}{k(k+3)}$. The intuitive idea of the remaining part of the proof is to choose a starting position such that it induces a small gain for every deterministic strategy from Alg(R). Let g be such that $g \leq k/M$ and $g \geq \frac{k+1}{M} - \frac{n-k+1}{kM^2}$. We call a starting position p *large* if $O_{i,p} \geq gM$, for some i, i. e., there is a single algorithm A_i that makes a large contribution to O_p; otherwise, we call this starting position *small*. Note that

$$g \geq \frac{k+1}{M} - \frac{n-k+1}{kM^2} \iff gM - 1 \geq k - \frac{n-k+1}{Mk},$$

which implies $n - k + 1 \geq Mk(k - \lfloor gM \rfloor)$. Moreover, $g \leq k/M$ easily implies $k - \lfloor gM \rfloor \geq 0$ and thus $Mk(k - \lfloor gM \rfloor) \geq 0$. Hence, due to our assumptions on g, we have that $Mk(k - \lfloor gM \rfloor) \in [0, n-k+1]$.

For any A_i, p is made large by A_i if $O_{i,p}$ takes values between $\lceil gM \rceil$ and k, which leads to $k - \lceil gM \rceil + 1 \geq k - \lfloor gM \rfloor$ different values. For any such value x, there are at most k positions p such that $O_{i,p} = x$, and, since there are M algorithms in total to consider, there are at most $Mk(k - \lfloor gM \rfloor)$ large positions. Accordingly, there are at least

$$n - k + 1 - Mk(k - \lfloor gM \rfloor) \geq 0$$

small positions.

In the following, let S denote the set of all small positions. Since no algorithm is allowed to contribute strictly more than $\lfloor gM \rfloor$ for these positions, we get that, for any algorithm A_i,

$$\sum_{p \in S} O_{i,p} \leq k(1 + 2 + \cdots + \lfloor gM \rfloor) = \frac{k\lfloor gM \rfloor(\lfloor gM \rfloor + 1)}{2}$$

and consequently

$$\sum_{p \in S} O_p \leq \frac{Mk\lfloor gM \rfloor(\lfloor gM \rfloor + 1)}{2}.$$

In particular, there is some small position p (as above, we use that the minimum cannot be larger than the average) with

$$\begin{aligned} O_p &\leq \frac{Mk\lfloor gM\rfloor(\lfloor gM\rfloor+1)}{2|S|} \leq \frac{Mk\lfloor gM\rfloor(\lfloor gM\rfloor+1)}{2(n-k+1-Mk(k-\lfloor gM\rfloor))} \\ &\leq \frac{MkgM(gM+1)}{2(n-k+1-Mk(k-gM+1))}, \end{aligned} \tag{7.3}$$

and, since $\mathbb{E}[B_p] = O_p/M$, there exists a starting position s such that

$$\mathbb{E}[B_s] \leq \frac{kgM(gM+1)}{2(n-k+1-Mk(k-gM+1))}. \tag{7.4}$$

We now set

$$g := \frac{2\left(k+\frac{3}{2}\right)}{M} - \frac{2(n-k+1)}{M^2k} = \frac{2kM\left(k+\frac{3}{2}\right)-2(n-k+1)}{M^2k},$$

which satisfies our assumptions on g since

$$\begin{aligned} \frac{Mk^2}{M^2k} &\geq \frac{2kM\left(k+\frac{3}{2}\right)-2(n-k+1)}{M^2k} \\ \iff Mk^2 &\geq 2k^2M+3kM-2(n-k+1) \\ \iff 2(n-k+1) &\geq Mk(k+3) \end{aligned}$$

holds due to $M \leq \frac{2(n-k+1)}{k(k+3)}$, and

$$\begin{aligned} \frac{2kM(k+\frac{3}{2})-2(n-k+1)}{M^2k} &\geq \frac{kM(k+1)-(n-k+1)}{M^2k} \\ \iff 2kM\left(k+\frac{3}{2}\right) &\geq kM(k+1)+(n-k+1) \\ \iff M(k^2+2k) &\geq n-k+1 \end{aligned}$$

holds due to $M \geq \frac{n-k+1}{k^2}$. Next, note that

$$\begin{aligned} g &= \frac{kgM(gM+1)}{2(n-k+1-Mk(k-gM+1))} \\ \iff 1 &= \frac{kM(gM+1)}{2(n-k+1-Mk(k-gM+1))} \\ \iff kM(gM+1) &= 2(n-k+1)-2Mk^2+2M^2kg-2Mk \\ \iff 2Mk^2 &= 2(n-k+1)+M^2kg-3Mk \\ \iff 2k &= \frac{2(n-k+1)}{kM}+Mg-3 \\ \iff 2k+3 &= \frac{2(n-k+1)}{kM}+\frac{2Mk\left(k+\frac{3}{2}\right)-2(n-k+1)}{Mk} \\ \iff 2k+3 &= 2\left(k+\frac{3}{2}\right) \\ \iff k &= k \end{aligned}$$

is always true, which, together with (7.4), implies that $\mathbb{E}[B_s] \leq g$. To finish the proof, we choose $M = c(n-k+1)/k^2$, yielding

$$\mathbb{E}[B_s] \leq \frac{2k^2\left(k+\frac{3}{2}\right)}{c(n-k+1)} - \frac{2k^3(n-k+1)}{c^2(n-k+1)^2} = \frac{2k^2c\left(k+\frac{3}{2}\right) - 2k^3}{c^2(n-k+1)}$$
$$= \frac{2(c-1)}{c^2} \cdot \frac{k^3}{n-k+1} + \frac{3k^2}{c(n-k+1)}$$

as we claimed.

The theorem follows. □

We now complement this lower bound by an upper bound that is tight up to a small constant factor.

Theorem 7.15. *Let M be an even number. There exists a randomized online algorithm $\mathtt{R} \in \mathcal{R}and(M)$ for (n,k)-Boxes such that, for every starting position s,*

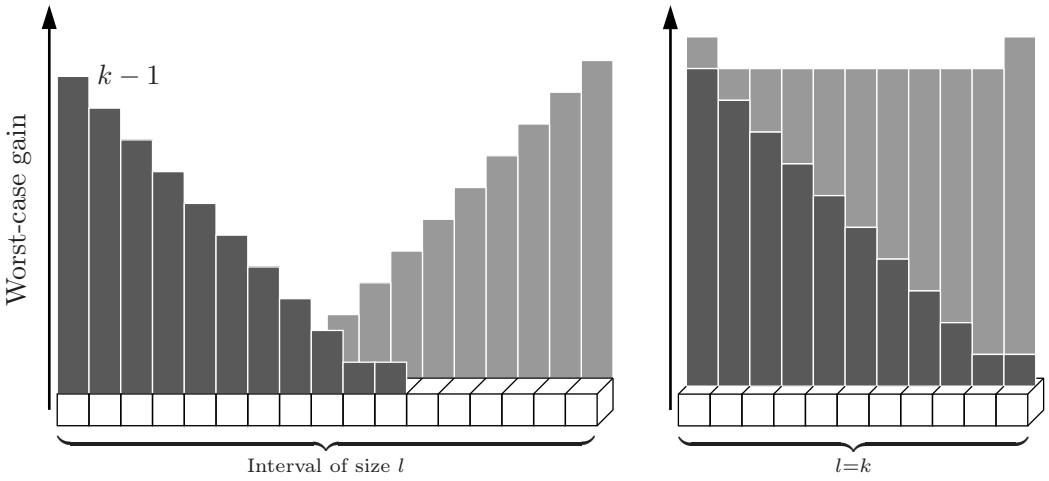
1. *if $M \geq \frac{2n}{k(k-1)}$, then $\mathbb{E}[B_s] \geq \frac{8}{9} \cdot \frac{k^3}{2n} - 1$, and*
2. *if $\frac{n-k+1}{k^2} < M \leq \frac{2n}{k(k-1)}$, then $\mathbb{E}[B_s] \geq \frac{2k-2}{M} - \frac{2n}{M^2k}$.*

Proof. As we have mentioned, $\mathtt{R}$ is a probability distribution over Alg($\mathtt{R}$), where each deterministic algorithm $A \in$ Alg($\mathtt{R}$) gets chosen with the same probability $1/M$. Every algorithm A opens boxes within some interval of fixed length and performs a straightforward local search when a box is found, which enables A to find at least $k-i$ full boxes if an item is discovered in time step i. Moreover, consider an adversary $\textsc{Adv}$ that tries to hide the full boxes as well as possible from every algorithm in Alg($\mathtt{R}$) at once.

In the following, we focus on an equivalent problem to analyze the $\mathtt{R}$'s performance. We shrink the instance to an instance of size $\lfloor n/k \rfloor$, i. e., we compress k boxes into one so-called *hyper-box*, thereby neglecting the last $d = k\lfloor n/k \rfloor < k$ original boxes. There is exactly one non-empty hyper-box that has a value of $k-1$ at the beginning and whose value is decreasing by one with every unsuccessful opening of a hyper-box (except for the last step). The algorithm can open up to k hyper-boxes. When it opens the full hyper-box in the jth trial, it achieves a gain of $k-j$ if $j < k$, and a gain of 1 if $j = k$.

Next, we show that it is possible to reduce (n,k)-Boxes to its shrunk counterpart. To this end, assume that we have an algorithm $\mathtt{A}'$ for the shrunk version. We can construct an algorithm $\mathtt{A}$ for (n,k)-Boxes as follows. Whenever $\mathtt{A}'$ opens some hyper-box, $\mathtt{A}$ opens the last box corresponding to this hyper-box. As soon as $\mathtt{A}$ finds some full box, it continues with a local search. Consider any input instance for $\mathtt{A}$, which is specified by the starting position p. Then, $\mathtt{A}$ achieves at least the same gain as $\mathtt{A}'$ running on an instance where the hyper-box corresponding to p is full. Since p cannot be within the last $k-1$ boxes, the hyper-box corresponding to p exists. If $\mathtt{A}'$ opens the full hyper-box, the starting position is within distance k to the left of the box opened by $\mathtt{A}$, therefore $\mathtt{A}$ opens a full box as well. If this happens in time step j, the local search of $\mathtt{A}$ guarantees a gain of at least $k-j$ if $j < k$, and of 1 if $j = k$.

In the sequel, we provide a randomized online algorithm $\mathtt{R}$ that solves the shrunk version of (n,k)-Boxes, thereby proving that an equally good algorithm for the original problem exists. Consider some constant $l \in \{1, \ldots, 2k-1\}$. Starting with some hyper-box q, a deterministic algorithm $A \in$ Alg($\mathtt{R}$) opens k consecutive hyper-boxes until k empty hyper-boxes are inspected or it finds the full hyper-box at the jth trial; as mentioned before,



(a) The gain of both A and $\overline{A}$ within one interval.

(b) Optimally compressed interval and the sums of the gains of A and $\overline{A}$.

Figure 7.2: The worst-case gain within one interval assigned to the $\lfloor n/k \rfloor$ hyper-boxes.

it gains $k-j$ if $j < k$, and 1 if $j = k$. Next, we define the symmetric algorithm to A, denoted by $\overline{A} \in \mathrm{Alg}(\mathtt{R})$: $\overline{A}$ initially opens hyper-box $q+l-1$ and then continues to open $k-1$ consecutive hyper-boxes left of $q+l-1$ in reverse order until it arrives at $s+l-k$ or finds the full hyper-box; A and $\overline{A}$ are called an *algorithm pair*, because they work on the same interval.

Let us now bound the minimum of the total gain of the two algorithms within one interval of length l. Clearly, if $l = k$, the gain is at least $k-1$ for every hyper-box, and, more general, if $l = k+i$, for $-k < i < k$, we get a gain of at least $k-1-i = 2k-l-1$ (see Figure 7.2). Starting at the first hyper-box, we assign every of the $M/2$ algorithm pairs to one interval of length l in such a way that, by allowing *wrap-arounds*, every hyper-box is covered by exactly c intervals.

It follows that

$$\mathbb{E}[B_s] \geq \frac{c(2k-l-1)}{M}$$

if we can guarantee a number of c wrap-arounds (see Figure 7.3); to do so, it clearly has to hold that

$$\frac{M}{2} \cdot l \geq \left\lfloor \frac{n}{k} \right\rfloor \cdot c,$$

which can be guaranteed by satisfying

$$l \geq \frac{2cn}{Mk}, \tag{7.5}$$

and we obviously aim at minimizing l while satisfying (7.5). This means that we may set $l := \lceil 2cn/Mk \rceil$ yielding

$$\mathbb{E}[B_s] \geq \frac{c\left(2k - \left\lceil \frac{2cn}{Mk} \right\rceil - 1\right)}{M} \geq \frac{c\left(2k - \frac{2cn}{Mk} - 2\right)}{M} = \frac{2ck-2c}{M} - \frac{2c^2n}{M^2k}. \tag{7.6}$$

We distinguish two cases according to the size of M.

Case 1. Suppose that $M \geq \frac{2n}{k(k-1)}$. Let

$$\delta := \frac{k^3}{2n} - \frac{k(16k - 8 + k^2)}{18n}.$$

In the following, we want to guarantee that

$$\frac{2ck - 2c}{M} - \frac{2c^2 n}{M^2 k} \geq \delta$$

and therefore

$$0 \geq k\delta M^2 - (2ck^2 - 2ck)M + 2c^2 n \tag{7.7}$$

to prove the bound we claimed. In other words, we have to show that there exists a number of wrap-arounds c, for any $M \geq \frac{2n}{k(k-1)}$, such that (7.7) holds. To this end, let us treat the right-hand side of (7.7) as a function in M, i. e., consider

$$f(M) := \left(\frac{k^4}{2n} - \frac{k^2(16k - 8 + k^2)}{18n}\right)M^2 - (2ck^2 - 2ck)M + 2c^2 n.$$

We obtain

$$\begin{aligned}
&\left(\frac{9k^4 - k^2(16k - 8 + k^2)}{18n}\right)M^2 - (2ck^2 - 2ck)M + 2c^2 n = 0 \\
&\quad\iff M^2 - \frac{18n(2ck - 2c)}{8k(k^2 - 2k + 1)}M + \frac{36c^2 n^2}{8k^2(k^2 - 2k + 1)} = 0,
\end{aligned}$$

and thus it follows that $f(M)$ has roots at

$$\begin{aligned}
M_{1,2} &= \frac{9n(2ck - 2c)}{8k(k-1)^2} \pm \sqrt{\left(\frac{9n(2ck - 2c)}{8k(k-1)^2}\right)^2 - \frac{9c^2 n^2}{2k^2(k-1)^2}} \\
&= \frac{9cn(k-1)}{4k(k-1)^2} \pm \sqrt{\left(\frac{9cn(k-1)}{4k(k-1)^2}\right)^2 - \frac{9c^2 n^2}{2k^2(k-1)^2}} \\
&= \frac{9cn}{4k(k-1)} \pm \sqrt{\frac{81c^2 n^2}{16k^2(k-1)^2} - \frac{9c^2 n^2}{2k^2(k-1)^2}} \\
&= \frac{9cn}{4k(k-1)} \pm \sqrt{\frac{9c^2 n^2}{16k^2(k-1)^2}} = \frac{9cn}{4k(k-1)} \pm \frac{3cn}{4k(k-1)},
\end{aligned}$$

which gives $M_1 = \frac{3}{2} \cdot \frac{nc}{k(k-1)}$ and $M_2 = 3 \cdot \frac{nc}{k(k-1)}$. Clearly,

$$f(M) = \left(M - \frac{3}{2} \cdot \frac{nc}{k(k-1)}\right) \cdot \left(M - 3 \cdot \frac{nc}{k(k-1)}\right) \leq 0$$

is satisfied (and therefore (7.7)) if and only if

$$\frac{3}{2} \cdot \frac{nc}{k(k-1)} \leq M \leq 3 \cdot \frac{nc}{k(k-1)}.$$

To show that we can cover all possible values of M for the right choice of c, note that, for $c = 1$, we have

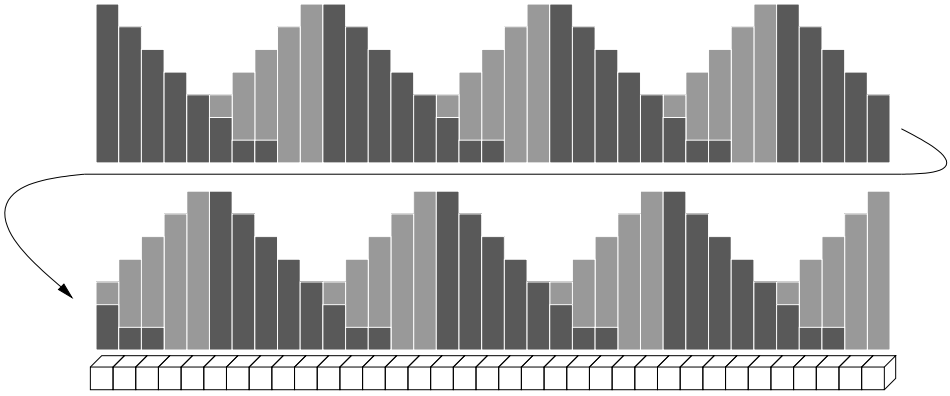


Figure 7.3: The gain of R with wrap-around $c = 2$.

$$\frac{3}{2} \cdot \frac{n}{k(k-1)} \leq 2 \cdot \frac{n}{k(k-1)} \leq M,$$

and, for two consecutive values c' and c'' (i. e., for $c'' = c' + 1$), we have

$$\frac{3c'n}{k(k-1)} \geq \frac{3nc''}{2k(k-1)} = \frac{3n(c'+1)}{2k(k-1)} \iff 2 \geq \frac{c'+1}{c'},$$

which holds for any $c' \geq 1$. From (7.6) and (7.7), we get

$$\mathbb{E}[B_s] \geq \frac{2ck - 2c}{M} - \frac{2c^2 n}{M^2 k} \geq \frac{k^3}{2n} - \frac{k(16k - 8 + k^2)}{18n} \geq \left(1 - \frac{1}{9}\right)\frac{k^3}{2n} - 1.$$

Case 2. Now suppose that $\frac{n-k+1}{k^2} < M \leq \frac{2n}{k(k-1)}$. In this case, we do not have enough randomness to do wrap-arounds. Thus, fixing $c := 1$ in (7.6), we immediately get

$$\mathbb{E}[B_s] \geq \frac{2k - l - 1}{M} \geq \frac{2k-2}{M} - \frac{2n}{M^2 k}.$$

The theorem follows. □

The intuitive idea behind the proof above is the following. With increasing M, as long as $M \leq \frac{2n}{k(k-1)}$, the gain of R increases, because we can choose from more and more deterministic strategies and shrink l. However, if M grows too much, the gain gets less, and we start the wrap-around technique and thereby get a bound that does not depend on M anymore. If M increases over some threshold, another wrap-around is made and the intervals get decompressed a little. If M increases further, the intervals shrink until, for some next threshold value, another wrap-around is made.

Up to this point, we have restricted ourselves to even values of M. However, a simple observation resolves this issue.

Theorem 7.16. *For any M and any randomized online algorithm $\mathtt{R} \in \mathcal{R}and(M)$, the bounds from Theorem 7.15 hold up to a multiplicative factor of $1 - 1/M$.*

Proof. Theorem 7.15 holds for any even M. Now if M is odd, R acts as above for any random choice from $\{1, \ldots, M-1\}$. In any of the cases, the expected gain X is the same

as in Theorem 7.15. If M is chosen, R may open some arbitrary boxes. In this case, we assume that the gain is 0. We therefore get a total expected gain of at least

$$\left(1 - \frac{1}{M}\right) \cdot X + \frac{1}{M} \cdot 0$$

as we claimed. □

Observe that the above theorems provide us with two sharp thresholds on the amount of randomness. If $M < (n + k - 1)/k^2$, randomness does not help at all. On the other hand, if $M > 2n/(k(k-1))$, which corresponds to roughly $\log n - 2 \log k$ random bits, any further randomness does not help to improve the gain.

Randomized Online Algorithms with Advice

Next, we present the main result of this section by analyzing the tradeoff between randomness and advice for (n, k)-Boxes. To this end, we consider online algorithms that base their computations on both advice bits and private randomness (analogously to the concept of private random bits as introduced in Section 7.3, i. e., the oracle does not know the outcome of the random decisions). In essence, we prove that, for the same amount of randomness, every additional advice bit allows us to find the same expected number of full boxes within a sequence of roughly twice the length. This implies that, for instance, the bound of $\frac{8}{9} \cdot \frac{k^3}{2n} - 1$ on the expected number of opened full boxes from the first claim of Theorem 7.15 can already be reached with a random number of roughly half the size.

To achieve this goal, we introduce the following notation. By $F(n, k, M, b)$ we denote the expected number of full boxes opened by the best algorithm that solves (n, k)-Boxes with randomness M and b bits of advice.

Next, we generalize (n, k)-Boxes to (S, n, k)-Boxes, where $S \subseteq \{1, \ldots, n - k + 1\}$. In (S, n, k)-Boxes, the starting position Adv chooses for the full boxes has to be picked from the set S, otherwise it is identical to (n, k)-Boxes. An algorithm that solves (S, n, k)-Boxes is called *faithful* if it only opens boxes whose positions are in S until the first full box is encountered.

Lemma 7.17. *For every algorithm* A *that solves* (S, n, k)*-Boxes with randomness* M *and* b *advice bits, there exists a faithful algorithm* A′ *that also solves* (S, n, k)*-Boxes with randomness* M *and* b *advice bits such that* $\mathbb{E}[B_s(\mathtt{A}')] \geq \mathbb{E}[B_s(\mathtt{A})]$*, for every starting position* s*.*

Proof. Suppose we are given A as stated by the lemma. The following strategy is carried out by A′ until it finds the first full box: If A opens box b_i, then A′ opens box b_j with

$$j := \max\{s \in S \mid s \leq i\},$$

i. e., the next smaller box that is in S, or it opens no box at all if this maximum does not exist. It is easy to see that A′ opens its first full box not later than A: Assume A opens the first full box b_i, then $b_p, b_{p+1}, \ldots, b_i$ are all full, where p is the starting position of the sequence chosen by Adv. Of course, $p \in S$ and $p \leq j \leq i$. Afterwards, A′ opens the same boxes as A, except for the case where A wants to open the box b_j which was already opened by A′. In this case, A′ opens b_i; that way, A′ opens at least the same number of full boxes as A. □

In the following, we relate the expected gain of (S, n, k)-Boxes to the one of (n, k)-Boxes.

Lemma 7.18. *Let* A *be an algorithm that solves* (S, n, k)*-*Boxes *with randomness* M *and no advice. Then* $\min_s \mathbb{E}[B_s(\mathtt{A})] \leq F(|S| + k - 1, k, M, 0) + 1$.

Proof. Due to Lemma 7.17, we can assume that A is faithful without loss of generality. We now convert A into an algorithm A′ that solves $(|S| + k - 1, k)$-Boxes with randomness M and 0 advice bits such that $\min_s \mathbb{E}[B_s(\mathtt{A}')] \geq \min_s \mathbb{E}[B_s(\mathtt{A})] - 1$.

Let $S = \{p_1, \ldots, p_{|S|}\}$, where $p_1 < p_2 < \cdots < p_{|S|}$. Let us further assume that A would open boxes at positions $p_{i_1}, \ldots, p_{i_k}$ if we would report them all as empty. Then A′ opens boxes at positions $i_1, \ldots, i_r$ until it finds the first full box b_{i_r}; afterwards, A′ continues with local search.

Let i be a worst-case starting position for A′. As we have already discussed, the last $k-1$ boxes cannot be starting positions. We therefore have $i \leq |S| + k - 1 - (k-1) = |S|$. We choose p_i as the starting position for A. If A′ does not open a full box in the first r rounds, then neither does A, because A is faithful and therefore only opens boxes from S. By construction, any full box it finds this way corresponds to a full box A′ opens. □

We are now ready to prove the following two theorems that show the power of advice for (n, k)-Boxes.

Theorem 7.19. $F(n, k, M, b) \leq F(\lceil (n - k + 1)/2^b \rceil + k - 1, k, M, 0) + 1$.

Proof. If an algorithm A solves (n, k)-Boxes with b advice bits, by the pigeonhole principle, there is at least one advice string that is used for at least $\lceil (n-k+1)/2^b \rceil$ different starting positions; let S be the set of these starting positions. Then the algorithm A can be used to solve (S, n, k)-Boxes without advice (but with randomness M). Hence, if A is optimal, then $F(n, k, M, b) \leq \min_s \mathbb{E}[B_s(\mathtt{A})] \leq F(|S| + k - 1, k, M, 0) + 1$ by Lemma 7.18. Clearly, $F(n, k, M, b)$ is monotonically decreasing in n (if k, M, and b are fixed, Adv obtains more positions to hide the boxes with growing n), and thus $F(|S| + k - 1, k, M, 0) \leq F(\lceil (n - k + 1)/2^b \rceil + k - 1, k, M, 0)$. □

Theorem 7.20. $F(n, k, M, b) \geq F(\lceil (n - k + 1)/2^b \rceil + k - 1, k, M, 0)$.

Proof. Consider an algorithm A for (n, k)-Boxes that uses b bits of advice. Again, there are $n - k + 1$ possible starting positions for Adv. On the other hand, we may subdivide these boxes into 2^b groups of size

$$\left\lceil \frac{n - k + 1}{2^b} \right\rceil$$

and encode the position of the group that contains the starting position using b bits. Clearly, we can extend this interval by $k-1$ boxes which cannot contain a starting position. Afterwards, A simulates an optimal algorithm A′ for an instance of this size. It directly follows that A gains at least as much as A′. □

Combining our results regarding randomized computation with Theorems 7.19 and 7.20 immediately yields the following upper and lower bounds on the expected number of items found by randomized online algorithms with advice.

Corollary 7.21. *We can bound the function* F *from above as follows.*

1. *If* $M < \lceil \frac{n-k+1}{2^b} \rceil / k^2$*, then* $F(n, k, M, b) \leq 1$.

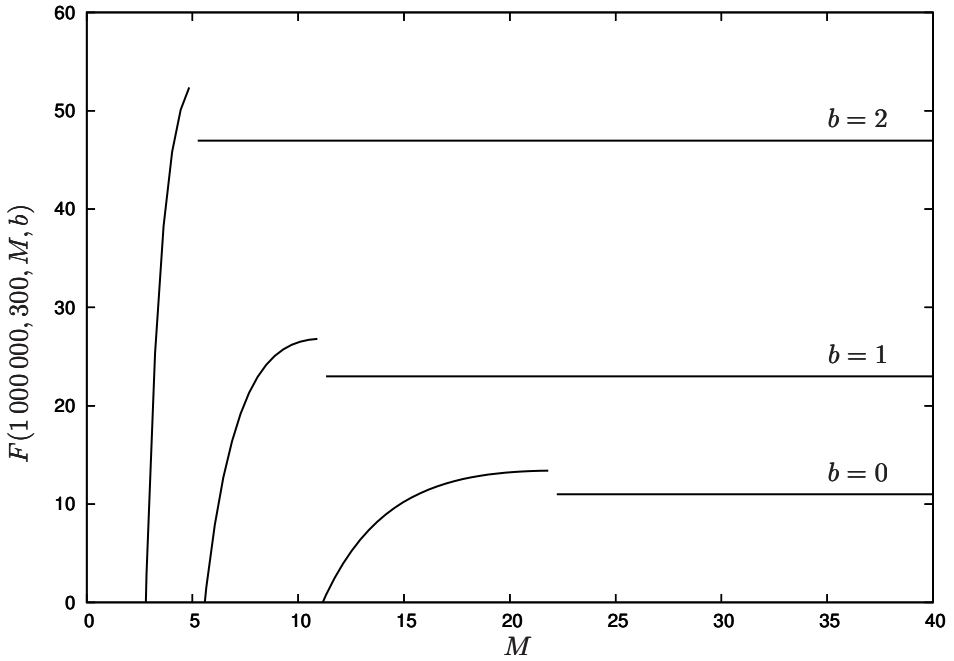


Figure 7.4: The expected gain using randomness M and b bits of advice for $n = 1\,000\,000$, $k = 300$, different values of M, and up to two advice bits.

2. For any M,

$$F(n,k,M,b) \leq \frac{k^2(k+1)}{2\lceil \frac{n-k+1}{2^b} \rceil} + 1.$$

3. If $M = c\lceil \frac{n-k+1}{2^b} \rceil / k^2$ with $c \in [1, 2k/(k+3)]$, then

$$F(n,k,M,b) \leq \frac{2(k+\frac{3}{2})}{M} - \frac{2\lceil \frac{n-k+1}{2^b} \rceil}{M^2 k} + 1 \quad \text{and}$$
$$F(n,k,M,b) \leq \frac{2(c-1)}{c^2} \cdot \frac{k^3}{\lceil \frac{n-k+1}{2^b} \rceil} + \frac{3k^2}{c \cdot \lceil \frac{n-k+1}{2^b} \rceil} + 1.$$

□

Corollary 7.22. *Let M be an even number. We can bound the function F from below as follows.*

1. If $M \geq 2(\lceil \frac{n-k+1}{2^b} \rceil + k - 1)/(k(k-1))$, then

$$F(n,k,M,b) \geq \frac{8}{9} \cdot \frac{k^3}{2 \cdot \lceil \frac{n-k+1}{2^b} \rceil + k - 1} - 1.$$

2. If $\lceil \frac{n-k+1}{2^b} \rceil \cdot \frac{1}{k^2} < M < 2(\lceil \frac{n-k+1}{2^b} \rceil + k - 1)/(k(k-1))$, then

$$F(n,k,M,b) \geq \frac{2k-2}{M} - \frac{2 \cdot \lceil \frac{n-k+1}{2^b} \rceil + 2k - 2}{M^2 k}.$$

□

Note that these upper and lower bounds are almost tight. Obviously, Corollary 7.22 can easily be extended to the case of odd values of M using Theorem 7.16. A graphical illustration of the connection between the amount of randomness, the number of advice bits, and the expected number of full boxes that are opened is given in Figure 7.4.

Chapter 8

Concluding Discussion

In this thesis, we have studied online problems within the framework of advice complexity. This concept tries to measure the amount of information that we miss when computing in an online environment, which prevents online algorithms from achieving results of high quality (compared to offline algorithms). To this end, we used a model that is as generic as possible, only describing the *pure* information necessary by means of an additional advice tape. As we have seen, problems may behave very differently. We conclude by giving some first ideas about how to extend this model to other fields of computer science in Section 8.2, point to related work from different areas in Section 8.3, and finally try to give some directions for further research in Section 8.4. However, before that, we want to describe an important observation concerning our proof techniques that might be crucial for further work.

8.1 A Note on the Model

In Section 1.6, we have formally introduced our model in a way such that the adversary ADV knows the oracle O. For the ease of presentation, let us consider online minimization problems and speak about strict competitiveness only. Our aim was to construct pairs of algorithms and oracles that achieve a specific competitive ratio if the algorithm reads a particular number of advice bits. Formally,

$$\exists\ (\mathtt{A}, \mathtt{O})\ \forall\ \textsc{Adv}\text{ creating } I\colon\ \operatorname{cost}(\mathtt{A}^{\phi}(I)) \leq c \cdot \operatorname{cost}(\textsc{Opt}(I)).$$

When proving lower bounds, we usually showed that, for some advice string of some fixed length, there exist different inputs within one class of inputs that cannot be distinguished sufficiently to achieve some specific competitive ratio; this becomes very visible when we show lower bounds on achieving optimality, e. g., in Theorem 4.3, where A has to know a unique permutation. However, ADV must know the (at least) two inputs that correspond to one advice string to cause A to fail when being given one of them, i. e., ADV has to know how O prepares the advice strings to select the concrete input. Formally, what we show is that

$$\forall\ (\mathtt{A}, \mathtt{O})\ \exists\ \textsc{Adv}\text{ creating } I\colon\ \operatorname{cost}(\mathtt{A}^{\phi}(I)) > c \cdot \operatorname{cost}(\textsc{Opt}(I)). \tag{8.1}$$

Again, here, the oracle is fixed (i. e., known to ADV), and we show that, for every algorithm and oracle, such an adversary exists.

Interestingly, in some of our proofs we are using an adversary that is seemingly weaker. For instance, consider the proof of Theorem 2.12 where we were dealing with a lower bound

on achieving c-competitiveness for JSS. Here, we subdivided the whole grid induced by the input into sub-grids such that every algorithm from Alg(A) is assigned to one of them. Note that these sub-grids do not interfere with each other. Therefore, ADV does not have to know which advice string is given; it is sufficient that it knows how A behaves for all possible advice strings. Thus, what we have really shown is that

$$\forall\ \mathtt{A}\ \exists\ \mathtt{ADV}\text{ creating } I \text{ such that } \forall\ \mathtt{O}\colon\ \mathrm{cost}(\mathtt{A}^{\phi}(I)) > c \cdot \mathrm{cost}(\mathtt{OPT}(I)). \tag{8.2}$$

The same argument is valid for the lower bound proofs of the set cover problem (see Theorems 5.6 and 5.11). While it seems obvious that (8.2) is stronger than (8.1), we want to point out that, as a matter of fact, both models are equally strong: Suppose that ADV merely knows A and the number b of advice bits used (which, of course, depends on n). ADV may then, for every I, proceed by using a procedure as shown in Algorithm 8.1 and *learn* which one is the *strongest* advice for every input. It follows that there must exist an oracle that constructs this optimal advice for every input, thus never being worse than any other oracle on any input.

Algorithm 8.1: `get_best_advice(`I, b`)`

```
i := 1;
BEST := A with advice 1;
while i ≤ 2^b
    A_i := A with advice i;
    simulate A_i on I;
    if cost(A_i(I)) ≤ cost(BEST(I))
        BEST := A_i;
    i := i + 1;
output index of BEST;
end
```

This way, ADV can determine the worst input for the best oracle. We therefore can safely say that ADV, as we have used it in this thesis, is indeed *oblivious* (as defined in Section 1.3), with the restriction that it knows the number of advice bits b supplied for the online algorithm A with advice. Consequently, it does not matter which technique is used for proving lower bounds, and it seems that it depends on the concrete problem which of the two approaches is the better choice.

8.2 Extending the Model

In [89], Hromkovič, Královič, and Královič proposed to use advice complexity for a broader set of scenarios to investigate what kind of information needs to be extracted from some problem instance of a computational problem to solve it with respect to some measurement (see also [34]): As we have *Kolmogorov complexity*[1] [48, 105, 117] or *Shannon's concept of entropy*[2] [141] to measure the information some string carries, it was suggested to use advice complexity to specify the information that is hidden in some problem description (the *information complexity* or *information content* [89]). It is clear that both of the above mentioned concepts cannot provide such a measurement. As pointed out, in online computation, the advice complexity measures what we lose for not knowing the future. A

[1] Andrei N. Kolmogorov, ⋆25.04.1903, †20.10.1987, Russian mathematician.
[2] Claude E. Shannon, ⋆30.04.1916, †24.02.2001, American mathematician.

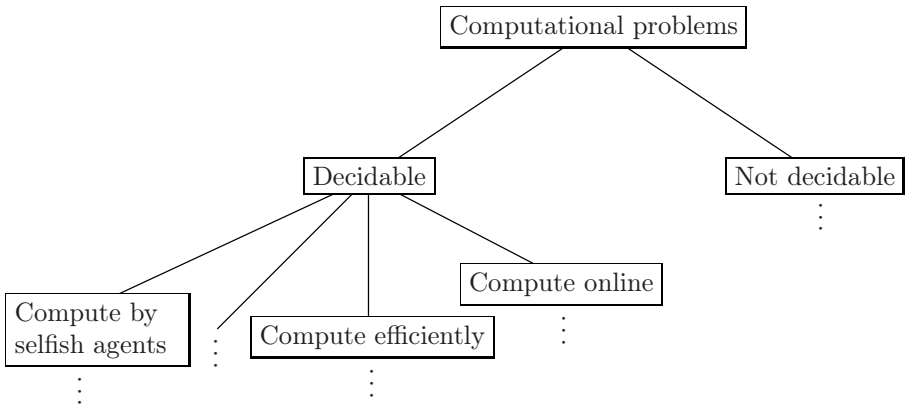


Figure 8.1: Computational problems.

next step could be to study advice complexity in scenarios where other drawbacks, which are implied by the model, can be compensated.

On the highest level, computational problems are subdivided into those that are not solvable (i. e., undecidable problems, e. g., the *halting problem* [87], as shown by Turing in 1936 [149]) and the ones that are. Problems that are not decidable may be distinguished further by the fact whether they are recursively enumerable, etc. Then again, when dealing with problems that we can solve by means of Turing machines that halt on any given input (i. e., algorithms), we are often still not completely satisfied by this circumstance. In practical applications, we do not only want to know that a problem is solvable, but we also want the solution to be computed in some specific framework while obeying some certain rules. Thus, a number of restrictions may be imposed on the way of how the solutions to these problems are obtained; let us only name a few (see Figure 8.1 for a sketch).

(i) *Computing online.* As discussed in this thesis, here, we are facing the fact that the input arrives successively as it is the case for a large number of computational problems encountered in practice. Classically, we use the competitive ratio to measure the hardness of these problems. What we were aiming at, was to see how much information is both necessary and sufficient to overcome bad results induced by the above restriction.

(ii) *Restricting the allowed running time.* As described in Section 1.2, we call algorithms *efficient* if they run in polynomial time. Now let us consider our model of advice for any *offline* decision problem [86, 87] (in $\mathcal{NP}$ or even beyond). It is immediately clear that one single bit of advice is sufficient to produce optimal output. If the problem is in $\mathcal{P}$, trivially, no advice is needed to obtain an efficient algorithm by definition; if the problem is $\mathcal{NP}$-hard, however, one bit is also necessary, unless $\mathcal{P} = \mathcal{NP}$. This needs no further investigation, still it is remarkable that, by the nature of this huge class of computational problems, there are absolutely no differences in hardness in terms of advice complexity for all problems that are $\mathcal{NP}$-hard. Only one single bit is sufficient to be as efficient as for any trivial computational problem. It gets more interesting, if we consider optimization problems, i. e., problems, for which we want to find good solutions among a huge number of feasible ones. As with decision problems, we are particularly interested in designing algorithms that are efficient. In other words, we are not satisfied by the fact that, for some optimization problem P, we are guaranteed to get a good solution eventually, but we also want guarantees on the time it takes to get it. Here, the *approximation ratio* takes the place of the competitive ratio in online scenarios. For $\mathcal{NP}$-hard problems, we

cannot hope to get an optimal solution in polynomial time, if $\mathcal{P} \neq \mathcal{NP}$. Thus, if we want to work efficiently, we pay by only getting an approximate solution. Depending on the problem, the quality of this solution might vary drastically. Analogously to what we did when dealing with online problems, we may ask how much information is sufficient and necessary to overcome this disadvantage, i. e., how much advice is needed to efficiently compute an *optimal* solution (we give some more details in the next subsection).

(iii) *Game-theoretic settings.* When considering games, we are not dealing with one central algorithm that processes the input, but are facing the fact that a number of selfish *agents* compute the solution to a problem (see, e. g., [130, 146] for an introduction). Many games admit so-called *Nash equilibria*[3] in which no agent has an incentive to change its strategy, given that the strategies of all other agents are fixed [129]. However, these equilibria are not necessarily optimal considering the overall cost (usually referred to as the *social welfare*). In 1999, Koutsoupias and Papadimitriou introduced the *price of anarchy* as the ratio of a worst Nash equilibrium and the optimal solution of a game [113]. Similar to the above scenarios, we may state that the price of anarchy tells us what we lose for not computing the solution to a problem by one central algorithm that processes the whole input, but by selfish agents. Again, we can ask whether advice helps in these environments. We can propose different types of advice that might either be distributed over all agents or that just forces a strict subset of agents to obey some strategies that are not necessarily optimal for them, but increase the social welfare. This would require agents that act selfishly as long as they are left alone, but that still obey commands given by a central authority (i. e., the oracle $\mathtt{O}$).

To sum up, the competitive ratio, the approximation ratio, and the price of anarchy are tools that tell us what we lose in particular scenarios for satisfying some type of restriction; they are all defined as the ratio of the cost achievable by obeying this restriction and the optimal cost that we can, in general, only obtain when neglecting these restrictions (under the assumption that $\mathcal{P} \neq \mathcal{NP}$ for optimization problems). Advice may then be used to measure the hardness of particular problems in a particular framework. As we briefly discuss in Section 8.4, these concepts might also be combined.

Optimization

When considering optimization problems, we are given an instance of a computational problem together with a set of feasible solutions (we do not want to give a formal definition and refer to the standard literature [11, 86, 131, 150]). We are particularly interested in studying optimization problems that are within the complexity class $\mathcal{NPO}$ [86].

Definition 8.1 ($\mathcal{NPO}$). *The class $\mathcal{NPO}$ contains all optimization problems P that have the following properties.*

(i) It is possible to efficiently verify whether a given string is an instance of P.

(ii) The size of any solution O for any instance I of P has a size that is polynomial in the size of I; furthermore, it is possible to efficiently verify whether O is a solution for I.

(iii) The cost function associated with P is computable in polynomial time.

[3] John F. Nash, ⋆13.06.1928, American mathematician.

Now suppose that we construct algorithms to solve some $\mathcal{NP}$-hard optimization problem P from $\mathcal{NPO}$ that have access to an advice tape with content ϕ, which was created beforehand by an oracle that inspects the given input. An algorithm $\mathtt{A}$ with advice for P then computes a solution O based on ϕ and I. Due to Definition 8.1 (ii), we have that any solution has a size of at most $p(n)$, for some polynomial p and with n being the size of I. Thus, there exists a polynomial p such that *all* solutions for I are at most of size $p(n)$. A polynomial upper bound follows. The oracle writes the solution O onto the advice tape, and $\mathtt{A}$ simply copies the content to its output tape. Note that, to do so, the advice has to be given in a self-delimiting form, using, e. g., Observation 1.14.

Observation 8.2. *For any optimization problem in $\mathcal{NPO}$, polynomial advice suffices to create optimal output.*

For some problems, it is easy to see that linear advice is sufficient. For instance, consider the *maximum satisfiability problem*, MaxSAT for short, formally defined as follows.

Definition 8.3 (MaxSAT). *Given a set $\Phi = \{F_1, \ldots, F_m\}$ of m clauses over k Boolean variables*[4] *$X(\Phi) = \{x_1, \ldots, x_k\}$, find an assignment that maximizes the number of clauses that evaluate to* 1.

Obviously, an optimal assignment can be specified by $\mathtt{O}$ by using k advice bits, so we get that the information content of MaxSAT is at most linear in the number of the given variables. Since the length of the input n is larger than k, it follows that the number of advice bits is at most linear in the input size. On the other hand, considering a lower bound on the advice necessary to compute an optimal solution, we can make the following general statement.

Observation 8.4. *Let I be an instance of length n of some $\mathcal{NP}$-hard optimization problem P from $\mathcal{NPO}$ and let d be a constant. There is no efficient algorithm that computes an optimal solution for I and that uses at most $\log(n^d)$ bits of advice, unless $\mathcal{P} = \mathcal{NP}$.*

Proof. Towards contradiction, suppose there exists an optimal algorithm $\mathtt{A}$ with advice that runs in $\mathcal{O}(p_1(n))$ time, for some polynomial p_1 of degree d_1, and that uses at most $\log(n^d)$ bits of advice. We can then construct an exact algorithm $\mathtt{B}$ that simulates $\mathtt{A}$ on every possible advice string and outputs the best feasible solution. To this end, $\mathtt{B}$ has to check, for every possible advice string, whether the output of $\mathtt{A}$ is a feasible solution for I and, if so, calculate its cost. Due to Definition 8.1 (ii) and (iii), both can be done, for one output, in time $\mathcal{O}(p_2(n))$, for some polynomial p_2 of degree d_2. Since there are $2^{\log(n^d)} = n^d$ possible advices of the above length, the running time of $\mathtt{B}$ does not exceed $n^d \cdot (p_1(n) + p_2(n))$ which clearly is in $\mathcal{O}(q(n))$, for some polynomial q of degree $d + \max\{d_1, d_2\}$. But this contradicts the $\mathcal{NP}$-hardness of P. □

The gap between the bounds stated by Observations 8.2 and 8.4 is rather large. For some problems, including MaxSAT, we can, using stronger assumptions, even show lower bounds $f(n)$, for all $f(n) \in o(n)$. The *Exponential-Time Hypothesis* (ETH), formally posed by Impagliazzo and Paturi in 1999 [92], implies that there does not exist any algorithm for 3-SAT that runs in time $2^{o(n)}$ [77]. Since any exact algorithm for MaxSAT running in sub-exponential time immediately yields a sub-exponential algorithm to decide any given instance of 3-SAT, we get the following observation, using a similar argument as in the proof of Observation 8.4.

[4] George Boole, ⋆02.11.1815, †08.12.1864, English mathematician.

Observation 8.5. *For* MaxSAT*, there is no efficient optimal algorithm that uses $o(n)$ bits of advice, unless the ETH fails.*

Consequently, if we believe the ETH, the upper bound of linear advice for MaxSAT is essentially (i. e., asymptotically) tight. Moreover, if the ETH is true, there are no optimal sub-exponential algorithms, and thus no efficient optimal algorithms with sub-linear advice, for a number of other problems, including *maximum independent set*, *minimum vertex cover*, or *maximum clique* [77].

Reoptimization

It is remarkable how counterintuitive things may behave; for instance, in optimization, there are problems which stay as hard even when given some seemingly very powerful advice. In particular, we want to point to *reoptimization*. This concept is somewhat incomparable to the one of advice complexity. Here, we are given some instance I_{new} of an $\mathcal{NP}$-hard optimization problem P together with a second instance I_{old} of P that differs from I_{new} only locally (i. e., very slightly) and an optimal solution $\mathcal{O}pt_{\text{old}}$ for I_{old}. We then want to answer how much knowing $\mathcal{O}pt_{\text{old}}$ helps us to efficiently obtain an optimal or high-quality solution $\mathcal{O}pt_{\text{new}}$ for the instance I_{new} of P given the additional information $(I_{\text{old}}, \mathcal{O}pt_{\text{old}})$. As a reoptimization variant of MaxSAT, we might, e. g., consider the following problem DelMaxSAT where a variable is removed from the input set.

Definition 8.6 (DelMaxSAT). *Given a triple $(I_{\text{new}}, I_{\text{old}}, \mathcal{O}pt_{\text{old}})$, where I_{old} is an instance of* MaxSAT*, $\mathcal{O}pt_{\text{old}}$ is an optimal solution for I_{old}, and I_{new} is obtained by removing one variable from the variable set of I_{old}, the problem* DelMaxSAT *is to find an optimal solution for I_{new}.*

In the last subsection, we have seen that m advice bits are sufficient to produce optimal output. Still, let us point out that, if we want to solve the problem optimally, the kind of advice presented here is useless. This might contradict our intuition, since we are given (for free) a seemingly very powerful piece of information together with the instance I_{new} that we want to solve. We can easily prove that this knowledge does not help at all by giving a straightforward reduction from the original problem.

Lemma 8.7. *There is no efficient optimal algorithm for* DelMaxSAT*, unless $\mathcal{P} = \mathcal{NP}$.*

Proof. Towards contradiction, suppose that there exists an optimal algorithm `A` for DelMaxSAT that runs in polynomial time. Let $\Phi =: I_{\text{new}}$ be any instance of MaxSAT. We introduce a new variable $x \notin X(\Phi)$ and add it to every clause from Φ. The obtained instance, which we call I_{old}, is obviously trivial: an optimal solution $\mathcal{O}pt_{\text{old}}$ sets $x = 1$ and, without loss of generality, all other variables to 0, which means that all clauses are satisfied. We set $I := (I_{\text{new}}, I_{\text{old}}, \mathcal{O}pt_{\text{old}})$, which we can clearly do in linear time, and run `A` on I. Clearly, we can adopt `A`'s output as an output for the original instance Φ of MaxSAT and thus have just constructed an optimal polynomial-time algorithm for MaxSAT, which contradicts its $\mathcal{NP}$-hardness. □

Similar (more sophisticated) hardness proofs were given for reoptimization variants of other optimization problems, e. g., for the shortest common superstring problem [22]. Many other problems on graphs have been investigated in terms of reoptimization, e. g., the metric traveling salesman problem [9, 13, 20, 24, 115], the metric traveling salesman problem with deadlines [29, 30], and the Steiner tree[5] problem [21, 25, 27, 157]. Further-

[5] Jakob Steiner, ⋆18.03.1796, †01.04.1863, Swiss mathematician.

more, the knapsack problem [10] and different covering problems [23] were investigated within this scope. In many cases, positive results were obtained like constructing a PTAS for a reoptimization variant of a problem that is $\mathcal{APX}$-hard. However, surprisingly, the following was shown for the traveling salesman problem on general graphs: even if *all* optimal solutions (in particular, exponentially many, thus using a large amount of advice) are known, there still does not exist an efficient algorithm that achieves improved results [28].

8.3 Related Work

The concept of using advice (in other words, using additional information) to study, classify, and evaluate problems and their hardness has been used in many settings different from ours. In what follows, we only mention a few examples. Note that the oracles used in these settings work differently from the ones in our model.

The question of how much additional information is sufficient to solve an otherwise infeasible problem is exceedingly interesting, also in practice, in cryptography (for an introduction, we refer to the standard literature, e. g., [60]). It is a well-known fact that many of the cryptographic systems in use rely on the assumption that factoring, i. e., to determine the prime factors of some composite number $N \in \mathbb{N}$, is infeasible for large numbers. For instance, the extensively used cryptographic system RSA[6] [138] would be considered broken if we would be able to efficiently factor large numbers; actually, it might be easier to break RSA [2, 37], whereas for the cryptographic system Rabin[7] [134], it is even *proven* that it is *as hard* to break as factoring. Remarkably, it is not yet known whether factoring is $\mathcal{NP}$-hard. The decision version of the problem, i. e., given a composite number N and a threshold t, decide whether N has a factor $d \in \{2, \ldots, t\}$, obviously is in $\mathcal{NP}$. It is, however, conjectured to be neither in $\mathcal{P}$ nor $\mathcal{NP}$-complete. Therefore, it might be an $\mathcal{NP}$-intermediate [11] problem (indeed, if $\mathcal{P} \neq \mathcal{NP}$, such problems *do* exist, as shown by Ladner [116]), but there are no proofs for that conjecture up to this day. Intriguingly, the problem to decide whether a given number is prime, denoted by Primes, was proven to be in $\mathcal{P}$ by Agrawal, Kayal, and Saxena in 2002 [3].

In 1985, Rivest and Shamir studied how many advice bits are necessary to efficiently factor an n-bit natural number $N = p \cdot q$, where p and q are $n/2$-bit primes [137]. These bits are supplied by an oracle $\mathtt{O}$ that correctly answers questions asked by the algorithm by "yes" or "no". The authors improved the trivial upper bound of $n/2$ to $n/3 + \mathcal{O}(1)$. Ten years later, Maurer significantly improved this result by showing that, for sufficiently large n, $n \cdot \varepsilon$ bits of advice suffice to construct a randomized polynomial-time algorithm, for any $\varepsilon > 0$ [123]. Moreover, oracles for the *Diffie-Hellman* key exchange protocol[8] (introduced in [62]) were studied by Maurer and Wolf [124].

As we have pointed out in the previous section, our model is not suited to study decision problems with given advice. However, there exist other approaches where oracles are employed for a further exploration of decision problems and the classes $\mathcal{P}$ and $\mathcal{NP}$ (comprehensive introductions are given in [11, 131]). Here, Turing machines are equipped with an oracle that instantly answers questions concerning special decision problems; $\mathtt{M}^{\mathrm{SAT}}$,

[6] Ronald L. Rivest, ⋆06.05.1947, American computer scientist;
Adi Shamir, ⋆06.07.1952, Israeli computer scientist;
Leonard Adleman, ⋆31.12.1945, American computer scientist.
[7] Michael O. Rabin, ⋆01.09.1931, Israeli computer scientist.
[8] Martin E. Hellman, ⋆02.10.1945, American electrical engineer;
Whitfield Diffie, ⋆05.06.1944, American mathematician.

for instance, is a Turing machine that can query an oracle for SAT asking whether some Boolean formula is decidable, and it gets a correct answer immediately. The differences to our model are obvious: The oracles considered do not create their answers depending on the input, but give some general advice when queried. Let $\mathcal{P}^{\mathtt{O}}$ [$\mathcal{NP}^{\mathtt{O}}$] be the class of decision problems that can be decided by deterministic [nondeterministic] Turing machines with an oracle $\mathtt{O}$. In 1975, Baker, Gill, and Solovay showed that there exist oracles $\mathtt{O}_1$ and $\mathtt{O}_2$ such that both $\mathcal{P}^{\mathtt{O}_1} = \mathcal{NP}^{\mathtt{O}_1}$ and $\mathcal{P}^{\mathtt{O}_2} \neq \mathcal{NP}^{\mathtt{O}_2}$ [15]. This result is of great relevance as it renders a large number of proof techniques for resolving the $\mathcal{P}$ vs. $\mathcal{NP}$ problem useless (such approaches are called *relativizing* as they work both with and without oracles).

In Sections 7.3 and 7.4, we studied the collaboration of an oracle and a random source. For decision problems, a similar idea is followed by *interactive proof systems* (see, e. g., [11, 86]). Here, a Turing machine $\mathtt{M}$ has access to both a random tape and a so-called *proof tape*; $\mathtt{M}$ has *random access* to this proof tape. For some "yes"-instance I of length n of a decision problem D (e. g., SAT), the proof tape contains a *witness* that proves that I is indeed a "yes"-instance of D (e. g., an assignment of the variables such that the given formula is true). $\mathtt{M}$ is also called a *probabilistic verifier*. The complexity class $\mathcal{PCP}(f(n), g(n))$ contains all decision problems that can be solved efficiently and with high probability by a probabilistic verifier that accesses at most $f(n)$ random bits and $g(n)$ bits of the proof tape. Clearly, $\mathcal{P} = \mathcal{PCP}(0, 0)$ and $\mathcal{NP} = \mathcal{PCP}(0, \text{poly}(n))$, where $\text{poly}(n)$ denotes the class of all polynomials over $\mathbb{N}$. The PCP theorem, proven by Arora et al. in 1992 [12], states that $\mathcal{NP} = \mathcal{PCP}(\mathcal{O}(\log n), \mathcal{O}(1))$, i. e., there is a probabilistic verifier for any problem in $\mathcal{NP}$ that uses $\mathcal{O}(\log n)$ random bits and that reads a constant number of bits of the proof tape. The PCP theorem provides a strong technique for proving lower bounds on the approximation ratio of optimization problems [12, 11, 86, 83], unless $\mathcal{P} = \mathcal{NP}$.

Finally, we want to point to *dynamic data structures*, where, recently, the concept of advice was also introduced and used as a lower bound technique [49, 132].

8.4 Further Research and Open Questions

Let us conclude with some suggestions for future work. First of all, many online problems have not yet been studied in terms of their advice complexity. Moreover, for the problems that were already addressed, many upper and lower bounds are far apart. As pointed out, for instance, there is quite some room for improvement for k-Server; as recent results suggest, it seems likely that the upper bound of Theorem 4.6 might still be drastically improved, at least for many instances [16]. Additionally, we do not know any lower bound on achieving some competitive ratio $c > 1$. As for SetCover, there remains an exponential gap between the lower bound (see Theorem 5.11) and the upper bound (see Theorem 5.10) when measuring the advice complexity with respect to the size of the set family $\mathcal{S}$. For a large number of the problems studied, at least the constants for many lower and upper bounds can be improved. Furthermore, we studied a rather special class of job shop scheduling problems in Chapter 2. It would certainly be reasonable to study more general settings (e. g., to have more than 2 jobs) that require more advice to be solved near-optimally. Also, so far, a weighted version of SetCover has not yet been investigated; in this thesis, we assumed that all members of $\mathcal{S}$ have a weight of 1. If we allow arbitrary weights, we cannot assume anymore that supersets of sets are always taken by any online algorithm to satisfy requests. This might lead to higher lower bounds.

Some of the lower bounds (see Chapters 3 and 4) are only valid for *strictly* competitive algorithms. A next step towards more general statements could be to obtain lower bounds for non-strict competitiveness. This will probably require more complex arguments as it does not allow for an optimal solution of constant size anymore.

As usual, we followed a concept of *unconditional hardness* [38] when analyzing online algorithms neglecting their running time and focussing on what we lose for not knowing the whole input. Nevertheless, the majority of these algorithms are in fact efficient; an exception are the presented algorithms for SetCover. In Section 5.1, we encoded information about the actual input on the advice tape such that the algorithm still had to compute the solution itself. Therefore, it was left with solving an $\mathcal{NP}$-hard problem. On the other hand, the upper bounds with respect to the size of $\mathcal{S}$, presented in Section 5.2, use information about the *solution*, and the resulting algorithms obviously run in polynomial time. In general, it makes sense to analyze the advice complexity of efficient online algorithms. Almost nothing is known about the comparison of online algorithms with advice with unrestricted and restricted runtime, opening an interesting field for further research.

In Section 7.4, we have introduced (n, k)-Boxes as a simple problem to extensively study the cooperation of an oracle and a random source. However, so far, randomized online algorithms with this kind of advice have not been investigated for any other problems. More generally, it seems very interesting to investigate the relation between *public* and *private* random bits in more detail and to search for online algorithms that produce good solutions with high probability (see Section 7.2). In fact, results with high probability can provide a valid alternative to the expected competitive ratio in online computation. Moreover, as shown in Section 7.1, for some problems, we might derive powerful barely random algorithms from online algorithms with advice that use a constant number of advice bits. This was, to the best of our knowledge, so far, only done for JSS, SimpleKnapsack, and Paging, although it might prove to be a new approach for the construction of barely random algorithms for a broader class of online problems (once more, let us stress that it does not provide any general technique). Furthermore, it is easy to see that lower bounds on the advice necessary carry over to the amount of random bits required to obtain some specific output quality (see Chapter 7). A more general question, asking how many advice bits we need instead of random bits to achieve the same competitive ratio, has already been partly answered by Theorem 7.2. Then again, only minimization problems were considered, and it is unclear whether such statements can also be made for maximization problems. This tradeoff between advice and randomization seems to be of great theoretical interest.

Before we have introduced our model, in Section 1.5, we have comprised some of the measurements that were proposed as alternatives to the competitive ratio. As we have already stated, some of these measurements aim at a somewhat different direction than we do here as they do not try to give another way to classify the problems themselves, but want to give a more realistic way to compare existing algorithms. Examples include the access graph model to compare LRU and FIFO or the comparative ratio to compare algorithms with lookahead to those that do not possess this feature. In Section 6.1, we have shown that resource augmentation is very powerful with respect to advice complexity: For SimpleKnapsack, instead of using a logarithmic number of advice bits, it suffices to only use a constant number if we allow some small overpacking. It seems reasonable to use this approach for the study of further alternative measures.

Throughout this thesis, we have considered an adversary that knows the number b of advice bits the online algorithm reads. It might be worth further research to investigate how the model changes if we drop this strong requirement. Actually, b cannot be obtained by merely studying the online algorithm, because it may depend on the advice itself. Clearly, the lower bounds will get worse, but it is not clear to which extent.

Finally, as pointed out in the previous section, it is surely worth trying to extend the model of *advice in computation* to a more general framework. An (admittedly rather ambitious) long-term goal would be to prove general statements of the following form:

> *For the problem P, the crucial information that must be extracted from inputs of length n in the framework W is $b(n)$.*

With our ideas from Section 8.2, we may even pose more general questions. We already mentioned the study of efficient online algorithms with advice. Suppose we know that there exists a c-competitive online algorithm with advice for an $\mathcal{NP}$-hard problem P that uses b bits of advice, but requiring exponential runtime. We may then investigate how many further bits of advice we need to solve P efficiently. A similar idea would be to combine two other fields; for instance, in [71], Engelberg and Naor have introduced *online games*, i. e., problem definitions in which selfish agents work in an online environment. Thus, we lose performance for two reasons. Here, we could study advice that is supplied for both decreasing the price of anarchy and revealing yet unknown parts of the input.

Bibliography

[1] Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theoretical Computer Science*, 234(1–2):203–218, Elsevier Science Publishers, 2000. 11

[2] Divesh Aggarwal and Ueli M. Maurer. Breaking RSA generically is equivalent to factoring. In *Proceedings of the 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2009)*, volume 5479 of *Lecture Notes in Computer Science*, pages 36–53. Springer-Verlag, 2009. 115

[3] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in $\mathcal{P}$. *Annals of Mathematics*, 160(2):781–793, Department of Mathematics, Princeton University, 2004. 115

[4] Sheldon B. Akers. A graphical approach to production scheduling problems. *Operations Research*, 4:244–245, Informs, 1956. 29

[5] Susanne Albers. On the influence of lookahead in competitive paging algorithms. *Algorithmica*, 18(3):283–305, Springer-Verlag, 1997. 8, 12

[6] Susanne Albers. Online algorithms: A survey. *Mathematical Programming*, 97(1):3–26, Springer-Verlag, 2003. 6, 8, 11, 53

[7] Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. The online set cover problem. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC 2003)*, pages 100–105. Association for Computing Machinery, 2003. 67

[8] Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. The online set cover problem. *SIAM Journal on Computing*, 39(2):361–370, Society for Industrial and Applied Mathematics, 2009. 67

[9] Claudia Archetti, Luca Bertazzi, and M. Grazia Speranza. Reoptimizing the traveling salesman problem. *Networks*, 42(3):154–159, Wiley, 2003. 114

[10] Claudia Archetti, Luca Bertazzi, and M. Grazia Speranza. Reoptimizing the 0-1 knapsack problem. *Discrete Applied Mathematics*, 158(17):1879–1887, Elsevier Science Publishers, 2010. 115

[11] Sanjeev Arora and Boaz Barak. *Computational Complexity*. Cambridge University Press, 2009. 112, 115, 116

[12] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and hardness of approximation problems. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science (FOCS 1992)*, pages 14–23. IEEE Computer Society, 1992. 116

[13] Giorgio Ausiello, Bruno Escoffier, Jérôme Monnot, and Vangelis Th. Paschos. Reoptimization of minimum and maximum traveling salesman's tours. *Journal of Discrete Algorithms*, 7(4):453–463, Elsevier Science Publishers, 2009. 114

[14] Baruch Awerbuch, Yair Bartal, Amos Fiat, and Adi Rosén. Competitive non-preemptive call control. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1994)*, pages 312–320. Society for Industrial and Applied Mathematics, 1994. 7, 50

[15] Theodore P. Baker, John Gill, and Robert Solovay. Relativizations of the $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$ question. *SIAM Journal on Computing*, 4(4):431–442, Society for Industrial and Applied Mathematics, 1975. 116

[16] Nikhil Bansal, Niv Buchbinder, Aleksander Mądry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem (extended abstract). In *Proceedings of the 52th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2011)*, pages 267–276. IEEE Computer Society, 2011. 53, 88, 116

[17] Laszlo A. Belady. A study of replacement algorithms for virtual-storage computer. *IBM Systems Journal*, 5(2):78–101, IBM, 1966. 10

[18] Shai Ben-David and Allan Borodin. A new measure for the study of on-line algorithms. *Algorithmica*, 11(1):73–91, Springer-Verlag, 1994. 11, 12

[19] Shai Ben-David, Allan Borodin, Richard M. Karp, Gábor Tardos, and Avi Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, Springer-Verlag, 1994. 8

[20] Tobias Berg and Harald Hempel. Reoptimization of traveling salesperson problems: Changing single edge-weights. In *Proceedings of the 3rd International Conference on Language and Automata Theory and Applications (LATA 2009)*, volume 5457 of *Lecture Notes in Computer Science*, pages 141–151. Springer-Verlag, 2009. 114

[21] Davide Bilò, Hans-Joachim Böckenhauer, Juraj Hromkovič, Richard Královič, Tobias Mömke, Peter Widmayer, and Anna Zych. Reoptimization of Steiner trees. In *Proceedings of the 11th Scandinavian Workshop on Algorithm Theory (SWAT 2008)*, volume 5124 of *Lecture Notes in Computer Science*, pages 258–269. Springer-Verlag, 2008. 114

[22] Davide Bilò, Hans-Joachim Böckenhauer, Dennis Komm, Richard Královič, Tobias Mömke, Sebastian Seibert, and Anna Zych. Reoptimization of the shortest common superstring problem. *Algorithmica*, 61(2):227–251, Springer-Verlag, 2011. 114

[23] Davide Bilò, Peter Widmayer, and Anna Zych. Reoptimization of weighted graph and covering problems. In *Proceedings of the 6th International Workshop on Approximation and Online Algorithms (WAOA 2008)*, volume 5426 of *Lecture Notes in Computer Science*, pages 201–213. Springer-Verlag, 2008. 115

[24] Hans-Joachim Böckenhauer, Luca Forlizzi, Juraj Hromkovič, Joachim Kneis, Joachim Kupke, Guido Proietti, and Peter Widmayer. On the approximability of TSP on local modifications of optimally solved instances. *Algorithmic Operations Research*, 2(2):83–93, Preeminent Academic Facets, 2007. 114

[25] Hans-Joachim Böckenhauer, Karin Freiermuth, Juraj Hromkovič, Tobias Mömke, Andreas Sprock, and Björn Steffen. The Steiner tree reoptimization problem with sharpened triangle inequality. In *Proceedings of the 7th International Conference on Algorithms and Complexity (CIAC 2010)*, volume 6078 of *Lecture Notes in Computer Science*, pages 180–191. Springer-Verlag, 2010. 114

[26] Hans-Joachim Böckenhauer, Juraj Hromkovič, Dennis Komm, Richard Královič, and Peter Rossmanith. On the power of randomness versus advice in online computation. Accepted for publication in *Lecture Notes in Computer Science*, 2012. 2

[27] Hans-Joachim Böckenhauer, Juraj Hromkovič, Richard Královič, Tobias Mömke, and Peter Rossmanith. Reoptimization of Steiner trees: Changing the terminal set. *Theoretical Computer Science*, 410(36):3428–3435, Elsevier Science Publishers, 2009. 114

[28] Hans-Joachim Böckenhauer, Juraj Hromkovič, and Andreas Sprock. Knowing all optimal solutions does not help for TSP reoptimization. In Jozef Kelemen and Alica Kelemenovà, editors, *Computation, Cooperation and Life – Essays Dedicated to Gheorghe Păun on the Occasion of His 60th Birthday*, volume 6610 of *Lecture Notes in Computer Science*, pages 7–15. Springer-Verlag, 2011. 115

[29] Hans-Joachim Böckenhauer, Joachim Kneis, and Joachim Kupke. Approximation hardness of deadline-TSP reoptimization. *Theoretical Computer Science*, 410(21–23):2241–2249, Elsevier Science Publishers, 2009. 114

[30] Hans-Joachim Böckenhauer and Dennis Komm. Reoptimization of the metric deadline TSP. *Journal of Discrete Algorithms*, 8(1):87–100, Elsevier Science Publishers, 2010. 114

[31] Hans-Joachim Böckenhauer, Dennis Komm, Richard Královič, and Peter Rossmanith. On the advice complexity of the knapsack problem. Technical Report 740, Department of Computer Science, ETH Zurich, 2011. 2

[32] Hans-Joachim Böckenhauer, Dennis Komm, Richard Královič, and Peter Rossmanith. On the advice complexity of the knapsack problem. In *Proceedings of the 10th Latin American Symposium on Theoretical Informatics (LATIN 2012)*, Lecture Notes in Computer Science. Springer-Verlag, 2012. To appear. 2

[33] Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, and Richard Královič. On the advice complexity of the k-server problem. Technical Report 703, Department of Computer Science, ETH Zurich, 2010. 2

[34] Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, and Richard Královič. On the advice complexity of the k-server problem. In *Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP 2011)*, volume 6755 of *Lecture Notes in Computer Science*, pages 207–218. Springer-Verlag, 2011. 2, 88, 89, 110

[35] Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, Richard Královič, and Tobias Mömke. On the advice complexity of online problems. In *Proceedings of the 20th International Symposium on Algorithms and Computation (ISAAC 2009)*, volume 5878 of *Lecture Notes in Computer Science*, pages 331–340. Springer-Verlag, 2009. 2, 11, 19, 20, 25, 26, 27, 29, 30, 31, 32, 36, 39, 47, 89, 92

[36] Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, Richard Královič, and Tobias Mömke. Online algorithms with advice. Technical Report 614, Department of Computer Science, ETH Zurich, 2009. 2, 92

[37] Dan Boneh and Ramarathnam Venkatesan. Breaking RSA may not be equivalent to factoring. In *Proceedings of the 17th International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT 1998)*, volume 1403 of *Lecture Notes in Computer Science*, pages 59–71. Springer-Verlag, 1998. 115

[38] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998. 1, 6, 7, 8, 11, 13, 20, 43, 44, 51, 53, 54, 90, 93, 117

[39] Allan Borodin, Sandy Irani, Prabhakar Raghavan, and Baruch Schieber. Competitive paging with locality of reference. *Journal of Computer and System Sciences*, 50(2):244–258, Elsevier Science Publishers, 1995. 11

[40] Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal online algorithm for metrical task systems. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC 1987)*, pages 373–382. Association for Computing Machinery, 1987. 20

[41] Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *Journal of the ACM*, 39(4):745–763, Association for Computing Machinery, 1992. 20

[42] Joan Boyar and Kim S. Larsen. The seat reservation problem. *Algorithmica*, 25(4):403–417, Springer-Verlag, 1999. 13

[43] Joan Boyar, Kim S. Larsen, and Morten N. Nielsen. The accommodating function: A generalization of the competitive ratio. *SIAM Journal on Computing*, 31(1):233–258, Society for Industrial and Applied Mathematics, 2001. 11, 13

[44] Ilja N. Bronstein and Konstantin A. Semendjajev. *Handbook of Mathematics*. Springer-Verlag, 3rd edition, 1997. 60

[45] Peter Brucker. An efficient algorithm for the job-shop problem with two jobs. *Computing*, 40(4):353–359, Springer-Verlag, 1988. 29

[46] Peter Brucker. *Scheduling Algorithms*. Springer-Verlag, 4th edition, 2004. 13, 29

[47] Ioannis Caragiannis, Aleksei V. Fishkin, Christos Kaklamanis, and Evi Papaioannou. Randomized on-line algorithms and lower bounds for computing large independent sets in disk graphs. *Discrete Applied Mathematics*, 155(2):119–136, Elsevier Science Publishers, 2007. 47

[48] Gregory J. Chaitin. On the length of programs for computing finite binary sequences. *Journal of the ACM*, 13(4):547–569, Association for Computing Machinery, 1966. 110

[49] Arkadev Chattopadhyay, Jeff Edmonds, Faith Ellen, and Toniann Pitassi. A little advice can be very helpful. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1994)*, 2012. To appear. 116

[50] Francis Y. L. Chin, Marek Chrobak, Stanley P. Y. Fung, Wojciech Jawor, Jiří Sgall, and Tomás Tichý. Online competitive algorithms for maximizing weighted throughput of unit jobs. *Journal of Discrete Algorithms*, 4(2):255–276, Elsevier Science Publishers, 2006. 6

[51] Andrew Chou, Jeremy Cooperstock, Ran El-Yaniv, Michael Klugerman, and Tom Leighton. The statistical adversary allows optimal money-making trading strategies. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1995)*, pages 467–476. Society for Industrial and Applied Mathematics, 1995. 12

[52] Marek Chrobak, Wojciech Jawor, Jiří Sgall, and Tomás Tichý. Improved online algorithms for buffer management in QoS switches. *ACM Transactions on Algorithms*, 3(4), Association for Computing Machinery, 2007. 6

[53] Marek Chrobak and Lawrence L. Larmore. An optimal on-line algorithm for k-servers on trees. *SIAM Journal on Computing*, 20(1):144–148, Society for Industrial and Applied Mathematics, 1991. 53

[54] Marek Chrobak and John Noga. LRU is better than FIFO. *Algorithmica*, 23(2):180–185, Springer-Verlag, 1999. 11

[55] Vašek Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, Informs, 1979. 67

[56] Alan Cobham. The intrinsic computational difficulty of functions. In *Proceedings of the 2nd International Congress for Logic, Methodology and Philosophy of Science*, pages 24–30. North-Holland, 1965. 5

[57] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC 1971)*, pages 151–158. Association for Computing Machinery, 1971. 5

[58] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009. 5, 29

[59] János Csirik and Gerhard J. Woeginger. Resource augmentation for online bounded space bin packing. *Journal of Algorithms*, 44(2):308–320, Elsevier Science Publishers, 2002. 12, 81

[60] Hans Delfs and Helmut Knebl. *Introduction to Cryptography. Principles and Applications*. Springer-Verlag, 2002. 115

[61] Reinhard Diestel. *Graph Theory*. Springer-Verlag, 4th edition, 2010. 5

[62] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, IEEE Computer Society, 1976. 115

[63] Stefan Dobrev, Rastislav Královič, and Dana Pardubská. How much information about the future is needed? In *Proceedings of the 34th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2008)*, volume 4910 of *Lecture Notes in Computer Science*, pages 247–258. Springer-Verlag, 2008. 1, 13, 14

[64] Stefan Dobrev, Rastislav Královič, and Dana Pardubská. Measuring the problem-relevant information in input. *Theoretical Informatics and Applications (RAIRO)*, 43(3):585–613, EDP Sciences, 2009. 1, 11, 13, 15, 19

[65] Reza Dorrigiv. *Alternative Measures for the Analysis of Online Algorithms*. PhD thesis, University of Waterloo, 2011. 13

[66] Tomáš Ebenlendr and Jiří Sgall. Semi-online preemptive scheduling: One algorithm for all variants. *Theory of Computing Systems*, 48(3):577–613, Springer-Verlag, 2011. 13

[67] Jack Edmonds. Minimum partition of a matroid into independent subsets. *Journal of Research of the National Bureau of Standards*, 69B:67–72, United States Government Printing Office, 1965. 5

[68] Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. Online computation with advice. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP 2009)*, volume 5555 of *Lecture Notes in Computer Science*, pages 427–438. Springer-Verlag, 2009. 19, 20

[69] Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. On the additive constant of the k-server work function algorithm. *Information Processing Letters*, 110(24):1120–1123, Elsevier Science Publishers, 2010. 53

[70] Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. Online computation with advice. *Theoretical Computer Science*, 412(24):2642–2656, Elsevier Science Publishers, 2011. 20, 59, 66

[71] Roee Engelberg and Joseph Naor. Equilibria in online games. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2007)*, pages 149–158. Society for Industrial and Applied Mathematics, 2007. 118

[72] Leah Epstein and Lene M. Favrholdt. Optimal preemptive semi-online scheduling to minimize makespan on two related machines. *Operations Research Letters*, 30(4):269–275, Elsevier Science Publishers, 2002. 13

[73] Uriel Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, Association for Computing Machinery, 1998. 67

[74] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel D. Sleator, and Neal E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, Elsevier Science Publishers, 1991. 11

[75] Amos Fiat and Gerhard J. Woeginger. Competitive odds and ends. In *Online Algorithms, The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*, pages 385–394. Springer-Verlag, 1998. 11, 13

[76] Amos Fiat and Gerhard J. Woeginger, editors. *Online Algorithms, The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998. 1, 6, 8, 13

[77] Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Springer-Verlag, 2010. 113, 114

[78] Michael R. Garey and David S. Johnson. *Computers and Intractability*. Freeman, 1979. 5, 29

[79] Michael R. Garey, David S. Johnson, and Ravi Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, Informs, 1976. 29

[80] Ronald Graham, Donald Knuth, and Oren Patashnik. *Concrete Mathematics*. Addison-Wesley, 2nd edition, 1994. 3, 11

[81] William W. Hardgrave and George L. Nemhauser. A geometric model and a graphical algorithm for a sequencing problem. *Operations Research*, 11(6):889–900, Informs, 1963. 29

[82] David Harel and Yishai Feldman. *Algorithmics: The Spirit of Computing*. Addison-Wesley, 3rd edition, 2004. 5

[83] Johan Håstad. Some optimal inapproximability results. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC 1997)*, pages 1–10. Association for Computing Machinery, 1997. 116

[84] Yong He and Yiwei Jiang. Optimal algorithms for semi-online preemptive scheduling problems on two uniform machines. *Acta Informatica*, 40(5):367–383, Springer-Verlag, 2004. 13

[85] Yong He and Guochuan Zhang. Semi on-line scheduling on two identical machines. *Computing*, 62(3):179–187, Springer-Verlag, 1999. 13

[86] Juraj Hromkovič. *Algorithmics for Hard Problems*. Springer-Verlag, 2nd edition, 2004. 4, 5, 67, 68, 111, 112, 116

[87] Juraj Hromkovič. *Theoretical Computer Science*. Springer-Verlag, 2004. 5, 6, 111

[88] Juraj Hromkovič. *Design and Analysis of Randomized Algorithms*. Springer-Verlag, 2005. 1, 3, 5, 6, 7, 8, 10, 29, 30, 77, 89, 91, 93, 94

[89] Juraj Hromkovič, Rastislav Královič, and Richard Královič. Information complexity of online problems. In *Proceedings of the 35th International Symposium on Mathematical Foundations of Computer Science (MFCS 2010)*, volume 6281 of *Lecture Notes in Computer Science*, pages 24–36. Springer-Verlag, 2010. 14, 110

[90] Juraj Hromkovič, Tobias Mömke, Kathleen Steinhöfel, and Peter Widmayer. Job shop scheduling with unit length tasks: Bounds and algorithms. *Algorithmic Operations Research*, 2(1):1–14, Preeminent Academic Facets, 2007. 29, 30, 35, 38, 39, 40, 89

[91] Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, 22(4):463–468, Association for Computing Machinery, 1975. 77

[92] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. In *Proceedings of the 14th IEEE Conference on Computational Complexity (CCC 1999)*, pages 237–240. IEEE Computer Society, 1999. 113

[93] Sandy Irani and Anna R. Karlin. On online computation. In Dorit S. Hochbaum, editor, *Approximation Algorithms for $\mathcal{NP}$-hard Problems*, chapter 13, pages 521–564. PWS Publishing Company, 1997. 1, 6, 7, 8, 11, 13, 20, 53

[94] Sandy Irani, Anna R. Karlin, and Steven Phillips. Strongly competitive algorithms for paging with locality of reference. *SIAM Journal on Computing*, 25(3):477–497, Society for Industrial and Applied Mathematics, 1996. 11

[95] Kazuo Iwama and Guochuan Zhang. Online knapsack with resource augmentation. *Information Processing Letters*, 110(22):1016–1020, Elsevier Science Publishers, 2010. 12, 81

[96] David S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9(3):256–278, Elsevier Science Publishers, 1974. 67

[97] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS 1995)*, pages 214–221. IEEE Computer Society, 1995. 12

[98] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, Association for Computing Machinery, 2000. 12, 81

[99] Anna R. Karlin, Mark S. Manasse, Lyle A. McGeoch, and Susan Owicki. Competitive randomized algorithms for non-uniform problems. In *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1990)*, pages 301–309. Society for Industrial and Applied Mathematics, 1990. 14, 88

[100] Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):79–119, Springer-Verlag, 1988. 7

[101] Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a Symposium on the Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972. 67, 77

[102] Hans Kellerer, Vladimir Kotov, M. Grazia Speranza, and Zsolt Tuza. Semi on-line algorithms for the partition problem. *Operations Research Letters*, 21(5):235–242, Elsevier Science Publishers, 1997. 13

[103] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*. Springer-Verlag, 2004. 77

[104] Jon Kleinberg and Éva Tardos. *Algorithm Design*. Addison-Wesley, 2005. 5, 10

[105] Andrei N. Kolmogorov. Three approaches to the definition of the concept "quantity of information". *Problemy Peredachi Informatsii*, 1(1):3–11, Russian Academy of Sciences, 1965. In Russian. 110

[106] Dennis Komm, Richard Královič, and Tobias Mömke. On the advice complexity of the set cover problem. In *Proceedings of the 7th International Computer Science Symposium in Russia (CSR 2012)*, Lecture Notes in Computer Science. Springer-Verlag, 2012. To appear. 2

[107] Dennis Komm and Richard Královič. Advice complexity and barely random algorithms. Technical Report 703, Department of Computer Science, ETH Zurich, 2010. 2

[108] Dennis Komm and Richard Královič. Advice complexity and barely random algorithms. In *Proceedings of the 37th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2011)*, volume 6543 of *Lecture Notes in Computer Science*, pages 332–343. Springer-Verlag, 2011. 2

[109] Dennis Komm and Richard Královič. Advice complexity and barely random algorithms. *Theoretical Informatics and Applications (RAIRO)*, 45(2):249–267, EDP Sciences, 2011. 2, 11, 92, 93

[110] Dennis Komm, Richard Královič, and Tobias Mömke. On the advice complexity of the set cover problem. Technical Report 738, Department of Computer Science, ETH Zurich, 2011. 2

[111] Elias Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105–118, Elsevier Science Publishers, 2009. 53

[112] Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *Journal of the ACM*, 42(5):971–983, Association for Computing Machinery, 1995. 53

[113] Elias Koutsoupias and Christos H. Papadimitriou. Worst-case equilibria. In *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS 1999)*, volume 1563 of *Lecture Notes in Computer Science*, pages 404–413. Springer-Verlag, 1999. 112

[114] Elias Koutsoupias and Christos H. Papadimitriou. Beyond competitive analysis. *SIAM Journal on Computing*, 30(1):300–317, Society for Industrial and Applied Mathematics, 2000. 11, 12

[115] Richard Královič and Tobias Mömke. Approximation hardness of the traveling salesman reoptimization problem. In *Proceedings of the 3rd Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS 2007)*, pages 97–104. Masaryk University and Brno University of Technology, 2007. 114

[116] Richard E. Ladner. On the structure of polynomial time reducibility. *Journal of the ACM*, 22(1):155–171, Association for Computing Machinery, 1975. 115

[117] Ming Li and Paul Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, 3rd edition, 2008. 23, 110

[118] Zvi Lotker and Boaz Patt-Shamir. Nearly optimal FIFO buffer management for DiffServ. In *Proceedings of the 21st Annual ACM Symposium on Principles of Distributed Computing (PODC 2002)*, pages 134–142. Association for Computing Machinery, 2002. 14

[119] László Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13(4):383–390, Elsevier Science Publishers, 1975. 67

[120] Mark S. Manasse, Lyle A. McGeoch, and Daniel D. Sleator. Competitive algorithms for on-line problems. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC 1988)*, pages 322–333. Association for Computing Machinery, 1988. 53

[121] Eli Maor. e*: The Story of a Number*. Princeton University Press, 1998. 3

[122] Alberto Marchetti-Spaccamela and Carlo Vercellis. Stochastic on-line knapsack problems. *Mathematical Programming*, 68:73–104, Springer-Verlag, 1995. 78

[123] Ueli M. Maurer. On the oracle complexity of factoring integers. *Computational Complexity*, 5(3/4):237–247, Birkhäuser-Verlag, 1995. 115

[124] Ueli M. Maurer and Stefan Wolf. The relationship between breaking the diffie-hellman protocol and computing discrete logarithms. *SIAM Journal on Computing*, 28(5):1689–1721, Society for Industrial and Applied Mathematics, 1999. 115

[125] Lyle A. McGeoch and Daniel D. Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, Springer-Verlag, 1991. 11

[126] Tobias Mömke. *Algorithmic Approaches for Solving Hard Problems: Approximation and Complexity*. PhD thesis, ETH Zurich, 2009. 29, 89

[127] Tobias Mömke. On the power of randomization for job shop scheduling with k-units length tasks. *Theoretical Informatics and Applications (RAIRO)*, 43:189–207, EDP Sciences, 2009. 29

[128] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995. 3, 10

[129] John F. Nash. Equilibrium points in N-person games. *Proceedings of the National Academy of Sciences of the United States of America*, 36(1):48–49, National Academy of Sciences, 1950. 112

[130] Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay V. Vazirani, editors. *Algorithmic Game Theory*. Cambridge University Press, 2007. 112

[131] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1993. 5, 10, 112, 115

[132] Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC 2010)*, pages 603–610. Association for Computing Machinery, 2010. 116

[133] Cynthia A. Phillips, Clifford Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32(2):163–200, Springer-Verlag, 2002. 12, 81

[134] Michael O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical report, Laboratory for Computer Science, Massachusetts Institute of Technology, 1979. 115

[135] Prabhakar Raghavan. A statistical adversary for on-line algorithms. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 7:79–83, American Mathematical Society, 1991. 12

[136] Nick Reingold, Jeffery Westbrook, and Daniel D. Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11(1):15–32, Springer-Verlag, 1994. 11, 90

[137] Ronald L. Rivest and Adi Shamir. Efficient factoring based on partial information. In *Proceedings of the 4th Workshop on the Theory and Application of Cryptographic Techniques (EUROCRYPT 1985)*, volume 219 of *Lecture Notes in Computer Science*, pages 31–34. Springer-Verlag, 1985. 115

[138] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, Association for Computing Machinery, 1978. 115

[139] Alexander Schrijver. *Combinatorial Optimization*. Springer-Verlag, 2003. 5

[140] Steven S. Seiden, Jiří Sgall, and Gerhard J. Woeginger. Semi-online scheduling with decreasing job sizes. *Operations Research Letters*, 27(5):215–221, Elsevier Science Publishers, 2000. 13

[141] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3,4):379–423, 623–656, Bell Labs, 1948. 110

[142] Gerald S. Shedler and Chien-Yi Tung. Locality in page reference strings. *SIAM Journal on Computing*, 1(3):218–241, Society for Industrial and Applied Mathematics, 1972. 11

[143] Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1996. 5

[144] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, Association for Computing Machinery, 1985. 1, 7, 9, 10

[145] Emanuel Sperner. Ein Satz über Untermengen einer endlichen Menge. *Mathematische Zeitschrift*, 27(1):544–548, Springer-Verlag, 1928. In German. 4

[146] Philip D. Straffin. *Game Theory and Strategy*. Mathematical Association of America Textbooks, 1993. 20, 81, 112

[147] Włodzimierz Szwarc. Solution of the Akers-Friedman scheduling problem. *Operations Research*, 8(6):782–788, Informs, 1960. 29

[148] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, 2nd edition, 2001. 8, 11

[149] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(2):230–265, Oxford University Press, 1936. 5, 111

[150] Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2003. 5, 67, 112

[151] Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, 2nd edition, 2001. 5

[152] Andrew C.-C. Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS 1977)*, pages 222–227. IEEE Computer Society, 1977. 88

[153] Andrew C.-C. Yao. New algorithms for bin packing. *Journal of the ACM*, 27(2):207–227, Association for Computing Machinery, 1980. 7

[154] Neal E. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, Springer-Verlag, 1994. 11

[155] Neal E. Young. Bounding the diffuse adversary. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1998)*, pages 420–425. Society for Industrial and Applied Mathematics, 1998. 12

[156] Neal E. Young. On-line file caching. *Algorithmica*, 33(3):371–383, Springer-Verlag, 2002. 11

[157] Anna Zych and Davide Bilò. New reoptimization techniques employed to Steiner tree problem. *Electronic Notes in Discrete Mathematics*, 37:387–392, Elsevier Science Publishers, 2011. 114