Randomized Approximation and Online Algorithms for Assignment Problems

Dissertation
zur Erlangung des mathematisch-naturwissenschaftlichen Doktorgrades
"Doctor rerum naturalium"
der Georg-August-Universität Göttingen

im Promotionsprogramm Mathematik
der Georg-August University School of Science (GAUSS)

vorgelegt von
Marco Bender
aus Simmern/Hunsrück

Göttingen, 2015

Betreuungsausschuss

Prof. Dr. Stephan Westphal, Institut für Angewandte Stochastik und Operations Research, Technische Universität Clausthal

Prof. Dr. Anita Schöbel, Institut für Numerische und Angewandte Mathematik, Georg-August-Universität Göttingen

Mitglieder der Prüfungskomission

Referent: Prof. Dr. Stephan Westphal, Institut für Angewandte Stochastik und Operations Research, Technische Universität Clausthal

Korreferent: Prof. Dr. Tjark Vredeveld, Department of Quantitative Economics, Maastricht University

Weitere Mitglieder der Prüfungskomission:

Jun.-Prof. Dr. Felix Krahmer, Institut für Numerische und Angewandte Mathematik, Georg-August-Universität Göttingen

PD Dr. Hartje Kriete, Mathematisches Institut, Georg-August-Universität Göttingen

Prof. Dr. Anita Schöbel, Institut für Numerische und Angewandte Mathematik, Georg-August-Universität Göttingen

Prof. Dr. Dominic Schuhmacher, Institut für Mathematische Stochastik, Georg-August-Universität Göttingen

Tag der mündlichen Prüfung: 23.04.2015

Acknowledgments

I would like to thank everyone who contributed to this thesis.

First of all, I am indebted to my supervisor Stephan Westphal for being a fountain of new ideas and his enthusiasm, which helped me never to forget how much fun mathematics is. Anita Schöbel was a great co-supervisor, who gave me good advice and the confidence to be on the right track. I thank Tjark Vredeveld for accepting to be the secondary examiner without any hesitation.

The financial support of the German Research Foundation (DFG) within the Research Training Group 1703 "Resource Efficiency in Interorganizational Networks" enabled me to visit several conferences and workshops and is gratefully acknowledged.

A warm thank-you goes to all my (former) colleagues from the Optimization group for providing such a pleasant environment for both work and leisure time. In particular, I thank Morten Tiedemann for being the best office mate one could wish for.

I enjoyed the many fun moments when working together with Clemens Thielen, and I highly appreciate the help of Sabine Büttner and Sebastian Müller in the proofreading process of this thesis.

Finally, I would like to thank my parents for their never-ending support, and Betty for all the love and making my life more beautiful.

Contents

— 1 —
Introduction

Combinatorial optimization is a branch of mathematics that deals with the question how to determine a best possible solution among a given finite set of alternatives. One of the fundamental problems in this field is the *knapsack problem*: Imagine you want to go on a hiking trip and when packing your knapsack, you realize that it is not possible to pack everything you wanted, the knapsack is simply too heavy. You decide to assign a positive value to each item depending on how important it is for your trip and pack only a set of items that does not weigh more than you can carry such that the sum of the packed items' values is maximized. The knapsack problem is a special case of the *generalized assignment problem*, for which we study several variants throughout this thesis.

Although it is theoretically possible to obtain an optimal solution for a combinatorial optimization problem by enumerating all feasible alternatives, this is, in general, a hopeless endeavor as the number of possible alternatives often exceeds the estimated number of atoms in the known universe. Some combinatorial optimization problems allow for "efficient" algorithms that are able to compute an optimal solution in polynomial time. Despite a lot of research, there is still a large class of problems for which not a single polynomial-time algorithm is known. In fact, the existence of polynomial-time algorithms for these "hard" problems is still an open question (this problem, known as the $\mathcal{P}$-vs-$\mathcal{NP}$ problem, is one of the unsolved Millennium problems). However, it is widely believed that no such algorithms exist.

This motivates the analysis of *approximation algorithms* that yield in polynomial time a solution with a guaranteed performance ratio, i.e., the solution is always within a range of the optimal solution. Our key results in this thesis resort to *randomized algorithms* that use, in contrast to deterministic algorithms, random decisions. One of the advantages of randomized algorithms is that they often yield (in expectation) better approximation factors.

Another issue when dealing with optimization problems is that we rarely have full information: Imagine that after successfully packing your knapsack, you are finally sitting in your car and heading towards the start of your hike.

Unfortunately, you find a mud slide blocking the road. What should you do: wait for a clearing vehicle or choose another route which is hopefully open to traffic? Problems of this kind fall into the field of *online optimization*.

An *online algorithm* has no knowledge about the future and has to take a decision right after a new piece of information is revealed. The quality of online algorithms is measured by *competitive analysis*, which compares the value of the algorithm's solution to the value of the optimal solution achievable if all data would be known beforehand.

For the combinatorial optimization problems analyzed in this thesis, we make use of the mathematical structure inherent in these problems in order to make statements about their computational complexity, design approximation and online algorithms, and obtain results on their quality.

1.1 Preliminaries

In this section we give a short overview of important notations, definitions, and results that we frequently use throughout this thesis.

For an overview on (integer) linear programming we refer to (Chvatal, 1983; Schrijver, 1998; Nemhauser and Wolsey, 1988). An introduction to graph-theoretic concepts can be found in (Krumke and Noltemeier, 2005; Ahuja et al., 1993; Korte and Vygen, 2007).

1.1.1 Complexity and Approximation

An instance of an *optimization problem* is given by a set X of feasible solutions and an objective function $f : X \to \mathbb{R}$. In a maximization problem the task is to find an $x^* \in X$ such that $f(x^*) \geq f(x)$ for all $x \in X$, and for the corresponding *decision problem* we are given a value $B \in \mathbb{R}$ and need to decide whether there exists an $x^* \in X$ such that $f(x^*) \geq B$ or not (for minimization, "$\geq$" is replaced by "$\leq$"). If the set of feasible solutions is finite, i.e., $|X| < \infty$, we speak of combinatorial optimization problems.

In complexity theory, one wants to express the worst-case running time of an algorithm as a function of the "input size". The *encoding length* of an instance of a problem is the number of binary bits that are necessary to encode the instance. We say that an algorithm runs in *polynomial time* if its running time can be bounded for all instances by a polynomial in the encoding length. If we have an algorithm that is only polynomial with respect to a unary encoding of the numeric data, we say that it runs in *pseudo-polynomial time*.

The class $\mathcal{P}$ consists of all decision problems that can be solved in polynomial time. The class $\mathcal{NP}$ contains all decision problems which have for every "yes" instance a certificate that can be verified in polynomial time. A decision problem is called $\mathcal{NP}$*-complete* if it is contained in $\mathcal{NP}$ and any problem in $\mathcal{NP}$ can be reduced to it in polynomial time. We say that an optimization

problem is $\mathcal{NP}$*-hard* if its corresponding decision problem is $\mathcal{NP}$-complete. A special class of $\mathcal{NP}$-hard problems are the *strongly* $\mathcal{NP}$*-hard* problems, which remain $\mathcal{NP}$-hard even if a unary encoding is used for the numeric data.

It is widely believed that $\mathcal{P} \neq \mathcal{NP}$, which would imply that there does not exist a polynomial-time algorithm for any $\mathcal{NP}$-hard problem. Therefore, one is interested in approximations. The following definitions are for the case of maximization problems, which we mostly consider throughout this thesis, but they easily carry over to the minimization case as shown in the last paragraph of this section.

A deterministic polynomial-time algorithm ALG is a β*-approximation algorithm* for an optimization problem (with $0 < \beta \leq 1$) if it yields for every instance I a feasible solution $\text{ALG}(I)$ such that

$$\text{ALG}(I) \geq \beta \cdot \text{OPT}(I), \tag{1.1}$$

where $\text{OPT}(I)$ denotes the optimal solution for instance I. For short, we also write $\text{ALG} \geq \beta \cdot \text{OPT}$.

A special kind of approximation algorithms are the so called *(fully) polynomial time approximation schemes*, which we abbreviate by (F)PTAS. A PTAS yields for every $\epsilon > 0$ and instance I a feasible solution $\text{PTAS}(I, \epsilon)$ with

$$\text{PTAS}(I, \epsilon) \geq (1 - \epsilon) \cdot \text{OPT}(I), \tag{1.2}$$

and the running time is bounded by a polynomial in the encoding length of the problem. For an FPTAS the running time is additionally bounded by a polynomial in $\frac{1}{\epsilon}$. Under the assumption $\mathcal{P} \neq \mathcal{NP}$, a strongly $\mathcal{NP}$-hard problem admits neither an FPTAS nor a pseudo-polynomial algorithm.

In contrast to deterministic algorithms, *randomized algorithms* use random decisions. A randomized polynomial-time algorithm ALG (whose output depends on some random variable X) is a β*-approximation algorithm* if it yields for every instance I a feasible solution $\text{ALG}_X(I)$ such that

$$\mathbb{E}_X\left[\text{ALG}_X(I)\right] \geq \beta \cdot \text{OPT}(I), \tag{1.3}$$

and we often write for short $\mathbb{E}[\text{ALG}] \geq \beta \cdot \text{OPT}$.

For minimization problems, it is common to express the quality of an approximation algorithm ALG by means of a factor $\alpha \geq 1$, and, using the same notation as in (1.1), we say that ALG is an α-approximation algorithm for a minimization problem if it holds for all instances I that

$$\text{ALG}(I) \leq \alpha \cdot \text{OPT}(I).$$

Definitions (1.2) and (1.3) carry over accordingly.

For more details on computational complexity we refer to (Papadimitriou, 1993; Garey and Johnson, 1979). An introduction to approximation algorithms is given in (Vazirani, 2001; Williamson and Shmoys, 2011).

1.1.2 The Generalized Assignment Problem

The *generalized assignment problem* (GAP) is one of the classical combinatorial optimization problems and the starting point for the problems studied in this thesis. It is defined as follows:

Definition 1.1 (GAP)**.**
An instance of GAP *is given by a set of items* $\mathcal{I} = \{1, \dots, n\}$ *and a set of bins* $\mathcal{B} = \{1, \dots, m\}$*. An item* $i \in \mathcal{I}$ *yields a profit of* $p_{ij} \in \mathbb{Z}$ *and has a weight of* $w_{ij} \in \mathbb{Z}_{\geq 0}$ *when assigned to bin* $j \in \mathcal{B}$*. There is a capacity* $B_j \in \mathbb{Z}_{\geq 0}$ *for the maximum amount of weight that can be assigned to each bin* $j \in \mathcal{B}$*.*

The task is to find an assignment of a subset of the items to the bins, i.e., every item is assigned to at most one bin, such that every bin's capacity is respected and the total profit is maximized.

GAP has the following natural integer programming formulation:

$$\max \quad \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{B}} p_{ij} x_{ij} \tag{1.4a}$$

$$\text{s.t.} \quad \sum_{j \in \mathcal{B}} x_{ij} \leq 1 \qquad \forall\, i \in \mathcal{I} \tag{1.4b}$$

$$\sum_{i \in \mathcal{I}} w_{ij} x_{ij} \leq B_j \quad \forall\, j \in \mathcal{B} \tag{1.4c}$$

$$x_{ij} \in \{0, 1\} \qquad \forall\, i \in \mathcal{I}, j \in \mathcal{B}. \tag{1.4d}$$

Here, x_{ij} is one if and only if item i is assigned to bin j, constraints (1.4b) ensure that every item is assigned to at most one bin, and constraints (1.4c) ensure that all bin capacities are respected.

GAP generalizes the *(multiple) knapsack problem* (cf. (Kellerer et al., 2004)), where the items' profits and weights do not depend on the bin they are assigned to, i.e., $p_{ij} = p_i$ and $w_{ij} = w_i$.

The *bin packing problem* (cf. Nemhauser and Wolsey (1988)) can also be seen as a special case of GAP: checking whether a set of items can be packed into a given number of bins is equivalent to solving the corresponding GAP instance with unit profits, i.e., $p_{ij} = 1$.

Previous Work

GAP is a well-studied problem in literature. A comprehensive introduction is given in (Martello and Toth, 1990), a survey of algorithms can be found in (Cattrysse and Van Wassenhove, 1992), and different variants of assignment problems are summarized in (Pentico, 2007).

Cohen et al. (2006) show that any approximation algorithm for the knapsack problem with approximation factor $\frac{1}{\gamma}$, for some $\gamma \geq 1$, can be used to obtain an approximation algorithm for GAP with approximation factor $\frac{1}{1+\gamma}$.

GAP is known to be APX-hard[1] (Chekuri and Khanna, 2006), but there exists a deterministic $\frac{1}{2}$-approximation algorithm (Shmoys and Tardos, 1993; Chekuri and Khanna, 2006). The currently best-known approximation factor of $1 - \frac{1}{e} + \delta$, for some small constant $\delta > 0$, is achieved by a randomized algorithm due to Feige and Vondrák (2006). If the profit of each item does not depend on the bin it is assigned to, i.e., $p_{ij} = p_i$, Nutov et al. (2006) show how the randomized algorithm of Fleischer et al. (2011) can be derandomized using the method of conditional expectations (cf. (Alon and Spencer, 1992)) to obtain a deterministic algorithm with approximation factor $1 - \frac{1}{e}$.

Shmoys and Tardos (1993) study a minimization version of GAP where all items have to be assigned to some bin. Since already the feasibility problem is $\mathcal{NP}$-complete, they provide a dual $(1, 2)$-approximation that yields for every feasible instance a solution that violates the capacity constraints by at most a factor of 2 and the total costs are a lower bound on the optimal solution.

1.1.3 Configuration IPs and Randomized Rounding

Throughout this thesis, we will design approximation algorithms for generalizations of GAP that resort to randomized rounding based on a configuration integer programming formulation. This is a well-known technique (cf., e.g., (Fleischer et al., 2011; Bampis et al., 2013)).

A *configuration* t of bin j is a subset of the items that can be assigned to bin j, and we denote its profit by p_t. We write $T(j)$ for the set of all feasible configurations for bin j, and we denote the disjoint union of all bins' configurations by $T := \dot{\bigcup}_{j \in \mathcal{B}} T(j)$. We then consider the configuration-based integer programming formulation:

$$
\begin{aligned}
\text{(IP)} \quad \max \quad & \sum_{t \in T} p_t x_t && && \text{(1.5a)} \\
\text{s.t.} \quad & \sum_{t \in T(j)} x_t = 1 && \forall\, j \in \mathcal{B} && \text{(1.5b)} \\
& \sum_{t \in T : i \in t} x_t \leq 1 && \forall\, i \in \mathcal{I} && \text{(1.5c)} \\
& x_t \in \{0, 1\} && \forall\, t \in T. && \text{(1.5d)}
\end{aligned}
$$

Here, variable x_t, for $t \in T(j)$, is one if and only if configuration t is selected for bin j. Constraints (1.5b) ensure that one configuration is selected for each bin, while constraints (1.5c) ensure that each item is assigned to at most one bin.

[1] APX contains all problems in $\mathcal{NP}$ that allow for a constant-factor approximation. A problem is APX-hard if there exists a polynomial-time and PTAS-preserving reduction from any problem in APX to this problem (cf. (Crescenzi, 1997)).

The problem GAP is given by defining the set of feasible configurations for bin j as all subsets whose total weight does not exceed the capacity B_j, i.e.,

$$T(j) := \{t \subseteq \mathcal{I} : \sum_{i \in t} w_{ij} \leq B_j\},$$

and setting the profit for a configuration $t \in T(j)$ to the sum of the items' profits, i.e.,

$$p_t := \sum_{i \in t} p_{ij}.$$

Although the number of variables in (IP) is, in general, exponential in the encoding length of the problem, it is sometimes possible to obtain an optimal (or approximate) solution to its linear relaxation in polynomial time. This depends on the underlying optimization problem.

We denote the linear relaxation of (IP) by (LP) and remark that the integrality constraint (1.5d) is relaxed to $x_t \geq 0$ (as $x_t \leq 1$ is already implied by the other constraints).

In the following, we assume that we are given a β-approximation algorithm for solving the linear relaxation (LP), i.e., we can obtain in polynomial time a fractional solution x^{LP} with objective value

$$\sum_{t \in T} p_t x_t^{\mathrm{LP}} \geq \beta \cdot \mathrm{OPT}_{\mathrm{LP}}, \tag{1.6}$$

where $\mathrm{OPT}_{\mathrm{LP}}$ denotes the optimal solution of (LP).

We can then use x^{LP} for a randomized rounding procedure and consider the resulting values x_t^{LP} as the probabilities for using configuration $t \in T(j)$ for bin j. More precisely, we select for every bin j a configuration independently at random, where configuration $t \in T(j)$ is selected with probability x_t^{LP}. Note that, by constraints (1.5b), $(x_t^{\mathrm{LP}})_{t \in T(j)}$ defines a probability distribution for every bin j.

Since we select exactly one configuration for each bin, the resulting vector $x^{\mathrm{IP}} \in \{0,1\}^{|T|}$ obtained by our randomized rounding (where $x_t^{\mathrm{IP}} = 1$ if and only if configuration t is selected) then satisfies constraints (1.5b), i.e., exactly one configuration is chosen for each bin, and we have for the expected profit of the rounded solution that

$$\mathbb{E}\left[\sum_{t \in T} p_t x_t^{\mathrm{IP}}\right] = \sum_{t \in T} p_t \mathbb{E}\left[x_t^{\mathrm{IP}}\right] = \sum_{t \in T} p_t x_t^{\mathrm{LP}} \overset{(1.6)}{\geq} \beta \cdot \mathrm{OPT}_{\mathrm{LP}}. \tag{1.7}$$

However, the solution x^{IP} is, in general, not a feasible solution for (IP) since it may violate constraints (1.5c) (an item might be assigned several times to different bins).

It now depends on the underlying optimization problem if it is possible to design a polynomial-time procedure that obtains, based on x^{IP}, a feasible solution and guarantees a constant approximation factor.

1.1.4 Online Optimization

While classical optimization theory assumes that we are given complete knowledge about the problem instance, online optimization deals with optimization problems in the presence of uncertainty. An *online algorithm* has to take a series of decisions without complete knowledge about the future. New information is revealed piece by piece and an online algorithm has to take a decision immediately after a new piece of information is presented. This means that for a finite sequence $\sigma = (r_1, \ldots, r_n)$ of requests, it has to produce a sequence $(a_1, \ldots, a_n)$ of answers, where answer a_i may only depend on $r_1, \ldots, r_i$.

Competitive analysis, as introduced by Sleator and Tarjan (1985), is a popular measure for the quality of online algorithms. It compares for every instance σ the value $\textsc{alg}(\sigma)$ of an online algorithm ALG with the objective value $\textsc{opt}(\sigma)$ of an *optimal offline algorithm*, which knows the complete sequence σ in advance.

In the following, we consider minimization problems. A deterministic online algorithm ALG is said to be *c-competitive* (with $c \geq 1$) if it holds for all request sequences σ that

$$\textsc{alg}(\sigma) \leq c \cdot \textsc{opt}(\sigma). \tag{1.8}$$

While some authors allow for an additional additive constant, we consider this strict version of competitiveness.

The *competitive ratio* of an online algorithm is defined as the infimum over all c such that the algorithm is c-competitive.

Online optimization can also be seen as a game between an online player and an *adversary*. The adversary presents a sequence of requests which he processes using the optimal offline algorithm and aims at maximizing the competitive ratio, while the online player chooses a strategy that minimizes the worst-case ratio.

A randomized online algorithm RALG is a probability distribution X over the set of deterministic online algorithms. RALG is *c-competitive against an oblivious adversary* if it holds for every request sequence σ that

$$\mathbb{E}_X\left[\textsc{ralg}(\sigma)\right] \leq c \cdot \textsc{opt}(\sigma).$$

Here, the entire request sequence must be constructed by the adversary before any request is processed. There are other adaptive adversary models, but we restrict ourselves in this thesis to the oblivious adversary, which is also the most-studied model in the literature.

Yao's Principle (Yao, 1977) is an important result for proving lower bounds on the competitive ratio of randomized algorithms. It states that it suffices to choose a suitable probability distribution over the set of request sequences such that no *deterministic* online algorithm "performs well" in expectation. Formally, it is given as follows:

Theorem 1.2 (Yao (1977))**.** *If there is a probability distribution $\bar{p}$ over the set of request sequences such that it holds for all deterministic online algorithms* ALG *that*

$$\mathbb{E}_{\bar{p}}\left[\text{ALG}(\sigma)\right] \geq \underline{c} \cdot \mathbb{E}_{\bar{p}}\left[\text{OPT}(\sigma)\right],$$

then $\underline{c}$ is a lower bound on the competitive ratio achievable by any randomized online algorithm.

For maximization problems, we replace (1.8) by $\text{ALG}(\sigma) \geq \frac{1}{c} \cdot \text{OPT}(\sigma)$, and the other definitions carry over accordingly.

More details on online optimization can be found in (Borodin and El-Yaniv, 1998; Fiat and Woeginger, 1998).

1.2 Outline

In Chapter 2 we consider a variant of the *generalized assignment problem* (GAP), where all items have *unit weight*, and we have the additional constraint that the amount of space used in each bin is restricted to be either zero (if the bin is not used) or above a given lower bound, the *minimum quantity* (if the bin is used). It is known that this problem does not admit a PTAS, but no further approximation results were known so far (Krumke and Thielen, 2013).

Initially, we study the special cases of bin-independent and item-independent profits. While the first case can be solved optimally using dynamic programming, a similar formulation for the second case has only pseudo-polynomial running time, and we show that there exists a $(\frac{1}{3} - \epsilon)$-approximation for every $\epsilon > 0$. For the general case, we present a randomized algorithm that is based on a configuration integer programming formulation. We show that the linear relaxation can be solved in polynomial time and yields an appropriate probability distribution for a randomized rounding procedure. While the rounded solution is, in general, infeasible, we show how it can be turned into a feasible solution with high probability by a two-stage procedure while guaranteeing a constant approximation factor: for every $c \geq 2$, our solution is feasible with probability at least $1 - \frac{1}{c}$, and we obtain an approximation factor of $\frac{1}{2c-1} \cdot (1 - \frac{1}{e})$.

In Chapter 3 we consider a generalization of GAP. We relax the hard constraints for the bin capacities and introduce for every bin a cost function that is convex in the total load on this bin. These costs are subtracted from the profits of assigned items, and the task is to find an assignment maximizing the resulting net profit. This setting generalizes the work of (Barman et al., 2012; Antoniadis et al., 2013), where a knapsack problem with additional convex costs is studied.

We show that even restricted cases of our problem remain strongly $\mathcal{NP}$-hard and identify two cases that can be solved in polynomial time. Further-

more, we present a $(1 - \frac{1}{e})$-approximation algorithm for the general case that resorts to a randomized rounding procedure. In order to turn the rounded solution into a feasible solution, we define appropriate estimators that linearize the convex costs.

In Chapter 4 we consider a variant of the *separable assignment problem* (SAP). As in GAP one wants to find a maximum-profit assignment of a subset of the items to the bins. However, SAP can model more general packing constraints of the bins. In the setting we study, every item is allowed to be assigned at most k times to different bins. We show that for this problem a randomized $((1 - \frac{1}{e^k})\beta)$-approximation algorithm exists whenever the single bin subproblem admits a β-approximation. This generalizes the result known for the case $k = 1$ (Fleischer et al., 2011). Finally, we show that it is possible to derandomize this algorithm if the profits of the items do not depend on the bin they are assigned to.

In Chapter 5 we consider the problem of scheduling intervals on identical machines, where each interval can be seen as a job with fixed start and end time. The goal is to accept a maximum cardinality subset of the given intervals and assign these intervals to the machines subject to the constraint that no two intervals assigned to the same machine overlap. This problem is a special case of SAP.

We analyze a novel online version of this problem where, initially, a set of potential intervals and an upper bound k on the number of failing intervals is given. If an interval fails, it can neither be accepted by the online algorithm nor by the adversary. An online algorithm learns that an interval fails at the time when it is supposed to be started. If a non-failing interval is accepted, it cannot be aborted and must be processed non-preemptively until completion. For a single machine, we prove a lower bound of k on the competitive ratio for deterministic algorithms and present a $(k + 1)$-competitive online algorithm. We show that even randomization does not help to obtain a competitive ratio better than $\Omega(\log k)$ and give a randomized $(\log(k+2))$-competitive algorithm for laminar sets of intervals. Moreover, we show how this algorithm can be generalized to multiple machines, where we also obtain a competitive ratio of $\log(k + 2)$ for the laminar case.

In Chapter 6 we consider the *k-Canadian traveller problem* (k-CTP), which asks for a shortest path between two nodes s and t in an undirected graph. However, up to k edges may be blocked, and an online algorithm learns that an edge fails when reaching one of its endpoint.

It is known that no randomized online algorithm for k-CTP can achieve a competitive ratio better than $k + 1$, even on graphs where all paths are node-disjoint (Westphal, 2008). We show that this bound is tight by constructing a randomized online algorithm for this case that achieves a competitive ratio of $k + 1$ against an oblivious adversary and is therefore best possible. This is the first result on randomized algorithms for k-CTP. Furthermore, we con-

sider a setting of the k-CTP where blocked edges might become reopened after some time that is unknown to the online algorithm. We present a $(2k + 3)$-competitive algorithm for this problem and show extensions to the traveling salesman problem with such a recoverable blockage scenario.

1.3 Credits

Most of the results presented in Chapter 2 were obtained jointly with Clemens Thielen and Stephan Westphal and have been published in the *Proceedings of the 38th International Symposium on Mathematical Foundations of Computer Science (MFCS 2013)* (Bender et al., 2013a). An error in the first version of this paper was pointed out to us by Alexander Souza and corrected in the accompanying erratum (Bender et al., 2013b).

The idea to use linear estimators for a randomized rounding procedure in Chapter 3 has been developed together with Stephan Westphal, and parts of this chapter appeared in the *Proceedings of the Third International Symposium on Combinatorial Optimization (ISCO 2014)* (Bender and Westphal, 2014).

Chapter 4 contains material from joint work with Clemens Thielen and Stephan Westphal that has been published in *Information Processing Letters* (Bender et al., 2015b).

Initial ideas for Chapter 5 were obtained in cooperation with Clemens Thielen and Stephan Westphal, and parts of this chapter have been submitted for publication (Bender et al., 2015a).

The analysis of the randomized online algorithm presented in Chapter 6 has been developed together with Stephan Westphal and has been published in the *Journal of Combinatorial Optimization* (Bender and Westphal, 2013).

— 2 —

The Generalized Assignment Problem with Minimum Quantities and Unit Weight Items

Recently, Krumke and Thielen (2013) introduced the *generalized assignment problem with minimum quantities* (GAP-MQ), which is a generalization of GAP where the total weight of items assigned to each bin must be either zero (if the bin is not used) or above a given lower bound, the *minimum quantity* (if the bin is used). This additional restriction can be motivated from many practical applications. Consider, e.g., power plants that will always produce a base load of energy if they run that cannot be undercut.

In this chapter we consider the special case of GAP-MQ where all items have unit weight ($w_{ij} = 1$), which we denote by GAP-MQ-UNIT. This can be motivated, e.g., by the assignment of students to seminars (cf. (Krumke and Thielen, 2013)): A number of students want to participate in a seminar and declare their preferences for all seminars that are possibly offered. The students correspond to the items in our setting and the seminars to the bins. As every student occupies exactly one spot in a seminar if he is assigned, all items have unit weight. The goal is to assign students to seminars such that their total satisfaction is maximized, where the students' preferences are modeled by the profits. Naturally, every course has an upper bound on the number of participants, e.g., due to the room size or the number of available time slots for presentations. Furthermore, a seminar can only take place if it has at least a minimum number of participants.

From a theoretical point of view, GAP-MQ-UNIT is also interesting to study. While it is known that GAP-MQ-UNIT does not allow for a PTAS, there are no further results on the approximability of this problem.

Previous Work

The *generalized assignment problem with minimum quantities* (GAP-MQ) was introduced by Krumke and Thielen (2013). They show that the general version of GAP-MQ (with arbitrary weights) does not admit a polynomial-time approximation algorithm unless $\mathcal{P} = \mathcal{NP}$, but there exists a dual (1,2)-approximation (which yields a solution that violates the minimum quantity and capacity con-

straints at most by a factor of 2 and the total profit is an upper bound on the optimal solution).

There are a number of other problems involving minimum quantities that have been researched in literature.

The *minimum-cost flow problem* has been generalized by adding minimum quantity constraints for each arc (Seedig, 2011; Krumke and Thielen, 2011; Zhu et al., 2011), i.e., the flow on an arc has to be either zero or above the minimum quantity. This problem does not admit a polynomial-time approximation algorithm unless $\mathcal{P} = \mathcal{NP}$, even on series-parallel graphs (Krumke and Thielen, 2012). The *maximum flow problem* with minimum quantities was studied in (Thielen and Westphal, 2013). In general, it does not allow for a polynomial-time approximation algorithm unless $\mathcal{P} = \mathcal{NP}$, but there exists a $(\frac{\lambda}{2\lambda-1})$-approximation if all arcs have the same minimum quantity λ.

In (Assmann, 1983; Assmann et al., 1984) the *bin covering problem* with minimum quantity constraints is analyzed. In this problem, one is given a set of items, each with a size independent of the bin it is assigned to, and the task is to assign all items to bins such that the number of bins used is maximized. However, if a bin is used, it has to be filled at least to some threshold T. The best approximation algorithm presented in (Assmann et al., 1984) guarantees an asymptotic ratio of $\frac{3}{4}$. This result was improved in the following by Csirik et al. (2001) and Jansen and Solis-Oba (2003) to obtain an asymptotic PTAS and an asymptotic FPTAS, respectively.

Chapter Outline

In this chapter we consider GAP with additional minimum quantity restrictions for the bins and unit weight items. A formal definition of the problem GAP-MQ-UNIT, some initial observations, and important previous results can be found in Section 2.1. We first analyze two special cases: If the profits are bin-independent ($p_{ij} = p_i$), the problem can be solved in polynomial time using dynamic programming as shown in Section 2.2. In Section 2.3 we see that, although a similar dynamic programming formulation can be used to solve the case of item-independent profits ($p_{ij} = p_j$), its running time is only pseudo-polynomial. This problem is $\mathcal{NP}$-hard, and we give, for every $\epsilon > 0$, a $(\frac{1}{3} - \epsilon)$-approximation. We complement these results by showing that this variant becomes polynomially solvable if, in addition, all bins have the same capacity and the same minimum quantity ($B_j = B, q_j = q$).

The case of arbitrary profits is analyzed in Section 2.4. We present a randomized approximation algorithm that uses a configuration-based integer program. The linear relaxation of this program is shown to be polynomially solvable and a fractional solution can be used as a probability distribution for a randomized rounding procedure. We show how the rounded solution can be turned into a feasible solution with high probability by a two-stage procedure while guaranteeing a constant approximation factor.

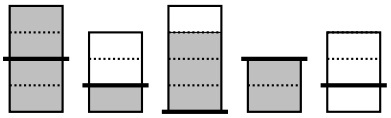


Figure 2.1: Illustration of a feasible solution for GAP-MQ-UNIT. The thick lines correspond to the minimum quantities.

2.1 Problem Definition and Preliminaries

Formally, the *generalized assignment problem with minimum quantities and unit weight items* (GAP-MQ-UNIT) is defined as follows:

Definition 2.1 (GAP-MQ-UNIT)**.**
An instance of GAP-MQ-UNIT *is given by a set of items* $\mathcal{I} = \{1, \dots, n\}$ *and a set of bins* $\mathcal{B} = \{1, \dots, m\}$*, where every bin* $j \in \mathcal{B}$ *has a minimum quantity* $q_j \in \mathbb{Z}_{\geq 0}$ *and a capacity* $B_j \in \mathbb{Z}_{>0}$ *with* $q_j \leq B_j$*. If item* $i \in \mathcal{I}$ *is assigned to bin* $j \in \mathcal{B}$*, it yields a profit of* $p_{ij} \in \mathbb{Z}_{\geq 0}$*.*

The task is to find a feasible assignment of a subset of the items to the bins, i.e., every item is assigned to at most one bin, and the number of items in each bin $j \in \mathcal{B}$ *is either zero (if bin* j *is not used) or at least* q_j *and at most* B_j *(if bin* j *is used), such that the total profit is maximized.*

An illustration of a feasible assignment of an instance of GAP-MQ-UNIT is given in Figure 2.1.

Since there are in total n items, solving an instance of GAP-MQ-UNIT where $B_j > n$ for some bin j is equivalent to solving the instance with $B_j = n$. We summarize this observation for later reference in the following:

Observation 2.2. *For all instances of* GAP-MQ-UNIT *it holds, without loss of generality, that* $B_j \leq n$ *for all* $j \in \mathcal{B}$*.*

If we drop the minimum quantity condition (by setting $q_j = 0$ for all $j \in \mathcal{B}$), the problem can be solved in polynomial time. This can, e.g., be seen by considering the integer programming formulation (1.4) for GAP: if $w_{ij} = 1$, the constraint matrix is totally unimodular and the right-hand side is integer, and, thus, it suffices to solve the linear relaxation to obtain an integer solution (Hoffman and Kruskal, 1956).

Based on formulation (1.4), the most natural integer programming formulation for GAP-MQ-UNIT is given by introducing an additional variable y_j for each bin j (that attains the value one if and only if bin j is used) as follows:

$$\max \quad \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{B}} p_{ij}x_{ij} \tag{2.1a}$$

$$\text{s.t.} \quad \sum_{j\in\mathcal{B}} x_{ij} \leq 1 \qquad \forall\, i \in \mathcal{I} \tag{2.1b}$$

$$q_j y_j \leq \sum_{i\in\mathcal{I}} x_{ij} \leq B_j y_j \quad \forall\, j \in \mathcal{B} \tag{2.1c}$$

$$x_{ij} \in \{0,1\} \qquad \forall\, i \in \mathcal{I}, j \in \mathcal{B} \tag{2.1d}$$

$$y_j \in \{0,1\} \qquad \forall\, j \in \mathcal{B}. \tag{2.1e}$$

Here, constraints (2.1c) ensure that if a bin is used, it satisfies both the minimum quantity and the capacity condition.

Although this formulation might be sufficient for many practical purposes, its integrality gap can become arbitrary large, as the following result shows. Thus, it is not applicable for the design of LP-based approximation algorithms, and we will make use of another formulation in Section 2.4.

Proposition 2.3. *The integer programming formulation* (2.1) *has an unbounded integrality gap.*

Proof. Consider an instance of GAP-MQ-UNIT consisting of n items and n bins with $q_j = B_j = n$ for all bins j and profits given by

$$p_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{else.} \end{cases}$$

In an optimal solution of (2.1) all items have to be assigned to one bin in order to ensure feasibility. In this case, only one item contributes to the profit and we have $\text{OPT} = 1$.

In the linear relaxation of (2.1), constraints (2.1d) and (2.1e) are replaced by $0 \leq x_{ij} \leq 1$ and $0 \leq y_j \leq 1$, respectively. In this case, we can choose $y_j = \frac{1}{n}$ for all j, and $x_{ij} = 1$ for $i = j$ (and $x_{ij} = 0$, otherwise), and we obtain a feasible solution with total profit $\text{OPT}_{\text{LP}} = n$. For $n \to \infty$, the claim follows. □

In (Krumke and Thielen, 2013) it was shown by a gap-preserving reduction from *3-bounded 3-dimensional matching* that there exists a constant $\epsilon_0 > 0$ such that it is strongly $\mathcal{NP}$-hard to approximate GAP-MQ-UNIT within a factor smaller than $(1 + \epsilon_0)$ (even if all profits p_{ij} are in $\{0, 1\}$ and the minimum quantities and bin capacities of all bins are fixed to three). In particular, under the assumption $\mathcal{P} \neq \mathcal{NP}$, there exists no PTAS. Furthermore, even the case of unit profits, i.e., $p_{ij} = 1$, was shown to be $\mathcal{NP}$-hard. This implies that the problem of deciding whether all items can be assigned to the bins is $\mathcal{NP}$-complete. We summarize these results in the following:

Theorem 2.4 (Krumke and Thielen (2013))**.** GAP-MQ-UNIT *...*

(i) ... does not admit a PTAS, unless $\mathcal{P} = \mathcal{NP}$*.*

(ii) ... is $\mathcal{NP}$*-hard for unit profits* ($p_{ij} = 1$)*.*

Remark 2.5. *Note that the encoding length of* GAP-MQ-UNIT *depends on the kind of instance we consider:*

For an arbitrary instance of GAP-MQ-UNIT*, the encoding length is a polynomial in* n *and* m *as there are* n *different items and* m *different bins for which profits, minimum quantities, and capacities need to be stored.*

If we assume, as in the second part of Theorem 2.4, that all items are identical, i.e., $p_{ij} = 1$*, it suffices to store the number of items. The encoding length is therefore a polynomial in* $\log n$ *and* m*.*

2.2 Bin-Independent Profits ($p_{ij} = p_i$)

In this section we consider the case where the profit of assigning item i to bin j does not depend on the bin, i.e., $p_{ij} = p_i$. We show that this problem can be solved in polynomial time using dynamic programming.

For $k \in \{1, \dots, m\}$ and $l \in \{0, \dots, n\}$, we define

$$f_k(l) = \begin{cases} 1, & \text{if there exists a subset } S \subseteq \{1, \dots, k\} \text{ of the first } k \text{ bins} \\ & \text{such that exactly } l \text{ items can be assigned to bins } S \\ 0, & \text{else.} \end{cases}$$

In order to obtain an optimal solution, we need to determine a maximal $l^* \in \{0, \dots, n\}$ such that $f_m(l^*) = 1$, i.e., we want to find the maximum number of items that can feasibly be assigned to some subset of the bins. As we assume that the profit of an item does not depend on the bin it is assigned to, the total profit is then given by the sum of the l^* most profitable items.

Next, we deal with the question how to compute the values $f_k(l)$.

For every $l \in \{0, \dots, n\}$, we can easily compute $f_1(l)$ by checking whether l items can be assigned to the first bin and set

$$f_1(l) = \begin{cases} 1, & \text{if } l = 0 \text{ or } q_1 \leq l \leq B_1 \\ 0, & \text{else.} \end{cases}$$

In order to compute $f_{k+1}(l)$, we use a recursive argument and assume that we have previously computed $f_k(i)$ for all values $i \in \{0, \dots, n\}$.

There are two possible cases, when we have $f_{k+1}(l) = 1$: In the first case, we have $f_k(l) = 1$, i.e., l items can already be assigned to some subset of the first k bins and no item needs to be assigned to bin $k+1$. In the second case, we have $f_k(l - l') = 1$ for some $l' \leq l$ with $q_{k+1} \leq l' \leq B_{k+1}$. Then, $l - l'$ items

can be assigned to some of the first k bins, and the remaining l' items can be assigned to bin $k+1$.

Otherwise, l items cannot be assigned to bins $1, \dots, k+1$, i.e., $f_{k+1}(l) = 0$. Thus, we can compute $f_{k+1}(l)$ using the following rule:

$$f_{k+1}(l) = \begin{cases} 1, & \text{if } f_k(l) = 1 \text{ or} \\ & \quad f_k(l - l') = 1 \text{ for some } l' \leq l \text{ with } q_{k+1} \leq l' \leq B_{k+1} \\ 0, & \text{else.} \end{cases}$$

Since the bin capacities are at most n (cf. Observation 2.2), determining $f_{k+1}(l)$ given $f_k(i)$ for all $i \in \{0, \dots, l\}$ takes at most $\mathcal{O}(n)$. Thus, the total running time of this dynamic program is $\mathcal{O}(mn^2)$.

Note that the encoding length of an instance of GAP-MQ-UNIT with $p_{ij} = p_i$ is polynomial in n and m (cf. Remark 2.5). Thus, we obtain the following result:

Theorem 2.6. *For bin-independent profits ($p_{ij} = p_i$),* GAP-MQ-UNIT *can be solved in polynomial time using dynamic programming.*

So far, we were only concerned with computing the maximum total profit that can be achieved. If we are interested in the underlying assignment, we need to keep track of the number of items that are assigned to each bin throughout the procedure. Recall that it does not make a difference to which bin an item is assigned.

In the following, let $S_k(l)$ be a k-dimensional vector, where the j-th entry contains the number of items assigned to bin $j \in \{1, \dots, k\}$ in the solution corresponding to $f_k(l)$. Initially, we have for $l \in \{1, \dots, n\}$,

$$S_1(l) = \begin{cases} l, & \text{if } q_1 \leq l \leq B_1 \\ 0, & \text{else,} \end{cases}$$

and recursively we can set

$$S_{k+1}(l) = \begin{cases} (S_k(l), 0), & \text{if } f_k(l) = 1 \\ (S_k(l - l'), l'), & \text{if } f_k(l - l') = 1 \text{ for some } l' \text{ with } q_j \leq l' \leq B_j \\ (0, \dots, 0), & \text{else.} \end{cases}$$

2.3 Item-Independent Profits ($p_{ij} = p_j$)

In this section we consider the case that the profit of assigning item i to bin j does not depend on the item, i.e., $p_{ij} = p_j$.

Note that, similar to Remark 2.5, the encoding length of this variant is no longer polynomial in n. As it suffices to store the number of items, it is a polynomial in $\log n$, and it follows by the second part of Theorem 2.4 that this variant is $\mathcal{NP}$-hard:

Corollary 2.7. GAP-MQ-UNIT *with item-independent profits ($p_{ij} = p_j$) is $\mathcal{NP}$-hard.*

2.3.1 Dynamic Program

By Corollary 2.7 we know that there does not exist a polynomial-time algorithm unless $\mathcal{P} = \mathcal{NP}$. However, we can use a dynamic program similar to the one from Section 2.3 to solve this case in pseudo-polynomial time.

For $k \in \{1, \dots, m\}$ and $l \in \{0, \dots, n\}$ we define $\pi_k(l)$ as the maximum profit that can be achieved if up to l items can be assigned to some subset of the first k bins.

For $k = 1$, we can easily check whether l items can be assigned to the first bin. In this case, every item contributes p_1 to the total profit, and we compute

$$\pi_1(l) = \begin{cases} l\,p_1, & \text{if } q_1 \leq l \leq B_1 \\ 0, & \text{else.} \end{cases}$$

Recursively, we can compute $\pi_{k+1}(l)$ if we are given the values $\pi_k(i)$ for $i \in \{0, \dots, l\}$. We make use of the fact that the profits are item-independent, and check how many items should be assigned to bin $k+1$, where every item yields a profit of p_{k+1}. Thus, we have

$$\pi_{k+1}(l) = \max \left\{ \pi_k(l), \max_{\substack{l' \in \{q_{k+1}, \dots, B_{k+1}\}: \\ l' \leq l}} \left\{ \pi_k(l - l') + l'\,p_{k+1} \right\} \right\}. \tag{2.2}$$

The optimal solution is then given by $\pi_m(n)$, and we can obtain the underlying optimal assignment analogously to Section 2.2. Since we can again assume that the bin capacities are at most n (cf. Observation 2.2), determining the maximum in (2.2) takes at most $\mathcal{O}(n \log n)$. Thus, the total running time of the dynamic program is $\mathcal{O}(mn^2 \log n)$, and we obtain the following result:

Theorem 2.8. *For item-independent profits ($p_{ij} = p_j$),* GAP-MQ-UNIT *can be solved in pseudo-polynomial time using dynamic programming.*

2.3.2 Approximation Algorithm

In this section we design an approximation algorithm for GAP-MQ-UNIT with $p_{ij} = p_j$.

A Note on the Knapsack Problem

First, we show a property of the *knapsack problem*, which will turn out to be useful later on. It relates the optimal solution of a knapsack instance to the total profit that could be achieved if more capacity was available.

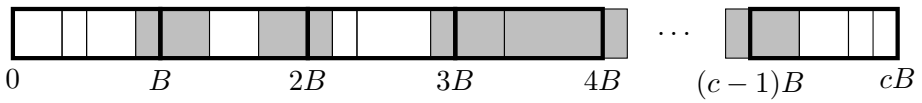


Figure 2.2: Illustration for the proof of Lemma 2.9: a fractional assignment of the items to c knapsacks of capacity B each, where fractionally assigned items are shown in grey.

Lemma 2.9. *Consider an instance of the knapsack problem (with items $\mathcal{I}$, profits p_i and weights w_i for all $i \in \mathcal{I}$, and knapsack capacity B). Let $\mathcal{I}' \subseteq \mathcal{I}$ be a subset of the items such that $\sum_{i\in\mathcal{I}'} w_i \leq cB$ for some $c \in \mathbb{Z}_{>0}$. Then, there exists a solution for the knapsack problem yielding a profit of at least $\frac{1}{2c-1}\sum_{i\in\mathcal{I}'} p_i$.*

Proof. As the total weight of all items in $\mathcal{I}'$ is at most cB, we can assign all these items fractionally to at most c knapsacks of capacity B each, where there are at most $c-1$ items that are fractionally assigned. This is illustrated in Figure 2.2.

Note that we can assume that there is no item with weight exceeding B, i.e., $w_i \leq B$ for all $i \in \mathcal{I}$ (otherwise, we could simply discard i from the set of items as it can never be packed). We can then remove all fractionally assigned items from the c knapsacks used so far, and put each of them into its own (additional) knapsack, which yields an integral assignment of all items to at most $2c-1$ knapsacks (of capacity B each).

Since all items in $\mathcal{I}'$ together have a profit of $\sum_{i\in\mathcal{I}'} p_i$, this implies that the items in the most profitable one among these $2c-1$ knapsacks have a profit of at least $\frac{1}{2c-1}\sum_{i\in\mathcal{I}'} p_i$, and the claim follows. □

Observe that the proof of Lemma 2.9 is constructive in the sense that it yields a set of items with the desired property. Even computing an optimal solution for the knapsack instance (which is possible in polynomial time by dynamic programming if the bin capacity is bounded by a polynomial in the encoding length of the problem, cf. (Kellerer et al., 2004)) does not yield a better bound in general, as the following example shows.

Example 2.10. *Choose some integer $c \leq \frac{\lfloor B/2\rfloor+1}{2}$, and consider a knapsack instance with $2c-1$ items of weight $\lfloor B/2\rfloor + 1$ and profit 1, and knapsack capacity B.*

Since $\lfloor B/2\rfloor + 1 > B/2$, at most one item can be packed into a knapsack of capacity B. However, the total weight of all items is

$$(2c-1)\left(\lfloor B/2\rfloor + 1\right) \leq cB + 2c - \left(\lfloor B/2\rfloor + 1\right) \leq cB,$$

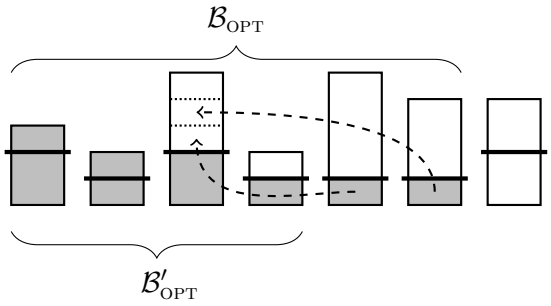


Figure 2.3: Illustration for the proof of Lemma 2.11.

and, thus, even an optimal solution for the knapsack problem can achieve only a factor $\frac{1}{2c-1}$ *of the total profit of all items.*

Approximation Algorithm for $p_{ij} = p_j$

Next, we bound the total capacity of the bins used in an optimal solution. This will turn out to be useful in the analysis later on.

Lemma 2.11. *If* $p_{ij} = p_i$*, there exists an optimal solution where the total capacity of all bins that are used is at most* $2n$*.*

Proof. Let $\mathcal{B}_{\text{OPT}} \subseteq \mathcal{B}$ be the set of bins that are used by OPT and assume that their total capacity exceeds $2n$, i.e., $\sum_{j \in \mathcal{B}_{\text{OPT}}} B_j > 2n$.

We now start removing bins from $\mathcal{B}_{\text{OPT}}$ in non-decreasing order of profits p_j until we remain with a subset $\mathcal{B}'_{\text{OPT}} \subseteq \mathcal{B}_{\text{OPT}}$ with $\sum_{j \in \mathcal{B}'_{\text{OPT}}} B_j \leq 2n$. By Observation 2.2 it follows that $\sum_{j \in \mathcal{B}'_{\text{OPT}}} B_j \geq n$, i.e., $\mathcal{B}'_{\text{OPT}}$ offers enough space to accommodate all items.

Recall that the profits are item-independent, i.e., there is a profit p_j associated with each bin that is gained for every item that is assigned to bin j. Thus, moving the items that have previously been assigned to $\mathcal{B}_{\text{OPT}} \setminus \mathcal{B}'_{\text{OPT}}$ to $\mathcal{B}'_{\text{OPT}}$ yields a feasible solution without decreasing the profit. This is illustrated in Figure 2.3. □

Our approximation algorithm is based on solving a particular instance of the knapsack problem, which we denote by KP:

The items of KP correspond to the bins of our problem, and the profit of item (bin) j is defined as $\overline{p_j} := p_j B_j$, its weight is $\overline{w_j} := B_j$, and the knapsack capacity is set to $\overline{B} := n$. Choosing an item j in KP then corresponds to opening bin j and filling it to its capacity B_j with items.

We can then solve this instance approximately by applying the well-known FPTAS for the knapsack problem (cf. (Kellerer et al., 2004)), and we obtain,

for every $\epsilon > 0$, in time polynomial in the encoding length of the problem and $\frac{1}{\epsilon}$ a solution $\mathcal{B}' \subseteq \mathcal{B}$ such that

$$\sum_{j \in \mathcal{B}'} B_j = \sum_{j \in \mathcal{B}'} \overline{w_j} \leq \overline{B} = n, \tag{2.3}$$

$$\sum_{j \in \mathcal{B}'} p_j B_j = \sum_{j \in \mathcal{B}'} \overline{p_j} \geq (1 - \epsilon) \cdot \text{OPT}_{\text{KP}}, \tag{2.4}$$

where OPT_{KP} denotes the optimal solution of KP.

If we fill, in our instance of GAP-MQ-UNIT, the bins $\mathcal{B}'$ to their capacity, all minimum quantities and capacities are trivially respected. By (2.3) we assign at most n items in total, i.e., we have a feasible assignment. Furthermore, (2.4) yields a lower bound on the total profit of this assignment, which we will use in the following.

Next, we show that an optimal solution for KP with knapsack capacity set to $\overline{B} = 2n$ (which we denote by $\text{OPT}_{\text{KP}}^{2n}$) yields an upper bound on the optimal solution OPT of our problem GAP-MQ-UNIT with $p_{ij} = p_j$.

Lemma 2.12. *It holds that* $\text{OPT}_{\text{KP}}^{2n} \geq \text{OPT}$.

Proof. By Lemma 2.11 we know that we can assume that OPT uses bins of total capacity at most $2n$. Hence, the set of bins that are used by OPT is also a feasible solution for our problem KP if we set the capacity to $\overline{B} = 2n$, and the claim follows. □

We are now ready to show the performance guarantee of our algorithm, which is the main result of this section.

Theorem 2.13. *For* GAP-MQ-UNIT *with item-independent profits ($p_{ij} = p_j$), there exists, for every $\epsilon > 0$, a $(\frac{1}{3} - \epsilon)$-approximation with running time polynomial in the encoding length of the problem and $\frac{1}{\epsilon}$.*

Proof. If we apply Lemma 2.9 with $c = 2$ to KP, it follows that

$$\text{OPT}_{\text{KP}} \geq \frac{1}{3} \cdot \text{OPT}_{\text{KP}}^{2n}, \tag{2.5}$$

where OPT_{KP} and $\text{OPT}_{\text{KP}}^{2n}$ denote, as before, the optimal solutions for KP with $\overline{B} = n$ and $\overline{B} = 2n$, respectively.

Combining (2.4), (2.5), and Lemma 2.12, we see that our solution $\mathcal{B}'$ yields a total profit of

$$\sum_{j \in \mathcal{B}'} p_j B_j \geq (1 - \epsilon) \cdot \text{OPT}_{\text{KP}} \geq \frac{1}{3} \cdot (1 - \epsilon) \cdot \text{OPT}_{\text{KP}}^{2n} \geq \left(\frac{1}{3} - \epsilon\right) \cdot \text{OPT}.$$

□

2.3.3 Identical Minimum Quantities and Capacities

If we assume, in addition to $p_{ij} = p_j$, that for all bins minimum quantities and capacities are the same, i.e., $q_j = q$ and $B_j = B$ for some q and B, we can compute an optimal solution in polynomial time as we will show below. In the following, we assume that the bins are sorted in non-increasing order of profits $p_1 \geq \cdots \geq p_m$.

As all bins are identical, the $\lceil n/B \rceil$ most profitable bins have a total capacity of $\lceil n/B \rceil \cdot B \geq n$, i.e., they can accommodate all n items. In fact, an optimal solution only uses, without loss of generality, (a subset of) theses bins: Assume an additional bin $\lceil n/B \rceil + 1$ was used in an optimal solution. Then we could increase the profit by moving the items assigned to this bin to bins $1, \ldots, \lceil n/B \rceil$. We summarize this in the following:

Observation 2.14. *When determining an optimal solution, it suffices to consider the $\lceil n/B \rceil$ most profitable bins.*

If $n/B \in \mathbb{Z}$, filling all bins to their capacity B is obviously optimal. Now, let $n/B \notin \mathbb{Z}$. For this case, we can obtain an optimal solution by the following procedure: We fill bins $1, \ldots, \lfloor n/B \rfloor$ to their capacity B. Then, there are $l := n - B \cdot \lfloor n/B \rfloor < B$ remaining items that are not assigned, and we distinguish the following cases:

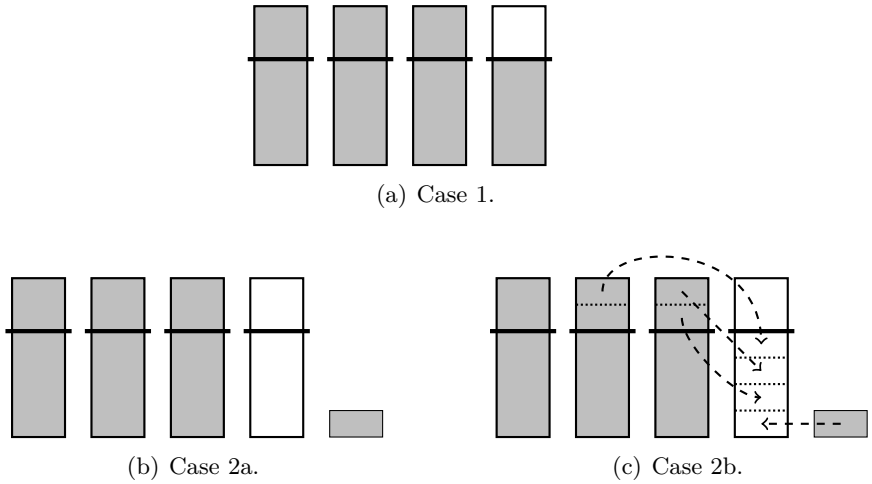
Case 1: If $l \geq q$, we fill bin $\lceil n/B \rceil$ with all remaining $q \leq l \leq B$ items and obtain an optimal solution. This is illustrated in Figure 2.4(a).

Case 2: If $l < q$, the remaining items alone do not suffice to fill bin $\lceil n/B \rceil$ to its minimum quantity. The question of interest is thus whether to use this bin or not, and we choose the better of the following two alternatives:

a) If bin $\lceil n/B \rceil$ is not used, the l remaining items are dropped and do not contribute to the total profit. This is illustrated in Figure 2.4(b).

b) If bin $\lceil n/B \rceil$ is used, it is filled to its minimum quantity q by "shifting" items from bins $1, \ldots, \lfloor n/B \rfloor$. There are $B - q$ items that can possibly be shifted from any of the previous bins, where bins are considered in non-decreasing order of profits. This is illustrated in Figure 2.4(c).

   Note that moving more items does not make sense as it cannot be optimal to close a bin in favor of opening an identical bin which yields only less profit per item. By the same argument it holds in this case that the last bin $\lceil n/B \rceil$ is never assigned more than q items.



(a) Case 1.

(b) Case 2a.

(c) Case 2b.

Figure 2.4: Illustration of the different cases in the optimal solution for identical minimum quantities and capacities. The thick lines correspond to the minimum quantity.

Hence, we obtain the following result:

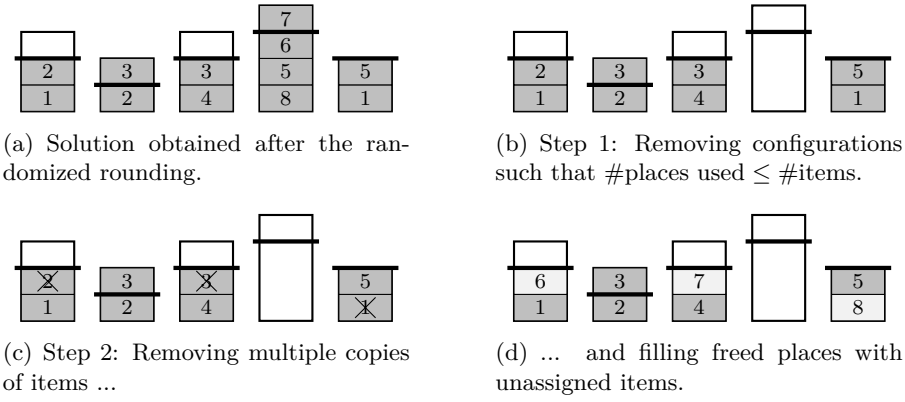
Theorem 2.15. *For item-independent profits ($p_{ij} = p_j$) and identical minimum quantities ($q_j = q$) and capacities ($B_j = B$),* GAP-MQ-UNIT *can be solved in polynomial time.*

2.4 Approximation Algorithm for the General Case

In this section we present an approximation algorithm for the case of arbitrary profits p_{ij}. The algorithm resorts to randomized rounding based on a configuration integer programming formulation as described in Section 1.1.3. Before we present our algorithm in detail, we give a brief overview of the different steps of our procedure and its analysis. For an illustration of the algorithm see Figure 2.5.

In Section 2.4.1 we show that, although our integer programming formulation has an exponential number of variables, its linear relaxation can be solved in polynomial time by column generation. We then use a solution to the linear relaxation as a probability distribution for our algorithm that selects for each bin a configuration independently at random.

The set of chosen configurations will, in general, not correspond to a feasible solution. Hence, in order to obtain a feasible integral solution, we apply a clean-up procedure that works in two steps as described in Section 2.4.2: In the first step, we discard a subset of the bins opened in order to ensure that



(a) Solution obtained after the randomized rounding.

(b) Step 1: Removing configurations such that #places used ≤ #items.

(c) Step 2: Removing multiple copies of items ...

(d) ... and filling freed places with unassigned items.

Figure 2.5: Illustration of the randomized rounding algorithm.

the overall occupied space (i.e., the total number of places used in the bins) is at most n. In the second step, we then replace remaining multiply assigned items in the solution by unassigned items in order to obtain a feasible integral solution. Our solution is feasible with high probability and we have a constant approximation factor.

2.4.1 Randomized Rounding Procedure

GAP-MQ-UNIT can be modeled as a configuration integer program (cf. (1.5)). We therefore define the set of feasible configurations for bin j as the union of the empty set and all subsets of the items with cardinality at least q_j and at most B_j, i.e.,

$$T(j) := \{t \subseteq \mathcal{I} : q_j \leq |t| \leq B_j\} \cup \{\emptyset\}.$$

As before, $T := \dot{\bigcup}_{j \in \mathcal{B}} T(j)$ denotes the disjoint union of all bins' configurations. If we set the profit of a configuration $t \in T(j)$ to

$$p_t := \sum_{i \in t} p_{ij},$$

GAP-MQ-UNIT is then given as follows:

$$\begin{aligned}
\text{(IP)} \quad \max \quad & \sum_{t \in T} p_t x_t && && \text{(2.6a)}\\
\text{s.t.} \quad & \sum_{t \in T(j)} x_t = 1 && \forall\, j \in \mathcal{B} && \text{(2.6b)}\\
& \sum_{t \in T : i \in t} x_t \leq 1 && \forall\, i \in \mathcal{I} && \text{(2.6c)}\\
& x_t \in \{0,1\} && \forall\, t \in T. && \text{(2.6d)}
\end{aligned}$$

We denote by (LP) the linear relaxation of (IP) (recall that (2.6d) is relaxed to $x_t \geq 0$).

Solving the Linear Relaxation

We now show that, even though the number of variables in (IP) is, in general, exponential in the encoding length of the instance of GAP-MQ-UNIT, we can solve (LP) in polynomial time.

To this end, it suffices to show that we can find a column of (LP), i.e., a configuration, with maximum reduced costs in polynomial time (this problem is also called the *pricing problem*). This follows, e.g., by results of (Mehlhorn and Ziegelmann, 2000; Minoux, 1987), and is based on the observation that a linear program can be solved in polynomial time if and only if its separation problem can be solved in polynomial time (cf. (Grötschel et al., 1988)): If we consider the dual program to (LP), which is given by

$$\begin{aligned}
\text{(DLP)} \quad \min \quad & \sum_{j \in \mathcal{B}} y_j + \sum_{i \in \mathcal{I}} z_i && && \text{(2.7a)}\\
\text{s.t.} \quad & y_j + \sum_{i \in t} z_i \geq p_t && \forall\, t \in T(j)\ \forall\, j \in \mathcal{B} && \text{(2.7b)}\\
& z_i \geq 0 && \forall\, i \in \mathcal{I}, && \text{(2.7c)}
\end{aligned}$$

the separation problem of (DLP) asks for some given y and z (with $z \geq 0$), whether (2.7b) holds for all $t \in T(j)$ and for all $j \in \mathcal{B}$, or if there exists a bin $j' \in \mathcal{B}$ and a configuration $t' \in T(j')$ such that $y_{j'} + \sum_{i \in t'} z_i < p_{t'}$. This problem can be solved by determining a column of (LP) with maximum reduced costs, i.e., by solving

$$\max_{j \in \mathcal{B}} \max_{t \in T(j)} p_t - y_j - \sum_{i \in t} z_i. \tag{2.8}$$

If this value is larger than 0, we have found a violating constraint, and otherwise, (2.7b) holds for all $t \in T(j)$ and for all $j \in \mathcal{B}$. Hence, if we can solve the pricing problem in polynomial time, we can also solve (DLP) and by duality theory also (LP) in polynomial time, and we have:

Theorem 2.16. *If the pricing problem of (LP) can be solved in polynomial time, then (LP) can be solved in polynomial time.*

In the following, we describe how the pricing problem (2.8) can be solved in our setting in polynomial time.

For each bin $j \in \mathcal{B}$, finding a configuration $t \in T(j)$ of maximum reduced costs is equivalent to solving

$$\max_{t \in T(j)} \sum_{i \in t} \underbrace{(p_{ij} - z_i)}_{=:p'_i}, \tag{2.9}$$

which is a knapsack problem with n unit-weight items, knapsack capacity B_j, profit $p'_i := p_{ij} - z_i$ for item i, and an additional minimum quantity constraint q_j.

This single bin version of GAP-MQ-UNIT can be solved by greedily selecting the item i with maximum value p'_i until we have either selected B_j items, or the next item i satisfies $p'_i < 0$. If this procedure returns an infeasible configuration with less than q_j items, we continue selecting the item i with maximum value p'_i (which is from now on always negative) until we have selected exactly q_j items. If the resulting total profit is non-negative, we are done. Otherwise, the empty configuration is optimal. This is summarized in Algorithm 2.1.

Algorithm 2.1 Greedy algorithm for GAP-MQ-UNIT with $|\mathcal{B}| = 1$

1: Sort the items in non-decreasing order of profits $p'_1 \geq \cdots \geq p'_n$ and initialize $\mathcal{A} := \emptyset$, $i := 1$.
2: **while** $|\mathcal{A}| < B_j$ **do**
3: **if** $p'_i < 0$ and $|\mathcal{A}| \geq q_j$ **then**
4: **if** $\sum_{a \in \mathcal{A}} p_a < 0$ **then**
5: $\mathcal{A} := \emptyset$
6: **end if**
7: **break**
8: **else**
9: $\mathcal{A} := \mathcal{A} \cup \{i\}$, $i := i + 1$
10: **end if**
11: **end while**
12: **return** $\mathcal{A}$

Lemma 2.17. *For a single bin ($|\mathcal{B}| = 1$), GAP-MQ-UNIT can be solved in polynomial time by Algorithm 2.1.*

Proof. Let $\mathcal{A}$ and $\mathcal{O}$ be the sets of items accepted by ALG and OPT, respectively.

If $\mathcal{O} = \emptyset$, the profit of any feasible subset of the items is non-positive. In particular, the profit of the q_j best items is non-positive, and the statement in step 4 is satisfied for $|\mathcal{A}| = q_j$, i.e., ALG sets $\mathcal{A} = \emptyset$ in step 5 and returns this solution.

If $\mathcal{O} \neq \emptyset$, we know that OPT accepts the $|\mathcal{O}| \in \{q_j, \ldots, B_j\}$ most profitable items with $\sum_{i \in \mathcal{O}} p_i \geq 0$, and accepting more items would violate the bin capacity B_j or decrease the total profit. ALG also accepts the items $\mathcal{O}$ and terminates afterwards. □

In order to determine a column of (LP) with maximum reduced costs, we solve the pricing problem (2.9) for every bin j, which yields one configuration for each bin. If we choose out of these configurations the one which maximizes $-y_j + (2.9)$, we obtain an optimal solution to the pricing problem (2.8). By Theorem 2.16 and Lemma 2.17 we thus obtain:

Theorem 2.18. *(LP) can be solved in time polynomial in the encoding length of* GAP-MQ-UNIT.

The Randomized Rounding Step

We can use an optimal fractional solution $x^{\mathrm{LP}} \in [0,1]^{|T|}$ of (LP) to perform a randomized rounding as described in Section 1.1.3: We choose for each bin j a configuration independently at random, where a configuration $t \in T(j)$ is chosen with probability x_t^{LP}. The resulting vector $x^{\mathrm{IP}} \in \{0,1\}^{|T|}$ (where $x_t^{\mathrm{IP}} = 1$ if and only if configuration t is chosen) then satisfies constraints (2.6b), and for the expected profit of the rounded solution it holds (as shown in (1.7)), that

$$\mathbb{E}\left[\textsc{profit}(x^{\mathrm{IP}})\right] = \mathbb{E}\left[\sum_{t\in T} p_t x_t^{\mathrm{IP}}\right] = \sum_{t\in T} p_t \mathbb{E}\left[x_t^{\mathrm{IP}}\right] = \sum_{t\in T} p_t x_t^{\mathrm{LP}} = \textsc{opt}_{\mathrm{LP}},$$

where we denote by $\textsc{opt}_{\mathrm{LP}}$ the optimal solution of (LP) and write for short $\textsc{profit}(x) := \sum_{t\in T} p_t x_t$. Hence, we have the following:

Observation 2.19. *The vector $x^{\mathrm{IP}} \in \{0,1\}^{|T|}$ obtained from the randomized rounding process satisfies constraints* (2.6b) *and $\mathbb{E}[\textsc{profit}(x^{\mathrm{IP}})] = \textsc{opt}_{\mathrm{LP}}$.*

Observe that x^{IP} is, in general, not a feasible solution to (IP) since it may violate constraints (2.6c) (an item might be assigned several times to different bins). In particular, the total number of items assigned to bins in the solution x^{IP} may be larger than n (when counted with multiplicities). We will come back to this issue in the next section.

2.4.2 Obtaining a Feasible Solution

As we have seen, the solution x^{IP} is, in general, not feasible for (IP), and we now show how we can turn it into a feasible solution (with high probability) while only decreasing the expected profit by a constant factor.

Our procedure works in two steps: In the first step, we discard (in some cases) a subset of the bins opened in x^{IP} in order to ensure that the total number of places used in the bins is at most n. Formally, the *number of places used* by x^{IP} in bin j is defined as $\sum_{t\in T(j)} |t| \cdot x_t^{\mathrm{IP}}$, i.e., items that are assigned to multiple bins are also counted multiple times. In the second step, we then replace all remaining multiply assigned items in the solution by unassigned items in order to respect the minimum quantity constraints and, thus, obtain a feasible integral solution.

First Step

We start by describing the first step of the procedure where we discard a subset of the bins. In the following, let $c \geq 2$ be an arbitrary but fixed integer. We distinguish two cases:

Case 1: If the number of places used in x^{IP} is at most cn, we can obtain a subset of the configurations of x^{IP} such that the total number of places used is at most n, and the expected profit is at least $\frac{1}{2c-1} \cdot \textsc{profit}(x^{\mathrm{IP}})$ (the proof is given below).

Case 2: If the number of places used in x^{IP} exceeds cn, we discard none of the configurations and leave the solution unchanged (and infeasible).

The lower bound for the profit in Case 1 can be seen by the following argument:

Given the vector $x^{\mathrm{IP}} \in \{0,1\}^{|T|}$ obtained from the randomized rounding process, we consider the following instance of the knapsack problem: The objects that can be packed into the knapsack are the configurations $t \in T$ with $x_t^{\mathrm{IP}} = 1$, i.e., the configurations selected by x^{IP}. The weight of object t is the number $|t|$ of items contained in configuration t, and its profit is the profit p_t of configuration t. The knapsack capacity is set to n.

As we assume in Case 1 that the total number of places used in the bins in x^{IP} is at most cn, the sum of the weights of all objects in the knapsack instance is at most cn. Thus, it follows by Lemma 2.9 that there exists a set of objects, i.e., a subset of the configurations in x^{IP}, with total weight at most n and profit at least a fraction $\frac{1}{2c-1}$ of the sum of all objects' profits. This computation can be performed in polynomial time based on the idea of the proof of Lemma 2.9:

We therefore place all the objects $t \in T$ with $x_t^{\mathrm{IP}} = 1$ (in a random order) in a large knapsack of capacity cn. As described before, we can then find an integral assignment of all items to at most $2c-1$ knapsacks of capacity n each (cf. Figure 2.2). Since the sum of all items' profits is $\textsc{profit}(x^{\mathrm{IP}})$, choosing one of these $2c-1$ knapsacks uniformly at random then yields an expected profit of at least $\frac{1}{2c-1} \cdot \textsc{profit}(x^{\mathrm{IP}})$.

In Case 2, the solution remains unchanged. Thus, together with Observation 2.19 we have:

Observation 2.20. *The solution obtained after the first clean-up step has expected profit at least* $\frac{1}{2c-1} \cdot \mathrm{OPT}_{\mathrm{LP}}$.

Second Step

After the first step, there might be still some items that are assigned to multiple bins. Therefore, we remove in the second step each multiply assigned item from all bins but the one where it yields the highest profit. The following analysis shows that we lose at most a factor of $(1 - \frac{1}{e})$ in the total profit by this removal process.

We fix an item i and denote by y_{ij} the probability that item i is assigned to bin j after the first clean-up step. We assume, without loss of generality, that

the bins are sorted by non-increasing profit of item i, i.e., $p_{i1} \geq p_{i2} \geq \cdots \geq p_{im}$. Hence, item i is assigned to bin 1 in the second step with probability y_{i1} and it yields a profit of p_{i1}.

Observe that the bins' configurations are initially chosen independently at random and the first clean-up step does not cause any dependencies. The events "item i is assigned to bin j after the first clean-up step" are therefore independent for $j = 1, \ldots, m$, and the probability that item i is assigned to bin 2 in the second step is given by $(1 - y_{i1})\, y_{i2}$. In this case, it contributes p_{i2} to the total profit. By applying the same argumentation for the remaining bins, we see that the expected profit obtained from item i is then given as

$$\begin{aligned} & y_{i1}p_{i1} + (1 - y_{i1})y_{i2}p_{i2} + \cdots + \prod_{j=1}^{m-1} (1 - y_{ij})y_{im}p_{im} \\ \geq\ & \left(1 - \left(1 - \frac{1}{m}\right)^m\right) \sum_{j=1}^{m} p_{ij}y_{ij} \\ \geq\ & \left(1 - \frac{1}{e}\right) \sum_{j=1}^{m} p_{ij}y_{ij}. \end{aligned}$$

Here, we used the arithmetic-geometric mean inequality and the fact that $(1 - \frac{1}{m})^m \leq \frac{1}{e}$ for all $m \geq 1$ (cf. (Goemans and Williamson, 1994)).

Since the total expected profit is the sum of the expected profits obtained from each item, we can perform this procedure separately for every item, and altogether, we obtain a solution with expected profit at least

$$\begin{aligned} \sum_{i \in \mathcal{I}} \left(1 - \frac{1}{e}\right) \sum_{j=1}^{m} p_{ij}y_{ij} &= \left(1 - \frac{1}{e}\right) \sum_{i \in \mathcal{I}} \sum_{j=1}^{m} p_{ij}y_{ij}. \\ &\geq \frac{1}{2c - 1} \left(1 - \frac{1}{e}\right) \operatorname{OPT}_{\mathrm{LP}}, \end{aligned}$$

where the last inequality follows from Observation 2.20.

After the removal of multiply assigned items, some of the bins that are opened may not be filled to their minimum quantities anymore, i.e., our solution is not feasible.

In Case 1, we constructed our solution such that the total number of places used after the first step was no more than the number n of available items. Hence, for each item i that is assigned to $l \geq 2$ bins, there must be $l - 1$ items that were not assigned to any bin after the first step. Thus, we can refill the $l - 1$ places vacated by deleting copies of item i with items that were previously unassigned. Doing this for all multiply assigned items yields a feasible solution that respects the minimum quantities of the bins.

Observe that the procedure of filling freed places does not decrease the total profit since we assume that all profits p_{ij} are non-negative.

Observation 2.21. *The solution obtained after the second clean-up step has expected profit at least* $\frac{1}{2c-1} \cdot \left(1 - \frac{1}{e}\right) \cdot \text{OPT}_{\text{LP}}$ *and is feasible in Case 1.*

Observe that this last step is, in general, not possible in Case 2, and our algorithm will end up with an infeasible solution. However, we can bound the probability of this "bad" event. Therefore, we make use of *Markov's inequality* (cf. (Mitzenmacher and Upfal, 2005)), which is a well-known estimate for the tail probability:

Theorem 2.22 (Markov's inequality)**.** *If* X *is a non-negative random variable, then it holds for all* $a > 0$ *that*

$$\Pr(X \geq a) \leq \frac{\mathbb{E}[X]}{a}.$$

In order to apply Theorem 2.22 to our problem, we define the random variable X as the number of places used in x^{IP} for which we have

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{t \in T} |t| \cdot x_t^{\text{IP}}\right] = \sum_{t \in T} |t| \cdot \mathbb{E}\left[x_t^{\text{IP}}\right] = \sum_{t \in T} |t| \cdot x_t^{\text{LP}}$$
$$= \sum_{i \in \mathcal{I}} \sum_{t \in T : i \in t} x_t^{\text{LP}} \overset{(2.6c)}{\leq} n. \quad (2.10)$$

If we set $a := cn$, it thus follows that

$$\Pr(X > cn) \leq \Pr(X \geq cn) \leq \frac{\mathbb{E}[X]}{cn} \overset{(2.10)}{\leq} \frac{n}{cn} = \frac{1}{c}. \quad (2.11)$$

Hence, our algorithm outputs a feasible solution with probability at least $1 - \frac{1}{c}$. Together with Observation 2.21 and the fact that $\text{OPT}_{\text{LP}} \geq \text{OPT}$ we obtain:

Theorem 2.23. *For every* $c \geq 2$*, the randomized rounding algorithm yields an expected profit of at least* $\frac{1}{2c-1} \cdot \left(1 - \frac{1}{e}\right) \cdot \text{OPT}$ *and outputs a feasible solution with probability at least* $1 - \frac{1}{c}$.

As an alternative to Markov's inequality, estimates for the tail probabilities can also be obtained using *Chernoff-Hoeffding bounds* (cf. Mitzenmacher and Upfal (2005)). These bounds are often stronger in certain settings, and there are estimates for different variants that have in common that the random variable of interest X is required to be the sum of independent random variables $X_1, \ldots, X_m$.

If we define for our problem for every bin j the random variable X_j as the number of places used in bin j (in the solution x^{IP} from the randomized rounding), we have that the total number of places used in x^{IP} is $X = \sum_{j=1}^{m} X_j$. Note that $X_1, \ldots, X_m$ are independent as the configurations are chosen independently for each bin.

While the classical Chernoff bound assumes that we have independent 0-1-random variables, this is not the case in our setting, and we make use of the extension to bounded random variables due to Hoeffding (1963):

Theorem 2.24 (Hoeffding's inequality)**.** *Let $X_1, \dots, X_m$ be independent random variables with $X_j \in [a_j, b_j]$ for all $j \in \{1, \dots, m\}$, and $X := \sum_{j=1}^{m} X_j$. Then,*

$$\Pr(X - \mathbb{E}[X] \geq \alpha) \leq e^{\frac{-2\alpha^2}{\sum_{j=1}^{m}(b_j - a_j)^2}}.$$

As every bin's capacity is at most n (cf. Observation 2.2), we can apply Theorem 2.24 with $a_j := 0$, $b_j := n$, and $\alpha := (c-1)n$, and obtain

$$\begin{aligned} \Pr(X \geq cn) &= \Pr(X - n \geq (c-1)n) \\ &\overset{(2.10)}{\leq} \Pr(X - \mathbb{E}[X] \geq (c-1)n) \\ &\leq e^{\frac{-2(c-1)^2 n^2}{mn^2}} = e^{\frac{-2(c-1)^2}{m}} =: H_m(c). \end{aligned} \tag{2.12}$$

As we can see, the bound $H_m(c)$ obtained in (2.12) depends on m, and it holds for fixed c that $H_m(c) \to 1$ for $m \to \infty$, i.e., for large m the tail bound becomes weak. However, for a fixed m, the exponential decrease of H_m ensures that there exists a value $c'(m)$ such that $H_m(c) \leq \frac{1}{c}$ for all $c \geq c'(m)$ as illustrated in Figure 2.6. This means that there are cases where (2.12) yields sharper estimates for our problem than the Markov bound (2.11).

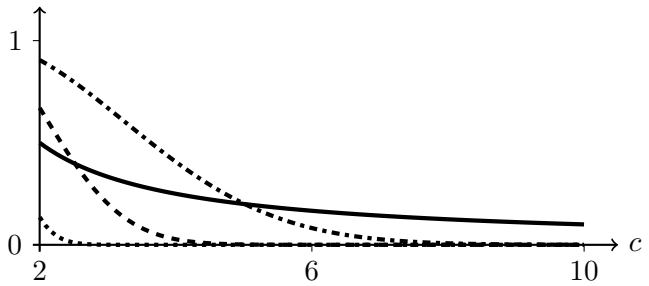


Figure 2.6: Illustration of the estimates for the tail probabilities as a function of c obtained from Markov's inequality (2.11) (solid line) and Hoeffding's inequality (2.12) for $m = 1$ (dotted line), $m = 5$ (dashed line), and $m = 20$ (dash-dotted line).

2.5 Conclusions

In this chapter we considered the problem GAP with unit weight items and an additional minimum quantity constraint for the bins. We first analyzed the

two special cases of item-independent and bin-independent profits. For the general case of arbitrary profits, we presented a randomized approximation algorithm.

The performance guarantee of our algorithm does, however, depend on a value c, which can be interpreted as the trade-off between high probability for a feasible solution and good approximation factor. If we want to ensure that we obtain a solution that is *always* feasible, we need to choose $c = m$ (every bin's capacity is at most n, and, thus, there can also be at most mn places to be occupied by the algorithm after the randomized rounding). The approximation factor we derived in Theorem 2.23 depends in this case on the value m. It thus remains an open question whether there exists a (randomized) approximation algorithm with constant approximation factor that always outputs a feasible solution.

In contrast to most related problems in literature (cf. (Fleischer et al., 2011)), the sets of feasible configurations of our problem are not an independence system, i.e., after removing an item from a bin, the packing for this bin might not be feasible anymore. As we have seen, this makes the procedure of obtaining a feasible solution from the rounded solution more difficult. In the context of assigning students to seminars one could think of other problems with this property: if we wish to arrange groups that are in some sense balanced (e.g., with respect to previous performance or gender of the participants), not every subset of a feasible group is again feasible. It would be interesting to study further problems of this kind.

— 3 —

The Generalized Assignment Problem with Convex Costs

In this chapter we consider a natural generalization of GAP by dropping the "hard" constraints for the bin capacities and introducing for every bin j a cost function c_j that depends on the total weight of items assigned to this bin. Then, we define the net profit of a solution as the sum over all profits for assigned items minus the costs incurred by the bins. We focus our attention on the class of convex cost functions and call the resulting problem *generalized assignment problem with convex costs* (GAP-CC).

The motivation for considering convex cost functions stems from scheduling problems that are concerned with energy-efficient computing environments, which is known as *speed scaling* (cf. (Albers, 2010)). If the workload on a processor increases, it has to run at a higher speed and, thus, its energy consumption increases. This dependence follows approximately the *cube-root rule*, which states that the energy consumption increases cubic in the speed.

Most optimization problems that have been researched in this field so far have "hard" constraints, e.g., minimizing the total energy consumption such that all jobs are completed, or minimizing the flow time given a fixed energy budget (cf. (Albers, 2010)). Pruhs and Stein (2010) note that it would be of more practical relevance to consider an objective function that takes both the rewards that are earned for a job and a convex cost function depending on the current workload into account. This would, e.g., allow to model the situation of the operator of a large data center that has to decide whether to accept or reject requests for computation times on his servers. In our setting, the requests correspond to the items and the servers to the bins.

Previous Work

Special cases of GAP-CC have been subject to recent research. Barman et al. (2012) consider the case $|\mathcal{B}| = 1$ and provide a $\frac{1}{3}$-approximation algorithm that first sorts the items in non-increasing order of value-to-weight ratio and then applies a greedy procedure. Furthermore, they analyze the problem under different additional feasibility constraints and show that the related

online problems (where the items arrive over time and an immediate decision whether to accept or reject the item has to be made without knowledge about future items) allow for competitive ratios that are only a constant factor worse than for the corresponding problems without the convex cost function.

Antoniadis et al. (2013) improve upon the results in (Barman et al., 2012) and show how the ideas of the $\frac{1}{3}$-approximation algorithm can be modified and analyzed to guarantee an approximation factor of $\frac{1}{2}$. Furthermore, they propose a dynamic program and show how this can be scaled in order to obtain an FPTAS. For the case of concave cost functions, they show that the problem can be solved optimally in polynomial time by a greedy procedure.

Chapter Outline

In this chapter we consider a generalization of the problem GAP where we drop the capacity constraints for the bins and add convex cost functions to the objective function that depend on the load on each bin. A formal definition of this problem GAP-CC and some preliminaries can be found in Section 3.1. In Section 3.2 we consider the single bin version of GAP-CC, before we deal in the remainder of this chapter with the case of multiple bins. We show strong $\mathcal{NP}$-hardness for restricted cases in Section 3.3, and we identify two polynomially solvable cases in Section 3.4. For the general case we present in Section 3.5 an approximation algorithm based on randomized rounding of a configuration integer program. In order to turn the rounded solution into a feasible solution, we define appropriate estimators that linearize the convex costs.

3.1 Problem Definition and Preliminaries

The *generalized assignment problem with convex costs* (GAP-CC) is formally defined as follows:

Definition 3.1 (GAP-CC)**.**
An instance of GAP-CC *is given by a set of items* $\mathcal{I} = \{1, \dots, n\}$ *and a set of bins* $\mathcal{B} = \{1, \dots, m\}$*. If item* $i \in \mathcal{I}$ *is assigned to bin* $j \in \mathcal{B}$*, it yields a profit of* $p_{ij} \in \mathbb{Z}_{\geq 0}$ *and increases the load of bin* j *by the weight* $w_{ij} \in \mathbb{Z}_{>0}$*. For each bin* j *there is a convex, non-decreasing cost function* $c_j \colon \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ *with* $c_j(0) = 0$*, and we assume that* c_j *can be evaluated in constant time. For pairwise disjoint subsets* $I_1, \dots, I_m \subseteq \mathcal{I}$*, the assignment of items* $I_1, \dots, I_m$ *to bins* $1, \dots, m$*, respectively, yields a* total net profit *of*

$$\pi(I_1, \dots, I_m) := \sum_{j \in \mathcal{B}} \left(\sum_{i \in I_j} p_{ij} - c_j \left(\sum_{i \in I_j} w_{ij} \right) \right).$$

The task is to find an assignment of a subset of the items to the bins, i.e., every item is assigned to at most one bin, such that the total net profit is maximized.

This can be summarized in the following *non-linear* integer program:

$$\max \quad \sum_{j\in\mathcal{B}}\left(\sum_{i\in\mathcal{I}} p_{ij}x_{ij} - c_j\left(\sum_{i\in\mathcal{I}} w_{ij}x_{ij}\right)\right) \tag{3.1a}$$

$$\text{s.t.} \quad \sum_{j\in\mathcal{B}} x_{ij} \leq 1 \qquad \forall\, i \in \mathcal{I} \tag{3.1b}$$

$$x_{ij} \in \{0,1\} \qquad \forall\, i \in \mathcal{I}, j \in \mathcal{B}. \tag{3.1c}$$

In Definition 3.1 we require the cost functions c_j to be convex. If we allow for arbitrary non-convex cost functions, the following result shows that we cannot hope for the existence of constant-factor approximation algorithms:

Theorem 3.2 (Antoniadis et al. (2013))**.** *For* GAP-CC *with arbitrary (non-convex) cost functions, there does not exist a constant-factor approximation algorithm (even for a single bin), unless* $\mathcal{P} = \mathcal{NP}$.

Note that GAP-CC contains GAP as a special case, e.g., by choosing the cost function for bin j as

$$c_j(w) = \begin{cases} 0, & \text{if } w \leq B_j \\ M_j(w - B_j), & \text{else,} \end{cases} \tag{3.2}$$

as illustrated in Figure 3.1. Here, the constant $M_j := \sum_{i\in\mathcal{I}} p_{ij} + 1$ ensures that in an optimal solution the load on bin j is no more than the bin capacity B_j in GAP. Hence, we have even for piecewise-linear cost functions the following:

Observation 3.3. GAP-CC *contains* GAP *as a special case.*

Note that it is, besides this negative result, possible to model GAP-CC as a *linear* integer program:

Since the weights are by assumption integer, it suffices to evaluate c_j at the integer points $0, 1, \ldots, W_j$, where $W_j := \sum_{i\in\mathcal{I}} w_{ij}$. If we interpolate the cost function c_j using these points, we obtain a piecewise linear function $\bar{c_j}$ consisting of W_j pieces $\phi_j^1, \phi_j^2, \ldots, \phi_j^{W_j}$ with $\phi_j^l(x) = a_j^l x + b_j^l$ for some values $a_j^l, b_j^l \in \mathbb{R}$.

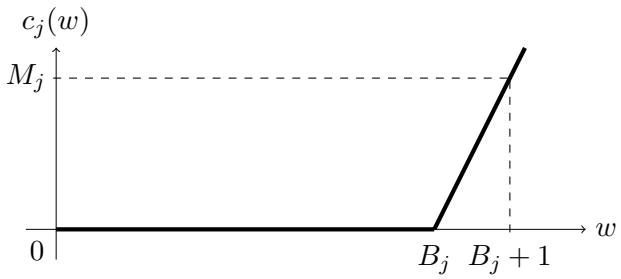


Figure 3.1: Using cost functions c_j as given in (3.2) shows that GAP is a special case of GAP-CC.

The function $\bar{c_j}$ is again convex, we have that $\bar{c_j}(x) = \max_{l=1}^{W_j} \phi_j^l(x)$ for all $x \in \{0, 1, \ldots, W_j\}$, and we can rewrite (3.1) using standard arguments as

$$
\begin{aligned}
\max \quad & \sum_{j \in \mathcal{B}} \left(\sum_{i \in \mathcal{I}} p_{ij} x_{ij} \right) - z_j & & \\
\text{s.t.} \quad & \sum_{j \in \mathcal{B}} x_{ij} \leq 1 & & \forall\, i \in \mathcal{I} \\
& z_j \geq \phi_j^l \left(\sum_{i \in \mathcal{I}} w_{ij} x_{ij} \right) & & \forall\, j \in \mathcal{B},\ l \in \{1, \ldots, W_j\} \\
& x_{ij} \in \{0, 1\} & & \forall\, i \in \mathcal{I}, j \in \mathcal{B} \\
& z_j \geq 0 & & \forall\, j \in \mathcal{B}.
\end{aligned}
$$

Note that, however, the number of interpolation points (and thus also the number of constraints in this program) is exponential in the encoding length of GAP-CC.

If we wish to restrict ourselves to a constant number of interpolation points, we are not guaranteed to obtain an optimal solution anymore. In fact, even dropping one interpolation point can imply that we do not find a solution with positive net profit as the following example shows:

Example 3.4. *Suppose there is a single bin with cost function $c_1(w) = w^2$ and two items that both yield a profit of $p_{11} = p_{21} = 2$ and have a weight of $w_{11} = w_{21} = 1$. One can easily see that an optimal solution would accept one of the two items yielding a total net profit of 1.*

If we used an estimate $\tilde{c_1}$ (instead of $\bar{c_1}$) for the cost function with only two interpolation points 0 and 2, i.e., $\tilde{c_1}(w) = 2w$, it would be optimal (with respect to $\tilde{c_1}$) to assign none of the items, which yields a actual net profit of 0.

3.2 The Single Bin Case

In this section we consider the single bin version of GAP-CC, i.e., $|\mathcal{B}| = 1$, which corresponds to the *knapsack problem with convex costs*. Therefore, we also denote this problem in the following by KP-CC. As there is only one bin, we write for short w_i and p_i for the weight and profit of item i, respectively, and denote the cost function of the knapsack by c.

By Observation 3.3, KP-CC contains the knapsack problem as a special case and is therefore $\mathcal{NP}$-hard. We show that it remains $\mathcal{NP}$-hard even if the cost function is quadratic, and we have that the weight of each item equals its profit, i.e., $w_i = p_i$ for all items i.

Theorem 3.5. KP-CC *is $\mathcal{NP}$-hard, even if the weight of each item i equals its profit ($w_i = p_i$) and the cost function is quadratic.*

Proof. We perform a reduction from SUBSET-SUM, which is known to be $\mathcal{NP}$-complete (cf. (Garey and Johnson, 1979)). An instance of SUBSET-SUM is given by a set A and integers $a_i \in \mathbb{Z}_{\geq 0}$ for all $i \in A$ and some $B \in \mathbb{Z}_{\geq 0}$. The task is to decide whether there exists a subset $A' \subseteq A$ such that $\sum_{i \in A'} a_i = B$.

We construct an instance of KP-CC as follows: Consider the cost function $c(w) = \frac{w^2}{2B}$, and define for every $i \in A$ an item $i \in \mathcal{I}$ with $p_i := w_i := a_i$. The net profit of a subset $\mathcal{I}' \subseteq \mathcal{I}$ is given by

$$\pi(\mathcal{I}') = \sum_{i \in \mathcal{I}'} a_i - c\left(\sum_{i \in \mathcal{I}'} a_i\right) = \sum_{i \in \mathcal{I}'} a_i - \frac{1}{2B}\left(\sum_{i \in \mathcal{I}'} a_i\right)^2.$$

Note that

$$0 \leq \frac{1}{2B}\left(\sum_{i \in \mathcal{I}'} a_i - B\right)^2 = \frac{1}{2B}\left(\sum_{i \in \mathcal{I}'} a_i\right)^2 - \sum_{i \in \mathcal{I}'} a_i + \frac{B}{2}$$

implies that $\pi(\mathcal{I}') \leq \frac{B}{2}$ for all subsets $\mathcal{I}' \subseteq \mathcal{I}$ and $\pi(\mathcal{I}') = \frac{B}{2}$ if and only if $\sum_{i \in \mathcal{I}'} a_i = B$. □

Next, we identify two special cases that can be solved in polynomial time:

Proposition 3.6. KP-CC *can be solved in polynomial time if all items have the same weight ($w_i = w$) or the same profit ($p_i = p$).*

Proof. If all items have the same weight, we sort the items in non-increasing order of profits, and greedily add items to our selection as long as the total net profit increases.

Assume this did not yield an optimal solution, i.e., there are items i and i' with $p_i > p_{i'}$, where i is not chosen by OPT, but i' is. Then, exchanging i by i' would increase the net profit.

The case that all items have the same profits can be solved analogously by sorting the items in non-decreasing order of weights. □

Although Theorem 3.5 rules out the possibility of a polynomial-time algorithm for solving KP-CC (assuming $\mathcal{P} \neq \mathcal{NP}$), we show next that there exists an algorithm with pseudo-polynomial running time, which is based on the ideas of the dynamic program known for the classical knapsack problem (cf. Kellerer et al. (2004)).

Our algorithm works as follows: We set an upper bound $P^{\max} := \sum_{i \in \mathcal{I}} p_i$, and define for all $k \in \{1, \dots, n\}$ and $l \in \{1, \dots, P^{\max}\}$ the value $f_k(l)$ as the minimum total weight of a subset $S \subseteq \{1, \dots, k\}$ of the first k items that yields a profit of exactly l, i.e.,

$$f_k(l) := \min \left\{ \sum_{i \in S} w_i \ : \ S \subseteq \{1, \dots, k\} \text{ and } \sum_{i \in S} p_i = l \right\}. \tag{3.3}$$

If the minimum does not exist, we set $f_k(l) := \infty$. By assumption the cost function c is non-decreasing, and it follows that under all assignments which yield a profit of l the one with minimal total weight also incurs minimal total costs. Thus, we obtain an optimal solution for KP-CC by computing

$$\max_{l \in \{0, \dots, P^{\max}\}} l - c(f_n(l)).$$

The values $f_k(l)$ can be determined by

$$f_1(l) = \begin{cases} w_1, & \text{if } l = p_1 \\ \infty, & \text{else,} \end{cases}$$

and the recursion formula

$$f_{k+1}(l) = \min\{f_k(l), f_k(l - p_{k+1}) + w_{k+1}\}.$$

As there are $nP^{\max}$ values that need to be determined and each one can be determined in constant time, the total running time of this procedure is $\mathcal{O}(nP^{\max})$. Note that the encoding length of an instance of KP-CC is only polynomial in $\log(P^{\max})$, and thus we have:

Proposition 3.7. KP-CC *can be solved in pseudo-polynomial time using dynamic programming.*

Independently of our work, Antoniadis et al. (2013) proposed a different dynamic programming formulation. We believe that our formulation is more natural. However, their formulation has the advantage that it uses a stronger upper bound (for the net profit), and it can be scaled appropriately in order to obtain an FPTAS:

Theorem 3.8 (Antoniadis et al. (2013))**.** *There exists an FPTAS for the single bin version of* GAP-CC*.*

3.3 Complexity

We have already seen in Observation 3.3 that GAP-CC contains GAP as a special case. In this section we show strong $\mathcal{NP}$-hardness for restricted cases of GAP-CC that use, in particular, more realistic cost functions.

Theorem 3.9. GAP-CC *is strongly $\mathcal{NP}$-hard, even if the cost functions of all bins are identical ($c_j = c$) and quadratic, the weight of an item is the same for each bin ($w_{ij} = w_i$), and either*

(i) the weight of each item equals its profit ($w_i = p_i$), or

(ii) all items have the same profit ($p_{ij} = p$).

Proof. We perform a reduction from 3-PARTITION, which is known to be strongly $\mathcal{NP}$-complete (cf. Garey and Johnson (1979)). In an instance of 3-PARTITION we are given non-negative integers $a_1, \ldots, a_{3m}, B \in \mathbb{Z}_{\geq 0}$ such that $\frac{B}{4} < a_i < \frac{B}{2}$ for all $i \in \{1, \ldots, 3m\}$ and $\sum_{i=1}^{3m} a_i = mB$. The task is to decide whether there exists a partition into sets of three elements each, i.e., $\dot{\cup}_{j=1}^{m} U_j = \{1, \ldots, 3m\}$ with $|U_j| = 3$ for all $j \in \{1, \ldots, m\}$ such that $\sum_{i \in U_j} a_i = B$ for all $j \in \{1, \ldots, m\}$.

Given an instance of 3-PARTITION, we construct an instance of GAP-CC, where every bin $j \in \{1, \ldots, m\}$ has the same convex, quadratic cost function c_j with $c_j(w) := \frac{w^2}{2B}$.

In (i), where it has to hold that $w_i = p_i$ for all items i, we define an item $i \in \mathcal{I}$ with $w_i := p_i := a_i$ for all $i \in \{1, \ldots, 3m\}$. The net profit when assigning $I_1, \ldots, I_m \subseteq \mathcal{I}$ to bins $1, \ldots, m$, respectively, is then given as

$$\pi(I_1, \ldots, I_m) = \sum_{j=1}^{m} \left(\sum_{i \in I_j} a_i - \frac{\left(\sum_{i \in I_j} a_i\right)^2}{2B} \right),$$

and we show next that it attains the value $\frac{mB}{2}$ if and only if there exists a 3-PARTITION:

It is easy to verify that if there is a 3-PARTITION, then the corresponding assignment yields a net profit of $\frac{mB}{2}$.

Vice versa, assume there is an assignment $I_1, \ldots, I_m$ with a total net profit of $\frac{mB}{2}$. Analogously to the proof of Theorem 3.5, it holds that the contribution of every bin j to the total net profit is at most $\frac{B}{2}$. Thus, we have for all bins $j \in \{1, \ldots, m\}$ that

$$\sum_{i \in I_j} a_i - \frac{\left(\sum_{i \in I_j} a_i\right)^2}{2B} = \frac{B}{2},$$

which is equivalent to $\sum_{i \in I_j} a_i = B$.

In (ii), where it has to hold that $p_{ij} = p$, we define an item $i \in \mathcal{I}$ with $w_i := a_i$ and $p_i := p := c\,(n \cdot \max_i w_i)+1$ for all $i \in \{1, \ldots, 3m\}$. Note that we assume that the cost function can be evaluated in constant time, and, hence, p can be determined in constant time. This "large" profit ensures that all items have to be assigned to some bin in an optimal solution, and we show that there exists an assignment with net profit $\pi^* := np - \frac{mB}{2}$ if and only if there exists a 3-PARTITION:

If there is a 3-PARTITION, it follows by definition that the corresponding assignment yields a net profit of π^*.

If there exists an assignment with net profit π^*, convexity of the cost functions implies that the load on every bin j is exactly B: Let $B + \delta_j$ be the load on bin j, where $\sum_{j\in\mathcal{B}} \delta_j = 0$. Then, the total costs incurred are

$$\begin{aligned}\frac{mB}{2} = \sum_{j\in\mathcal{B}} c_j(B+\delta_j) &= \frac{1}{2B}\Bigg(mB^2 + \underbrace{2B\sum_{j\in\mathcal{B}}\delta_j}_{=0} + \sum_{j\in\mathcal{B}}\delta_j^2\Bigg)\\ &= \frac{mB}{2} + \frac{1}{2B}\sum_{j\in\mathcal{B}}\delta_j^2,\end{aligned}$$

which implies that $\delta_j = 0$ for all j. Thus, there exists a 3-PARTITION. □

Observe that, when using a similar reduction from PARTITION instead of 3-PARTITION, we can see that the problem is also (weakly) $\mathcal{NP}$-hard for two bins under the previous assumptions.

Corollary 3.10. *Under the assumptions of Theorem 3.9,* GAP-CC *is $\mathcal{NP}$-hard, even for two bins, i.e., $|\mathcal{B}| = 2$.*

In particular, the case of constant profits, which can be solved in polynomial time for the single bin case (Proposition 3.6), becomes $\mathcal{NP}$-hard for two bins.

3.4 Polynomially Solvable Cases

We have seen that GAP-CC is strongly $\mathcal{NP}$-hard under the assumptions of Theorem 3.9. In this section we identify two special cases of GAP-CC that can be solved in polynomial time.

3.4.1 A Round-Robin Algorithm for Identical Cost Functions, Unit Weights, and Bin-Independent Profits ($w_{ij} = w, p_{ij} = p_i, c_j = c$)

If we consider the special case where we assume that the profits do not depend on the bin ($p_{ij} = p_i$), all cost functions are identical ($c_j = c$), and the

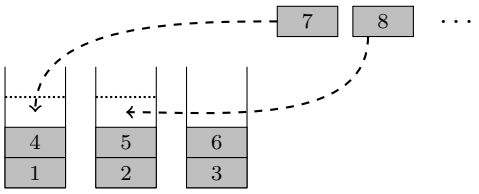


Figure 3.2: Illustration of a round-robin assignment of Algorithm 3.1.

weights are constant ($w_{ij} = w$), the problem can be solved in $\mathcal{O}(n \log n)$ by Algorithm 3.1.

The idea of the algorithm is to sort the items in non-increasing order of profits and then keep on adding them to the bins in a round-robin manner, i.e., item i is assigned to bin $(i \bmod m) + 1$ (see Figure 3.2) as long as the net profit increases.

Algorithm 3.1 Round-Robin Algorithm for $p_{ij} = p_i, w_{ij} = w, c_j = c$

```
1: Sort the items ℐ in non-increasing order of profits, i.e., p_1 ≥ p_2 ≥ ··· ≥ p_n.
2: for i = 1, ..., n do
3:    if the net profit increases when item i is assigned to bin (i mod m) + 1
      then
4:       assign item i to bin (i mod m) + 1
5:    end if
6: end for
```

Theorem 3.11. *If the profits do not depend on the bin ($p_{ij} = p_i$), the weights are constant ($w_{ij} = w$), and all bins have the same cost function ($c_j = c$),* GAP-CC *can be solved in $\mathcal{O}(n \log n)$ by Algorithm 3.1.*

Proof. We assume, without loss of generality, that $w_{ij} = 1$ (if this does not hold, we simply scale the common weight w appropriately and modify the cost functions).

Observe that an optimal solution assigns, without loss of generality, items to the bins such that the loads of the bins differ by at most one: otherwise, the items could be reassigned such that this property holds and the net profit is increased (since the profits do not change and the cost functions are convex and identical for all bins). Hence, we can assume that an optimal solution is given in a round robin-manner.

Since the net profit function can be evaluated in constant time, the running time is dominated by the time needed for the sorting of the items, which can be done in $\mathcal{O}(n \log n)$. □

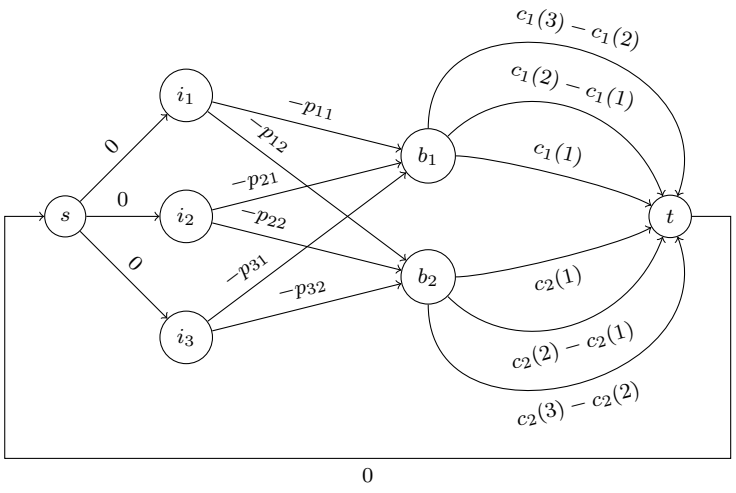


Figure 3.3: Illustration of the network for three items and two bins. The labels on the arcs correspond to the costs. Arc (t, s) has capacity ∞, all other arcs have capacity 1.

3.4.2 A Minimum-Cost Flow Model for Item-Independent Weights ($w_{ij} = w_j$)

The problem GAP-CC can also be solved in polynomial time if we restrict it to have weights that do not depend on the items, i.e., $w_{ij} = w_j$. In this case, we can model it as a minimum-cost flow problem.

Therefore, consider the following network: There is a node for every item i and for every bin j and two additional nodes s and t. There are arcs from s to all items with costs 0. Furthermore, there is an arc between item i and bin j with costs $-p_{ij}$, and there are n parallel arcs connecting bin j with the sink t that have costs of $c_j(a) - c_j(a-1)$ for $a \in \{1, \ldots, n\}$, respectively, representing the additional costs incurred by the a-th unit of weight on bin j. All of these arcs have unit capacity. Finally, there is an arc connecting t with s at cost 0 and capacity ∞. This network is illustrated in Figure 3.3.

Observe that we can assume all weights to be 1 (as argued in the proof of Theorem 3.11, we can modify the cost functions otherwise), and since the cost functions are convex and non-decreasing we have that $0 \leq c_j(a) - c_j(a-1) \leq c_j(b) - c_j(b-1)$ for all $a \leq b$. Thus, a minimum-cost circulation in the given network corresponds to an assignment of items to bins with maximum net profit.

In order to solve GAP-CC for $w_{ij} = w_j$, we simply need to compute a minimum-cost circulation in the given network. Since the minimum-cost flow problem can be solved in polynomial time (cf. Ahuja et al. (1993)) and the

graph has $m + n + 2 = \mathcal{O}(m + n)$ nodes and $n + 2mn + 1 = \mathcal{O}(mn)$ arcs, we obtain the following result:

Theorem 3.12. GAP-CC *can be solved in polynomial time if the weights do not depend on the item ($w_{ij} = w_j$).*

3.5 Approximation Algorithm for the General Case

In this section we design an approximation algorithm for the case of arbitrary profits. Following the general scheme as described in Section 1.1.3, it is based on randomized rounding of a configuration integer programming formulation. The linear relaxation of this program can be solved approximately by an FPTAS and the resulting fractional solution yields a suitable probability distribution for a randomized rounding procedure. However, the rounded solution is, in general, infeasible and in order to turn it into a feasible solution, we define appropriate estimators that linearize the convex costs.

3.5.1 Randomized Rounding Procedure

Besides the straight-forward formulation (3.1), GAP-CC can also be formulated as a configuration-based integer linear program (cf. (1.5)).

Although it might lead to negative net profits, it is allowed to assign any subset of the items to each of the bins, i.e., the set of feasible configurations for bin j is given by

$$T(j) := 2^{\mathcal{I}}.$$

As before, we write $T := \dot{\bigcup}_{j \in \mathcal{B}} T(j)$ for the disjoint union of all bins' configurations. Denoting the net profit of a configuration $t \in T(j)$ by

$$\pi_t := \sum_{i \in t} p_{ij} - c_j \left(\sum_{i \in t} w_{ij} \right),$$

GAP-CC is given as follows:

$$\begin{aligned}
\text{(IP)} \quad \max \quad & \sum_{t \in T} \pi_t x_t && && \text{(3.4a)} \\
\text{s.t.} \quad & \sum_{t \in T(j)} x_t = 1 && \forall\, j \in \mathcal{B} && \text{(3.4b)} \\
& \sum_{t \in T : i \in t} x_t \leq 1 && \forall\, i \in \mathcal{I} && \text{(3.4c)} \\
& x_t \in \{0, 1\} && \forall\, t \in T. && \text{(3.4d)}
\end{aligned}$$

We denote again the linear relaxation of (IP) by (LP) (recall that (3.4d) is relaxed to $x_t \geq 0$).

Observe that (LP) has an exponential number of variables ($|T| = m\,2^n$). However, we can use a result by Fleischer et al. (2011) that shows how an approximation for (LP) can be obtained using an approximation algorithm for the single bin subproblem, i.e., the problem of finding an optimal configuration for a fixed bin (we give a proof for a generalization of this result in Section 4.2.1):

Theorem 3.13 (Fleischer et al. (2011))**.** *If there exists an FPTAS for the single bin subproblem, then there exists an FPTAS for solving the linear relaxation (LP).*

As we have already stated in Theorem 3.8, Antoniadis et al. (2013) designed an FPTAS for the single bin subproblem. This yields together with Theorem 3.13 the following result:

Corollary 3.14. *There exists an FPTAS for solving (LP).*

Our randomized rounding can then be performed as described in Section 1.1.3: For a fixed $\epsilon > 0$, we first solve (LP) *approximately* as described above using the FPTAS and obtain a fractional solution $x^{\mathrm{LP}} \in [0,1]^{|T|}$ in time polynomial in the encoding length of the problem and $\frac{1}{\epsilon}$. Then, we choose for each bin j a configuration independently at random, where configuration $t \in T(j)$ is chosen with probability x_t^{LP}. The resulting vector $x^{\mathrm{IP}} \in \{0,1\}^{|T|}$ (where $x_t^{\mathrm{IP}} = 1$ if and only if configuration t is chosen) then satisfies constraints (3.4b) and the expected net profit of the rounded solution is given (as shown in (1.7)) by

$$\begin{aligned}\mathbb{E}\left[\textsc{net-profit}(x^{\mathrm{IP}})\right] = \mathbb{E}\left[\sum_{t\in T}\pi_t x_t^{\mathrm{IP}}\right] &= \sum_{t\in T}\pi_t \mathbb{E}\left[x_t^{\mathrm{IP}}\right] \\ &= \sum_{t\in T}\pi_t x_t^{\mathrm{LP}} \\ &\geq (1-\epsilon)\cdot \textsc{opt}_{\mathrm{LP}}, \qquad (3.5)\end{aligned}$$

where $\textsc{opt}_{\mathrm{LP}}$ denotes the optimal solution of (LP), and we use the notation $\textsc{net-profit}(x) := \sum_{t\in T}\pi_t x_t$. Hence, we have:

Observation 3.15. *The vector $x^{\mathrm{IP}} \in \{0,1\}^{|T|}$ obtained from the randomized rounding process satisfies* (3.4b) *and $\mathbb{E}[\textsc{net-profit}(x^{\mathrm{IP}})] \geq (1-\epsilon)\cdot \textsc{opt}_{\mathrm{LP}}$.*

Observe that x^{IP} does, in general, not fulfill (3.4c) since an item might be assigned to more than one bin.

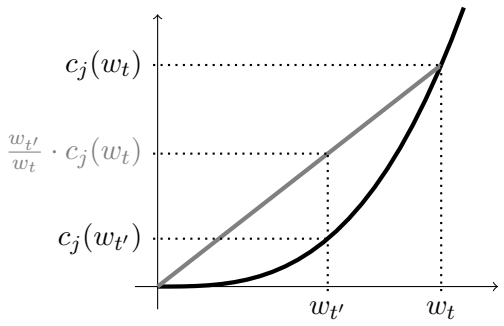


Figure 3.4: Illustration of the linearized costs as given in Definition 3.16 and Lemma 3.17 for configurations t (with total weight w_t) and $t' \subsetneq t$ (with total weight $w_{t'}$).

3.5.2 Obtaining a Feasible Solution

In order to turn x^{IP} into a feasible solution of (IP), we need to remove multiple copies of items from the bins they are assigned to. In classical GAP, where $c_j \equiv 0$ for all j, the straightforward way of doing this is to delete each item from all bins it is assigned to except for the one where it yields the highest profit (cf. (Fleischer et al., 2011)).

Due to the non-linear cost functions c_j, one also has to take the cost increase caused by the items' weights into account. One natural way to do this is to "linearize" the costs by charging every item an amount of the costs proportional to its weight as illustrated in Figure 3.4. This motivates the following definition:

Definition 3.16. *For a configuration $t \in T(j)$, we define the* net profit contribution of item i with respect to configuration t *as*

$$\mu_i^t := \begin{cases} p_{ij} - \frac{w_{ij}}{\sum_{k \in t} w_{kj}} c_j \left(\sum_{k \in t} w_{kj} \right), & \text{if } i \in t \\ 0, & \text{else.} \end{cases}$$

Note that for each $t \in T$ the net profit contributions of all items with respect to t sum up to the net profit of configuration t, i.e.,

$$\sum_{i \in t} \mu_i^t = \pi_t. \tag{3.6}$$

Moreover, the following result shows that we can use the net profit contributions with respect to a configuration in order to obtain a lower bound on the net profit of any subset of the configuration:

Lemma 3.17. *For every configuration $t' \subseteq t$, we have*

$$\sum_{i \in t'} \mu_i^t \leq \pi_{t'}.$$

Proof. For $t' = t$, the result holds by (3.6). For $t' \subsetneq t$, let j denote the bin to which configuration t corresponds, i.e., $t \in T(j)$. Observe that for $0 \leq x < y$ the convexity of c_j and the assumption that $c_j(0) = 0$ imply that

$$c_j(x) = c_j\left(\left(1 - \frac{x}{y}\right) 0 + \frac{x}{y}\, y\right) \leq \left(1 - \frac{x}{y}\right) \underbrace{c_j(0)}_{=0} + \frac{x}{y}\, c(y) = \frac{x}{y}\, c(y). \tag{3.7}$$

If we choose $x := \sum_{i \in t'} w_{ij}$ and $y := \sum_{i \in t} w_{ij}$ (which satisfy $0 \leq x < y$ since we assume that $w_{ij} > 0$ for all i, j and $t' \subsetneq t$), this shows that

$$c_j\left(\sum_{i \in t'} w_{ij}\right) \leq \frac{\sum_{i \in t'} w_{ij}}{\sum_{i \in t} w_{ij}} \cdot c_j\left(\sum_{i \in t} w_{ij}\right). \tag{3.8}$$

Using this inequality, we obtain

$$\sum_{i \in t'} \mu_i^t = \sum_{i \in t'} p_{ij} - \frac{\sum_{i \in t'} w_{ij}}{\sum_{k \in t} w_{kj}} \cdot c_j\left(\sum_{k \in t} w_{kj}\right) \leq \sum_{i \in t'} p_{ij} - c_j\left(\sum_{i \in t'} w_{ij}\right) = \pi_{t'}.$$

□

In order to decide to which bin a multiply assigned item remains assigned, we now use the net profit contributions we defined above. If an item i is assigned to multiple bins, we delete it from all bins except for the one where the expression

$$E_{ij} := \frac{\sum\limits_{\substack{t \in T(j):\\ i \in t}} \mu_i^t x_t^{\mathrm{LP}}}{y_j} \tag{3.9}$$

is maximal. Here, we denote by $y_j := \sum_{t \in T(j): i \in t} x_t^{\mathrm{LP}}$ the probability that item i is assigned to bin j. Note that for the classical GAP it holds that $\mu_i^t = p_{ij}$ for all $t \in T(j)$, i.e, $E_{ij} = p_{ij}$ (which coincides with the removal procedure of Fleischer et al. (2011)).

The expression E_{ij} can be interpreted as the conditional expectation of the net profit contribution of item i in bin j given that the configuration chosen for bin j contains i. To make this notion more precise, we define for an integral vector $x \in \{0,1\}^{|T|}$ (that is possibly infeasible for (3.4)) the *net profit contribution of item i in bin j* as

$$\mu_i^j(x) := \sum_{\substack{t \in T(j):\\ i \in t}} \mu_i^t x_t,$$

and, analogously, the *net profit contribution of an item i* is defined as the sum over all bins, i.e.,

$$\mu_i(x) := \sum_{\substack{t \in T: \\ i \in t}} \mu_i^t x_t = \sum_{j \in \mathcal{B}} \mu_i^j(x). \tag{3.10}$$

Note that we then have that

$$E_{ij} = \mathbb{E}\left[\mu_i^j(x^{\mathrm{IP}}) \mid \text{configuration chosen for bin } j \text{ contains item } i\right]. \tag{3.11}$$

We denote by $\bar{x}^{\mathrm{IP}}$ the feasible solution we obtain after removing multiple copies in our rounded solution x^{IP} (as defined above). In the following, we want to bound the expected net profit contribution of item i in $\bar{x}^{\mathrm{IP}}$ from below (summing over all items will then yield a lower bound for the expected actual net profit of our solution $\bar{x}^{\mathrm{IP}}$).

Let $i \in \mathcal{I}$ be an arbitrary but fixed item and number the bins in non-increasing order of (3.9) such that $E_{i1} \geq E_{i2} \geq \ldots \geq E_{im}$.

With probability y_1 the configuration that is chosen for bin 1 contains item i. In this case, item i is assigned to bin 1 and (as we will show in more detail below) its expected net profit contribution is at least E_{i1}.

With probability $(1 - y_1)y_2$ the configuration chosen for bin 1 does not contain item i, but the one for bin 2 does (this holds since the configurations for the bins are chosen independently). In this case, item i is assigned to bin 2 and its expected net profit contribution is at least E_{i2}. This argumentation can be applied in the same way to the remaining bins:

We consider for each $j \in \{1, \ldots, m\}$ the set $\mathcal{S}_j$ of outcomes of the random variable x^{IP} for which the configurations chosen for bins $1, \ldots, j-1$ do not contain item i, but the one for bin j does.

As the configurations are chosen independently for each bin, it holds that

$$\Pr(\mathcal{S}_j) = \left(\prod_{k=1}^{j-1}(1 - y_k)\right) y_j. \tag{3.12}$$

Furthermore, we have for the expected net profit contribution of item i in $\bar{x}^{\mathrm{IP}}$ conditioned on $\mathcal{S}_j$:

$$\begin{aligned}
& \mathbb{E}\left[\mu_i(\bar{x}^{\mathrm{IP}}) \mid \mathcal{S}_j\right] \\
= \; & \sum_{k \in \mathcal{B}} \mathbb{E}\left[\mu_i^k(\bar{x}^{\mathrm{IP}}) \mid \mathcal{S}_j\right] && (3.13) \\
= \; & \mathbb{E}\left[\mu_i^j(\bar{x}^{\mathrm{IP}}) \mid \mathcal{S}_j\right] && (3.14) \\
\geq \; & \mathbb{E}\left[\mu_i^j(x^{\mathrm{IP}}) \mid \mathcal{S}_j\right] && (3.15)
\end{aligned}$$

$$= \mathbb{E}\left[\mu_i^j(x^{\mathrm{IP}}) \mid \text{configuration chosen for bin } j \text{ contains } i\right] \tag{3.16}$$

$$\overset{(3.11)}{=} E_{ij}. \tag{3.17}$$

Here, the first equality (3.13) is due the definition of μ_i and linearity of the conditional expectation.

Equality (3.14) holds since the condition $\mathcal{S}_j$ implies that $\bar{x}_t^{\mathrm{IP}} = 0$ for all configurations $t \in T \setminus T(j)$ with $i \in t$: The configurations chosen by x^{IP} for bins $1, \ldots, j-1$ do not contain item i. As the configuration chosen by x^{IP} for bin j contains item i, the assumption that the bins are numbered in non-increasing order of (3.9) then implies that further copies of the item in bins $j+1, \ldots, m$ are deleted. Therefore, item i is only contained in bin j (in solution $\bar{x}^{\mathrm{IP}}$).

Inequality (3.15) is valid since for every $t \in T(j)$ with $i \in t$ that is chosen in x^{IP}, $\bar{x}^{\mathrm{IP}}$ chooses a subset $t' \subseteq t$ for bin j. As we argued above, the condition $\mathcal{S}_j$ implies that $i \in t'$, and by (3.8) it holds that $\mu_i^{t'} \geq \mu_i^t$, i.e., i has in t' at least the same net profit contribution as in t.

Finally, (3.16) holds since the expected net profit contribution of item i in bin j (in the solution x^{IP} obtained from the randomized rounding) is independent of the random choices for configurations of the other bins.

The events $\mathcal{S}_1, \ldots, \mathcal{S}_m$ cover all cases where item i is contained in the final solution $\bar{x}^{\mathrm{IP}}$. Thus, we obtain by (3.12) and (3.17) that the expected net profit contribution of item i in $\bar{x}^{\mathrm{IP}}$ is

$$\begin{aligned}
\mathbb{E}\left[\mu_i(\bar{x}^{\mathrm{IP}})\right] &= \sum_{j=1}^{m} \Pr(\mathcal{S}_j) \cdot \mathbb{E}\left[\mu_i(\bar{x}^{\mathrm{IP}}) \mid \mathcal{S}_j\right] \\
&\geq \sum_{j=1}^{m} \left(\prod_{k=1}^{j-1}(1-y_k)\right) y_j E_{ij} \\
&\geq \left(1 - \left(1 - \frac{1}{m}\right)^m\right) \sum_{j=1}^{m} y_j E_{ij} && (3.18) \\
&\geq \left(1 - \frac{1}{e}\right) \sum_{j=1}^{m} y_j E_{ij} && (3.19) \\
&\overset{(3.9)}{=} \left(1 - \frac{1}{e}\right) \sum_{\substack{t \in T: \\ i \in t}} \mu_i^t x_t^{\mathrm{LP}}. && (3.20)
\end{aligned}$$

Here, we used the arithmetic-geometric mean inequality for (3.18) and the fact that $(1 - \frac{1}{k})^k \leq \frac{1}{e}$ for all $k \geq 1$ for (3.19) (cf. Goemans and Williamson (1994)).

Hence, the expected total net profit of our solution $\bar{x}^{\mathrm{IP}}$ is

$$\begin{aligned}
\mathbb{E}\left[\textsc{net-profit}(\bar{x}^{\mathrm{IP}})\right] &= \mathbb{E}\left[\sum_{t\in T}\pi_t \bar{x}_t^{\mathrm{IP}}\right] \\
&\overset{(3.6)}{=} \mathbb{E}\left[\sum_{t\in T}\left(\sum_{i\in t}\mu_i^t\right)\bar{x}_t^{\mathrm{IP}}\right] \\
&= \sum_{i\in\mathcal{I}}\mathbb{E}\left[\sum_{\substack{t\in T:\\ i\in t}}\mu_i^t \bar{x}_t^{\mathrm{IP}}\right] \\
&\overset{(3.10)}{=} \sum_{i\in\mathcal{I}}\mathbb{E}\left[\mu_i(\bar{x}^{\mathrm{IP}})\right] \\
&\overset{(3.20)}{\geq} \sum_{i\in\mathcal{I}}\left(1-\frac{1}{e}\right)\sum_{t\in T: i\in t}\mu_i^t x_t^{\mathrm{LP}} \\
&= \left(1-\frac{1}{e}\right)\sum_{t\in T}\underbrace{\sum_{i\in t}\mu_i^t}_{=\pi_t} x_t^{\mathrm{LP}} \\
&\overset{(3.5)}{\geq} \left(1-\frac{1}{e}\right)(1-\epsilon)\,\mathrm{OPT}_{\mathrm{LP}} \\
&\geq \left(1-\frac{1}{e}-\epsilon\right)\mathrm{OPT}_{\mathrm{LP}}. \qquad (3.21)
\end{aligned}$$

Obviously, it holds that $\mathrm{OPT}_{\mathrm{LP}} \geq \mathrm{OPT}$, and we obtain the following result:

Proposition 3.18. *For every $\epsilon > 0$, there exists a randomized $\left(1-\frac{1}{e}-\epsilon\right)$-approximation algorithm for* GAP-CC *whose running time is polynomial in the encoding length of the problem and $\frac{1}{\epsilon}$.*

In fact, one can even slightly improve upon the previous result. If we use for (3.19) the sharper estimate

$$1-\left(1-\frac{1}{k}\right)^k \geq 1-\frac{1}{e}+\frac{1}{32k^2} \quad \text{for all } k\geq 1,$$

due to Nutov et al. (2006), we can bound the expected net profit from below similar to (3.21) by

$$\left(1-\frac{1}{e}+\frac{1}{32m^2}-\epsilon\right)\cdot\mathrm{OPT}_{\mathrm{LP}}.$$

If we then choose $\epsilon = \frac{1}{32m^2}$ (which guarantees that $\frac{1}{\epsilon}$ is polynomial in the encoding length of the problem), we obtain the following result:

Theorem 3.19. *There exists a randomized $(1-\frac{1}{e})$-approximation algorithm for* GAP-CC.

3.6 Conclusions

In this chapter we introduced and analyzed the problem GAP-CC, which generalizes the classical GAP. While we were able to solve certain special cases in polynomial time, we showed that even restricted cases of GAP-CC remain strongly $\mathcal{NP}$-hard. Finally, we presented a randomized approximation algorithm for the general case.

The idea of this algorithm can also be applied if we have additional feasibility constraints for the bins (Barman et al. (2012) consider, e.g., the single bin version of GAP-CC with additional matroid constraints). The existence of a β-approximation for the single bin subproblem implies that there also exists a β-approximation for the linear relaxation of the integer programming formulation (3.4) (cf. Section 4.2.1). Our analysis then yields that there exists a $((1 - \frac{1}{e})\beta)$-approximation for multiple bins.

For classical GAP with fixed profits ($p_{ij} = p_i$), the approximation algorithm from Fleischer et al. (2011) can be derandomized (Nutov et al., 2006). However, our setting is harder to analyze since the net profit obtained in a bin cannot be separated into the items' profits. It thus remains an open problem whether such a derandomization can also be performed for (some cases of) our problem, or whether there exist other deterministic approximation algorithms.

— 4 —

The Separable Assignment Problem with Multiple Copies of Items

In this chapter we consider the *separable assignment problem* (SAP) where we are given a set of bins $\mathcal{B}$ and a set of items $\mathcal{I}$ to pack into the bins. For each item $i \in \mathcal{I}$ and bin $j \in \mathcal{B}$, there is a profit p_{ij} that is obtained when assigning item i to bin j. Moreover, for every bin j, there is a *separate packing constraint*. This means that only certain subsets of the items fit into bin j, but if a set $S \subseteq \mathcal{I}$ of items is a feasible packing for bin j, then every subset $S' \subseteq S$ is also a feasible packing for bin j. The objective is to find an assignment of a subset of the items to the bins such that a feasible packing is obtained for each bin, no item is assigned to more than one bin, and the total profit is maximized.

As an important special case, SAP contains GAP, in which the feasible packings of bin j are all subsets of the items with total weight at most the given capacity B_j.

In the following, we define the problem k-SAP, in which each item can be assigned at most $k \geq 1$ times in total (but no item may be assigned more than once to the same bin). This obviously generalizes SAP. Moreover, we study a generalization of k-SAP in which we are given a different number $k_i \geq 1$ for each item i (that specifies the maximum number of bins it may be assigned to). As we will see below, k-SAP is, for any fixed $k \geq 1$, also a special case of SAP that, however, allows for better approximation results.

This version of SAP can be motivated, e.g., from the assignment of advertisements to magazines: A set of potential customers (items) want to place their advertisements in up to k different magazines (bins), and customer i is willing to pay p_{ij} if his ad is placed in magazine j. The media group that publishes the magazines aims at maximizing their total profit for accepted ads, while having to respect several constraints, e.g., only a limited number of ads can be assigned to every magazine (due to space restrictions), or advertisements of competitors should not be placed in the same magazine.[1]

[1]While the first constraint can also be modeled in GAP, forbidding certain subsets of items in such a general way is not possible in GAP.

Previous Work

For the general case of SAP, the best-known approximation result is the $((1-\frac{1}{e})\beta)$-approximation obtained by Fleischer et al. (2011) under the assumption that the single bin subproblem admits a β-approximation algorithm. If the single bin subproblem admits an FPTAS, their method yields a $(1-\frac{1}{e})$-approximation for SAP. Furthermore, they show that SAP can, in general, not be approximated in polynomial time with an approximation factor better than $(1-\frac{1}{e})$, unless $\mathcal{NP} \subseteq \text{DTIME}(n^{\mathcal{O}(\log\log n)})$.[2]

A $((1-\frac{1}{e})\beta)$-approximation for SAP can also be obtained from the results of Calinescu et al. (2011), who gave a $(1-\frac{1}{e})$-approximation for a general class of submodular maximization problems.

Chapter Outline

In this chapter we consider the problem k-SAP, which is a generalization of SAP where items may be assigned to at most $k \geq 1$ bins. A formal definition of this problem and some preliminaries can be found in Section 4.1. In Section 4.2 we show how to generalize the approach in (Fleischer et al., 2011) to obtain an approximation algorithm for k-SAP based on randomized rounding of a configuration-based integer program. Even though, in general, finding an *optimal* solution for an instance of k-SAP is not easier than for SAP, intuition suggests that for larger k the different bins are less linked,[3] which could facilitate finding a good *approximate* solution. We show that this intuition is correct by proving an approximation factor of $((1-\frac{1}{e^k})\beta)$ for each $k \geq 1$ under the assumption that the single bin subproblem admits a β-approximation algorithm. Whenever the single bin subproblem admits an FPTAS, our method yields a $(1-\frac{1}{e^k})$-approximation for k-SAP, which shows that k-SAP admits for $k \geq 2$ approximation algorithms which beat the upper bound of $(1-\frac{1}{e})$ known for SAP.

In Section 4.3 we show that the randomized algorithm from Section 4.2 can be derandomized using the method of conditional expectations (cf. (Alon and Spencer, 1992)) if we have for each item i and bin j that $p_{ij} = p_i$, i.e., the profits are bin-independent. In Section 4.4 we study a generalization of k-SAP, where we allow that there is a different number $k_i \geq 1$ for each item i that specifies the maximum number of bins item i may be assigned to. Given a β-approximation for the single bin subproblem, we show that our method yields a $((1-\frac{1}{e^k})\beta)$-approximation for this case, where $k := \min_{i\in\mathcal{I}} k_i$.

[2] DTIME(f) denotes the class of problems for which there exists a deterministic algorithm with running time $\mathcal{O}(f)$ (cf. (Vazirani, 2001)). In particular, it holds: $\mathcal{P} = \cup_{k\in\mathbb{N}}\text{DTIME}(n^k)$.

[3] For $k = n$, an optimal solution can be obtained by determining an optimal solution for each bin separately.

4.1 Problem Definition and Preliminaries

The *separable assignment problem* (SAP) is formally defined as follows:

Definition 4.1 (Separable assignment problem (SAP))**.**
An instance of SAP *is given by a set of items $\mathcal{I}$ and a set of bins $\mathcal{B}$. Every bin $j \in \mathcal{B}$ has a separate packing constraint which satisfies that if a subset $S \subseteq \mathcal{I}$ of the items is a feasible packing for bin j, then every subset $S' \subseteq S$ is also feasible for bin j. If item $i \in \mathcal{I}$ is assigned to bin $j \in \mathcal{B}$, it yields a profit of $p_{ij} \in \mathbb{Z}$.*

The task is to find a feasible assignment of a subset of the items to the bins, i.e., every item is assigned at most once and the packing constraints of all bins are satisfied, such that the total profit is maximized.

For SAP the following "inapproximability result" is known:

Theorem 4.2 (Fleischer et al. (2011))**.** SAP *cannot be approximated in polynomial time within an approximation factor better than $(1 - \frac{1}{e})$, unless $\mathcal{NP} \subseteq DTIME(n^{\mathcal{O}(\log \log n)})$.*

For a fixed number $k \in \mathbb{Z}_{>0}$, that is not part of the input, we consider the problem k-SAP, which is given as follows:

Definition 4.3 (k-SAP)**.**
*An instance of k-*SAP *is given as in* SAP*. The task is to find an assignment of the items to the bins, such that every item is assigned at most k times in total (but at most once to every bin), the packing constraints of all bins are satisfied and the total profit is maximized.*

Obviously, k-SAP generalizes SAP, and we have:

Proposition 4.4. *For any fixed $k \geq 1$, finding an optimal solution for k-*SAP *is not easier than finding an optimal solution for* SAP*.*

Proof. Given an instance of SAP, we construct an instance of k-SAP by adding $k - 1$ additional bins for which all subsets of the items are feasible and in which every item i yields profit one more than the maximum profit $\max_{j \in \mathcal{B}} p_{ij}$ obtainable from item i in the original bins $\mathcal{B}$.

It is then immediate that an optimal solution for the instance of k-SAP will assign every item once to each of the additional bins, and finding an optimal assignment for the remaining bin is equivalent to finding an optimal solution to the given instance of SAP. □

Conversely, it also holds that:

Proposition 4.5. *For any fixed $k \geq 1$, k-*SAP *is a special case of* SAP*.*

Proof. Given an instance of k-SAP, we can construct an instance of SAP by replacing each of the original items by k identical copies and forbidding more than one copy of an item to be packed into the same bin (by defining the packing constraints in a suitable way). Then, finding an optimal solution for the instance of k-SAP is equivalent to finding an optimal solution for the original instance of SAP. □

Note that, despite the previous results, the upper bound of $(1 - \frac{1}{e})$ known for the approximability of SAP (Theorem 4.2) does *not* carry over to k-SAP for $k \geq 2$. By the reduction used in the proof of Proposition 4.4, the existence of an α-approximation for k-SAP (with $k \geq 2$) does *not* immediately imply the existence of an α-approximation for SAP. In particular, the existence of an approximation algorithm for k-SAP with approximation factor better than $(1-\frac{1}{e})$ remains possible. In fact, we show in the remainder of this chapter that an approximation factor of $(1 - \frac{1}{e^k})$ can be achieved for k-SAP if the single bin subproblem admits an FPTAS. This beats the upper bound known for SAP for $k \geq 2$.

4.2 Approximation Algorithm

In this section we present an approximation algorithm for k-SAP that resorts to randomized rounding of a configuration integer programming formulation as presented in Section 1.1.3. Our analysis generalizes the work of Fleischer et al. (2011) for the case $k = 1$. When modifying the rounded solution to obtain a feasible solution, the different problem setting requires a more evolved analysis to bound the loss in profit.

4.2.1 Randomized Rounding Procedure

The problem k-SAP can be modeled as a configuration integer program, where the set $T(j)$ of feasible configurations for bin j is already (explicitly) given in the problem definition, and $T := \dot{\bigcup}_{j \in \mathcal{B}} T(j)$ denotes the disjoint union of all bins' configurations. As for GAP, the profit of a configuration $t \in T(j)$ is $p_t := \sum_{i \in t} p_{ij}$, and we have the following formulation:

$$\text{(IP)} \quad \max \sum_{t \in T} p_t x_t \tag{4.1a}$$

$$\text{s.t.} \sum_{t \in T(j)} x_t = 1 \quad \forall\, j \in \mathcal{B} \tag{4.1b}$$

$$\sum_{t \in T : i \in t} x_t \leq k \quad \forall\, i \in \mathcal{I} \tag{4.1c}$$

$$x_t \in \{0, 1\} \qquad \forall\, t \in T. \tag{4.1d}$$

Note that this formulation is slightly different from previous configuration-based formulations (cf. (1.5)) as constraints (4.1c) allow that every item i

is contained in at most k different configurations. While an (approximate) solution to the linear relaxation can obviously still be used for a randomized rounding (since (4.1b) guarantees that we are given probability distributions for the bins), another analysis in the procedure for obtaining a feasible solution becomes necessary.

An Approximation for the Linear Relaxation

For the case $k = 1$, it is known that one can obtain a $(\beta - \delta)$-approximation algorithm for the linear relaxation of (IP) for every $\delta > 0$ if a β-approximation exists for the single bin subproblem. Moreover, the running time of the corresponding algorithm depends polynomially on $\frac{1}{\delta}$. This result can be extended to our setting, and it holds that:

Theorem 4.6. *If there exists a β-approximation for the single bin subproblem, then, for every $\delta > 0$, there exists a $(\beta - \delta)$-approximation for the linear programming relaxation of (IP) whose running time is polynomial in the encoding length of the given instance of k-SAP and $\frac{1}{\delta}$.*

We prove Theorem 4.6 following the sketch of the proof of Lemma 2.2 and the subsequent paragraph in (Fleischer et al., 2011) for $k = 1$. As their proof is very compact, we also elaborate on the details that were skipped in the paper in order to show that the result can, in fact, be extended to our more general setting.

The linear programming relaxation of (4.1) is given as

$$\begin{aligned} \text{(LP)} \quad \max & \sum_{t \in T} p_t x_t \\ \text{s.t.} & \sum_{t \in T(j)} x_t = 1 && \forall\, j \in \mathcal{B} \\ & \sum_{t \in T : i \in t} x_t \leq k && \forall\, i \in \mathcal{I} \\ & x_t \geq 0 && \forall\, t \in T. \end{aligned}$$

Recall that, as stated before, (4.1d) is relaxed to $x_t \geq 0$. The corresponding dual linear program to (LP) is then

$$\begin{aligned} \text{(DLP)} \quad \min & \sum_{j \in \mathcal{B}} q_j + \sum_{i \in \mathcal{I}} k \cdot \lambda_i \\ \text{s.t.} & \quad q_j + \sum_{i \in t} \lambda_i \geq p_t && \forall\, j \in \mathcal{B}\ \forall\, t \in T(j) \\ & \quad \lambda_i \geq 0 && \forall\, i \in \mathcal{I}. \end{aligned}$$

If we define for $j \in \mathcal{B}$ the polyhedron

$$\mathcal{P}_j := \left\{ (q_j, \lambda) : q_j + \sum_{i \in t} \lambda_i \geq p_t \text{ for all } t \in T(j) \right\},$$

(DLP) is equivalent to

$$\begin{aligned} \min \quad & \sum_{j \in \mathcal{B}} q_j + \sum_{i \in \mathcal{I}} k \cdot \lambda_i & \\ \text{s.t.} \quad & (q_j, \lambda) \in \mathcal{P}_j & \forall\, j \in \mathcal{B} \\ & \lambda_i \geq 0 & \forall\, i \in \mathcal{I}. \end{aligned}$$

We define a *β-approximate separation algorithm for $\mathcal{P}_j$* as follows: given (q_j, λ), the algorithm either returns a violated constraint or guarantees that $(\frac{q_j}{\beta}, \lambda)$ is feasible for $\mathcal{P}_j$. Using this definition, we can show the following result:

Lemma 4.7. *For any $\delta > 0$, given a β-approximate separation algorithm for $\mathcal{P}_j$, there exists a $(\beta - \delta)$-approximation algorithm to solve (LP) in time polynomial in the encoding length of the given instance of k-SAP and $\frac{1}{\delta}$.*

Proof. We consider the following equivalent reformulation of (DLP):

$$\begin{aligned} \min \quad & v & & & (4.2a) \\ \text{s.t.} \quad & \sum_{j \in \mathcal{B}} q_j + \sum_{i \in \mathcal{I}} k \cdot \lambda_i \leq v & & & (4.2b) \\ & (q_j, \lambda) \in \mathcal{P}_j & \forall\, j \in \mathcal{B} & & (4.2c) \\ & \lambda_i \geq 0 & \forall\, i \in \mathcal{I}. & & (4.2d) \end{aligned}$$

For a given value of v, we can apply the ellipsoid method (using the β-approximate separation algorithm) to determine whether this linear program is feasible. A minimal v such that the ellipsoid method does not return a violated constraint can then be determined using a binary search (for some precision parameter $\delta' > 0$). We will focus on the number of iterations needed in the binary search below.

If this procedure terminates with a solution (q^*, λ^*) with $v^* = \sum_{j \in \mathcal{B}} q_j^* + \sum_{i \in \mathcal{I}} k \cdot \lambda_i^*$, we know that (4.2) is infeasible for any $v \leq v^* - \delta'$, i.e.,

$$\text{OPT}_{\text{DLP}} \geq v^* - \delta'. \tag{4.3}$$

We do not know if the solution (q^*, λ^*) we obtained is feasible for (DLP) since we only used an approximate separation algorithm for $\mathcal{P}_j$. But we know that $(\frac{q^*}{\beta}, \lambda^*)$ is feasible for (DLP), which implies that

$$\text{OPT}_{\text{DLP}} \leq \sum_{j \in \mathcal{B}} \frac{q_j^*}{\beta} + \sum_{i \in \mathcal{I}} k \cdot \lambda_i^* \leq \frac{v^*}{\beta}, \tag{4.4}$$

since $\beta \leq 1$.

When the ellipsoid method is executed for $v^* - \delta'$, it terminates with a violated constraint and only a polynomial number of constraints are checked to show that (4.3) holds.

If we consider the linear program resulting from considering only these constraints in (4.2), we obtain a relaxation of (DLP), which we denote by (DLP′), and it holds that also (DLP′) is infeasible for $v \leq v^* - \delta'$, i.e.,

$$\textsc{opt}_{\text{DLP}'} \geq v^* - \delta'. \tag{4.5}$$

We denote the dual linear program to (DLP′) by (LP′), which is simply (LP) restricted to the variables that correspond to the constraints checked by the ellipsoid method.

Hence, this restricted version of (LP) has polynomial size and, since (4.5) holds for its dual, linear programming duality shows that its optimum objective value is at least $v^* - \delta'$, i.e.,

$$\textsc{opt}_{\text{LP}'} = \textsc{opt}_{\text{DLP}'} \geq v^* - \delta'. \tag{4.6}$$

Thus, if we choose the precision δ' of the binary search such that

$$\delta' = \delta \cdot \textsc{opt}_{\text{LP}}, \tag{4.7}$$

solving the polynomially-sized restricted linear program (LP′) yields a solution to (LP) with objective value

$$\textsc{opt}_{\text{LP}'} \overset{(4.6)}{\geq} v^* - \delta' \overset{(4.7)}{=} v^* - \delta \cdot \textsc{opt}_{\text{LP}} \geq (\beta - \delta) \cdot \textsc{opt}_{\text{LP}},$$

where the last inequality holds as $\textsc{opt}_{\text{LP}} = \textsc{opt}_{\text{DLP}} \leq \frac{v^*}{\beta}$ by (4.4).

Finally, we discuss why the number of iterations of the binary search is polynomial in $\frac{1}{\delta}$ (and, thus, the running time of the entire procedure is polynomial in the encoding length of k-SAP and $\frac{1}{\delta}$).

Observe that by choosing $\bar{q}_j := \max_{t \in T(j)} p_t$ for all $j \in \mathcal{B}$, $\bar{\lambda}_i := 0$ for all $i \in \mathcal{I}$, and $\bar{v} := \sum_{j \in \mathcal{B}} \bar{q}_j$, we obtain a feasible solution for (DLP) with objective value $\bar{v}$, and we can apply the binary search with the initial interval $[0, \bar{v}]$ in order to obtain the value v^*.

As stated above, the procedure is supposed to guarantee a precision of $\delta' = \delta \cdot \textsc{opt}_{\text{LP}}$, i.e., it terminates as soon as the width of the current interval in the binary search drops below δ'. Hence, the number of iterations needed is the smallest integer i such that

$$\left(\frac{1}{2}\right)^i \bar{v} \leq \delta \cdot \textsc{opt}_{\text{LP}},$$

which is equivalent to

$$i \geq \log\left(\frac{1}{\delta}\right) + \log\left(\frac{\bar{v}}{\textsc{opt}_{\text{LP}}}\right).$$

Note that we can assume that $\text{OPT}_{\text{LP}} \geq 1$.[4] Since $(\bar{q}, \bar{\lambda}, \bar{v})$ is feasible for (DLP), it holds that $\bar{v} \geq \text{OPT}_{\text{LP}}$, and we have

$$\log\left(\frac{\bar{v}}{\text{OPT}_{\text{LP}}}\right) \leq \log(\bar{v}) = \log\left(\sum_{j\in\mathcal{B}} \max_{t\in T(j)} p_t\right) \leq \log(|\mathcal{B}|) + \log\left(\max_{i\in\mathcal{I}, j\in\mathcal{B}} p_{ij}\right), \tag{4.8}$$

which is bounded by a polynomial in the encoding length of the problem.

Since $\log\left(\frac{1}{\delta}\right) \in \mathcal{O}\left(\frac{1}{\delta}\right)$ for $\delta \to 0$, we thus have that the number of iterations of the binary search is bounded by a polynomial in the encoding length of the problem and $\frac{1}{\delta}$. □

Lemma 4.8. *If there exists a β-approximation algorithm for the single bin subproblem, then there exists a β-approximate separation algorithm for $\mathcal{P}_j$.*

Proof. Given a β-approximation algorithm for the single bin subproblem for bin j, we construct an approximate separation algorithm for $\mathcal{P}_j$ as follows:

If $q_j < 0$, the set $t = \emptyset$ yields a violated constraint. Otherwise, we define profits $p'_{ij} := p_{ij} - \lambda_i$ and we use the β-approximation algorithm for solving the single bin subproblem for bin j with profits p'. This yields a set $t^* \in T(j)$ such that

$$q_j^* := \sum_{i\in t^*} p'_{ij} = \sum_{i\in t^*} (p_{ij} - \lambda_i) \geq \beta \sum_{i\in t} (p_{ij} - \lambda_i) \text{ for all } t \in T(j).$$

If $q_j^* > q_j$, the set t^* yields a violated constraint since

$$q_j < q_{j^*} = \sum_{i\in t^*} (p_{ij} - \lambda_i) \iff q_j + \sum_{i\in t^*} \lambda_i < p_{t^*}.$$

Otherwise, we know that $(\frac{q}{\beta}, \lambda)$ is feasible since

$$q_j \geq q_j^* \geq \beta \sum_{i\in t} (p_{ij} - \lambda_i) \text{ for all } t \in T(j).$$

□

By Lemma 4.7 and Lemma 4.8, Theorem 4.6 follows immediately.

[4] If an item i is accepted in a bin j where it yields a non-positive profit $p_{ij} \leq 0$, we can remove it from the bin and remain with a configuration which is, by assumption, still feasible and yields at least the same profit. Thus, the only possibility for $\text{OPT} < 1$ is that $p_{ij} = 0$ for all $i \in \mathcal{I}, j \in B$, which can easily be checked beforehand.

The Randomized Rounding Step

In our randomized rounding procedure, we first use the given $(\beta - \delta)$-approximation to obtain a fractional solution $x^{\mathrm{LP}} \in [0,1]^{|T|}$. We then proceed as described in Section 1.1.3 and choose for each bin j a configuration independently at random, where we choose configuration $t \in T(j)$ with probability x_t^{LP}. The resulting vector $x^{\mathrm{IP}} \in \{0,1\}^{|T|}$, where $x_t^{\mathrm{IP}} = 1$ if and only if configuration t was selected, then satisfies constraint (4.1b), and we have for the expected profit of x^{IP} that

$$\mathbb{E}\left[\textsc{profit}(x^{\mathrm{IP}})\right] = \mathbb{E}\left[\sum_{t\in T} p_t x_t^{\mathrm{IP}}\right] = \sum_{t\in T} p_t \mathbb{E}\left[x_t^{\mathrm{IP}}\right] = \sum_{t\in T} p_t x_t^{\mathrm{LP}} \geq (\beta - \delta)\cdot \textsc{opt}_{\mathrm{LP}}, \quad (4.9)$$

where we denote by $\textsc{opt}_{\mathrm{LP}}$ the optimal solution of (LP), and write for short $\textsc{profit}(x) := \sum_{t\in T} p_t x_t$. We summarize this in the following:

Observation 4.9. *The vector $x^{\mathrm{IP}} \in \{0,1\}^{|T|}$ obtained from the randomized rounding process satisfies* (4.1b) *and $\mathbb{E}[\textsc{profit}(x^{\mathrm{IP}})] \geq (\beta - \delta)\cdot \textsc{opt}_{\mathrm{LP}}$.*

Recall that x^{IP} is, in general, not a feasible solution to (IP) since it may violate constraint (4.1c) (an item might be assigned to more than k bins).

4.2.2 Obtaining a Feasible Solution

In order to turn x^{IP} into a feasible solution of (IP), we need to remove copies of item i from the bins in case item i is assigned to more than k bins. Hence, we simply remove each item i that is assigned to more than k bins from all but the k bins with highest profit p_{ij} it is assigned to. Doing so for all items yields a feasible solution $\bar{x}^{\mathrm{IP}}$ of (IP).

In the following, we let $i \in \mathcal{I}$ be an arbitrary but fixed item. We denote by $y_j := \sum_{t\in T(j):i\in t} x_t^{\mathrm{LP}}$ the probability that item i is assigned to bin j, and sort the bins in non-increasing order of p_{ij}, i.e., $p_{i1} \geq p_{i2} \geq \cdots \geq p_{im}$. The profit obtained from item i in the fractional solution x^{LP} is then given as

$$\sum_{j=1}^{m} y_j p_{ij}. \quad (4.10)$$

In order to compute the expected profit of the solution $\bar{x}^{\mathrm{IP}}$ obtained after removing excess copies of items from the bins, we let $P_i(a,b)$ denote the expected profit obtained from item i (in $\bar{x}^{\mathrm{IP}}$) in the bins $b, b+1, \ldots, m$ conditioned on item i being contained in $k-a$ of the bins $1,\ldots,b-1$, i.e.,

$$P_i(a,b) := \mathbb{E}[\text{Profit of item } i \text{ in bins } b,\ldots,m \mid \text{item } i \text{ is assigned to } k-a \text{ of the bins } 1,\ldots,b-1].$$

Note that, since a configuration for each bin is chosen independently of the choices for all other bins, $P_i(a, b)$ can also be interpreted as the expected profit obtained from item i in the bins $b, b+1, \ldots, m$ given that item i may be assigned to at most a of these bins. The total expected profit of $\bar{x}^{\mathrm{IP}}$ can now be written as

$$\sum_{i \in \mathcal{I}} P_i(k, 1). \tag{4.11}$$

We now prove the main result for obtaining the approximation guarantee of our algorithm:

Lemma 4.10. *For each item i and all $1 \leq a \leq k$, $1 \leq b \leq m$, we have*

$$P_i(a, b) \geq \left(1 - \frac{1}{e^a} + \frac{1}{32m^2 e^{a-1}}\right) \cdot \sum_{j=b}^{m} y_j p_{ij}.$$

Proof. We fix an item i and prove the statement by induction on a.

For $a = 1$, the statement follows by arguments similar to the ones given in (Fleischer et al., 2011):

With probability y_b, the configuration assigned to bin b contains item i, so the profit obtained from item i is p_{ib}. Recall that the bins' configurations are chosen independently. Thus, with probability $(1 - y_b)y_{b+1}$, the configuration assigned to bin b does not contain item i but the configuration assigned to bin $b + 1$ contains item i. So the profit obtained from item i is in this case $p_{i,b+1}$. Proceeding in this way, we obtain that

$$P_i(1, b) = y_b p_{ib} + (1 - y_b)y_{b+1} p_{i,b+1} + \cdots + \left(\prod_{j=b}^{m-1} (1 - y_j)\right) \cdot y_m p_{im}. \tag{4.12}$$

The arithmetic-geometric mean inequality (cf. Goemans and Williamson (1994)) and the fact that

$$1 - \left(1 - \frac{1}{t}\right)^t \geq 1 - \frac{1}{e} + \frac{1}{32t^2} \tag{4.13}$$

for all $t \geq 1$ (cf. Nutov et al. (2006)) imply that (4.12) is at least

$$\begin{aligned}
&\left(1 - \left(1 - \frac{1}{m - b + 1}\right)^{m-b+1}\right) \cdot \left(\sum_{j=b}^{m} y_j p_{ij}\right) \\
\geq\ &\left(1 - \frac{1}{e} + \frac{1}{32(m - b + 1)^2}\right) \cdot \left(\sum_{j=b}^{m} y_j p_{ij}\right) \\
\geq\ &\left(1 - \frac{1}{e} + \frac{1}{32m^2}\right) \cdot \left(\sum_{j=b}^{m} y_j p_{ij}\right),
\end{aligned}$$

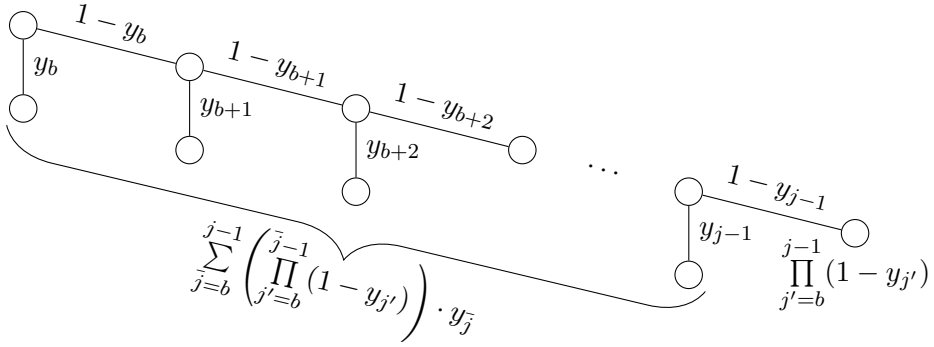


Figure 4.1: Illustration of Equation (4.16) – the sum of all probabilities for reaching the leaves of the binary tree equals 1.

which proves the statement for $a = 1$.

Now let $a \geq 2$ and assume that the statement holds for $a-1$. Then the definition of $P_i(a,b)$ implies that for each $1 \leq b \leq m$:

$$\begin{aligned} P_i(a,b) &= \sum_{j=b}^{m}\left[\left(\prod_{j'=b}^{j-1}(1-y_{j'})\right)\cdot y_j\cdot\Big(p_{ij}+P_i(a-1,j+1)\Big)\right] \\ &\geq \sum_{j=b}^{m}\left[\left(\prod_{j'=b}^{j-1}(1-y_{j'})\right)\cdot y_j\cdot\left(p_{ij}+\left(1-\frac{1}{e^{a-1}}\right)\cdot\left(\sum_{t=j+1}^{m}y_t p_{it}\right)\right)\right]. \end{aligned} \tag{4.14}$$

Here, the term $y_j p_{ij}$, for a fixed j, is summed up once with coefficient $\prod_{j'=b}^{j-1}(1-y_{j'})$ and, for every $\bar{j} \leq j-1$, with coefficient $\left(\prod_{j'=b}^{\bar{j}-1}(1-y_{j'})\right)\cdot y_{\bar{j}}\cdot\left(1-\frac{1}{e^{a-1}}\right)$. Hence, (4.14) can be rewritten as

$$\begin{aligned} &\sum_{j=b}^{m}\left[\prod_{j'=b}^{j-1}(1-y_{j'})+\sum_{\bar{j}=b}^{j-1}\left(\left(\prod_{j'=b}^{\bar{j}-1}(1-y_{j'})\right)\cdot y_{\bar{j}}\cdot\left(1-\frac{1}{e^{a-1}}\right)\right)\right]\cdot y_j p_{ij} \\ &= \sum_{j=b}^{m}\left[\left(1-\frac{1}{e^{a-1}}+\frac{1}{e^{a-1}}\right)\prod_{j'=b}^{j-1}(1-y_{j'})\right. \\ &\qquad\left.+\sum_{\bar{j}=b}^{j-1}\left(\left(\prod_{j'=b}^{\bar{j}-1}(1-y_{j'})\right)\cdot y_{\bar{j}}\cdot\left(1-\frac{1}{e^{a-1}}\right)\right)\right]\cdot y_j p_{ij}. \end{aligned} \tag{4.15}$$

As illustrated in Figure 4.1, it holds that

$$\left(\prod_{j'=b}^{j-1}(1-y_{j'})\right)+\sum_{\bar{j}=b}^{j-1}\left(\left(\prod_{j'=b}^{\bar{j}-1}(1-y_{j'})\right)\cdot y_{\bar{j}}\right)=1, \tag{4.16}$$

and, thus, expression (4.15) simplifies to

$$\sum_{j=b}^{m}\left[\left(1-\frac{1}{e^{a-1}}\right)+\frac{1}{e^{a-1}}\cdot\prod_{j'=b}^{j-1}(1-y_{j'})\right]\cdot y_j p_{ij}$$
$$=\left(1-\frac{1}{e^{a-1}}\right)\cdot\left(\sum_{j=b}^{m}y_j p_{ij}\right)+\frac{1}{e^{a-1}}\cdot\sum_{j=b}^{m}\prod_{j'=b}^{j-1}(1-y_{j'})\cdot y_j p_{ij}. \tag{4.17}$$

By using the arithmetic/geometric mean inequality and estimate (4.13) as before, expression (4.17) is at least

$$\left(1-\frac{1}{e^{a-1}}\right)\cdot\left(\sum_{j=b}^{m}y_j p_{ij}\right)+\frac{1}{e^{a-1}}\cdot\left(1-\left(1-\frac{1}{m-b+1}\right)^{m-b+1}\right)\cdot\left(\sum_{j=b}^{m}y_j p_{ij}\right)$$
$$\geq\left(1-\frac{1}{e^{a-1}}\right)\cdot\left(\sum_{j=b}^{m}y_j p_{ij}\right)+\frac{1}{e^{a-1}}\cdot\left(1-\frac{1}{e}+\frac{1}{32m^2}\right)\cdot\left(\sum_{j=b}^{m}y_j p_{ij}\right)$$
$$=\left(1-\frac{1}{e^{a}}+\frac{1}{32m^2e^{a-1}}\right)\cdot\left(\sum_{j=b}^{m}y_j p_{ij}\right).$$ □

Using (4.10) for the profit obtained from item i in x^{LP} and (4.11) for the expected profit of $\bar{x}^{\mathrm{IP}}$ together with Lemma 4.10 with $a := k$ and $b := 1$, we obtain

$$\begin{aligned}\mathbb{E}\left[\textsc{profit}(\bar{x}^{\mathrm{IP}})\right]&=\sum_{i\in\mathcal{I}}P_i(k,1)\\&\geq\left(1-\frac{1}{e^k}+\frac{1}{32m^2e^{k-1}}\right)\cdot\underbrace{\sum_{i\in\mathcal{I}}\sum_{j=1}^{m}y_j p_{ij}}_{=\mathbb{E}[\textsc{profit}(x^{\mathrm{IP}})]}\\&\overset{(4.9)}{\geq}\left(1-\frac{1}{e^k}+\frac{1}{32m^2e^{k-1}}\right)\cdot(\beta-\delta)\cdot\mathrm{OPT}_{\mathrm{LP}}.\\&\geq\left(\left(1-\frac{1}{e^k}\right)\beta+\left(\frac{\beta}{32m^2e^{k-1}}-\delta\right)\right)\cdot\mathrm{OPT}_{\mathrm{LP}}.\end{aligned}\tag{4.18}$$

Since, by Theorem 4.6, the time needed to obtain a $(\beta-\delta)$-approximate solution to the linear programming relaxation is polynomial in $\frac{1}{\delta}$, we can choose $\delta := \frac{\beta}{32m^2e^{k-1}}$ and obtain the following result:

Theorem 4.11. *There exists a $((1-\frac{1}{e^k})\beta)$-approximation algorithm for k-SAP whenever there exists a β-approximation algorithm for the single bin subproblem.*

The result from Theorem 4.11 can be slightly improved in case that there exists an FTPAS for the single bin subproblem. In this case, choosing $\beta := 1-\epsilon$

in (4.18) shows that there exists an algorithm with running time polynomial in the encoding length of the given instance of k-SAP and $\frac{1}{\epsilon}$ that achieves (for $\delta := \frac{1-\epsilon}{64m^2e^{k-1}}$) an approximation guarantee of

$$\left(1 - \frac{1}{e^k} + \frac{1}{64m^2e^{k-1}}\right)(1-\epsilon) \geq 1 - \frac{1}{e^k} + \frac{1}{64m^2e^{k-1}} - \epsilon.$$

Hence, if we choose $\epsilon := \frac{1}{64m^2e^{k-1}}$, we obtain an approximation factor of $(1-\frac{1}{e^k})$. Since we have for this choice of ϵ that $\frac{1}{\epsilon}$ is polynomial in the encoding length of k-SAP for any fixed k, we obtain the following result:

Corollary 4.12. *There exists a $(1-\frac{1}{e^k})$-approximation algorithm for k-SAP whenever there exists an FPTAS for the single bin subproblem.*

4.3 Derandomization

Nutov et al. (2006) showed that it is possible to use derandomization to obtain a deterministic approximation algorithm for classical GAP under the assumption that the profit of each item is independent of the bin it is assigned to. However, since GAP only allows that every item is assigned at most once, their analysis does not carry over to k-SAP.

Using the method of conditional expectations (cf. (Alon and Spencer, 1992)), we show how we can derandomize our approximation algorithm of the previous section for k-SAP if the profits are bin-independent, i.e., $p_{ij} = p_i$ for all items i and bins j.

For each bin j, we denote by $T^+(j) := \{t \in T(j) : x_t^{\mathrm{LP}} > 0\}$ the configurations of bin j that have a positive value in the (approximate) solution x^{LP}. Note that, since x^{LP} was obtained (from an approximation algorithm) in polynomial time, the size of $T^+(j)$ is also polynomially bounded.

In the following, we denote our randomized rounding algorithm from the previous section by ALG and write $\mathbb{E}\left[\mathrm{ALG} \mid t^1, \ldots, t^l\right]$ for the expected total profit obtained by ALG given that configurations $t^1, \ldots, t^l$ are chosen for bins $1, \ldots, l$, respectively. We show that it is possible to choose $t^{l+1} \in T^+(l+1)$ in polynomial time such that

$$\mathbb{E}\left[\mathrm{ALG} \mid t^1, \ldots, t^{l+1}\right] \geq \mathbb{E}\left[\mathrm{ALG} \mid t^1, \ldots, t^l\right].$$

First, we consider the case $l = 0$. By definition of the conditional expectation, we have

$$\mathbb{E}\left[\mathrm{ALG}\right] = \sum_{t^1 \in T^+(1)} \mathbb{E}\left[\mathrm{ALG} \mid t^1\right] \cdot x_{t^1}^{\mathrm{LP}}.$$

The fact that $\sum_{t^1 \in T^+(1)} x_{t^1}^{\mathrm{LP}} \overset{(4.1b)}{=} 1$ and $x_t^{\mathrm{LP}} \geq 0$ for all $t \in T$ then implies that there exists some $\overline{t^1} \in T^+(1)$ with $\mathbb{E}\left[\textsc{alg} \mid \overline{t^1}\right] \geq \mathbb{E}\left[\textsc{alg}\right]$. In the following, we show how to determine $\overline{t^1}$ in polynomial time.

Since ALG chooses the bins' configurations independently at random, it holds for each item i and for each $q \in \{0, \ldots, k\}$ that

$$\begin{aligned} &\Pr(\text{item } i \text{ is contained in exactly } q \text{ of the bins } 2, \ldots, m) \\ &= \sum_{\substack{\mathcal{B}' \subseteq \{2,\ldots,m\}: \\ |\mathcal{B}'|=q}} \left(\prod_{j \in \mathcal{B}'} y_j \prod_{j \in \{2,\ldots,m\} \setminus \mathcal{B}'} (1 - y_j) \right), \end{aligned} \tag{4.19}$$

where we denote, as in Section 4.2.2, by $y_j = \sum_{t \in T^+(j): i \in t} x_t^{\mathrm{LP}}$ the probability that item i is assigned to bin j in the randomized rounding procedure.

Note that the computation of each summand in (4.19) requires $\mathcal{O}(m)$ multiplications, and since there are $\binom{m-1}{q} \in \mathcal{O}(m^q)$ summands in total, determining (4.19) takes in total $\mathcal{O}(m^{q+1})$. This implies for each item i that

$$\begin{aligned} &\Pr(\text{item } i \text{ is contained in} \leq k-1 \text{ of the bins } 2, \ldots, m) \\ &= \sum_{q=0}^{k-1} \Pr(\text{item } i \text{ is contained in exactly } q \text{ of the bins } 2, \ldots, m) \end{aligned} \tag{4.20}$$

and

$$\begin{aligned} &\Pr(\text{item } i \text{ is contained in} \geq k \text{ of the bins } 2, \ldots, m) \\ &= 1 - \sum_{q=0}^{k-1} \Pr(\text{item } i \text{ is contained in exactly } q \text{ of the bins } 2, \ldots, m) \end{aligned} \tag{4.21}$$

can be computed in $\sum_{q=0}^{k-1} \mathcal{O}(m^{q+1}) \in \mathcal{O}(m^k)$.

In the following, let $t^1 \in T^+(1)$ be an arbitrary but fixed configuration chosen for bin 1, while the remaining configurations for bins $2, \ldots, m$ are chosen randomly.

If an item i is then contained in $q \leq k$ bins, it remains assigned to these q bins and contributes $q \cdot p_i$ to the total profit. If it is assigned $q > k$ times, it is deleted from $q - k$ bins and yields a profit of $k \cdot p_i$. Observe that this is a crucial point in the analysis where we make use of our assumption $p_{ij} = p_i$. Altogether, we thus have:

$$\begin{aligned} &\mathbb{E}\left[\textsc{alg} \mid t^1\right] \\ &= \sum_{i \in \mathcal{I}} \sum_{q=0}^{m} \Pr(i \text{ contained in exactly } q \text{ of the bins } 1, \ldots, m \mid t^1) \cdot \min\{q, k\} \cdot p_i \end{aligned}$$

$$
\begin{aligned}
&= \sum_{i \in t^1} \sum_{q=0}^{m-1} \Pr(i \text{ contained in exactly } q \text{ of the bins } 2, \dots, m) \cdot \min\{q+1, k\} \cdot p_i \\
&\quad + \sum_{i \in \mathcal{I} \setminus t^1} \sum_{q=0}^{m} \Pr(i \text{ contained in exactly } q \text{ of the bins } 2, \dots, m) \cdot \min\{q, k\} \cdot p_i \\
&= \sum_{i \in t^1} p_i \cdot \Bigg(\sum_{q=0}^{k-1} [\Pr(i \text{ contained in exactly } q \text{ of the bins } 2, \dots, m) \cdot (q+1)] \\
&\qquad\qquad + \Pr(i \text{ contained in} \geq k \text{ of the bins } 2, \dots, m) \cdot k \Bigg) \\
&\quad + \sum_{i \in \mathcal{I} \setminus t^1} p_i \cdot \Bigg(\sum_{q=0}^{k-1} [\Pr(i \text{ contained in exactly } q \text{ of the bins } 2, \dots, m) \cdot q] \\
&\qquad\qquad + \Pr(i \text{ contained in} \geq k \text{ of the bins } 2, \dots, m) \cdot k \Bigg).
\end{aligned}
$$

We argued that (4.20) and (4.21) can be computed in $\mathcal{O}(m^k)$ for every item i, and, therefore, $\mathbb{E}\left[\text{ALG} \mid t^1\right]$ can be determined in $\mathcal{O}(nm^k)$. Recall that k is a fixed parameter that is not part of the input.

As we already stated before, each set $T^+(j)$ has polynomial size, and, thus, we can determine $\overline{t^1}$ in polynomial time by comparing $\mathbb{E}\left[\text{ALG} \mid t^1\right]$ for all $t^1 \in T^+(1)$.

For $l > 0$, the argument follows exactly along the same lines. We have

$$
\mathbb{E}\left[\text{ALG} \mid t^1, \dots, t^l\right] = \sum_{t^{l+1} \in T^+(l+1)} \mathbb{E}\left[\text{ALG} \mid t^1, \dots, t^{l+1}\right] \cdot x^{\text{LP}}_{t^{l+1}},
$$

and know that there exists $\overline{t^{l+1}} \in T^+(l+1)$ such that

$$
\mathbb{E}\left[\text{ALG} \mid t^1, \dots, t^l, \overline{t^{l+1}}\right] \geq \mathbb{E}\left[\text{ALG} \mid t^1, \dots, t^l\right],
$$

which we can determine analogously to the case $l = 0$ in polynomial time.

By selecting a configuration for each bin $1, \dots, m$ with this iterative procedure, we deterministically obtain a feasible integral solution with profit at least $\mathbb{E}\left[\text{ALG}\right]$. By Theorem 4.11, we obtain the following result:

Theorem 4.13. *If all items' profits are independent of the bin they are assigned to ($p_{ij} = p_i$), there exists a deterministic $((1 - \frac{1}{e^k})\beta)$-approximation algorithm for k-SAP whenever there exists a deterministic β-approximation algorithm for the single bin subproblem.*

4.4 Generalizations

A natural generalization of k-SAP is that we are given a different number $k_i \geq 1$ for each item $i \in \mathcal{I}$ that specifies the maximum number of bins it may be assigned to, as summarized in the following program:

$$\max \sum_{t \in T} p_t x_t \tag{4.22a}$$

$$\text{s.t.} \sum_{t \in T(j)} x_t = 1 \qquad \forall\, j \in \mathcal{B} \tag{4.22b}$$

$$\sum_{t \in T : i \in t} x_t \leq k_i \quad \forall\, i \in \mathcal{I} \tag{4.22c}$$

$$x_t \in \{0, 1\} \qquad \forall\, t \in T. \tag{4.22d}$$

Theorem 4.6 still applies to this more general program (4.22), i.e., we can obtain a $(\beta - \delta)$-approximation for the linear programming relaxation of (4.22) if we are given a β-approximation for the single bin subproblem: in the proof given in Section 4.2.1 we simply need to change the objective function of (DLP) to $\sum_{j \in \mathcal{B}} q_j + \sum_{i \in \mathcal{I}} k_i \cdot \lambda_i$, which has no influence on the validity of Lemma 4.7 and Lemma 4.8. Thus, we can obtain a fractional solution and perform, as before, the randomized rounding.

If we remove each item i that is now assigned to more than k_i bins from all but the k_i bins with highest profit p_{ij} it is assigned to, we obtain a feasible integral solution $\bar{x}^{\mathrm{IP}}$. Moreover, the analysis from Section 4.2.2 immediately implies that, in expectation, the profit obtained from each item i in $\bar{x}^{\mathrm{IP}}$ is at least $(1 - \frac{1}{e^{k_i}})$ times the profit obtained from item i in the fractional solution x^{LP}. Hence, given a β-approximation for the single bin subproblem, our method yields a $((1 - \frac{1}{e^k})\beta)$-approximation for this case, where $k := \min_{i \in I} k_i$.

4.5 Conclusions

In this chapter we analyzed the problem k-SAP. We showed how the approximation algorithm by Fleischer et al. (2011) for $k = 1$ can be extended to this more general setting where each item can be assigned to at most k different bins yielding an approximation factor of $((1 - \frac{1}{e^k})\beta)$ whenever the single bin subproblem admits a β-approximation. We furthermore identified a special case where the approximation algorithm can even be derandomized.

Although solving k-SAP is, in general, not easier than solving SAP, we can obtain a $(1 - \frac{1}{e^k})$-approximation if there exists an FPTAS for the single bin subproblem. This shows that the inapproximability result known for SAP (Theorem 4.2) does not apply to k-SAP with $k \geq 2$.

It remains an open question whether a tight upper bound on the approximability of k-SAP can be proven or if there exist (deterministic) approximation algorithms with better approximation factors.

— 5 —

The Online Interval Scheduling Problem with Bounded Number of Failures

Scheduling is concerned with the allocation of activities (jobs) to scarce resources (machines). Besides the assignment decision, in most scheduling problems the scheduler also has to choose the start times of the jobs such that some given objective function is optimized. In contrast to this, we consider an *interval scheduling problem* (also known as *fixed job scheduling problem* or *k-track assignment problem*) in which fixed start and end times for all jobs (or *intervals*) are given. The task of the scheduler then consists of deciding which subset of the intervals should be accepted and to which machines these intervals should be assigned. The goal is to maximize the number of the accepted intervals subject to the constraint that no two intervals assigned to the same machine overlap.

Problems of this kind have many applications in fields where time-critical tasks have to be managed, e.g., in steel production, crew planning, and bandwidth allocation (cf. (Kolen et al., 2007; Kovalyov et al., 2007)).

Interval scheduling is a special case of the separable assignment problem (SAP) we studied in Chapter 4: Every interval (item) yields a profit of 1 if it is assigned to a machine (bin). The set of feasible packings for a machine is then given as all subsets of non-overlapping intervals.

We consider an online variant of interval scheduling in which an online algorithm initially has knowledge about the characteristics of all intervals. However, some of them might fail, i.e., they cannot be scheduled, and an online algorithm learns about the failure of an interval not before its start time. We show how to obtain competitive online algorithms and lower bounds on the competitive ratio for the case that we are given an upper bound k on the number of failing intervals. This online model is similar to a well-known online version of the shortest path problem, known as the *k-Canadian traveller problem*, where one is initially given a complete instance of the shortest path problem and the information that up to k (initially unknown) edges of the graph may be blocked. This problem will be studied in Chapter 6. In contrast to most previous work on online interval scheduling, we do not

need the possibility to abort accepted intervals in order to obtain a bounded competitiveness. Thus, our setting allows to model many situations where the acceptance of a job can be seen as a commitment of the scheduler to a customer and aborting a job is infeasible.

Previous Work

Surveys of different variants of interval scheduling problems studied in literature can be found in (Kolen et al., 2007; Kovalyov et al., 2007).

The offline problem on a single machine can be modeled as a shortest path problem or solved by the earliest finishing time rule in $\mathcal{O}(n \log n)$ time (cf. (Gupta et al., 1979)). The case of multiple identical machines can be solved using a minimum-cost flow formulation (Arkin and Silverberg, 1987; Bouzina and Emmons, 1996). If each interval may only be assigned to a given subset of the machines or the machines have different speeds, the problem is known to be $\mathcal{NP}$-hard (Arkin and Silverberg, 1987; Krumke et al., 2011), but there exists a $(1-\frac{1}{e})$-approximation algorithm even for unrelated machines (Fleischer et al., 2011).

There is also a lot of research on online interval scheduling. In contrast to our setting, most problems studied in previous work have in common that intervals arrive online (i.e., an online algorithm does not know about the existence of an interval before its start time) and it is allowed to abort an accepted interval during its execution (in which case the aborted interval is lost). If the goal is to maximize the number of accepted intervals, this problem can be solved optimally on a single machine or multiple identical machines by emulating the earliest finishing time rule (Carlisle and Lloyd, 1995; Faigle and Nawjin, 1995).

For the weighted version of the problem, where each interval has a positive weight and the goal is to maximize the total weight of accepted intervals, Woeginger (1994) showed that it is, in general, not possible to obtain deterministic algorithms with a bounded competitiveness even for a single machine. Later, Canetti and Irani (1998) showed that the same also holds for randomized algorithms. If a relation between the weights and lengths is imposed on the intervals (which is given by some benevolent function), Woeginger (1994) proposed a 4-competitive algorithm for the single machine problem and proved a matching lower bound. While this settles the deterministic case, it was shown that it is possible to improve upon this ratio using randomized algorithms (Epstein and Levin, 2010; Fung et al., 2009; Miyazawa and Erlebach, 2004; Seiden, 1998).

Lipton and Tomkins (1994) consider the single-machine problem in the case where aborting an accepted interval during its execution is not allowed. If the weight of an interval equals its length and there is an upper bound Δ on the ratio of the longest to the shortest interval, they provide a randomized

$\mathcal{O}(\log \Delta^{1+\epsilon})$-competitive algorithm for every $\epsilon > 0$ and prove a lower bound of $\Omega(\log \Delta)$ on the competitive ratio of randomized algorithms.

There are only a few publications dealing with multiple machines. The case of identical machines was studied in (Fung et al., 2008, 2012). Recently, online algorithms and lower bounds for the case of related machines have been proposed in (Epstein et al., 2013; Krumke et al., 2011).

Chapter Outline

In this chapter we study a novel online version of the interval scheduling problem. In Section 5.1 we formally define the problem k-OIS. In Sections 5.2 and 5.3 we provide several upper and lower bounds on the competitive ratio achievable by deterministic and randomized online algorithms for this problem. We summarize these results in Table 5.1 (where, as in the rest of the chapter, $\log(x)$ denotes the binary logarithm of x).

		$m = 1$		$m > 1$	
deterministic	l.b.	k	(Proposition 5.3)	$\Omega(\sqrt[m]{k})$	(Theorem 5.12)
	u.b.	$k+1$	(Theorem 5.7)	$4 + {}^{(k-3)}/_{m}$	(Theorem 5.11)
randomized	l.b.	$\Omega(\log k)$	(Theorem 5.13)	$\Omega(\log({}^{k}/_{m}))$	(Theorem 5.13)
	u.b.*	$\log(k+2)$	(Theorem 5.15)	$\log(k+2)$	(Theorem 5.21)

Table 5.1: Summary of our upper bounds (u.b.) and lower bounds (l.b.) on the competitive ratio for m machines (the results marked with a $\star$ are for the laminar case, see Definition 5.14).

Moreover, we remark in Section 5.4 that, for the weighted version of the problem, no deterministic online algorithm can achieve a bounded competitiveness even on a single machine. However, a competitive ratio of $(k+1)\cdot \frac{w^{\max}}{w^{\min}}$ can be achieved on a single machine if the weights of all intervals are between $w^{\min}$ and $w^{\max}$. Finally, we show how to obtain an online algorithm for our problem in the environment of related machines.

5.1 Problem Definition and Preliminaries

We consider the problem of scheduling intervals on m identical machines. An interval i is given by a *release date* $r_i \geq 0$ and a *processing requirement* (or *length*) $p_i > 0$. If an interval i is accepted, it must be assigned to start immediately at time r_i on a machine that is currently not occupied and is completed after p_i time units, i.e., it finishes at time $r_i + p_i$. The goal is to maximize the number of accepted intervals subject to the constraint that no two intervals assigned to the same machine overlap. This means that the open intervals $(r_{i_1}, r_{i_1} + p_{i_1})$ and $(r_{i_2}, r_{i_2} + p_{i_2})$ must not intersect whenever two intervals i_1 and i_2 are assigned to the same machine j.

In our online setting, we are initially given a set $\mathcal{I} = \{1, \dots, n\}$ of intervals with release dates and processing requirements and the information that up to k of these intervals might fail, i.e., they cannot be accepted. Hence, an instance $\sigma = (\mathcal{I}, F)$ of the problem is defined by the set $\mathcal{I}$ of intervals and a subset $F \subseteq \mathcal{I}$ of failing intervals with $|F| \leq k$. An online algorithm initially knows the set $\mathcal{I}$ of intervals and the upper bound k on the number of failures, but only learns that an interval i fails at the time r_i when it is supposed to be released. Once an interval is accepted on some machine, it may not be aborted, preempted, or moved to another machine. We refer to this problem as the *online interval scheduling problem with at most k failures* and write for short k-OIS. This is summarized in the following:

Definition 5.1 (k-OIS)**.**
An instance of k-OIS is given by a set of intervals $\mathcal{I} = \{1, \dots, n\}$ with release dates r_i and processing requirements p_i for $i \in \mathcal{I}$, a subset $F \subseteq \mathcal{I}$ of failing intervals with $|F| \leq k$, and a number of identical machines m.

The task is to maximize the number of accepted intervals and to assign them to one of m identical machines such that no two intervals assigned to the same machine overlap. An online algorithm learns that an interval $i \in \mathcal{I}$ fails at time r_i when it is supposed to be released, and for every non-failing interval the algorithm has to take an irrevocable decision whether to accept it (and to which machine it should be assigned) or to reject it.

An offline algorithm knows the complete set F of failing intervals in advance and can compute an optimal solution for the remaining intervals in $\mathcal{I} \setminus F$. As we already stated, the single machine case can be modeled as a shortest path problem (see Figure 5.1 for an illustration). Since interval scheduling is a special case of SAP, this implies that one can obtain a $(1 - \frac{1}{e})$-approximation for multiple unrelated machines (see Section 4.2).

We measure the quality of online algorithms for k-OIS by means of competitive analysis (see Section 1.1.4). For an instance σ, the profit (number of accepted intervals) obtained by an online algorithm ALG is denoted by $\text{ALG}(\sigma)$, and $\text{OPT}(\sigma)$ is the optimal profit achievable on this instance. A deterministic online algorithm ALG is c-competitive for k-OIS if $\text{OPT}(\sigma) \leq c \cdot \text{ALG}(\sigma)$ for every instance σ. Analogously, if ALG is a randomized online algorithm, it is c-competitive against an oblivious adversary if $\text{OPT}(\sigma) \leq c \cdot \mathbb{E}[\text{ALG}(\sigma)]$ for every instance σ.

In k-OIS, we assume that an online algorithm must not abort intervals. This assumption is reasonable since, otherwise, the online problem can be solved optimally as follows: an interval is accepted whenever the machine is idle, but it is dropped in favor of an overlapping interval that is released with earlier finishing time (cf. (Carlisle and Lloyd, 1995; Faigle and Nawjin, 1995)).

Proposition 5.2. *If abortion of intervals is allowed, k-OIS can be solved optimally.*

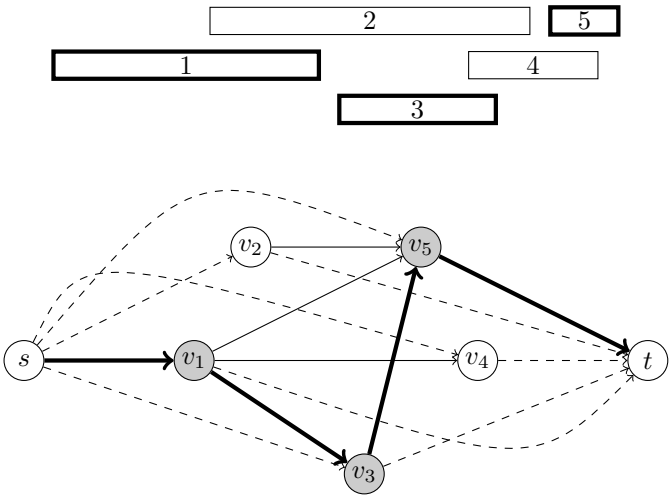


Figure 5.1: The offline interval scheduling problem for a single machine can be modeled as a shortest path problem. All arc costs are -1.

5.2 Deterministic Online Algorithms

5.2.1 A Single Machine

We start by providing a lower bound on the competitive ratio of deterministic online algorithms for the case of a single machine.

Proposition 5.3. *No deterministic online algorithm for k-*OIS *on a single machine ($m = 1$) can achieve a competitive ratio smaller than k.*

Proof. We consider an instance consisting of $k + 1$ intervals, where the first interval has release date $r_1 = 0$ and length $p_1 = 1$, and the remaining intervals $i \in \{2, \ldots, k + 1\}$ have release date $r_i = \frac{i-1}{k+1}$ and length $p_i = \frac{1}{k+1}$. This is illustrated in Figure 5.2.

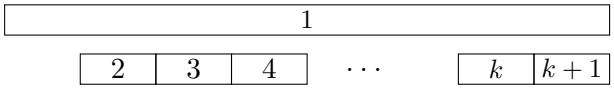


Figure 5.2: Illustration of the instance used in the proof of Proposition 5.3.

If the online algorithm ALG does not accept the first interval, the remaining k intervals fail and we have ALG $= 0$, whereas OPT would have accepted the first interval, i.e., OPT $= 1$.

Hence, in order to obtain a finite competitiveness, ALG has to accept the first interval. In this case, the remaining intervals do not fail. ALG cannot

accept any of these intervals since they overlap with the first one, so we have ALG $= 1$. An optimal offline algorithm, however, rejects the first interval and accepts intervals $2, \ldots, k+1$, which yields OPT $= k$. □

Remark 5.4. *Note that, for a single machine, we can assume that $r_{i_1} \neq r_{i_2}$ for all non-failing intervals i_1 and i_2 with $i_1 \neq i_2$. Otherwise, an online algorithm could simply discard all intervals with the same release date except for one with minimal length.*

In the following, we consider the algorithm GREEDY^1 (Algorithm 5.1) for a single machine that accepts an interval i if and only if the machine is available at time r_i and, until the finishing time $r_i + p_i$ of interval i, there are at most k intervals that are contained in i. We make this notion more precise:

Definition 5.5. *We say that an interval i' is* contained *in interval i if $r_i \leq r_{i'}$ and $r_i + p_i \geq r_{i'} + p_{i'}$, i.e., if $[r_{i'}, r_{i'} + p_{i'}] \subseteq [r_i, r_i + p_i]$. In this case, we also use the notation $i' \subseteq i$. Analogously, we write $i' \subsetneq i$ if $[r_{i'}, r_{i'} + p_{i'}] \subsetneq [r_i, r_i + p_i]$.*

Algorithm 5.1 GREEDY^1 for a single machine

1: Sort the intervals in non-decreasing order of release dates (for identical release dates in non-decreasing order of lengths).
2: **for all** intervals $i = 1, \ldots, n$ **do**
3: **if** interval i is non-failing, the machine is idle at the release date r_i of i, and i contains at most k intervals **then**
4: accept interval i
5: **end if**
6: **end for**

In the analysis below, we make use of the following argument: If an interval i that is accepted by OPT contains another non-failing interval $i' \subsetneq i$, then OPT can simply accept interval i' instead yielding the same number of accepted intervals. This directly yields the following observation:

Observation 5.6. *There always exists an optimal set $\mathcal{O}$ of non-failing intervals such that, for each interval $i \in \mathcal{O}$, all intervals that are contained in i fail.*

We now analyze the competitiveness of GREEDY^1:

Theorem 5.7. *Algorithm 5.1 is $(k+1)$-competitive for k-OIS on a single machine ($m = 1$).*

Proof. We consider an arbitrary instance of the problem and denote by $\mathcal{G}$ and $\mathcal{O}$ the sets of accepted intervals of GREEDY^1 and OPT, respectively, where the optimal set $\mathcal{O}$ is given as in Observation 5.6.

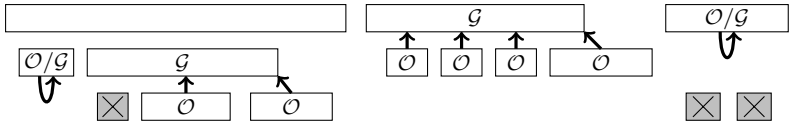


Figure 5.3: Illustration of the charging scheme used in the proof of Theorem 5.7 for $k = 3$. The intervals with labels $\mathcal{G}$ and $\mathcal{O}$ are accepted by GREEDY^1 and OPT, respectively. Failing intervals are crossed out.

In the following, we construct a mapping $\phi\colon \mathcal{O} \to \mathcal{G}$ such that each interval $i \in \mathcal{G}$ has at most $k+1$ preimages under ϕ.

For an interval $i \in \mathcal{O}$ accepted by OPT, GREEDY^1 can either accept or reject i. For rejection, there are two possibilities: Either i is rejected because GREEDY^1 was already processing another interval at the release date r_i, or because of the condition that it contains more than k intervals. The latter case, however, cannot occur: Since $\mathcal{O}$ is chosen as in Observation 5.6, every interval in $\mathcal{O}$ contains only failing intervals. Thus, since at most k intervals can fail in total, i can contain at most k intervals. Hence, if the machine is available at time r_i, GREEDY^1 accepts i.

We can now define the mapping ϕ as follows:

(i) For an interval $i \in \mathcal{O} \cap \mathcal{G}$, we set $\phi(i) := i$.

(ii) For an interval $i \in \mathcal{O} \setminus \mathcal{G}$, we know that i was rejected by GREEDY^1 because GREEDY^1 was already processing some other interval i' at time r_i. In this case, we set $\phi(i) := i'$.

An illustration of the mapping ϕ is given in Figure 5.3.

If $i \in \mathcal{O} \cap \mathcal{G}$, interval i has only itself as preimage under ϕ. In case that $i \in \mathcal{G} \setminus \mathcal{O}$, it has at most $k+1$ preimages under ϕ: Since i is accepted by GREEDY^1, it can contain at most k intervals that are accepted by OPT and mapped to i. Furthermore, there can be at most one additional interval $i \in \mathcal{O}$ that is mapped to i and starts before $r_i + p_i$ and finishes after $r_i + p_i$.

Thus, every interval $i \in \mathcal{G}$ has at most $k+1$ preimages under ϕ and the claim follows. □

Proposition 5.8. *The analysis in Theorem 5.7 is tight.*

Proof. Consider an instance consisting of $k+2$ non-failing intervals, where the first interval has release date $r_1 = 0$ and length $p_1 = 1$. The intervals $i \in \{2, \dots, k+1\}$ have release date $r_i = \frac{i-1}{k+2}$ and length $p_i = \frac{1}{k+2}$, and the last interval has release date $r_{k+2} = \frac{k+1}{k+2}$ and length $p_{k+2} = \frac{2}{k+2}$. This is illustrated in Figure 5.4.

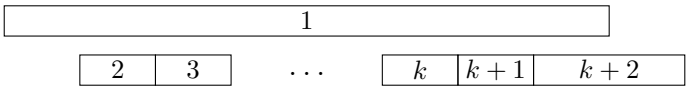


Figure 5.4: Illustration of the instance used in the proof of Proposition 5.8.

GREEDY^1 accepts only the first interval, whereas OPT accepts all other intervals $2, \ldots, k+2$, and we have OPT $= (k+1) \cdot$ GREEDY^1. □

Remark 5.9. *If all intervals have the same length, the analysis in the proof of Theorem 5.7 immediately implies that for every interval accepted by* GREEDY^1*,* OPT *can also accept at most one interval, i.e.,* GREEDY^1 *is 1-competitive.*

5.2.2 Multiple Machines

The idea of the single machine algorithm GREEDY^1 can also be used to obtain a competitive algorithm for multiple machines.

Our algorithm GREEDY^m runs m copies of (a slightly modified version of) GREEDY^1, one for each machine. If an interval i is released at time r_i, it is given to the first machine. If i is accepted and assigned to machine 1, the processing of interval i is terminated. Otherwise, interval i is rejected on the first machine and we pass it on to machine 2. The procedure is continued until the interval is either accepted on some machine or rejected on all of them.

In contrast to the previous section, where GREEDY^1 accepts an interval if it contains at most k intervals, we now set the threshold to $k+m-1$. The intuition for a higher threshold is the following: The threshold of k in GREEDY^1 was motivated by the fact that we can assume for $m = 1$ that OPT accepts only intervals that contain at most k intervals (see Observation 5.6 and the proof of Theorem 5.7). However, this is no longer true for multiple machines and there are instance where it makes sense for OPT to accept intervals that contain more than k intervals. Using a threshold of k for GREEDY^m would then lead to a competitive ratio of $\Omega(m)$, which can be seen by considering the instance of Figure 5.5, where GREEDY^m (with threshold k) would accept only one interval, whereas OPT $= m$.

In contrast to Remark 5.4, we cannot assume for $m > 1$ that an instance does not contain identical intervals. If there are identical intervals, we need to slightly modify the acceptance rule of GREEDY^m. Otherwise, the following problem might occur: if there are more than $k+m-1$ identical intervals, our algorithm would reject all of them (as each one contains the other ones in the sense of Definition 5.5). We therefore initially fix an ordering for identical intervals and, when determining the number of contained intervals at some

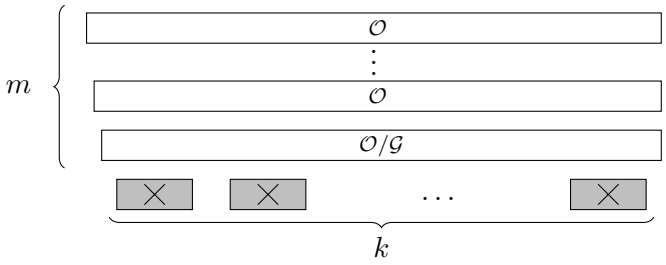


Figure 5.5: The intervals with labels $\mathcal{G}$ and $\mathcal{O}$ are accepted by GREEDY^m (if the acceptance threshold is set to k) and OPT, respectively. Failing intervals are crossed out.

point, we only consider the intervals that have not been processed so far. The entire greedy procedure for m machines is summarized in Algorithm 5.2.

Algorithm 5.2 GREEDY^m for m machines

1: Sort the intervals lexicographically according to non-decreasing release dates and lengths. For identical intervals, choose an arbitrary ordering.
2: **for all** intervals $i = 1, \ldots, n$ **do**
3: **for all** machines $j = 1, \ldots, m$ **do**
4: **if** interval i is non-failing, machine j is idle at the release date r_i of i, and i contains at most $k + m - 1$ of the intervals $i + 1, \ldots, n$ **then**
5: accept interval i on machine j
6: **break**
7: **end if**
8: **end for**
9: **end for**

Before we analyze GREEDY^m in detail, we prove the following result, which turns out to be useful later on:

Lemma 5.10. *If* GREEDY^m *rejects an interval i because i contains more than $k + m - 1$ intervals, it accepts a set of intervals $\tilde{\mathcal{I}}$ with $|\tilde{\mathcal{I}}| \geq m$ which all overlap with i.*

Proof. In the following, we denote the intervals contained in i by $i_1, \ldots, i_p$, where $p > k + m - 1$, and we assume that these intervals are sorted as in Step 1 of GREEDY^m.

It thus holds that i_1 contains at most $p - 1$ intervals, i_2 contains at most $p - 2$ intervals, and so on. Therefore, each of the intervals $i_{p-(k+m-1)}, \ldots, i_p$ contains at most $k + m - 1$ intervals. As no more than k of these $m + k$ many intervals can fail in total, at least m intervals do not fail and can be accepted

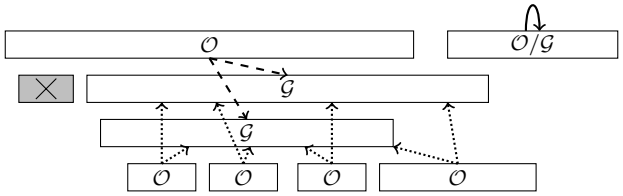


Figure 5.6: Illustration of the charging scheme used in the proof of Theorem 5.11 for $m = 2$ and $k = 3$. The intervals with labels $\mathcal{G}$ and $\mathcal{O}$ are accepted by GREEDY^m and OPT, respectively. The crossed-out interval is declared failing. Solid, dashed, and dotted lines correspond to cases (i), (ii), and (iii), respectively.

by GREEDY^m if the machine is not occupied at the release date. In particular, GREEDY^m can accept in total at least m intervals that overlap with i. □

For the competitiveness of GREEDY^m, we then obtain the following:

Theorem 5.11. *Algorithm 5.2 is* $(4 + \frac{k-3}{m})$*-competitive for* k-OIS *on* $m \geq 2$ *machines.*

Proof. We define a charging scheme that fractionally maps the set of intervals $\mathcal{O}$ accepted by OPT to the set of intervals $\mathcal{G}$ accepted by GREEDY^m. Formally, this is given by a mapping $\phi\colon \mathcal{O} \times \mathcal{G} \to [0, 1]$, where $\phi(i, j)$ can be seen as the fraction of interval $i \in \mathcal{O}$ that an interval $j \in \mathcal{G}$ is charged with. The mapping has to satisfy that $\sum_{j \in \mathcal{G}} \phi(i, j) = 1$ for all $i \in \mathcal{O}$. In order to show that GREEDY^m is c-competitive (for some $c \geq 1$), we show that at most a value of c is charged onto every interval accepted by GREEDY^m, i.e., $\sum_{i \in \mathcal{O}} \phi(i, j) \leq c$ for all $j \in \mathcal{G}$. This implies that

$$|\mathcal{O}| = \sum_{i \in \mathcal{O}} \underbrace{\sum_{j \in \mathcal{G}} \phi(i, j)}_{=1} = \sum_{j \in \mathcal{G}} \underbrace{\sum_{i \in \mathcal{O}} \phi(i, j)}_{\leq c} \leq c \cdot |\mathcal{G}|.$$

The mapping ϕ is illustrated in Figure 5.6 and formally defined as follows:

(i) For an interval $i \in \mathcal{O} \cap \mathcal{G}$, we set $\phi(i, i) := 1$.

(ii) For an interval $i \in \mathcal{O} \setminus \mathcal{G}$ that was rejected by GREEDY^m because it contained more than $k + m - 1$ intervals, GREEDY^m accepts a set of intervals $\tilde{\mathcal{I}}$ with $|\tilde{\mathcal{I}}| \geq m$ which all overlap with i (see Lemma 5.10). In this case, we set $\phi(i, \tilde{i}) := \frac{1}{|\tilde{\mathcal{I}}|} \leq \frac{1}{m}$ for all $\tilde{i} \in \tilde{\mathcal{I}}$.

(iii) For an interval $i \in \mathcal{O} \setminus \mathcal{G}$ that was rejected by GREEDY^m because all its machines were occupied at the release date r_i, there must be a set

of m intervals $\mathcal{I}'$ that GREEDY^m processes at time r_i. In this case, we set $\phi(i, i') := \frac{1}{|\mathcal{I}'|} = \frac{1}{m}$ for all $i' \in \mathcal{I}'$.

In the following, let $i \in \mathcal{G}$ be an arbitrary but fixed interval accepted by GREEDY^m. Observe that i is charged with a positive value by some interval $i' \in \mathcal{O}$ (i.e., $\phi(i', i) > 0$) only if i and i' are overlapping. We distinguish two cases:

$i \notin \mathcal{O}$: In this case, all intervals that charge onto i are given as in (ii) and (iii) and each of them charges a value of at most $\frac{1}{m}$ onto i.

There can be at most m intervals that are released before r_i and charge onto i: recall that all intervals that charge onto i need to overlap with i, and, thus, all intervals of this kind must pairwise overlap.

Since GREEDY^m accepts i, there are at most $k+m-1$ intervals contained in i that charge onto i.

Finally, there can be at most m intervals charging onto i that are released before r_i+p_i and finish thereafter (by the same argument as before, they must be pairwise overlapping).

Altogether, i is thus charged a value of at most

$$\frac{1}{m} \cdot (m + (k + m - 1) + m) = 3 + \frac{k-1}{m} \leq 4 + \frac{k-3}{m}.$$

$i \in \mathcal{O}$: If OPT accepts i and assigns it to some machine, it cannot accept another interval on the same machine that also charges onto i. In this case, i charges a value of 1 onto itself (see (i)).

Thus, we can restrict ourselves in the remaining analysis to $m-1$ machines, where all intervals that charge onto i are given as in (ii) and (iii). Analogously to the previous case, there can be at most $m-1$ intervals that are released before r_i, $k+m-1$ intervals contained in i, and $m-1$ intervals that are released before $r_i + p_i$ and finish thereafter. Each of these intervals charges a value of at most $\frac{1}{m}$.

Therefore, i is charged overall a value of at most

$$1 + \frac{1}{m} \cdot ((m-1) + (k + m - 1) + (m-1)) = 4 + \frac{k-3}{m}.$$

□

For multiple machines, we are able to establish the following lower bound for deterministic algorithms:

Theorem 5.12. *No deterministic online algorithm for k-OIS on m machines can be better than $\Omega(\sqrt[m]{k})$-competitive.*

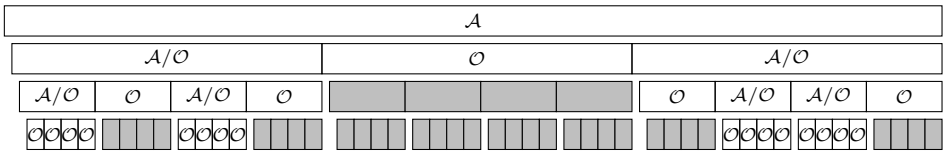


Figure 5.7: Illustration of the behaviour of the adversary in the proof of the lower bound in Theorem 5.12 for $m = 3$ and $k \geq 64$. The labels $\mathcal{A}$ and $\mathcal{O}$ correspond to the intervals accepted by ALG and OPT, respectively. Grey intervals are declared failing.

Proof. Let m be fixed and assume in the following that $k \geq 2^m$.

Consider the following instance consisting of $m+1$ layers of intervals (as illustrated in Figure 5.7): Layer 1 consists of a single interval with release date 0 and length 1. Layer 2 consists of $\lfloor \sqrt[m]{k} \rfloor - 1$ non-overlapping intervals that are all contained in the interval of layer 1 and their release dates are strictly larger than 0. For the remaining layers $3, \ldots, m+1$, for every interval i in layer $l \in \{2, \ldots, m\}$, there are $\lfloor \sqrt[m]{k} \rfloor$ intervals in layer $l+1$ that are non-overlapping and contained in i with release date strictly larger than r_i. Note that the total number of intervals is

$$1 + \sum_{j=0}^{m-1} \left(\lfloor \sqrt[m]{k} \rfloor - 1 \right) \left(\lfloor \sqrt[m]{k} \rfloor \right)^j \leq 1 + \sum_{j=0}^{m-1} \left(\sqrt[m]{k} - 1 \right) \left(\sqrt[m]{k} \right)^j$$
$$= 1 + \left(\sqrt[m]{k} - 1 \right) \cdot \frac{1 - (\sqrt[m]{k})^m}{1 - \sqrt[m]{k}} = k,$$

i.e., possibly all of them can fail.

Now let ALG be an arbitrary deterministic online algorithm. Note that ALG has at the release date r_i of interval i no information about the failure status of all intervals that are contained in i.

The adversary uses the following strategy: If ALG accepts some interval i, all intervals in the next layer that are contained in i will be non-failing. If ALG decides to reject some interval i, all intervals contained in i (in all further layers) will fail. This is also illustrated in Figure 5.7.

Observe that ALG can never accept any interval i in the last layer $m+1$ since this interval fails if ALG rejects at least one of the m intervals in layers $1, \ldots, m$ containing i. Hence, for every interval i that is accepted by ALG, OPT can accept the $\lfloor \sqrt[m]{k} \rfloor - 1$ non-failing intervals in the next layer that are contained in i, which shows that

$$\frac{\text{OPT}}{\text{ALG}} \geq \lfloor \sqrt[m]{k} \rfloor - 1 \in \Omega\left(\sqrt[m]{k} \right).$$

□

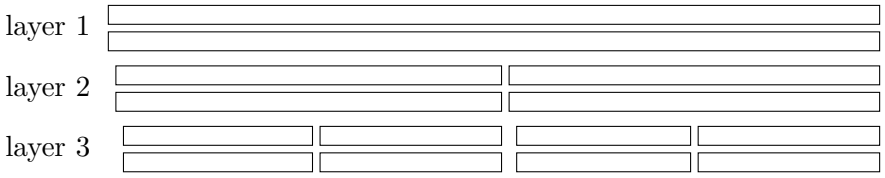


Figure 5.8: Illustration of the set of intervals used in the proof of Theorem 5.13 for $m = 2$ machines and three layers.

5.3 Randomized Online Algorithms

We now turn to randomized algorithms and establish the following lower bound:

Theorem 5.13. *No randomized online algorithm for k-OIS on m machines can be better than $\Omega\left(\log(k/m)\right)$-competitive.*

Proof. We want to apply Yao's Principle (Yao, 1977) and, therefore, define a probability distribution over $\lfloor \log(k/m) \rfloor$ instances $\sigma_1, \ldots, \sigma_{\lfloor \log(k/m) \rfloor}$ specified below. Throughout the proof, we assume that $k \geq 2m$, which ensures that $\lfloor \log(k/m) \rfloor \geq 1$.

The set $\mathcal{I}$ of intervals our instances consist of is partitioned into $\lfloor \log(k/m) \rfloor$ layers defined inductively as follows: Layer 1 contains m intervals with release date 0 and length 1. Inductively, for $l \in \{2, \ldots, \lfloor \log(k/m) \rfloor\}$ and for every interval i of layer $l-1$, layer l contains two non-overlapping intervals that are contained in i with release date strictly larger than r_i. Thus, layer l consists of $m \cdot 2^{l-1}$ intervals in total. This construction is illustrated for the case $m = 2$ and three layers in Figure 5.8.

Instance $\sigma_l = (\mathcal{I}, F_l)$ for $l \in \{1, \ldots, \lfloor \log(k/m) \rfloor\}$ is now defined by choosing the set F_l of failing intervals to consist of all the intervals in layers $l+1, \ldots, \lfloor \log(k/m) \rfloor$.

Note that this is feasible since all layers together contain

$$m \sum_{l=1}^{\lfloor \log(k/m) \rfloor} 2^{l-1} = m \, \frac{1 - 2^{\lfloor \log(k/m) \rfloor}}{1 - 2} = -m + m \, 2^{\lfloor \log(k/m) \rfloor} \leq m \, 2^{\log(k/m)} = k$$

intervals, i.e., any subset of the intervals is allowed to fail.

In this setting, every deterministic online algorithm is characterized by the number x_l^j of intervals it accepts in each layer l on each machine j. For $x = (x^1, \ldots, x^m)$ with $x^j = (x_1^j, \ldots, x_{\lfloor \log(k/m) \rfloor}^j)$, we let ALG_x denote a deterministic online algorithm that accepts exactly x_l^j intervals in layer $l \in \{1, \ldots, \lfloor \log(k/m) \rfloor\}$ on machine $j \in \{1, \ldots, m\}$. Note that, in order for

ALG_x to be feasible, we must have

$$\sum_{l=1}^{\lfloor \log(k/m) \rfloor} \frac{1}{2^{l-1}} x_l^j \leq 1 \quad \text{for all } j \in \{1, \dots, m\}, \tag{5.1}$$

since accepted intervals must not overlap.

We now define a probability distribution $\pi = (\pi_1, \dots, \pi_{\lfloor \log(k/m) \rfloor}\}$ over the set of instances $\{\sigma_l\}$ by setting $\pi_l := 2^{-l}$ for $l \in \{1, \dots, \lfloor \log(k/m) \rfloor - 1\}$ and $\pi_{\lfloor \log(k/m) \rfloor} := \frac{2}{2^{\lfloor \log(k/m) \rfloor}}$. Note that this defines a probability distribution since

$$\sum_{l=1}^{\lfloor \log(k/m) \rfloor} \pi_l = \sum_{l=1}^{\lfloor \log(k/m) \rfloor - 1} 2^{-l} + \frac{2}{2^{\lfloor \log(k/m) \rfloor}} = 1.$$

In order to calculate the expected number of intervals accepted by ALG_x with respect to this probability distribution, observe that ALG_x can accept intervals in layer j on instance σ_l only if they do not fail, i.e., if $j \leq l$. Hence, we obtain

$$\begin{aligned}
\mathbb{E}\left[\text{ALG}_x\right] &= \sum_{j=1}^{m} \sum_{l=1}^{\lfloor \log(k/m) \rfloor} x_l^j \left(1 - \sum_{p=1}^{l-1} \pi_p\right) \\
&= \sum_{j=1}^{m} \sum_{l=1}^{\lfloor \log(k/m) \rfloor} x_l^j \left(1 - \sum_{p=0}^{l-2} 2^{-p-1}\right) \\
&= \sum_{j=1}^{m} \sum_{l=1}^{\lfloor \log(k/m) \rfloor} \frac{1}{2^{l-1}} x_l^j \leq m,
\end{aligned}$$

where the last inequality follows from the feasibility constraints (5.1).

An optimal offline algorithm accepts the $m \cdot 2^{l-1}$ intervals of layer l on instance σ_l, so we have

$$\begin{aligned}
\mathbb{E}\left[\text{OPT}\right] = m \sum_{l=1}^{\lfloor \log(k/m) \rfloor} \pi_l \, 2^{l-1} &= m \left(\sum_{l=1}^{\lfloor \log(k/m) \rfloor - 1} 2^{-l} \, 2^{l-1} + \frac{2}{2^{\lfloor \log(k/m) \rfloor}} \, 2^{\lfloor \log(k/m) \rfloor - 1} \right) \\
&= m \, \frac{\lfloor \log(k/m) \rfloor + 1}{2}.
\end{aligned}$$

Hence, we have shown that every deterministic online algorithm ALG_x satisfies

$$\frac{\mathbb{E}\left[\text{OPT}\right]}{\mathbb{E}\left[\text{ALG}_x\right]} \geq \frac{\lfloor \log(k/m) \rfloor + 1}{2} \in \Omega\left(\log\left(k/m\right)\right),$$

and the claim follows by Yao's Principle. □

5.3.1 A Randomized Algorithm for Laminar Intervals on a Single Machine

In the proof of Theorem 5.13, we used a special structure of the intervals: The set of intervals we considered is *laminar*, i.e., for every pair of intervals, it holds that the intervals are either disjoint or one of them is contained in the other. Laminar set systems are a well-known concept in set theory. For intervals as considered here, laminarity is formally defined as follows:

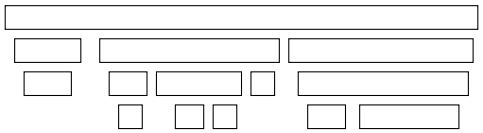
Definition 5.14. *A set $\mathcal{I}$ of intervals is called* laminar *if it holds for all $i_1, i_2 \in \mathcal{I}$ that $i_1 \subseteq i_2$, $i_2 \subseteq i_1$, or $i_1 \cap i_2 = \emptyset$.*

Next, we show how to construct a randomized $(\log(k+2))$-competitive algorithm for k-OIS on a single machine if the set of intervals is laminar. This competitiveness improves significantly upon the results for deterministic algorithms and almost matches the lower bound shown in Theorem 5.13 for randomized algorithms.

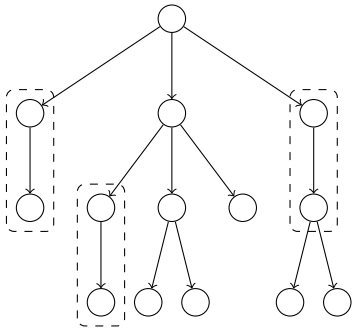
Given a laminar set $\mathcal{I}$ of intervals, we define a directed graph called the *inclusion graph* that has one node for every interval and an arc from node i to j if and only if $j \subseteq i$ and there is no other interval k such that $j \subseteq k \subseteq i$. We can then *shrink* this graph as follows: if a node i has only one outgoing arc (i, j), we *merge* nodes i and j. We call the graph we obtain after merging nodes as long as possible the *shrinked inclusion graph*. Note that, since $\mathcal{I}$ is laminar, the inclusion graph and the shrinked inclusion graph are collections of rooted *out-trees*, i.e., directed trees in which each node is reachable by a unique directed path from the root of the corresponding tree (the roots are the nodes without incoming arcs, which correspond to intervals that are maximal with respect to inclusion). An example of a laminar set of intervals and the corresponding (shrinked) inclusion graph is given in Figures 5.9(a)-(c)

Our algorithm now works as follows: We first remove all intervals that contain more than k intervals (recall that such an interval contains at least one non-failing interval, and it never makes sense for a competitive online algorithm to accept it on a single machine).

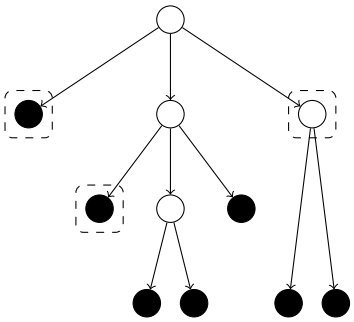
We then construct the shrinked inclusion graph for the remaining set of intervals and add all leaf nodes in this graph to a class, which we denote by C_1. Then, all these leaf nodes and their incident edges are removed from the graph, and we remain with an inclusion graph (which may contain nodes with only one outgoing arc). In this graph, we then merge nodes that have only one outgoing arc with their successors as described before, and add all leaf nodes in the resulting shrinked graph to a new class C_2. We continue with this procedure until we obtain a partition of the nodes of the original inclusion graph (and, thus, also of the original set of intervals) into $l_{\max}$ classes $C_1, \ldots, C_{l_{\max}}$. An illustration of such a partitioning of intervals into classes is given in Figure 5.9(d).



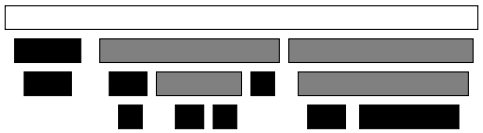
(a) Illustration of a laminar set of intervals.



(b) The corresponding inclusion graph ...



(c) ... and the shrinked inclusion graph. All leaf nodes (black) are assigned to the same class ...



(d) ... and then removed. In the remaining graph, we again merge nodes and add the leaf nodes to the next class (grey). Afterwards, the remaining graph only consists of a single node (the root), which is added to the third class (white).

Figure 5.9: Illustration of the shrinked inclusion graph and the partitioning of the intervals into classes.

The algorithm then chooses a number $l \in \{1, \ldots, l_{\max}\}$ uniformly at random and accepts intervals from class C_l greedily, i.e., a non-failing interval from C_l is accepted whenever the machine is currently not occupied. More precisely, this means that, for every set of intervals that belong to a merged node in C_l, the first non-failing interval is accepted. This is summarized in Algorithm 5.3.

Observe that, if we restrict ourselves to the subset of the intervals that belong to a single class C_l, OPT cannot accept more intervals than ALG since all intervals whose corresponding nodes were merged to a single node during the construction of the partition overlap. Formally, this means that

$$\text{ALG}(\sigma_l) = \text{OPT}(\sigma_l), \tag{5.2}$$

Algorithm 5.3 Randomized algorithm for laminar instances

1: Remove all intervals that contain more than k intervals.
2: Construct a partition of the remaining intervals into classes $C_1, \ldots, C_{l_{\max}}$ as described.
3: Choose $l \in \{1, \ldots, l_{\max}\}$ uniformly at random and accept the first non-failing interval within each set of intervals that belong to a merged node in C_l.

where σ_l denotes the subset of instance σ consisting only of the intervals of class l. Using this notation, we also see that

$$\textsc{opt}(\sigma) \leq \sum_{l=1}^{l_{\max}} \textsc{opt}(\sigma_l). \tag{5.3}$$

Thus, we have

$$\begin{aligned}\mathbb{E}\left[\textsc{alg}(\sigma)\right] = \sum_{l=1}^{l_{\max}} \frac{1}{l_{\max}} \cdot \textsc{alg}(\sigma_l) &\overset{(5.2)}{=} \sum_{l=1}^{l_{\max}} \frac{1}{l_{\max}} \cdot \textsc{opt}(\sigma_l) \\ &\overset{(5.3)}{\geq} \frac{1}{l_{\max}} \cdot \textsc{opt}(\sigma). \end{aligned} \tag{5.4}$$

In the following, we want to show that the number of different classes in our algorithm is at most $\log(k+2)$, i.e., $l_{\max} \leq \log(k+2)$.

Every leaf node in the shrinked inclusion graph of iteration $l+1$ corresponds to a set of non-leaf nodes in the shrinked inclusion graph of iteration l. Moreover, since we merge nodes whenever possible, we know that, among the successors of this set of nodes in the shrinked inclusion graph of iteration l, there are at least two leaf nodes of iteration l.

Hence, we obtain that, for every leaf node in the shrinked inclusion graph of iteration $l+1$ that is added to class C_{l+1}, there must be at least two leaf nodes in the shrinked inclusion graph of iteration l that are added to class C_l. Moreover, the definition of $l_{\max}$ implies that class $C_{l_{\max}}$ contains at least one node. Thus, we obtain that, for each $l \in \{1, \ldots, l_{\max}\}$, class C_l consists of at least $2^{l_{\max}-l}$ (merged) nodes.

Since there are at most $k+1$ nodes in every subtree of the shrinked inclusion graph (as we delete all intervals that contain more than k intervals at the beginning), we have that

$$k+1 \geq \sum_{l=1}^{l_{\max}} 2^{l_{\max}-l} = \sum_{l=0}^{l_{\max}-1} 2^l = \frac{1-2^{l_{\max}}}{1-2} = 2^{l_{\max}} - 1,$$

which implies that

$$l_{\max} \leq \log(k+2). \tag{5.5}$$

Using this in (5.4) yields the following:

Theorem 5.15. *Algorithm 5.3 is* $(\log(k+2))$*-competitive for* k-OIS *on a single machine* ($m = 1$) *when the set of intervals is laminar.*

Remark 5.16. *Note that there are instances for which the analysis in Theorem 5.15 is tight: For example, consider the instance illustrated in Figure 5.2 for* $k = 2$*, i.e., there are three intervals with release dates* $r_1 = 0, r_2 = \frac{1}{3}, r_3 = \frac{2}{3}$*, and lengths* $p_1 = 1, p_2 = p_3 = \frac{1}{3}$*. If intervals 2 and 3 fail, it holds that*

$$\mathbb{E}\left[\text{ALG}\right] = \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 0 = \frac{1}{2} = \frac{1}{\log(k+2)} \cdot \text{OPT}.$$

5.3.2 Extension of the Algorithm to Multiple Machines

In the following, we describe how to extend the idea of Algorithm 5.3 to $m \geq 2$ machines.

Throughout this section we use the following notation: As defined in Section 5.1, an instance $\sigma = (\mathcal{I}, F)$ of k-OIS is given by a set of intervals $\mathcal{I}$ and a set $F \subseteq \mathcal{I}$ of failing intervals.[1] When dealing with different instances, we occasionally write $\mathcal{I}(\sigma)$ and $F(\sigma)$ if it is necessary to make clear which set of intervals we are concerned with. For a set X of intervals and an instance $\sigma = (\mathcal{I}, F)$, we define $\sigma - X := (\mathcal{I} \setminus X, F \setminus X)$.

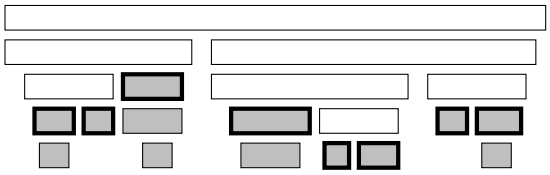
While we can assume for $m = 1$ that all release dates are pairwise different (Remark 5.4), this is no longer valid for $m \geq 2$. Hence, it is possible that several intervals are identical, and there is no longer a unique ordering with respect to inclusion (which is needed for the construction of the inclusion graph). We circumvent this problem by fixing the ordering of identical intervals arbitrarily in advance.

Description of the Algorithm

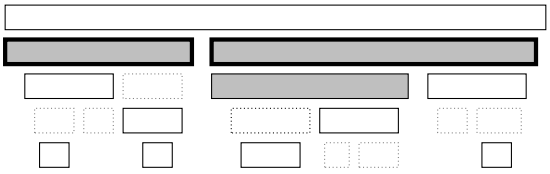
Our algorithm works as follows:

In the first iteration, we ignore all intervals that contain more than k intervals and construct a partitioning of the intervals into classes $C_1, \ldots, C_{l_{\max}}$ as described in the previous section. Among these classes, we then choose a class $\mathcal{C}^1$ uniformly at random for the first machine. We denote the set of possible choices for the random variable $\mathcal{C}^1$ by Ξ^1, i.e., $\Xi^1 = \{C_1, \ldots, C_{l_{\max}}\}$. Our algorithm will then later accept on machine 1 only intervals from this class $\mathcal{C}^1$ in a greedy manner, i.e., for every set of intervals that belong to a merged node in class $\mathcal{C}^1$, the first non-failing interval is accepted (where ties are broken in favor of the interval that is maximal with respect to the inclusion ordering).

[1] Formally, the number of machines m is also part of the instance. However, we will sometimes slightly abuse notation and say that an instance σ is processed on m machines.



(a) For the first machine, a class $\mathcal{C}^1$ (grey) of the interval partitioning is chosen uniformly at random. The intervals in $I^1(\mathcal{C}^1)$ are highlighted with a black frame.



(b) For the second machine, the intervals in $I^1(\mathcal{C}^1)$ are removed (dotted), and a class $\mathcal{C}^2$ (grey) of the partitioning of $\mathcal{I} \setminus I^1(\mathcal{C}^1)$ is chosen uniformly at random. The intervals in $I^2(\mathcal{C}^2)$ are highlighted with a black frame.

Figure 5.10: Illustration of the randomized algorithm for multiple machines.

When determining a set of intervals for the second machine, we choose only from a subset of the intervals: For each set of merged intervals in $\mathcal{C}^1$, we remove the inclusion-wise maximal one from $\mathcal{I}$. We denote the set of removed intervals by $I^1 = I^1(\mathcal{C}^1)$ (see Figure 5.10(a) for an illustration). The motivation for this definition stems from the fact that our algorithm will be defined in the following such that every non-failing interval from I^1 will be accepted on the first machine and cannot be accepted on any of the other machines.

For the remaining set $\mathcal{I} \setminus I^1$ of intervals, we again ignore all intervals still containing more than k intervals[2], construct a partitioning into classes, and choose among these a class $\mathcal{C}^2$ uniformly at random for the second machine, as illustrated in Figure 5.10(b). As before, we let $\Xi^2 = \Xi^2(\mathcal{C}^1)$ denote the set of possible choices for the random variable $\mathcal{C}^2$, and $I^2 = I^2(\mathcal{C}^2)$ denotes the set of inclusion-wise maximal intervals in $\mathcal{C}^2$. We continue with this procedure until we have obtained a set of intervals for every machine, i.e., in each iteration $l = 1, \ldots, m$ we construct a partitioning of the intervals $\mathcal{I} \setminus \cup_{i=1}^{l-1} I^i$ and choose a class $\mathcal{C}^l$ uniformly at random.

[2]Note that it can happen that some of the intervals we ignored in the first iteration are not ignored anymore in the second iteration.

Note that, in our randomized procedure, it can happen that an interval i is contained in multiple classes chosen for the different machines. In this case, the algorithm follows the rule that, when interval i is released and does not fail, we assign it to the machine with minimum index (among the machines j whose class $\mathcal{C}^j$ contains i) that is currently not occupied (if all such machines are already occupied, the interval is rejected).

Observe that our random variables $\mathcal{C}^1, \ldots, \mathcal{C}^m$ are not independent as we have $\mathcal{C}^1 \in \Xi^1$, $\mathcal{C}^2 \in \Xi^2(\mathcal{C}^1), \ldots, \mathcal{C}^m \in \Xi^m(\mathcal{C}^1, \ldots, \mathcal{C}^{m-1})$. All these random variables are realized, i.e., the classes for the machines are chosen, before any interval is released. In particular, they do not depend on the set of failing intervals $F \subseteq \mathcal{I}$, which will be important in the analysis later on.

Analysis of the Algorithm

In the following, we analyze the competitiveness of our algorithm.

For m machines, we denote by ALG^m the algorithm described above, and write $\text{ALG}^m(\sigma)$ for the number of intervals accepted by ALG^m on instance σ. Furthermore, we use the notation $\text{ALG}^m_S(\sigma)$ for the number of intervals that ALG^m assigns to a subset $S \subseteq \{1, \ldots, m\}$ of the machines on instance σ.

As the number of accepted intervals on the first machine is independent of $\mathcal{C}^2, \ldots, \mathcal{C}^m$, we have that

$$\mathbb{E}\left[\text{ALG}^m(\sigma)\right] = \mathbb{E}_{\mathcal{C}^1}\left[\text{ALG}^m_1(\sigma) + \mathbb{E}_{\mathcal{C}^2,\ldots,\mathcal{C}^m}\left[\text{ALG}^m_{2,\ldots,m}(\sigma)\right]\right]. \tag{5.6}$$

In the remainder of this section we want to show that, for any fixed m, $\mathbb{E}\left[\text{ALG}^m(\sigma)\right] \geq \frac{1}{\log(k+2)} \cdot \text{OPT}^m(\sigma)$ for all instances σ. Here, $\text{OPT}^m(\sigma)$ denotes the maximum number of intervals that can be accepted on instance σ if m machines are available. As for our algorithm, we use the notation $\text{OPT}^m_S(\sigma)$ for the number of intervals that a given optimal schedule for instance σ accepts on a subset $S \subseteq \{1, \ldots, m\}$ of the machines.

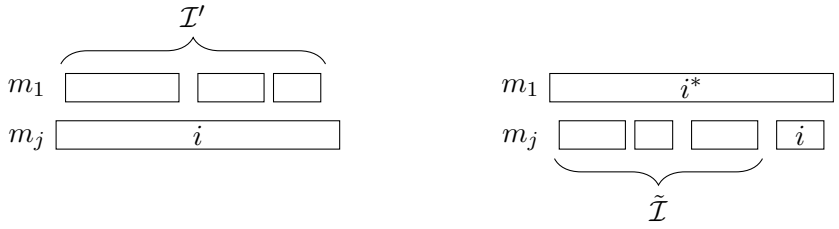
Next, we show some results that will turn out to be useful in the analysis of our algorithm.

Lemma 5.17. *For each instance σ and each subset $M \subseteq I(\sigma)$ of pairwise non-overlapping intervals, we can assume, without loss of generality, that*

$$\text{OPT}^m_{2,\ldots,m}(\sigma) \leq \text{OPT}^{m-1}(\sigma - M). \tag{5.7}$$

Proof. We show that there always exists an optimal schedule for σ on m machines in which all of the intervals in M are either assigned to the first machine or rejected. This implies the claim.

So assume that we are given an optimal schedule $\mathcal{S}_{\text{OPT}}$ on m machines in which some interval $i \in M$ is assigned to a machine $j \in \{2, \ldots, m\}$.



(a) First case. (b) Second case.

Figure 5.11: Illustration of the two cases in Lemma 5.17.

We now consider the intervals accepted in $\mathcal{S}_{\text{OPT}}$ on machine 1 that overlap with i. As the set of intervals is laminar, all these intervals are either contained in i or there exists an interval i^* containing i (see Figure 5.11).

First, we consider the case that there exists a non-empty subset $\mathcal{I}' \subseteq \mathcal{I}$ of the intervals that are accepted in $\mathcal{S}_{\text{OPT}}$ on machine 1 and contained in i as illustrated in Figure 5.11(a). As the instance is laminar, moving i to machine 1, and $\mathcal{I}'$ to machine j yields a feasible schedule containing the same number of accepted intervals.

If i is contained in some interval i^* that is assigned to machine 1 in $\mathcal{S}_{\text{OPT}}$, this interval may also contain some set $\tilde{\mathcal{I}}$ of other intervals (assigned to machine j in $\mathcal{S}_{\text{OPT}}$) as illustrated in Figure 5.11(b). As before, it is possible to change the assignment of the intervals to the machines by moving i^* to machine j and $\tilde{\mathcal{I}} \cup \{i\}$ to machine 1.

After this step, interval i is no longer assigned to machine j. Note that it is not possible that, after this exchange, another interval from M that was previously assigned to machine 1 is now assigned to machine j (as intervals from M do not overlap). Hence, performing the above procedure at most n times yields an optimal schedule with the desired property. □

Observe that, for each $j \in \{1, \ldots, m\}$, algorithm ALG^m acts on the first j machines in the same way as ALG^j does: This follows by our construction (ALG^m chooses a class $\mathcal{C}^1$ for the first machine from the same set of classes of the interval partitioning as ALG^j does, and, thus, the same intervals $\mathcal{I} \setminus I^1$ are considered when a class $\mathcal{C}^2$ for the second machine is chosen, and so on) and the fact that a non-failing interval i is accepted on the machine with minimum index (among those machines whose classes contain i). Thus, we obtain the following result:

Observation 5.18. *For all instances σ and $j \in \{1, \ldots, m\}$, it holds that*

$$\text{ALG}^m_{1,\ldots,j}(\sigma) = \text{ALG}^j(\sigma).$$

The most important result for proving the competitiveness of ALG^m is the following lemma. For an arbitrary instance σ, it bounds the expected number of intervals accepted by ALG^m on machines $2, \dots, m$ from below by means of OPT^{m-1} (on a sub-instance).

In the following, we denote by $\mathcal{A}_1^m(\sigma, \mathcal{C}^1)$ the set of intervals accepted by ALG^m on the first machine for instance σ if $\mathcal{C}^1 \in \Xi^1$ is chosen. If σ and $\mathcal{C}^1$ are clear from the context, we write for short $\mathcal{A}_1^m$.

Lemma 5.19. *For any instance σ, a fixed choice of a class $\mathcal{C}^1 \in \Xi^1$ for the first machine, and $I^1 = I^1(\mathcal{C}^1)$, it holds that*

$$\mathbb{E}_{\mathcal{C}^2,\dots,\mathcal{C}^m}\left[\text{ALG}^m_{2,\dots,m}(\sigma)\right] \geq \frac{1}{\log(k+2)} \cdot \text{OPT}^{m-1}\left((\sigma - I^1) - \mathcal{A}_1^m\left(\sigma, \mathcal{C}^1\right)\right).$$

Proof. In the following, we denote by $\overline{\text{ALG}^m}$ the algorithm that processes an instance in the same way as ALG^m, i.e., it constructs a partition of the intervals and chooses a class uniformly at random, which is then processed greedily. However, some of the intervals are not allowed to be accepted by $\overline{\text{ALG}^m}$ and have to be skipped in the acceptance procedure.

For an instance σ and a set $U \subseteq \mathcal{I}(\sigma)$ of intervals that have to be skipped, we denote the number of accepted intervals by $\overline{\text{ALG}^m}(\sigma \,|\, U)$. Using this notation, we claim that

$$\begin{aligned}\mathbb{E}_{\mathcal{C}^2,\dots,\mathcal{C}^m}\left[\text{ALG}^m_{2,\dots,m}(\sigma)\right] &= \mathbb{E}\left[\overline{\text{ALG}^{m-1}}(\sigma - I^1 \mid \mathcal{A}_1^m)\right] &(5.8)\\ &\geq \frac{1}{\log(k+2)} \cdot \text{OPT}^{m-1}((\sigma - I^1) - \mathcal{A}_1^m). &(5.9)\end{aligned}$$

Here, equality (5.8) holds by the following argument:

By assumption, ALG^m chooses (on instance σ) for machine 2 a class from the partitioning of the intervals $\mathcal{I} \setminus I^1$ uniformly at random, and, in the following, accepts these intervals in a greedy manner except that intervals that are accepted on machine 1 (i.e., the intervals $\mathcal{A}_1^m$) have to be skipped.

$\overline{\text{ALG}^{m-1}}$ chooses (on instance $\sigma - I^1$) for machine 1 a class from the same partitioning of the intervals $\mathcal{I} \setminus I^1$ and accepts from these intervals in a greedy manner while having to skip intervals in $\mathcal{A}_1^m$.

By the same reasoning, the behavior of ALG^m on machine j and $\overline{\text{ALG}^{m-1}}$ on machine $j-1$ coincide for each $j \in \{2, \dots, m\}$.

In order to prove inequality (5.9), we show the following:

Claim 5.20. *For all sets of intervals $U \subseteq I(\sigma)$ and instances $\hat{\sigma} \subseteq \sigma$,[3] it holds that*

$$\mathbb{E}\left[\overline{\text{ALG}^{m-1}}(\hat{\sigma} \mid U)\right] \geq \frac{1}{\log(k+2)} \cdot \text{OPT}^{m-1}(\hat{\sigma} - U).$$

[3]Here, $\hat{\sigma} \subseteq \sigma$ means that $I(\hat{\sigma}) \subseteq I(\sigma)$ and $F(\hat{\sigma}) \subseteq F(\sigma) \cap I(\hat{\sigma})$.

(Note that choosing $\hat{\sigma} := \sigma - I^1$ and $U := \mathcal{A}_1^m$ in the claim yields (5.9). Hence, proving the claim suffices to establish Lemma 5.19.)

Proof. We prove the claim by induction on m.

First, we consider the induction basis $m = 2$, where we can use similar arguments as in the analysis of Algorithm 5.3.

We denote by $\hat{\sigma}_l$ the subset of instance $\hat{\sigma}$ consisting only of the intervals belonging to class C_l in the partitioning of the set $\mathcal{I}(\hat{\sigma})$ (where we ignore, by definition of the algorithm, all intervals that contain more than k intervals). Furthermore, we let $\hat{l}_{\max}$ denote the total number of classes for instance $\hat{\sigma}$. Thus, analogously to (5.2), (5.3), and (5.5), it holds for all $l \in \{1, \ldots, \hat{l}_{\max}\}$ that

$$\overline{\text{ALG}^1}(\hat{\sigma}_l \mid U) = \text{OPT}^1(\hat{\sigma}_l - U), \tag{5.10}$$

$$\text{OPT}^1(\hat{\sigma} - U) \leq \sum_{l=1}^{\hat{l}_{\max}} \text{OPT}^1(\hat{\sigma}_l - U), \tag{5.11}$$

$$\hat{l}_{\max} \leq \log(k+2). \tag{5.12}$$

Altogether, we obtain

$$\begin{aligned}
\mathbb{E}\left[\overline{\text{ALG}^1}(\hat{\sigma} \mid U)\right] &= \sum_{l=1}^{\hat{l}_{\max}} \frac{1}{\hat{l}_{\max}} \cdot \overline{\text{ALG}^1}(\hat{\sigma}_l \mid U) \\
&\overset{(5.10)}{=} \sum_{l=1}^{\hat{l}_{\max}} \frac{1}{\hat{l}_{\max}} \cdot \text{OPT}^1(\hat{\sigma}_l - U) \\
&\overset{(5.11)}{\geq} \frac{1}{\hat{l}_{\max}} \cdot \text{OPT}^1(\hat{\sigma} - U) \\
&\overset{(5.12)}{\geq} \frac{1}{\log(k+2)} \cdot \text{OPT}^1(\hat{\sigma} - U).
\end{aligned}$$

For the inductive step, assume that the statement holds for $m-1$. We use the notation $\overline{\mathcal{A}_1^{m-1}} = \overline{\mathcal{A}_1^{m-1}}(\hat{\sigma}, \mathcal{C}^1 | U)$ for the set of intervals accepted by $\overline{\text{ALG}^{m-1}}$ on the first machine for instance $\hat{\sigma}$ and class $\mathcal{C}^1 \in \Xi_1$ when the set U of intervals has to be skipped. It holds that (see below for an explanation of the single steps):

$$\begin{aligned}
&\mathbb{E}\left[\overline{\text{ALG}^{m-1}}(\hat{\sigma} \mid U)\right] \\
&= \mathbb{E}_{\mathcal{C}_1}\left[\overline{\text{ALG}_1^{m-1}}(\hat{\sigma} \mid U) + \mathbb{E}_{\mathcal{C}^2,\ldots,\mathcal{C}^{m-1}}\left[\overline{\text{ALG}_{2,\ldots,m-1}^{m-1}}(\hat{\sigma} \mid U)\right]\right] && (5.13) \\
&= \mathbb{E}_{\mathcal{C}^1}\left[\overline{\text{ALG}_1^{m-1}}(\hat{\sigma} \mid U) + \mathbb{E}\left[\overline{\text{ALG}^{m-2}}\left(\hat{\sigma} - I^1(\mathcal{C}^1) \;\middle|\; U \cup \overline{\mathcal{A}_1^{m-1}}(\hat{\sigma}, \mathcal{C}^1 | U)\right)\right]\right] && (5.14)
\end{aligned}$$

$$\geq \mathbb{E}_{\mathcal{C}^1}\Bigg[\overline{\mathrm{ALG}_1^{m-1}}(\hat{\sigma} \mid U) + \frac{1}{\log(k+2)} \cdot \mathrm{OPT}^{m-2}\left((\hat{\sigma} - I^1(\mathcal{C}^1)) - \left(U \cup \overline{\mathcal{A}_1^{m-1}}(\hat{\sigma}, \mathcal{C}^1|U)\right)\right)\Bigg] \tag{5.15}$$

$$\geq \mathbb{E}_{\mathcal{C}^1}\left[\overline{\mathrm{ALG}_1^{m-1}}(\hat{\sigma} \mid U)\right] + \frac{1}{\log(k+2)} \cdot \min_{\widetilde{\mathcal{C}^1} \in \Xi^1} \mathrm{OPT}^{m-2}\left((\hat{\sigma} - I^1(\widetilde{\mathcal{C}^1})) - \left(U \cup \overline{\mathcal{A}_1^{m-1}}(\hat{\sigma}, \widetilde{\mathcal{C}^1}|U)\right)\right) \tag{5.16}$$

$$\geq \mathbb{E}_{\mathcal{C}^1}\left[\overline{\mathrm{ALG}_1^{m-1}}(\hat{\sigma} \mid U)\right] + \frac{1}{\log(k+2)} \cdot \min_{\widetilde{\mathcal{C}^1} \in \Xi^1} \mathrm{OPT}^{m-2}\left(\hat{\sigma} - \left(U \cup \overline{\mathcal{A}_1^{m-1}}(\hat{\sigma}, \widetilde{\mathcal{C}^1}|U)\right)\right) \tag{5.17}$$

$$= \mathbb{E}\left[\overline{\mathrm{ALG}^1}\,(\hat{\sigma} \mid U)\right] + \frac{1}{\log(k+2)} \cdot \min_{\widetilde{\mathcal{C}^1} \in \Xi^1} \mathrm{OPT}^{m-2}\left((\hat{\sigma} - U) - \overline{\mathcal{A}_1^{m-1}}(\hat{\sigma}, \widetilde{\mathcal{C}^1}|U)\right) \tag{5.18}$$

$$\geq \frac{1}{\log(k+2)} \cdot \Bigg(\mathrm{OPT}^1(\hat{\sigma} - U) + \min_{\widetilde{\mathcal{C}^1} \in \Xi^1} \mathrm{OPT}^{m-2}\left((\hat{\sigma} - U) - \overline{\mathcal{A}_1^{m-1}}(\hat{\sigma}, \widetilde{\mathcal{C}^1}|U)\right)\Bigg) \tag{5.19}$$

$$\geq \frac{1}{\log(k+2)} \cdot \left(\mathrm{OPT}_1^{m-1}(\hat{\sigma} - U) + \mathrm{OPT}_{2,\ldots,m-1}^{m-1}\,(\hat{\sigma} - U)\right) \tag{5.20}$$

$$\geq \frac{1}{\log(k+2)} \cdot \mathrm{OPT}^{m-1}(\hat{\sigma} - U).$$

Here, equality (5.13) follows analogously to (5.6). The second equality (5.14) holds by the argumentation presented for (5.8).[4] For (5.15), we use the induction hypothesis, while (5.16) holds since the minimum is no larger than the expected value.

Inequality (5.17) holds by the following argument: If we have $I^1(\widetilde{\mathcal{C}^1}) \subseteq U \cup \overline{\mathcal{A}_1^{m-1}}(\hat{\sigma}, \widetilde{\mathcal{C}^1}|U)$, the inequality obviously holds. So assume there exists an interval $i \in I^1(\widetilde{\mathcal{C}^1}) \setminus (U \cup \overline{\mathcal{A}_1^{m-1}}(\hat{\sigma}, \widetilde{\mathcal{C}^1}|U))$. By definition of our algorithm, this interval i is then not accepted by $\overline{\mathrm{ALG}^{m-1}}$ on the first machine because it fails, i.e., $i \in F(\hat{\sigma})$. In this case, it can neither be accepted by OPT^{m-2} on any sub-instance of $\hat{\sigma}$.

Equality (5.18) follows by choosing $j = 1$ in Observation 5.18.[5] Finally, for inequality (5.19), we use the induction basis $m = 2$, and (5.20) is a consequence

[4] Forbidding an algorithm to accept some intervals does not change the argumentation presented in the paragraph subsequent to (5.9).

[5] The argumentation for Observation 5.18 carries over to $\overline{\mathrm{ALG}^m}$.

of Lemma 5.17 applied to $\sigma = \hat{\sigma} - U$ and $M = \overline{\mathcal{A}_1^{m-1}}(\hat{\sigma}, \mathcal{C}^1_{\min}|U)$, where $\mathcal{C}^1_{\min} = \operatorname*{argmin}_{\mathcal{C}^1 \in \Xi^1} \textsc{opt}^{m-2}\left((\hat{\sigma} - U) - \overline{\mathcal{A}_1^{m-1}}(\hat{\sigma}, \widetilde{\mathcal{C}^1}|U)\right)$. □

We now complete the proof of the competitiveness of our algorithm by bounding the expected number of accepted intervals as follows:

$$
\begin{aligned}
&\mathbb{E}\left[\textsc{alg}^m(\sigma)\right] \\
&= \mathbb{E}_{\mathcal{C}^1}\left[\textsc{alg}_1^m(\sigma) + \mathbb{E}_{\mathcal{C}^2,\dots,\mathcal{C}^m}\left[\textsc{alg}^m_{2,\dots,m}(\sigma)\right]\right] && (5.21)\\
&\geq \mathbb{E}_{\mathcal{C}^1}\left[\textsc{alg}_1^m(\sigma) + \frac{1}{\log(k+2)} \cdot \textsc{opt}^{m-1}\left((\sigma - I^1(\mathcal{C}^1)) - \mathcal{A}_1^m(\sigma, \mathcal{C}^1)\right)\right] && (5.22)\\
&\geq \mathbb{E}_{\mathcal{C}^1}\left[\textsc{alg}_1^m(\sigma)\right] \\
&\qquad + \frac{1}{\log(k+2)} \cdot \min_{\widetilde{\mathcal{C}^1} \in \Xi^1} \textsc{opt}^{m-1}\left((\sigma - I^1(\widetilde{\mathcal{C}^1})) - \mathcal{A}_1^m(\sigma, \widetilde{\mathcal{C}^1})\right) && (5.23)\\
&\geq \mathbb{E}_{\mathcal{C}^1}\left[\textsc{alg}_1^m(\sigma)\right] + \frac{1}{\log(k+2)} \cdot \min_{\widetilde{\mathcal{C}^1} \in \Xi^1} \textsc{opt}^{m-1}\left(\sigma - \mathcal{A}_1^m(\sigma, \widetilde{\mathcal{C}^1})\right) && (5.24)\\
&= \mathbb{E}_{\mathcal{C}^1}\left[\textsc{alg}^1(\sigma)\right] + \frac{1}{\log(k+2)} \cdot \min_{\widetilde{\mathcal{C}^1} \in \Xi^1} \textsc{opt}^{m-1}\left(\sigma - \mathcal{A}_1^m(\sigma, \widetilde{\mathcal{C}^1})\right) && (5.25)\\
&\geq \frac{1}{\log(k+2)} \cdot \textsc{opt}^1(\sigma) + \frac{1}{\log(k+2)} \cdot \textsc{opt}^m_{2,\dots,m}(\sigma) && (5.26)\\
&\geq \frac{1}{\log(k+2)} \cdot \textsc{opt}^m_1(\sigma) + \frac{1}{\log(k+2)} \cdot \textsc{opt}^m_{2,\dots,m}(\sigma) \\
&= \frac{1}{\log(k+2)} \cdot \textsc{opt}^m(\sigma).
\end{aligned}
$$

Here, equality (5.21) holds by (5.6), and (5.22) is due to Lemma 5.19. Inequality (5.23) obviously holds since the minimum is no larger than the expected value, and (5.24) follows analogously to the argumentation presented for (5.17) in the proof of Lemma 5.19. Choosing $j = 1$ in Observation 5.18 yields (5.25). Finally, (5.26) holds by our result for the single machine algorithm (Theorem 5.15) and Lemma 5.17 with $M = \mathcal{A}_1^m(\sigma, \mathcal{C}^1_{\min})$ and $\mathcal{C}^1_{\min} = \operatorname*{argmin}_{\mathcal{C}^1 \in \Xi^1} \textsc{opt}^{m-1}\left(\sigma - \mathcal{A}_1^m(\sigma, \widetilde{\mathcal{C}^1})\right)$. Hence, we have shown:

Theorem 5.21. *There exists a randomized* $(\log(k+2))$*-competitive algorithm for* k*-*OIS *when the set of intervals is laminar.*

5.4 Weighted Interval Scheduling

A generalization of our problem k-OIS is given by introducing a *weight* $w_i > 0$ for each interval i and maximizing the total weight of accepted intervals. However, this setting does, in general, not allow for deterministic online algorithms with bounded competitiveness, as the following result shows:

Proposition 5.22. *For the weighted version of k-OIS, no deterministic online algorithm can achieve a bounded competitiveness, even if $m = 1$, $k = 1$, and*

(i) all intervals have the same length, or

(ii) the length of each interval equals its weight.

Proof. Consider the following instance consisting of two intervals with release dates $r_1 = 0$, $r_2 = \frac{1}{2}$ and weights $w_1 = 1$, $w_2 = M$ for some large value M. In the first case, both intervals have length $p_1 = p_2 = 1$, whereas we have in the second case $p_1 = 1$ and $p_2 = M$. In both cases the intervals are overlapping.

ALG has to accept the first interval in order to obtain a finite competitiveness. Then, it cannot accept the second interval, and for $M \to \infty$, we have $\text{OPT} \to \infty$, whereas $\text{ALG} = 1$. □

However, if there exist $w^{\min}$ and $w^{\max}$ such that $w^{\min} \leq w_i \leq w^{\max}$ for all intervals i, ignoring the weights and applying GREEDY^1 (Algorithm 5.1) yields a competitive ratio of $(k+1) \cdot \frac{w^{\max}}{w^{\min}}$ by Theorem 5.7. Similarly, we can see that no deterministic online algorithm can achieve a competitive ratio better than $k \cdot \frac{w^{\max}}{w^{\min}}$ by considering the instance from the proof of Proposition 5.3 with weights $w_1 := w^{\min}$, $w_2 := \ldots := w_{k+1} := w^{\max}$.

5.5 Conclusions

In this chapter we analyzed a novel online version of the interval scheduling problem where we are given a set of potential intervals and an upper bound k on the number of failing intervals.

While we obtained (almost) tight bounds on the competitive ratio for the case $m = 1$, the gaps between our upper and lower bounds increase for $m > 1$. Hence, a natural question is how to close these gaps.

The randomized algorithm we presented and the proof of its performance guarantee highly rely on the assumption that we consider laminar sets of intervals. It remains an open problem how to design randomized algorithms for arbitrary instances.

— 6 —
The Canadian Traveller Problem

In this chapter we consider the *Canadian traveller problem* (CTP), which is a variant of the *shortest path problem* without complete knowledge of the graph. We are given an undirected graph $G = (V, E)$ with non-negative edge costs (or lengths) $c : E \to \mathbb{R}_{\geq 0}$ and two designated vertices $s, t \in V$. The task is to find a path from s to t at minimum total costs. However, some of the edges may be blocked, and in an online setting we do not know whether an edge is blocked before reaching one of its endpoints.

In the *k-Canadian traveller problem* (k-CTP) we are additionally given an upper bound k on the number of blockages. Note that this online setting is similar to the online interval scheduling problem k-OIS considered in Chapter 5, where initially a set of potential intervals is given, but up to k of them can fail and an online algorithm learns that an interval fails not before the time it is supposed to be released.

Previous Work

The CTP has been introduced by Papadimitriou and Yannakakis (1991). They devise optimal online algorithms for special cases and show that the general problem of finding an online algorithm with bounded competitiveness is PSPACE-complete. Bar-Noy and Schieber (1991) consider the *recoverable* CTP where blocked edges may be reopened after some time. If a blocked edge is reached, an online algorithm is given the information when this edge will become traversable again. Furthermore, they present a strategy for the k-CTP that yields a solution with minimal worst-case length. Details to this worst-case analysis can be found in the work of Ben-David and Borodin (1994).

The k-CTP has been analyzed for the first time by means of competitive analysis by Westphal (2008). He shows that no deterministic online algorithm can be better than $(2k + 1)$-competitive, and the algorithm BACKTRACK that returns to s and recomputes a shortest path after a blockage is reached achieves this ratio. Furthermore, he shows that no randomized algorithm can achieve a competitive ratio better than $k + 1$. Recently, Demaine et al. (2014) designed a randomized algorithm that achieves a competitive ratio of $(1 + \frac{1}{\sqrt{2}})k + 1$

against an oblivious adversary, where $1 + \frac{1}{\sqrt{2}} \approx 1.71$.

Other aspects of the k-CTP have also been studied from a competitive analysis point of view. Xu et al. (2009) propose a $(2^{k+1} - 1)$-competitive greedy strategy. Although the competitiveness is significantly worse than the optimal ratio of $2k + 1$, they argue why this heuristic might be reasonable for navigating in certain urban traffic environments. Zhang and Xu (2011) analyze the k-CTP with multiple travelers that may use communication in order to find a shortest path. They show that a second traveler improves the competitive ratio to $k + 1$ and prove a lower bound of $2\lfloor k/L \rfloor + 1$, where L denotes the number of travelers. Su and Xu (2004) were the first ones to analyze simple strategies for the *recoverable* CTP. An optimal competitive online algorithm for this problem is given by Huang and LiaoShou (2012). Moreover, Su et al. (2008) analyze it from a risk-reward perspective, which goes back to Al-Binali (1999).

Büttner (2013) introduces several *Canadian type problems* that are classical routing problems with the additional online feature that up to k edges might be blocked as in our setting: In the *Canadian travelling salesman problem* (CTSP) one has to find a closed tour visiting every vertex at least once minimizing the total travel costs (this problem has also been studied by Liao and Huang (2014), who present an $\mathcal{O}(\sqrt{k})$-competitive algorithm for edge costs satisfying the triangle inequality). In the *Canadian latency travelling salesman problem*, every vertex has a due date, and the task is to find a closed tour minimizing the total (or maximal) latency. The *Canadian tour operator problem* allows to skip some vertices, and the task is to find a tour visiting the remaining vertices such that the sum of travel costs and penalty costs (for vertices that are not visited) is minimized.

Chapter Outline

In Section 6.1 definitions and previous results on the k-CTP are summarized. In particular, Theorem 6.3 states that no randomized online algorithm for the k-CTP can be better than $(k + 1)$-competitive (Westphal, 2008). In the proof an instance is used where all s-t-paths are node-disjoint. This is the motivation for Section 6.2, where we construct a randomized online algorithm for node-disjoint paths. Our algorithm achieves a competitive ratio of $k + 1$ and is therefore best possible. To the best of our knowledge, this is the first result on randomized algorithms for the k-CTP. We demonstrate possible extensions of our algorithm to other graph classes and limitations where this is not possible.

Finally, we consider in Section 6.3 the k-CTP with *uncertain recovery times*. In this setting, an online algorithm is given the information that an edge is currently blocked when reaching one of its endpoints. A blocked edge might, however, be reopened at some point in time which is not known to the online algorithm in advance. This generalizes the *recoverable* CTP where recovery

times are revealed as soon as a blockage is reached (cf. (Bar-Noy and Schieber, 1991)). We present a $(2k + 3)$-competitive algorithm for this problem and discuss possible extensions.

6.1 Problem Definition and Preliminaries

The *k-Canadian traveller problem* (k-CTP) we consider in this chapter is formally defined as follows:

Definition 6.1 (k-CTP)**.**
An instance of k-CTP is given by an undirected graph $G = (V, E)$ with non-negative edge costs $c : E \to \mathbb{R}_{\geq 0}$, designated vertices $s, t \in V$, and a set $\mathcal{B} \subseteq E$ (with $|\mathcal{B}| \leq k$) of blocked edges.

The task is to find a shortest path from s to t in $G - \mathcal{B}$, where an online algorithm learns that an edge e is blocked, i.e., $e \in \mathcal{B}$, only when reaching one of the endpoints of e.

We measure the quality of online algorithms for k-CTP by means of competitive analysis (see Section 1.1.4). For an instance σ, $\text{ALG}(\sigma)$ denotes the total distance traveled by an online algorithm ALG on instance σ, and $\text{OPT}(\sigma)$ denotes the length of the shortest s-t-path in this instance. A deterministic online algorithm ALG is c-competitive for k-CTP if $\text{ALG}(\sigma) \leq c \cdot \text{OPT}(\sigma)$ for every instance σ. Analogously, if ALG is a randomized online algorithm, it is c-competitive against an oblivious adversary if $\mathbb{E}[\text{ALG}(\sigma)] \leq c \cdot \text{OPT}(\sigma)$ for every instance σ.

For the basic setting of k-CTP, the notion that an edge is blocked means that it cannot be traversed at any point in time. In the following, we use the notation $c(P) := \sum_{e \in P} c(e)$ for the costs of an s-t-path P in G, and write $\mathcal{P}$ for the set of all (simple) s-t-paths.

Algorithm 6.1 BACKTRACK for k-CTP

```
1: while t is not reached do
2:    determine a shortest path P in G
3:    if a blocked edge e ∈ P is found then
4:       G := G − e, return to s
5:    end if
6: end while
```

Westphal (2008) proposed the BACKTRACK strategy, which is summarized in Algorithm 6.1 and based on the following idea: First, a shortest s-t-path P in the currently known graph G is computed. If a blocked edge $e \in P$ on this path is encountered, the algorithm returns to s, computes a new shortest path in $G - e$, and continues with this procedure until t is reached. Since at most k edges can fail, the following result holds:

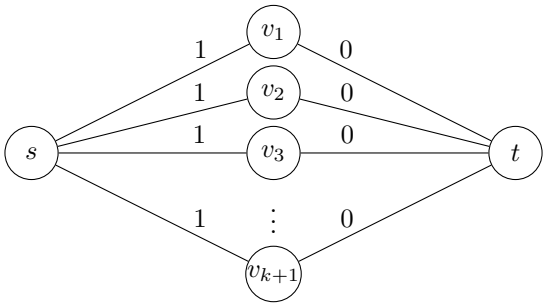


Figure 6.1: Illustration of the graph used in the proof of Theorem 6.3.

Theorem 6.2 (Westphal (2008))**.** *Algorithm 6.1 is* $(2k+1)$*-competitive for* k-CTP*.*

The next result shows that Algorithm 6.1 is best possible. Furthermore, it shows that even randomization does not help to obtain a competitive ratio better than $\Omega(k)$. We give a sketch of the proof as the considered instance will be important in the following.

Theorem 6.3 (Westphal (2008))**.** *For* k-CTP*,...*

(i) ... no deterministic online algorithm can achieve a competitive ratio better than $2k+1$*.*

(ii) ... no randomized online algorithm can achieve a competitive ratio better than $k+1$*.*

Proof. Consider the graph illustrated in Figure 6.1 consisting of $k+1$ node-disjoint s-t-paths $P_i = (s, v_i, t)$. Every deterministic online algorithm ALG is characterized by a permutation α of $\{1, \dots, k+1\}$, specifying the order in which the paths are chosen in order to find a path to t.

For the first part, assume that edges $(v_{\alpha(1)}, t), \dots, (v_{\alpha(k)}, t)$ are blocked. Then, ALG has to return to s in total k times before it reaches t in the last attempt via $P_{\sigma(k+1)}$, incurring total costs of $2k+1$. OPT chooses the non-blocked path $P_{\sigma(k+1)}$ at cost 1, yielding $\text{ALG} = (2k+1) \cdot \text{OPT}$.

For the second part, we apply Yao's Principle (Yao, 1977). We therefore define $k+1$ instances such that in instance i all edges (v_j, t) with $j \neq i$ are blocked. A probability distribution over these instances is given by choosing $i \in \{1, \dots, k+1\}$ uniformly at random.

The probability that a deterministic online algorithm ALG finds the unblocked path in the l-th attempt is $\frac{1}{k+1}$. In this case, its costs are $2l-1$, and the expected costs of the algorithm are therefore $\mathbb{E}[\text{ALG}] = \frac{1}{k+1} \sum_{l=1}^{k+1} 2l-1 = k+1$. Since $\mathbb{E}[\text{OPT}] = 1$, the claim follows. □

6.2 An Optimal Randomized Algorithm for Node-Disjoint Paths

In this section we consider graphs where all s-t-paths are node-disjoint (an example for such a graph was already given in the proof of Theorem 6.3). We assume that there are n such node-disjoint s-t-paths, i.e., $\mathcal{P} := \{P_1, \ldots, P_n\}$, and we denote the costs of path P_i by $c_i := c(P_i)$. Furthermore, we assume that the paths are sorted in non-decreasing order of costs, i.e., $c_1 \leq \ldots \leq c_n$.

One of the advantages of considering node-disjoint paths is the fact that the failure of one edge implies the blockage of one s-t-path (while for arbitrary graphs several paths might be blocked). The following result shows that we can restrict ourselves in the analysis to the case of $k+1$ different paths:

Lemma 6.4. *In the analysis of the k-CTP we can assume that there are in total $n = k+1$ s-t-paths, i.e., $\mathcal{P} = \{P_1, \ldots, P_{k+1}\}$.*

Proof. If $n < k+1$, we can simply add sufficiently many "dummy" paths with costs ∞.

If $n > k+1$, it suffices to consider the $k+1$ cheapest paths $P_1, \ldots, P_{k+1}$: Since at most k edges can fail and all s-t-paths are node-disjoint, at most k paths can be blocked. Thus, any competitive online algorithm would assign a positive probability only to these paths. □

The algorithm we present can be seen as a randomized version of BACKTRACK (Algorithm 6.1) and is based on the following idea: Initially, we consider the $k+1$ shortest s-t-paths in the graph. Although we do not know at this point which paths are blocked, we can be sure that at least one of these paths will be the actual shortest path since at most k of them can contain a blocked edge. We then define an appropriate probability distribution, and choose a path according to this distribution. If the chosen path is traversable, we reach t. If it is blocked, we return to s and repeat the procedure for the k shortest remaining paths.

First, we consider the case that only one edge can be blocked, i.e., $k = 1$:

Lemma 6.5. *Let P_1 and P_2 be the two cheapest s-t-paths with costs c_1 and c_2, respectively. The algorithm that chooses at the beginning path P_1 with probability $\pi_1 = \frac{c_2^2}{c_1^2+c_2^2}$ and path P_2 with probability $\pi_2 = \frac{c_1^2}{c_1^2+c_2^2}$ achieves a competitive ratio of 2.*

Proof. If P_1 is blocked, ALG has expected costs of at most $\pi_1(2c_1 + c_2) + \pi_2 c_2$. OPT chooses in this case P_2 and we have a ratio of

$$\frac{\mathbb{E}[\text{ALG}]}{\text{OPT}} \leq \frac{\pi_1(2c_1 + c_2) + \pi_2 c_2}{c_2}. \tag{6.1}$$

If P_2 is blocked, the ratio is analogously given by

$$\frac{\mathbb{E}\left[\text{ALG}\right]}{\text{OPT}} \leq \frac{\pi_1 c_1 + \pi_2(2c_2 + c_1)}{c_1}. \tag{6.2}$$

If neither P_1 nor P_2 is blocked, we have

$$\frac{\mathbb{E}\left[\text{ALG}\right]}{\text{OPT}} = \frac{\pi_1 c_1 + \pi_2 c_2}{c_1}. \tag{6.3}$$

Using the definitions of π_1 and π_2 for the expressions on the right-hand sides of (6.1), (6.2), and (6.3), we obtain that in all cases

$$\frac{\mathbb{E}\left[\text{ALG}\right]}{\text{OPT}} \leq \frac{(c_1 + c_2)^2}{c_1^2 + c_2^2},$$

which is bounded from above by 2. □

In the following, we consider an arbitrary $k \geq 1$ and first restrict ourselves to a special class of paths that have the *similar costs property* (see Definition 6.6). For these paths it is possible to derive a useful result which gives us access to a set of probability distributions that we can use for the randomized choice of a path (see Lemma 6.7). Afterwards, we show how it is possible to create for arbitrary graphs a suitable partitioning of the paths (see Definition 6.11) such that the paths within each class have the similar costs property, and we can make use of the probability distributions we derived before.

6.2.1 The Similar Costs Property

In Lemma 6.5 we have seen how to choose a suitable probability distribution for $k = 1$. Next, we show how this can be done for $k > 1$ if the costs of the paths do not differ "too much". We make this notion more precise by defining the *similar costs property*, which states that the costs of no path may exceed twice the average costs of all paths.

Definition 6.6. *Paths $\mathcal{P} \subseteq \{P_1, \ldots, P_{k+1}\}$ have the* similar costs property *if for all $P \in \mathcal{P}$ it holds that*

$$c(P) \leq \frac{2}{|\mathcal{P}|} \sum_{P' \in \mathcal{P}} c(P'). \tag{6.4}$$

For paths that fulfill the similar costs property, we are able to show the following result, which will be crucial in the analysis later on:

Lemma 6.7. *If paths $\mathcal{P} \subseteq \{P_1, \ldots, P_{k+1}\}$ have the similar costs property, the polyhedron $Q(\mathcal{P})$ is not empty, where*

$$Q(\mathcal{P}) := \Big\{ \pi \in \mathbb{R}_{\geq 0}^{|\mathcal{P}|} : (2 - |\mathcal{P}|)\, \pi_P + \frac{2}{c(P)} \sum_{\substack{P' \in \mathcal{P}: \\ P' \neq P}} c(P')\, \pi_{P'} \leq 1 \quad \forall\, P \in \mathcal{P}, \\ \sum_{P \in \mathcal{P}} \pi_P = 1 \Big\}.$$

Proof. In the proof we make use of the following description of the polyhedron $Q(\mathcal{P})$. For

$$A(\mathcal{P}) := \left\{ \pi \in \mathbb{R}_{\geq 0}^{|\mathcal{P}|} :\ (2 - |\mathcal{P}|)\, \pi_P + \frac{2}{c(P)} \sum_{\substack{P' \in \mathcal{P}: \\ P' \neq P}} c(P')\, \pi_{P'} \leq 1 \quad \forall\, P \in \mathcal{P} \right\}$$

$$B(\mathcal{P}) := \left\{ \pi \in \mathbb{R}_{\geq 0}^{|\mathcal{P}|} :\ \sum_{P \in \mathcal{P}} \pi_P = 1 \right\},$$

it holds that $Q(\mathcal{P}) = A(\mathcal{P}) \cap B(\mathcal{P})$.

Consider $\bar{\pi}$ with

$$\bar{\pi}_P := \frac{(2 - |\mathcal{P}|) \cdot c(P) + 2 \sum\limits_{\substack{P' \in \mathcal{P}: \\ P' \neq P}} c(P')}{c(P) \cdot |\mathcal{P}|^2}.$$

By the similar costs property we have $2c(P) \geq |\mathcal{P}| \cdot c(P)$, i.e., $\bar{\pi}_P$ is non-negative for all $P \in \mathcal{P}$. Note that this is the only point in the proof where this assumption is used.

Furthermore, $\bar{\pi} \in A(\mathcal{P})$ since we have for all $P \in \mathcal{P}$

$$\begin{aligned} &(2 - |\mathcal{P}|)\, \bar{\pi}_P + \frac{2}{c(P)} \sum_{\substack{P' \in \mathcal{P}: \\ P' \neq P}} c(P') \cdot \bar{\pi}_{P'} \\ &= (2 - |\mathcal{P}|) \underbrace{\frac{(2 - |\mathcal{P}|) \cdot c(P) + 2 \sum\limits_{\substack{P' \in \mathcal{P}: \\ P' \neq P}} c(P')}{c(P) \cdot |\mathcal{P}|^2}}_{=\bar{\pi}_P} \\ &\quad + \frac{2}{c(P)} \sum_{\substack{P' \in \mathcal{P}: \\ P' \neq P}} c(P') \underbrace{\frac{(2 - |\mathcal{P}|) \cdot c(P') + 2 \sum\limits_{\substack{P'' \in \mathcal{P}: \\ P'' \neq P'}} c(P'')}{c(P') \cdot |\mathcal{P}|^2}}_{=\bar{\pi}_{P'}} \end{aligned} \tag{6.5}$$

$$
\begin{aligned}
&= \frac{(2-|\mathcal{P}|)^2}{|\mathcal{P}|^2} + \frac{2\,(2-|\mathcal{P}|)}{c(P)\cdot|\mathcal{P}|^2} \sum_{\substack{P'\in\mathcal{P}:\\ P'\neq P}} c(P') \\
&\quad + \frac{2\,(2-|\mathcal{P}|)}{c(P)\cdot|\mathcal{P}|^2} \sum_{\substack{P'\in\mathcal{P}:\\ P'\neq P}} c(P') + \frac{4}{c(P)\cdot|\mathcal{P}|^2} \underbrace{\sum_{\substack{P'\in\mathcal{P}:\\ P'\neq P}} \sum_{\substack{P''\in\mathcal{P}:\\ P''\neq P'}} c(P'')}_{=(|\mathcal{P}|-1)\cdot c(P)+(|\mathcal{P}|-2)\cdot \sum_{\substack{P'\in\mathcal{P}:\\ P'\neq P}} c(P')} \\
&= \underbrace{\frac{(2-|\mathcal{P}|)^2}{|\mathcal{P}|^2} + \frac{4(|\mathcal{P}|-1)}{|\mathcal{P}|^2}}_{=1} \\
&\quad + \underbrace{\left(\frac{2\,(2-|\mathcal{P}|)}{c(P)\cdot|\mathcal{P}|^2} + \frac{2\,(2-|\mathcal{P}|)}{c(P)\cdot|\mathcal{P}|^2} + \frac{4\,(|\mathcal{P}|-2)}{c(P)\cdot|\mathcal{P}|^2}\right)}_{=0} \cdot \sum_{\substack{P'\in\mathcal{P}:\\ P'\neq P}} c(P') \\
&= 1.
\end{aligned} \tag{6.6}
$$

Next, observe that

$$
\begin{aligned}
\sum_{P\in\mathcal{P}} \bar{\pi}_P &= \sum_{P\in\mathcal{P}} \frac{(2-|\mathcal{P}|)\cdot c(P) + 2\sum_{\substack{P'\in\mathcal{P}:\\ P'\neq P}} c(P')}{c(P)\cdot|\mathcal{P}|^2} \\
&= \frac{2-|\mathcal{P}|}{|\mathcal{P}|} + \frac{2\sum_{P\in\mathcal{P}} \left(\sum_{\substack{P'\in\mathcal{P}:\\ P'\neq P}} c(P') \cdot \prod_{\substack{P''\in\mathcal{P}:\\ P''\neq P}} c(P'')\right)}{\prod_{P\in\mathcal{P}} c(P)\cdot|\mathcal{P}|^2} \\
&= \frac{2-|\mathcal{P}|}{|\mathcal{P}|} + \frac{2\sum_{\substack{P,P'\in\mathcal{P}:\\ P\neq P'}} \left(c(P')^2 \cdot \prod_{\substack{P''\in\mathcal{P}:\\ P\neq P''\neq P'}} c(P'')\right)}{\prod_{P\in\mathcal{P}} c(P)\cdot|\mathcal{P}|^2} \\
&= \frac{2-|\mathcal{P}|}{|\mathcal{P}|} + \frac{\sum_{\substack{P,P'\in\mathcal{P}:\\ P\neq P'}} \left(\overbrace{\left(c(P)^2+c(P')^2\right)}^{\geq 2c(P)c(P')} \cdot \prod_{\substack{P''\in\mathcal{P}:\\ P\neq P''\neq P'}} c(P'')\right)}{\prod_{P\in\mathcal{P}} c(P)\cdot|\mathcal{P}|^2} \\
&\geq \frac{2-|\mathcal{P}|}{|\mathcal{P}|} + \frac{2\cdot(|\mathcal{P}|-1)\cdot|\mathcal{P}|\cdot\prod_{P\in\mathcal{P}} c(P)}{\prod_{P\in\mathcal{P}} c(P)\cdot|\mathcal{P}|^2} \\
&= 1.
\end{aligned} \tag{6.7}
$$

By definition of $A(\mathcal{P})$ it holds that $0 \in A(\mathcal{P})$, and (6.6) shows that $\bar{\pi} \in A(\mathcal{P})$. Thus, it follows by the convexity of $A(\mathcal{P})$ that $\lambda\bar{\pi} \in A(\mathcal{P})$ for all $\lambda \in [0,1]$. Observe that by (6.7), we have $\lambda^* := 1/(\sum_{P\in\mathcal{P}} \bar{\pi}_P) \in [0,1]$, and, therefore, it holds that $\pi^* := \lambda^*\bar{\pi} \in A(\mathcal{P})$. Furthermore, π^* is defined such that $\sum_{P\in\mathcal{P}} \pi^*_P = 1$, i.e., $\pi^* \in B(\mathcal{P})$. This implies $\pi^* \in A(\mathcal{P}) \cap B(\mathcal{P}) = Q(\mathcal{P})$, and the claim follows. □

Note that the proof of Lemma 6.7 is constructive in the sense that we actually specify an element $\pi^* \in Q(\mathcal{P})$. We will use this probability distribution (or possibly any other element in $Q(\mathcal{P})$) as our "suitable" distribution over the set of s-t-paths in the algorithm.

6.2.2 An Algorithm for Strong Similar Costs

First, we present an algorithm for the case that *every* subset of the paths has the similar costs property. We define this formally as follows:

Definition 6.8. *Paths $\mathcal{P} \subseteq \{P_1, \ldots, P_{k+1}\}$ have the* strong similar costs property *if $\mathcal{P}'$ has the similar costs property for all $\mathcal{P}' \subseteq \mathcal{P}$.*

Note that this is a stronger assumption than the (weak) similar costs property (Definition 6.6):

Remark 6.9. *Obviously, every set of paths that has the strong similar costs property also fulfills the (weak) similar costs property. However, the converse does, in general, not hold true as the following example shows:*

Consider paths $\mathcal{P} = \{P_1, P_2, P_3, P_4\}$ with costs $c_1 = c_2 = 1, c_3 = c_4 = x$ for some $x \geq 1$. For every choice of x, $\mathcal{P}$ fulfills the similar costs property, since $\frac{2}{|\mathcal{P}|}\sum_{P\in\mathcal{P}} c(P) = 1 + x \geq x$.

But if we consider $\mathcal{P}' = \{P_1, P_2, P_3\} \subsetneq \mathcal{P}$, it holds that $\frac{2}{|\mathcal{P}'|}\sum_{P\in\mathcal{P}'} c(P) = \frac{4+2x}{3}$, which is strictly smaller than $c_3 = x$ for $x > 4$.

Our algorithm for strong similar costs then works as follows: We choose a path at random according to a probability distribution $\pi \in Q(\mathcal{P})$ over the set of all paths $\mathcal{P}$ that are not yet known to be blocked. If we reach t via the chosen path, we are done. Otherwise, we remove this path from our set, and repeat the procedure. Observe that since we assume the strong similar costs property, it always holds that the remaining set of paths fulfills the similar costs property, and by Lemma 6.7 we can be sure that there exists a suitable probability distribution. Formally, this is summarized in Algorithm 6.2.

Theorem 6.10. *If the strong similar costs property holds, Algorithm 6.2 is $(k+1)$-competitive against an oblivious adversary.*

Algorithm 6.2 RAND-BACKTRACK for strong similar costs

Input: Set of paths $\mathcal{P}$ satisfying the strong similar costs property.

1: **while** t is not reached **do**
2: determine a probability distribution $\pi \in Q(\mathcal{P})$ over the set of paths $\mathcal{P}$ (Lemma 6.7), and choose a path $P \in \mathcal{P}$ according to this distribution
3: **if** a blocked edge $e \in P$ is found **then**
4: set $\mathcal{P} := \mathcal{P} \setminus \{P\}$, and return to s
5: **end if**
6: **end while**

Proof. We prove the result by induction over k.

The basis for $k = 1$ follows from Lemma 6.5 since the specified probability distribution satisfies $\pi \in Q(\mathcal{P})$.

For the inductive step, assume that the statement has been shown for $k-1$. We show that the result then also holds for k, i.e., the competitive ratio of Algorithm 6.2 (ALG) can be bounded by $k+1$ if up to k blockages occur.

Every instance of the k-CTP is specified by a set of blocked paths $\mathcal{B}$ (with $|\mathcal{B}| \leq k$) and traversable paths $\mathcal{T} := \mathcal{P} \setminus \mathcal{B}$. Then,

$$\text{OPT} = \min_{P \in \mathcal{T}} c(P) =: c(P^*).$$

The competitive ratio is therefore given by

$$\frac{\mathbb{E}\left[\text{ALG}\right]}{\text{OPT}} \leq \frac{\sum\limits_{P \in \mathcal{T}} \pi_P \, c(P) + \sum\limits_{P \in \mathcal{B}} \pi_P \left(2c(P) + V_{P^*}^{-P}\right)}{c(P^*)}. \tag{6.8}$$

Here, the first sum in the enumerator corresponds to the case that ALG chooses a traversable path $P \in \mathcal{T}$. Then, t is reached at cost $c(P)$.

The second sum is for the case that a blocked path $P \in \mathcal{B}$ is chosen. Then, ALG has costs of at most $2c(P)$ for trying to traverse P and returning to s. After this unsuccessful attempt, at most $k-1$ additional edges can fail. We denote the expected remaining costs of ALG until reaching t by $V_{P^*}^{-P}$ (the subscript refers to P^* being the optimal path, and the superscript indicates that ALG already learned that P is blocked).

By the strong similar costs assumption, the set $\mathcal{P} \setminus \{P\}$ has the similar costs property. Thus, we can apply ALG to these paths, and we know that at most $k-1$ of them can possibly fail. By induction hypothesis ALG is k-competitive for this remaining problem, which yields that $V_{P^*}^{-P} \leq k \cdot c(P^*)$. Hence, the right-hand side of (6.8) can be bounded from above by

$$\sum_{P \in \mathcal{T}} \pi_P \frac{c(P)}{c(P^*)} + \sum_{P \in \mathcal{B}} \pi_P \left(\frac{2c(P)}{c(P^*)} + k\right) \leq \pi_{P^*} + \sum_{\substack{P \in \mathcal{P}: \\ P \neq P^*}} \pi_P \left(\frac{2c(P)}{c(P^*)} + k\right). \tag{6.9}$$

We need to show that the right hand side of (6.9) is bounded from above by $k+1$ for all instances of the problem. Using that $|\mathcal{P}| = k+1$ this is equivalent to showing that for all $P^* \in \{P_1, \ldots, P_{k+1}\}$:

$$\pi_{P^*} + \frac{2}{c(P^*)} \sum_{\substack{P \in \mathcal{P}: \\ P \neq P^*}} \pi_P + (|\mathcal{P}| - 1) \sum_{\substack{P \in \mathcal{P}: \\ P \neq P^*}} \pi_P \leq |\mathcal{P}| \cdot \underbrace{\sum_{P \in \mathcal{P}} \pi_P}_{=1}$$

$$\iff \qquad (2 - |\mathcal{P}|)\, \pi_{P^*} + \frac{2}{c(P^*)} \sum_{\substack{P \in \mathcal{P}: \\ P \neq P^*}} c(P)\, \pi_P \leq 1. \tag{6.10}$$

Algorithm 6.2 chooses $\pi \in Q(\mathcal{P})$ according to Lemma 6.7 exactly such that it is a probability distribution and constraints (6.10) are fulfilled. Hence, the claim follows. □

6.2.3 An Algorithm for Arbitrary Costs

In Algorithm 6.2 it was necessary to assume that paths $\mathcal{P}$ have the strong similar costs property, i.e., any subset of $\mathcal{P}$ has the similar costs property. For arbitrary costs, we must not apply Algorithm 6.2 since we cannot ensure that there always exists a suitable probability distribution (see the inductive step in the proof of Theorem 6.10).

Hence, we use the following idea for arbitrary costs: We first partition the set of paths into suitable classes. We then want to process each of these classes in non-decreasing order of costs. The classes are defined to be maximal with respect to the (weak) similar costs property, i.e., the paths in each class fulfill the similar costs property, and a class cannot be extended without violating this property. Therefore, we can perform at least the first iteration of Algorithm 6.2 for each class (since by Lemma 6.7 a suitable probability distribution exists).

If we reach a blocked edge and remove a path, it can happen that the corresponding class does not have the similar costs property anymore (see Remark 6.9 for an example). In this case, we cannot apply Algorithm 6.2 to the remaining paths. We therefore perform another partitioning of these paths and obtain classes that have again the similar costs property.

Before we present and analyze the algorithm for the case of arbitrary costs in more detail, we first define formally our concept of a *similar costs partition*:

Definition 6.11. *We call $\mathcal{K} = \{\mathcal{K}_1, \ldots, \mathcal{K}_l\}$ a* similar costs partition *of the paths $\mathcal{P} \subseteq \{P_1, \ldots, P_{k+1}\}$ if*

(i) $\mathcal{K}_i \subseteq \mathcal{P}$ for all $i \in \{1, \ldots, l\}$ and $\dot{\bigcup}_{i=1}^{l} \mathcal{K}_i = \mathcal{P}$,

(ii) it is possible to assign to each path $P \in \mathcal{P}$ *a* class $\mathcal{K}_P \in \mathcal{K}$ *and a* position $n_P \in \{1, \ldots, |\mathcal{K}_P|\}$ *in this class*[1] *such that we have a lexicographical ordering with respect to the costs, i.e.,*

$$\begin{aligned} &c(\mathcal{K}_1, 1) \leq c(\mathcal{K}_1, 2) \leq \ldots \leq c(\mathcal{K}_1, |\mathcal{K}_1|) \\ &\leq c(\mathcal{K}_2, 1) \leq c(\mathcal{K}_2, 2) \leq \ldots \leq c(\mathcal{K}_2, |\mathcal{K}_2|) \\ &\leq \ldots \\ &\leq c(\mathcal{K}_l, 1) \leq c(\mathcal{K}_l, 2) \leq \ldots \leq c(\mathcal{K}_l, |\mathcal{K}_l|), \end{aligned}$$

(iii) and it holds that

$$c(\mathcal{K}_i, n) \leq \frac{2}{n} \sum_{j=1}^{n} c(\mathcal{K}_i, j) \quad \text{for all } \mathcal{K}_i \in \mathcal{K} \text{ and } n \in \{1, \ldots, |\mathcal{K}_i|\}, \tag{6.11}$$

$$c(\mathcal{K}_{i+1}, 1) > \frac{2}{|\mathcal{K}_i| + 1} \left(\sum_{j=1}^{|\mathcal{K}_i|} c(\mathcal{K}_i, j) + c(\mathcal{K}_{i+1}, 1) \right) \quad \text{for all } i \in \{1, \ldots, l-1\}. \tag{6.12}$$

Observe that for $n = |\mathcal{K}_i|$, (6.11) states that the paths in class $\mathcal{K}_i$ have the similar costs property and (6.12) ensures that the classes are as large as possible.

Remark 6.12. *A similar costs partition can easily be constructed by going through the set of paths in non-decreasing order of costs. Paths are added to the current class as long as possible such that the similar costs property holds. If adding an additional path violates the property, a new class is opened.*

Example 6.13. *For a set of paths with costs as given below we have the following similar costs partitioning:*

$$[c(P_1), \ldots, c(P_{11})] = [\underbrace{1, 2, 3, 5}_{\mathcal{K}_1}, \underbrace{8, 9, 11, 12, 14}_{\mathcal{K}_2}, \underbrace{28, 35}_{\mathcal{K}_3}].$$

Formally, our algorithm is summarized in Algorithm 6.3. Initially, the algorithm is executed as RAND-BACKTRACK($\mathcal{P}$) with the complete set of all s-t-paths $\mathcal{P} = \{P_1, \ldots, P_{k+1}\}$. Observe that if after the first partitioning step every class already has the strong similar costs property, this algorithm breaks down to applying Algorithm 6.2 to each class.

[1] Formally, this is given by a bijective mapping $\phi\colon \mathcal{P} \to \mathcal{K} \times \dot{\cup}_{i=1}^{l} \{1, \ldots, |\mathcal{K}_i|\}$ with $\phi(P) := (\mathcal{K}_P, n_P)$. The costs of a path $P \in \mathcal{P}$ are therefore $c(P) = c(\phi^{-1}(\mathcal{K}_P, n_P))$. In order to simplify notation, we write for short $c(\mathcal{K}_P, n_P)$.

Algorithm 6.3 RAND-BACKTRACK for general costs

Input: Set of paths $\mathcal{P}$ with arbitrary costs.

while t is not reached **do**
  if $\mathcal{P}$ has the similar costs property **then**
    determine a probability distribution $\pi \in Q(\mathcal{P})$ over the set of paths $\mathcal{P}$ (Lemma 6.7), and choose a path $P^* \in \mathcal{P}$ according to this distribution
    if a blocked edge $e \in P^*$ is found **then**
      return to s
      RAND-BACKTRACK$(\mathcal{P} \setminus \{P^*\})$
    end if
  else
    determine a similar costs partition $\mathcal{K} = (K_1, \ldots, K_{|\mathcal{K}|})$ of $\mathcal{P}$
    for $K \in \mathcal{K}$ **do**
      return to s
      RAND-BACKTRACK(K)
    end for
  end if
end while

Theorem 6.14. *Algorithm 6.3 is* $(k+1)$*-competitive against an oblivious adversary.*

Proof. We prove the result again by induction over k. As before, the basis for $k = 1$ follows by Lemma 6.5 since $\pi \in Q(\mathcal{P})$.

For the inductive step, assume that the statement has already been shown for $k-1$. In order to show that the result then also holds for k, we distinguish two cases.

First, we consider the case that $\mathcal{P} = \{P_1, \ldots, P_{k+1}\}$ has the similar costs property. Then, we know by Lemma 6.7 that there exists a probability distribution $\pi \in Q(\mathcal{P})$, which is used in the first iteration of the algorithm for choosing a path $P \in \mathcal{P}$ at random.

If the chosen path P is not blocked, we reach t at costs $c(P)$. If a blocked edge $e \in P \cap \mathcal{B}$ is encountered, the algorithm returns to s and incurs costs of at most $2c(P)$. As before, we denote the expected costs of ALG in the remaining network by $V_{P^*}^{-P}$, and the path chosen by OPT by P^*. Hence, the competitive ratio is given as in (6.8) by

$$\frac{\mathbb{E}\left[\text{ALG}\right]}{\text{OPT}} \leq \frac{\sum\limits_{P \in \mathcal{T}} \pi_P \, c(P) + \sum\limits_{P \in \mathcal{B}} \pi_P \left(2c(P) + V_{P^*}^{-P}\right)}{c(P^*)}. \tag{6.13}$$

The set $\mathcal{P} \setminus \{P\}$ contains at most $k-1$ failing paths, and, thus, we know by induction hypothesis that the algorithm can be applied to this remaining

problem[2], and the expected costs for reaching t are at most $k \cdot c(P^*)$. Analogously to the proof of Theorem 6.10 it follows that (6.13) is at most $k+1$ (cf. (6.9) and the remainder of the proof).

Next, we consider the case that $\mathcal{P}$ does not have the similar costs property. In this case, ALG constructs first a partitioning $\mathcal{K} = \{K_1, \ldots, \mathcal{K}_l\}$.

Let the path chosen by OPT be contained in class $\mathcal{K}_o$ for some $o \in \{1, \ldots, l\}$. Hence, we know that all paths in classes $\mathcal{K}_1, \ldots, \mathcal{K}_{o-1}$ are blocked (since they have by construction smaller costs).

ALG tries to traverse all of these paths first before choosing any path in class $\mathcal{K}_o$, and costs of $2\sum_{P \in \mathcal{K}_i : i<o} c(P)$ are incurred for these unsuccessful attempts. When processing class $\mathcal{K}_o$, we know by construction that it has the similar costs property and contains at most $|\mathcal{K}_o| < k$ paths that can fail.[3] Thus, it follows by induction hypothesis that the expected costs for processing $\mathcal{K}_o$ until t is reached are at most $|\mathcal{K}_o| \cdot \text{OPT}$. Altogether, we have

$$\begin{aligned}
\mathbb{E}[\text{ALG}] &\leq 2\sum_{i=1}^{o-1} \sum_{P \in \mathcal{K}_i} c(P) + |\mathcal{K}_o| \cdot \text{OPT} \\
&\overset{(6.12)}{\leq} \sum_{i=1}^{o-1} \left((|\mathcal{K}_i| - 1) \cdot \underbrace{c(\mathcal{K}_{i+1}, 1)}_{\leq \text{OPT}} \right) + |\mathcal{K}_o| \cdot \text{OPT} \\
&\leq \left(\underbrace{-l+1}_{\leq 0} + \underbrace{\sum_{i=1}^{o} |\mathcal{K}_i|}_{\leq k+1} \right) \cdot \text{OPT} \\
&\leq (k+1) \cdot \text{OPT}.
\end{aligned}$$

Here, the second inequality follows since (6.12) holds (i.e., the classes are maximal with respect to the similar costs property). The last inequality holds since there are $k+1$ paths in total (see Lemma 6.4). □

6.2.4 Extensions and Limitations

Serial Compositions

Our result about the existence of a $(k+1)$-competitive randomized online algorithm generalizes straightforward to graphs that are serial compositions of graphs with node-disjoint paths.

[2]Note that, in order to apply the induction hypothesis, we do *not* need to require that $\mathcal{P} \setminus \{P\}$ has the similar costs property.

[3]Note that there are at least two classes and every class contains at least two elements. By Lemma 6.4 there are at most $k+1$ paths in total, and, thus, it holds that $|\mathcal{K}_o| < k$.

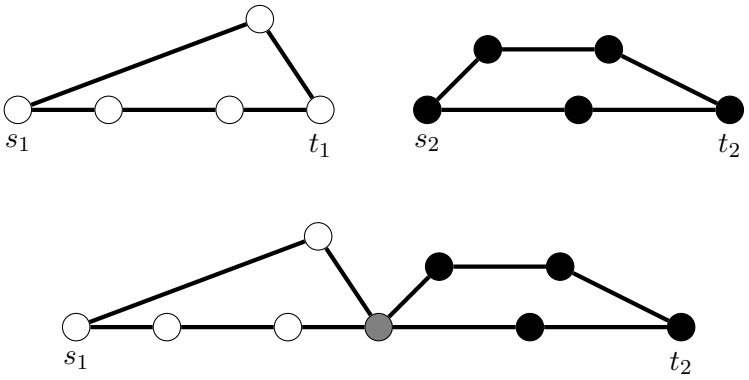


Figure 6.2: Illustration of a serial composition of two graphs consisting of node-disjoint paths.

A *serial composition* of two graphs G_1 and G_2 (with origin-destination pairs (s_1, t_1) and (s_2, t_2), respectively) is the graph that emerges from identifying t_1 with s_2. The new origin-destination pair is (s_1, t_2), as illustrated in Figure 6.2.

We can apply Algorithm 6.3 to all graphs G that are serial compositions of a number of graphs $G_1, \ldots, G_p$ that consist of node-disjoint paths[4]: We use our algorithm to find a path from s_i to t_i for every component G_i. By Theorem 6.14 we then have that $\mathbb{E}[\textsc{alg}(G_i)] \leq (k+1) \cdot \textsc{opt}(G_i)$, and the structure of G implies $\sum_{i=1}^{p} \textsc{opt}(G_i) = \textsc{opt}(G)$. Thus, it holds that

$$\mathbb{E}[\textsc{alg}] = \sum_{i=1}^{p} \mathbb{E}[\textsc{alg}(G_i)] \leq (k+1) \cdot \sum_{i=1}^{p} \textsc{opt}(G_i) = (k+1) \cdot \textsc{opt}.$$

Series-Parallel Compositions

We have seen that our algorithm can be applied sequentially to serial compositions of graphs whose paths are node-disjoint. Thus, it is natural to study the more general concept of series-parallel compositions.

A *parallel composition* of two graphs G_1 and G_2 (with origin-destination pairs (s_1, t_1) and (s_2, t_2), respectively) is the graph that emerges from identifying s_1 with s_2 and t_1 with t_2.

Unfortunately, our result does not carry over to graphs that are constructed by a sequence of serial and parallel compositions of node-disjoint paths, since we cannot simply choose a (parallel) component randomly and apply our algorithm to all paths in this component:

[4]The serial composition is defined iteratively as follows: Assume we have constructed the serial composition $\mathring{G}_l$ of $G_1, \ldots, G_l$ for some l. The serial composition of $G_1, \ldots, G_{l+1}$ is then defined as the serial composition of $\mathring{G}_l$ and G_{l+1}.

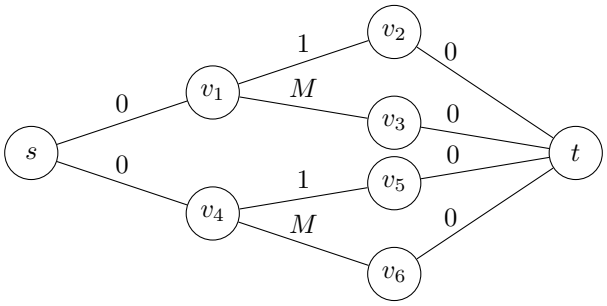


Figure 6.3: A straightforward application of the randomized algorithm to series-parallel compositions does not yield a bounded competitiveness.

Consider the graph shown in Figure 6.3 consisting of two parallel components. By the symmetry of the instance the best we can do (by means of minimizing the competitive ratio) is to choose to go to v_1 and v_2 with probability $\frac{1}{2}$ each in the first step.

Assume we choose to go to v_1 first and the edge (v_2, t) is blocked. If we stay in this component and explore the other path v_1-v_3-t, we incur a total cost of at least M, i.e., $\mathbb{E}[\text{ALG}] \to \infty$ for $M \to \infty$. If, however, the path s-v_4-v_5-t is open, we have $\text{OPT} = 1$.

Considering the $k+1$ Cheapest Paths

In Algorithm 6.3 we could restrict ourselves to the $k+1$ cheapest paths. This will not help in order to obtain a competitive ratio better than the deterministic ratio of $2k+1$ on general graphs:

Consider the graph shown in Figure 6.4 with three possible s-t-paths and $k = 1$. Assume we choose a probability distribution over the cheapest $k+1 = 2$ paths, which are in this case the two (s, v_1, t)-paths, i.e., we go to v_1 in the first step with probability 1. If (v_1, t) is blocked, we have to return to s, and this yields a ratio of

$$\frac{\mathbb{E}[\text{ALG}]}{\text{OPT}} = \frac{3+\epsilon}{1+\epsilon},$$

which tends to $3 = 2k+1$ as $\epsilon \to 0$.

Considering $k+1$ Node-Disjoint Paths

It is neither possible to improve the competitive ratio for arbitrary graphs when choosing a set of $k+1$ node-disjoint paths. Therefore, consider the graph shown in Figure 6.5 where up to $k = 1$ edge may fail.

The graph consists of three s-t-paths: $P_1 = (s, v_1, v_3, t)$, $P_2 = (s, v_2, v_3, t)$, and $P_3 = (s, t)$. Paths P_1 and P_2 are not node-disjoint. A probability distribu-

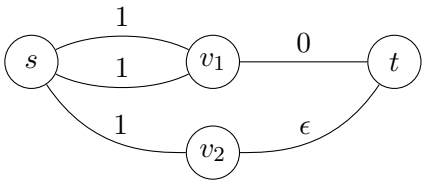


Figure 6.4: Defining a probability distribution over the $k+1$ cheapest paths does not improve the competitive ratio.

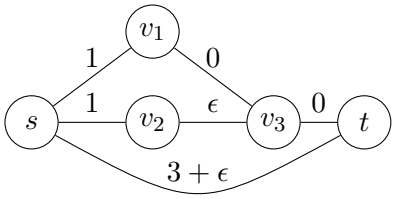


Figure 6.5: Defining a probability distribution over $k+1$ node-disjoint paths does not improve the competitive ratio.

tion over the set of node-disjoint paths is therefore specified by probabilities π_1 for path P_1, and π_3 for P_3 (assigning a positive probability to P_2 instead of P_1 does not improve the competitive ratio). If only the edge (v_1, v_3) is blocked, we have

$$\frac{\mathbb{E}\left[\text{ALG}\right]}{\text{OPT}} \geq \frac{\pi_1(3+\epsilon) + \pi_3(3+\epsilon)}{1+\epsilon}.$$

Since $\pi_1 + \pi_3 = 1$, this ratio goes to $3 = 2k+1$ for $\epsilon \to 0$ and is therefore not better than the deterministic ratio.

6.3 Uncertain Recovery Times

In this section we consider the k-CTP with *uncertain recovery times*, which we denote by k-CTP-URT. As in the classical k-CTP, up to k edges are initially blocked, but now each of these edges can be reopened after some time, which we call the *recovery time*. An online algorithm moves through the network and learns about the current blockage state of a particular edge e when it is located at one of the endpoints of e. If an edge is known to be traversable, it will always be open from this point in time on (a justification for this assumption is given in Proposition 6.17).

Due to the recovery possibility of an edge, it could make sense for an algorithm to wait at some of the vertices. These *idle times* should also be included

in the objective function, and we minimize the *total travel time* needed by an algorithm, which is given as the sum of all edge travel times on the final path from s to t and all idle times.

Note that, in order to have a consistent notion, we refer in this section to travel times for the edges rather than edge costs as we did before.

Formally, the problem k-CTP-URT is defined as follows:

Definition 6.15 (k-CTP-URT)**.**
An instance of k-CTP-URT is given by an undirected graph $G = (V, E)$ with non-negative travel times $c : E \to \mathbb{R}_{\geq 0}$ for the edges, designated vertices $s, t \in V$, a set $\mathcal{B} \subseteq E$ (with $|\mathcal{B}| \leq k$) of (initially) blocked edges*, and recovery times $b : \mathcal{B} \to \mathbb{R}_{\geq 0}$ for the blocked edges (from which point in time on the edges are traversable).*

An online algorithm learns that $e \in \mathcal{B}$ only when reaching one of the endpoints of e. The recovery time of e remains unknown until the edge is reached at some point in time later than $b(e)$. The task is to find a path from s to t in G that minimizes the total travel time, where an edge $e \in \mathcal{B}$ must not be used before time $b(e)$.

An optimal offline algorithm OPT knows all blocked edges and their recovery times in advance. Since open edges will never become blocked again, we can assume that OPT does not wait at any node other than s.

Note that our setting is different from the *recoverable* CTP, which was analyzed by means of competitive analysis, e.g., in (Su and Xu, 2004; Huang and LiaoShou, 2012): in their setting, an online algorithm already obtains the recovery time for a blocked edge as soon as an endpoint of this edge is reached for the first time.

If we consider the instance used in the proof of Theorem 6.3 and set the recovery times to infinity, we immediately obtain the following lower bound:

Corollary 6.16. *No deterministic online algorithm for the k-CTP-URT can be better than $(2k + 1)$-competitive.*

In our setting, we do not allow that edges can arbitrarily change their blockage state. This is motivated by the following result:

Proposition 6.17. *If edges that are traversable may become blocked again at some later point in time, no randomized online algorithm can achieve a finite competitiveness.*

Proof. Consider the graph illustrated in Figure 6.6. We want to apply Yao's Principle (Yao, 1977) and therefore consider the following two instances:

In instance σ_1, the edge (v_1, t) is blocked and never reopened, whereas (v_2, t) is initially traversable but blocked from time $1 + \epsilon$ on, for some small $\epsilon > 0$. Instance σ_2 has a similar blockage scenario: (v_2, t) is blocked forever,

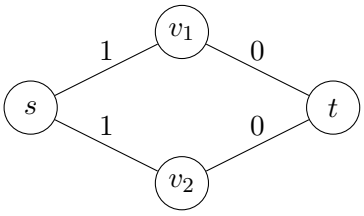


Figure 6.6: Graph used in the proof of Proposition 6.17.

whereas (v_1, t) is only traversable until time $1 + \epsilon$. We define a probability distribution by choosing each instance with probability $\frac{1}{2}$.

Let a deterministic algorithm ALG choose, without loss of generality, path s-v_1-t first. Then, its total travel time is ∞ on instance σ_1 (since t cannot be reached), and 1 on σ_2, i.e., $\mathbb{E}[\text{ALG}] = \frac{1}{2} \cdot \infty + \frac{1}{2} \cdot 1 = \infty$. For σ_1 and σ_2, OPT chooses s-v_2-t and s-v_1-t, respectively, and we have $\mathbb{E}[\text{OPT}] = 1$. It follows by Yao's Principle that no randomized online algorithm can be competitive. □

6.3.1 A Competitive Algorithm

Next, we present a deterministic online algorithm for k-CTP-URT which works on general graphs and is based on the following idea:

Similar to the BACKTRACK strategy, the algorithm chooses in each iteration an s-t-path and tries to traverse it. If an edge on this path is reached in a blocked state, the algorithm returns to s (without waiting) and chooses again another path until t is reached. Obviously, the algorithm should initially prefer "short" paths. However, in contrast to Algorithm 6.1, it can now make sense to attempt paths several times in order to hedge against the adversary.

We keep for each path $P \in \mathcal{P}$ a value $\pi(P)$ that is a lower bound on the total travel time needed for reaching t via this path. We then choose in each iteration a path with minimal π-value and try to traverse it. If we reach t, we are done. Otherwise, we encounter a blocked edge e on our chosen path, and we check for all paths $P \in \mathcal{P}$ with $e \in P$ if we can update $\pi(P)$ to the current time (if this improves the lower bound). We then return to s (without waiting) and repeat this procedure. Since we know that at the time of our arrival at the blocked edge it was still blocked, $\pi(P)$ remains a lower bound on the total travel time needed for reaching t via this path P.[5] This is summarized in Algorithm 6.4.

In order to simplify notation, we denote by `currentTime` the total time traveled by ALG until the current point in time.

[5]Note that, although it is possible to improve the lower bound by adding the travel time for the remaining part of the path (from the endpoint of e to t), this does not help in the analysis.

Algorithm 6.4 BACKTRACK for k-CTP-URT

1: **for** $P \in \mathcal{P}$ **do**
2: $\pi(P) := c(P)$ *// initialize lower bounds*
3: **end for**
4: **while** t is not reached **do**
5: choose a path $P^* \in \mathcal{P}$ with $\pi(P^*) = \min_{P \in \mathcal{P}} \pi(P)$
6: **if** a blocked edge $e \in P^*$ is reached **then**
7: **for** $P \in \mathcal{P}$ with $e \in P$ **do**
8: **if** $\pi(P) <$ `currentTime` **then**
9: $\pi(P) :=$ `currentTime` *// update lower bounds*
10: **end if**
11: **end for**
12: return to s
13: **end if**
14: **end while**

Remark 6.18. *Note that the minimum in Step 5 of Algorithm 6.4 is, in general, not unique. The algorithm can therefore choose any path with currently minimal lower bound.*

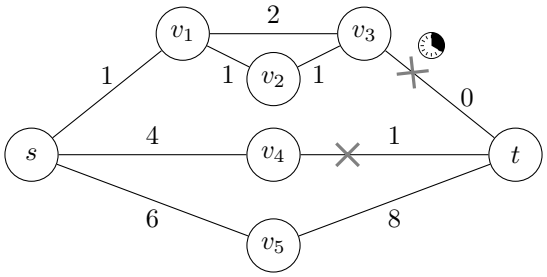


Figure 6.7: Illustration for Example 6.19: blocked edges are crossed out, and the recovery times are $b(v_1, t) = 10$, $b(v_4, t) = \infty$.

In order to get a better understanding of Algorithm 6.4, we give an example (we remark that this is no worst-case instance):

Example 6.19. *Consider the graph illustrated in Figure 6.7 consisting of four s-t-paths $P_1 = (s, v_1, v_3, t)$, $P_2 = (s, v_1, v_2, v_3, t)$, $P_3 = (s, v_4, t)$, and $P_4 = (s, v_5, t)$.*

The edge (v_1, t) is initially blocked and recovered after $b(v_1, t) = 10$ time units, whereas the edge (v_4, t) is blocked and never recovered, i.e., $b(v_4, t) = \infty$. The behavior of Algorithm 6.4 on this instance is shown in Table 6.1.

currentTime	position	event	$\pi(P_1)$	$\pi(P_2)$	$\pi(P_3)$	$\pi(P_4)$
0	s	choose P_1	3	3	5	14
3	v_3	(v_3, t) blocked, return to s	3	3	5	14
6	s	choose P_2	3	3	5	14
9	v_3	(v_3, t) blocked, update $\pi(P_1)$ and $\pi(P_2)$, return to s	9	9	5	14
12	s	choose P_3	9	9	4	5
16	v_4	(v_4, t) blocked, update $\pi(P_3)$, return to s	9	9	16	14
20	s	choose P_1	9	9	16	14
23	t	t reached	9	9	16	14

Table 6.1: The behavior of Algorithm 6.4 in Example 6.19.

In our algorithm, we initially set $\pi(P) := c(P)$, and we update it only when we reach a blockage at time $\texttt{currentTime} > \pi(P)$. Thus, we have:

Observation 6.20. *Throughout Algorithm 6.4 it holds for all $P \in \mathcal{P}$:*

(i) $c(P) \leq \pi(P)$,

(ii) $\pi(P)$ *is a lower bound on the time needed by* OPT *for reaching t via path P.*

Theorem 6.21. *Algorithm 6.4 is $(2k+3)$-competitive for k-*CTP-URT*.*

Proof. Let OPT choose some path $P^* \in \mathcal{P}$ that possibly contains (temporarily) blocked edges. We can assume that all blocked edges that are not on P^* have recovery time set to infinity. This does not change OPT and only increases the total travel time of ALG.

Throughout the algorithm, the lower bound $\pi(P)$ of a path P is possibly updated several times, and we define $\pi^{\max}(P)$ to be the largest value that is set by ALG for this path. By Observation 6.20, it thus holds

$$\text{OPT} \geq \pi^{\max}(P^*).$$

As long as t is not reached, ALG chooses an s-t-path in the graph and travels along this path until a blocked edge is reached and then returns to s. In order to analyze the algorithm, we divide the complete path traveled by ALG (that possibly traverses some edges several times) into the following phases:

Phase 1
: contains the subpath of ALG until the last update of $\pi(P^*)$ and the subsequent return to s.

Phase 2
: contains the subpath that consists of all subsequent unsuccessful attempts to traverse blocked paths and the last return to s.

Phase 3
: contains the final subpath from s to t where no blockage is encountered.

We distinguish two cases depending on whether Phase 1 is empty or not.

Case 1: Phase 1 is not empty.

First, we consider the case that Phase 1 is not empty. The last point in time when ALG encounters an edge on path P^* in a blocked state is therefore $\pi^{\max}(P^*) > c(P^*)$.

We denote this edge by $e' \in P^*$, and let P' be the chosen path for which ALG encounters this blockage $e' \in P^* \cap P'$ at time $\pi^{\max}(P^*)$. By Observation 6.20 and since P' is chosen as path with minimal π-value in this iteration, we have $c(P') \leq \pi(P') \leq \pi^{\max}(P^*)$ and, therefore, returning from e' to s takes at most $c(P') \leq \pi^{\max}(P^*)$ time units. Thus, Phase 1 ends after at most $2\pi^{\max}(P^*) \leq 2 \cdot \text{OPT}$ time units.

Let P'' be a path chosen in Phase 2 that encounters some blocked edge $e'' \in P''$. Therefore, we have $\pi(P'') \leq \pi^{\max}(P^*)$,[6] and the time needed to go from s to e'' and back to s takes at most $2c(P'') \leq 2\pi(P'') \leq 2\pi^{\max}(P^*)$. When the blocked edge e'' is reached, ALG updates the lower bounds such that after the update it holds $\pi(P) > \pi^{\max}(P^*)$ for all paths $P \in \mathcal{P}$ with $e'' \in P$.[7]

Since there are in total not more than $k-1$ blocked edges different from e', we know that $k-1$ attempts (on blocked paths) in Phase 2 suffice to ensure that the lower bounds for all paths that contain a blocked edge are raised to some value larger than $\pi^{\max}(P^*)$. Therefore, Phase 2 takes at most $2(k-1)\,\pi^{\max}(P^*) \leq 2(k-1) \cdot \text{OPT}$ time units.

In Phase 3, ALG then reaches t via a path P''' with minimal π-value, i.e., Phase 3 takes $c(P''') \leq \pi(P''') \leq \pi^{\max}(P^*) \leq \text{OPT}$ time units.

Altogether, ALG needs in Case 1 at most $(2k+1) \cdot \text{OPT}$ time units.

Case 2: Phase 1 is empty.

Next, we consider the case that Phase 1 is empty, i.e., we never encounter a blockage on P^* later than $c(P^*)$, and it holds throughout the algorithm that $\pi^{\max}(P^*) = c(P^*)$, i.e., $\text{OPT} \geq c(P^*)$. Recall that ALG thus chooses some path P only if the π-value satisfies $\pi(P) \leq c(P^*)$.

For the analysis of Phase 2, we consider the point in time T^* which has the following property: ALG is positioned in s, and all subsequent unsuccessful attempts of ALG on paths P (that contain a blocked edge)

[6] If this was not true, ALG would choose P^* before P''. By definition, we know that we do not encounter a blockage on P^* after $\pi^{\max}(P^*)$, and we would thus reach t before using P''. This contradicts the assumption that P'' is chosen in Phase 2.

[7] Note that we have at the end of Phase 1 `currentTime` $> \pi^{\max}(P^*)$.

lead to an update of the corresponding lower bound such that we have afterwards $\pi(P) > c(P^*)$.

We show that $T^* \leq 2 \cdot \text{OPT}$: Let P' be the last path that is chosen by ALG *before* returning to s at time T^*. Thus, *at* time T^*, it has to hold that $\pi(P') \leq c(P^*)$ (as, by definition of T^*, only the iterations *after* T^* raise the lower bounds of the blocked paths to a value larger than $c(P^*)$). Hence, a blockage on P' was reached after at most $\pi(P') \leq c(P^*) \leq \text{OPT}$ time units, and returning from this blockage to s takes again at most $c(P') \leq c(P^*) \leq \text{OPT}$, i.e., $T^* \leq 2 \cdot \text{OPT}$.

If a path P'' that contains a blocked edge $e'' \in P''$ is chosen by ALG *after* T^*, the time to go from s to e'' and back to s is at most $2c(P'') \leq 2\pi(P'') \leq 2c(P^*) \leq 2 \cdot \text{OPT}$ (note that, as ALG chooses P'', it holds at the beginning of this iteration that $\pi(P'') \leq c(P^*)$).

By definition, we have after the update that $\pi(P) > c(P^*)$ for all paths $P \in \mathcal{P}$ with $e'' \in P$. By the same reasoning as presented in Case 1, the total time until the π-values of all paths that contain a blocked edge are raised to a value larger than $\pi^{\max}(P^*)$, is at most $2k \cdot \text{OPT}$, and, thus, Phase 2 takes in total at most $(2k+2) \cdot \text{OPT}$ time units.

In Phase 3, ALG chooses a path P''' with minimal π-value which is not blocked. As in the first case, it thus holds that $\pi(P''') \leq \pi^{\max}(P^*) \leq c(P^*)$, i.e., Phase 3 takes at most $c(P''') \leq \text{OPT}$ time units.

Altogether, ALG needs at most $(2k+3) \cdot \text{OPT}$ time units in Case 2. □

Proposition 6.22. *The analysis of Algorithm 6.4 is tight.*

Proof. Consider for the case $k = 2$ a graph with three node-disjoint paths $P_1 = (s, v_1, t)$, $P_2 = (s, v_2, t)$, and $P_3 = (s, v_3, t)$, as shown in Figure 6.8, and assume that edges (v_1, t) and (v_2, t) are blocked and never reopened.

ALG attempts to traverse P_1, P_2, and again P_1. Finally, it reaches t using path P_3 after $7 + \epsilon$ time units, whereas OPT only needs $1 + \epsilon$, and we have

$$\frac{\text{ALG}}{\text{OPT}} = \frac{7+\epsilon}{1+\epsilon},$$

which goes to $7 = 2k + 3$ for $\epsilon \to 0$. □

Remark 6.23. *The competitive ratio of Algorithm 6.4 almost matches the lower bound of $2k + 1$. However, it remains an open problem whether there exist better algorithms.*

One of the "problems" of Algorithm 6.4 becomes apparent in Proposition 6.22: After choosing paths P_1 and P_2, it would be better to choose immediately P_3 (instead of attempting again P_1). Our algorithm does not make use of the information that P_3 cannot be blocked.

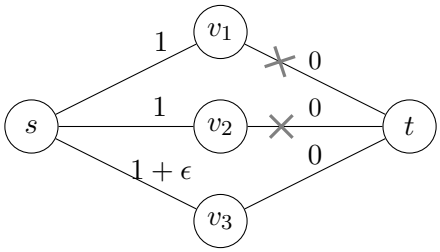


Figure 6.8: The analysis of Algorithm 6.4 is tight.

The algorithm could also be "improved" by implementing a waiting policy. For $k = 1$, a competitive ratio of $2k + 1 = 3$ is guaranteed as follows: Assume we choose initially a path P_1 with $c(P_1) = \min_{P \in \mathcal{P}} c(P)$. If we reach a blocked edge $e \in P_1$, we wait up to $w := c(P_2) - c(P_1)$ time units. Here, P_2 denotes the shortest path in $G - e$, i.e., $c(P_2) = \min_{P \in \mathcal{P}: e \notin P} c(P)$. If the blocked edge is recovered within this waiting time, we reach t after ALG $\leq c(P_1) + w =$ OPT *time units. Otherwise, we return to s, use P_2 (which cannot be blocked), and reach t after* ALG $\leq c(P_1) + w + c(P_1) + c(P_2) = 2 \cdot c(P_2) + c(P_1) \leq 3 \cdot$ OPT.

However, it is not clear how to generalize these ideas to arbitrary graphs and $k > 1$. We believe that it is necessary to resort to a more evolved analysis different from ours.

6.3.2 Extensions to the Canadian TSP

Another problem which is related to the CTP is the *Canadian travelling salesman problem* (CTSP) (cf. (Büttner, 2013)): An online algorithm has to find a minimum-cost tour visiting every vertex of an undirected graph at least once starting and ending in a designated vertex s under the same online setting as in the k-CTP (cf. Definition 6.1). As in the previous section, we consider a generalization of the problem with uncertain recovery times.

This problem, the *k-Canadian travelling salesman problem with uncertain recovery times* (k-CTSP-URT), is defined as follows:

Definition 6.24 (k-CTSP-URT)**.**
*An instance of k-*CTSP-URT *is given by an undirected graph $G = (V, E)$ with non-negative travel times $c : E \to \mathbb{R}_{\geq 0}$ for the edges, a designated vertex $s \in V$, a set $\mathcal{B} \subseteq E$ (with $|\mathcal{B}| \leq k$) of (initially) blocked edges, and recovery times $b : \mathcal{B} \to \mathbb{R}_{\geq 0}$ for the blocked edges (from which point in time on the edges are traversable).*

An online algorithm learns that $e \in \mathcal{B}$ only when reaching one of the endpoints of e. The recovery time of e remains unknown until the edge is reached at some point in time later than $b(e)$. The task is to find a closed tour starting

and ending in s visiting every vertex in G that minimizes the total travel time where an edge $e \in \mathcal{B}$ must not be used before time $b(e)$.

Büttner (2013) showed that a straightforward BACKTRACK strategy (where a shortest tour in the currently known network is chosen and traverse in an arbitrary direction) is $(2k+1)$-competitive for the k-CTSP.

Our extension of the BACKTRACK strategy for the k-CTP (Algorithm 6.2) can also be applied to k-CTSP-URT: We define for each tour T a value $\pi(T)$ that is a lower bound on the total travel time for using this tour. As long as we have not visited every vertex once, we choose a tour T^* with minimal π-value and traverse it in an arbitrary direction. If we reach a blocked edge $e^* \in T^*$, we update $\pi(T)$ for all paths T with $e^* \in T$ and return to s.

One can prove the competitiveness of this strategy completely analogously to Theorem 6.21, and we obtain the following:

Corollary 6.25. *There exists a $(2k+3)$-competitive online algorithm for k-CTSP-URT.*

The best-known lower bound for the competitive ratio of CTSP due to Büttner (2013) is $\sqrt{\frac{3}{2}} \approx 1.224$. This bound also holds for k-CTSP-URT (by setting the recovery time in her setting to infinity). We are able to improve upon this result:

Theorem 6.26. *No deterministic online algorithm for k-CTSP-URT can obtain a competitive ratio better than $1 + \frac{1}{\sqrt{3}} \approx 1.577$, even on cycle graphs and for $k = 1$.*

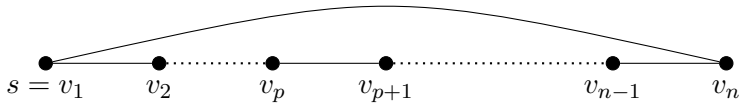
Proof. Consider for $n \geq 6$ a graph $G = (V, E)$ with nodes $V = \{v_1, \ldots, v_n\}$, edges $E = \{(v_{i-1}, v_i) : i = 2, \ldots, n\} \cup \{(v_1, v_n)\}$, and travel times $c(e) = 1$ for all $e \in E$ as illustrated in Figure 6.9(a).

We assume, without loss of generality, that a deterministic online algorithm ALG starting in $s = v_1$ initially chooses the tour $v_1, v_2, v_3, \ldots, v_n, v_1$.

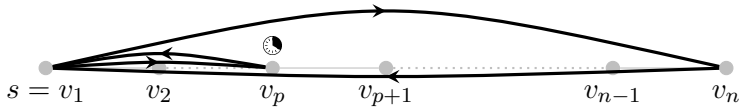
Now, let the p-th edge (v_p, v_{p+1}) on this path be blocked, where $p \in \mathbb{Z}_{\geq 0}$ is defined by $\left\lfloor \frac{n}{2\sqrt{3}} \right\rfloor \leq p \leq \left\lceil \frac{n}{2\sqrt{3}} \right\rceil$. Every deterministic online algorithm ALG is characterized by the time $w^{\max} \in \mathbb{R}_{\geq 0}$ it is willing to wait at v_p, before returning in the direction of s. We distinguish two cases:

Case 1: $w^{\max} \leq \frac{2(n-1)}{\sqrt{3}}$.

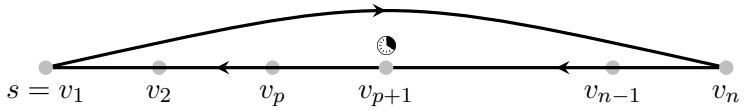
> In this case, the recovery time of the p-th edge (v_p, v_{p+1}) is set to $b(v_p, v_{p+1}) := p - 1 + w^{\max} + \epsilon$ for some small $\epsilon > 0$. Hence, after waiting $w^{\max}$ time units, ALG traverses the cycle in opposite direction, and the previously blocked edge will be recovered right after ALG left v_p.



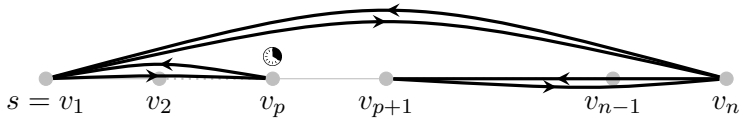
(a) Graph used in the proof of Theorem 6.26.



(b) In case 1, ALG waits at the blockage v_p and then traverses the graph in opposite direction,...



(c) ... whereas OPT traverses the graph in opposite direction and (maybe) waits at v_{p+1} until (v_p, v_{p+1}) is recovered.



(d) In case 2, (v_p, v_{p+1}) is blocked forever. ALG and OPT follow the same path, but ALG waits $w^{\max}$ time units at v_p.

Figure 6.9: Illustration of the proof of Theorem 6.26.

The final tour of ALG is illustrated in Figure 6.9(b), and we have

$$\begin{aligned}\text{ALG} &= \underbrace{(p-1)}_{\text{from } v_1 \text{ to } v_p} + \underbrace{w^{\max}}_{\text{waiting at } v_p} + \underbrace{(p-1)+n}_{\substack{\text{returning to } v_1 \text{ and} \\ \text{visiting remaining nodes}}} \\ &\geq n + 2\left(\left\lfloor \frac{n}{2\sqrt{3}} \right\rfloor - 1\right) + w^{\max}.\end{aligned}$$

OPT can traverse the cycle in opposite direction, i.e., visiting nodes in the order $v_1, v_n, v_{n-1}, \ldots$, and reach the other endpoint v_{p+1} of the blocked edge at time $n-p$. After waiting $\max\{b(v_p, v_{p+1}) - (n-p), 0\}$, the edge becomes traversable and OPT can go along the remainder of the cycle, which takes p time units (see Figure 6.9(c)). Thus, we have

$$\begin{aligned}\text{OPT} &\leq \underbrace{(n-p)}_{\text{from } v_1 \text{ to } v_{p+1}} + \underbrace{\max\{b(v_p, v_{p+1}) - (n-p), 0\}}_{\text{waiting at } v_{p+1}} + \underbrace{p}_{\text{from } v_{p+1} \text{ to } v_1} \\ &= n + \max\{2p - 1 + w^{\max} + \epsilon - n, 0\}. \qquad (6.14)\end{aligned}$$

We distinguish two further cases depending on which value attains the maximum in (6.14):

a) If $2p-1+w^{\max}+\epsilon-n \leq 0$, (6.14) yields the upper bound $\text{OPT} \leq n$, and we have

$$\frac{\text{ALG}}{\text{OPT}} \geq \frac{n+2\left(\left\lfloor \frac{n}{2\sqrt{3}} \right\rfloor - 1\right) + \overbrace{w^{\max}}^{\geq 0}}{n} \geq \frac{n+2\left(\frac{n}{2\sqrt{3}}-2\right)}{n},$$

which goes to $1+\frac{1}{\sqrt{3}}$ for $n \to \infty$.

b) If $2p-1+w^{\max}+\epsilon-n > 0$, (6.14) implies $\text{OPT} \leq 2p-1+w^{\max}+\epsilon$, which yields

$$\begin{aligned}
\frac{\text{ALG}}{\text{OPT}} &\geq \frac{n+2\left(\left\lfloor \frac{n}{2\sqrt{3}} \right\rfloor - 1\right) + w^{\max}}{2\left\lceil \frac{n}{2\sqrt{3}} \right\rceil - 1 + w^{\max} + \epsilon} \\
&\geq \frac{n+2\left(\frac{n}{2\sqrt{3}}-2\right) + w^{\max}}{2\left(\frac{n}{2\sqrt{3}}+1\right) - 1 + w^{\max} + \epsilon} \\
&\geq \frac{n+2\left(\frac{n}{2\sqrt{3}}-2\right) + \frac{2(n-1)}{\sqrt{3}}}{2\left(\frac{n}{2\sqrt{3}}+1\right) - 1 + \frac{2(n-1)}{\sqrt{3}} + \epsilon} \\
&\geq \frac{\left(1+\sqrt{3}\right)n - \left(4+\frac{2}{\sqrt{3}}\right)}{\sqrt{3}n + 1 - \frac{2}{\sqrt{3}} + \epsilon},
\end{aligned} \tag{6.15}$$

which goes to $1+\frac{1}{\sqrt{3}}$ for $n \to \infty$.

Here, the second-to-last inequality holds by the following argument: If we define $\alpha(n) := n+2\left(\frac{n}{2\sqrt{3}}-2\right)$ and $\beta(n) := 2\left(\frac{n}{2\sqrt{3}}+1\right) - 1 + \epsilon$, it holds for $n \geq 6$ that $\alpha(n), \beta(n) \geq 0$ and $\alpha(n) \geq \beta(n)$. Therefore, (6.15) $= \frac{\alpha(n)+w^{\max}}{\beta(n)+w^{\max}}$ is, for fixed $n \geq 6$, a decreasing function in $w^{\max}$ on the interval $[0, \frac{2(n-1)}{\sqrt{3}}]$ and attains its minimum for $w^{\max} = \frac{2(n-1)}{\sqrt{3}}$.

Case 2: $w^{\max} > \frac{2(n-1)}{\sqrt{3}}$

In this case, the blocked edge (v_p, v_{p+1}) will never be reopened, i.e., $b(v_p, v_{p+1}) := \infty$. Thus, ALG has to traverse all edges except (v_p, v_{p+1}) twice, and we have

$$\text{ALG} \geq 2(n-1) + w^{\max} > 2(n-1) + \frac{2(n-1)}{\sqrt{3}},$$

whereas OPT does not wait at all, i.e.,

$$\text{OPT} = 2(n-1).$$

This is illustrated in Figure 6.9(d), and, altogether, we have in this case

$$\frac{\text{ALG}}{\text{OPT}} \geq \frac{2(n-1) + \frac{2(n-1)}{\sqrt{3}}}{2(n-1)} = 1 + \frac{1}{\sqrt{3}}.$$

□

6.4 Conclusions

In the first part of this chapter we analyzed the problem k-CTP. We presented the first randomized online algorithm for this problem. Our algorithm applies to graphs where all s-t-paths are node-disjoint and its competitive ratio matches the lower bound of $k+1$ that was presented in (Westphal, 2008), i.e., our algorithm is best possible from a competitive analysis point of view. We discussed possible extensions to more general graph classes and limitations where this is not possible.

Recently, Demaine et al. (2014) presented the first randomized online algorithm that beats the deterministic ratio for general graphs and achieves a competitive ratio of $(1 + \frac{1}{\sqrt{2}})k + 1 < 2k + 1$. It remains an open problem whether there exist better randomized algorithms for general graphs.

In the second part of this chapter we analyzed the problem k-CTP-URT, which is a generalization of k-CTP, where blocked edges can become traversable at some later point in time that is initially not known. We presented a deterministic $(2k + 3)$-competitive online algorithm and showed how this idea can be carried over to the problem k-CTSP-URT. A natural direction of future research would be to study other routing problems under this new online setting.

Bibliography

R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows*. Prentice Hall, 1993.

S. Al-Binali. A risk-reward framework for the competitive analysis of financial games. *Algorithmica*, 25(1):99–115, 1999.

S. Albers. Energy-efficient algorithms. In *Communications of the ACM*, pages 86–96, 2010.

N. Alon and J. H. Spencer. *The Probabilistic Method*. John Wiley & Sons, 1992.

A. Antoniadis, C.-C. Huang, S. Ott, and J. Verschae. How to pack your items when you have to buy your knapsack. In *Proceedings of the 38th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 62–73, 2013.

E. M. Arkin and E. B. Silverberg. Scheduling jobs with fixed start and end times. *Discrete Applied Mathematics*, 18(1):1–8, 1987.

S. F. Assmann. *Problems in Discrete Applied Mathematics*. PhD thesis, Massachusetts Institute of Technology, 1983.

S. F. Assmann, D. S. Johnson, D. J. Kleinman, and J. Y.-T. Leung. On a dual version of the one-dimensional bin packing problem. *Journal of Algorithms*, 5(4):502–525, 1984.

E. Bampis, A. Kononov, D. Letsios, G. Lucarelli, and M.Sviridenko. Energy efficient scheduling and routing via randomized rounding. In *Proceedings of the 33rd International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 449–460, 2013.

A. Bar-Noy and B. Schieber. The canadian traveller problem. In *Proceedings of the 2nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 261–270, 1991.

S. Barman, S. Umboh, S. Chawla, and D. Malec. Secretary problems with convex costs. In *Proceedings of the 39th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 75–87, 2012.

S. Ben-David and A. Borodin. A new measure for the study of on-line algorithms. *Algorithmica*, 11(1):73–91, 1994.

M. Bender and S. Westphal. An optimal randomized online algorithm for the k-canadian traveller problem on node-disjoint paths. *Journal of Combinatorial Optimization*, pages 1–10, 2013.

M. Bender and S. Westphal. Maximum generalized assignment with convex costs. In *Proceedings of the Third International Symposium on Combinatorial Optimization (ISCO)*, pages 75–86, 2014.

M. Bender, C. Thielen, and S. Westphal. A constant factor approximation for the generalized assignment problem with minimum quantities and unit size items. In *Proceedings of the 38th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 135–145, 2013a.

M. Bender, C. Thielen, and S. Westphal. Erratum to "A constant factor approximation for the generalized assignment problem with minimum quantities and unit size items" (Bender et al., 2013a). In *Proceedings of the 38th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages E1–E3, 2013b.

M. Bender, C. Thielen, and S. Westphal. The online interval scheduling problem with bounded number of failures. submitted, 2015a.

M. Bender, C. Thielen, and S. Westphal. Packing items into several bins facilitates approximating the separable assignment problem. *Information Processing Letters*, 115(6–8):570–575, 2015b.

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

K. I. Bouzina and H. Emmons. Interval scheduling on identical machines. *Journal of Global Optimization*, 9(3–4):379–393, 1996.

S. Büttner. *Online Disruption and Delay Management*. PhD thesis, University of Kaiserslautern, 2013.

G. Calinescu, C. Chekuri, M. Pal, and J. Vondrák. Maximizing a submodular set function subject to a matroid constraint. *SIAM Journal on Computing*, 40(6):1740–1766, 2011.

R. Canetti and S. Irani. Bounding the power of preemption in randomized scheduling. *SIAM Journal on Computing*, 27(4):993–1015, 1998.

M. C. Carlisle and E. L. Lloyd. On the k-coloring of intervals. *Discrete Applied Mathematics*, 59(3):225–235, 1995.

D. G. Cattrysse and L. N. Van Wassenhove. A survey of algorithms for the generalized assignment problem. *European Journal of Operational Research*, 60(3):260–272, 1992.

C. Chekuri and S. Khanna. A PTAS for the multiple knapsack problem. *SIAM Journal on Computing*, 35(3):713–728, 2006.

V. Chvatal. *Linear Programming*. W. H. Freeman, 1983.

R. Cohen, L. Katzir, and D. Raz. An efficient approximation algorithm for the generalized assignment problem. *Information Processing Letters*, 100 (4):162–166, 2006.

P. Crescenzi. A short guide to approximation preserving reductions. In *Proceedings of the 12th Annual IEEE Conference on Computational Complexity*, pages 262–273, 1997.

J. Csirik, D. S. Johnson, and C. Kenyon. Better approximation algorithms for bin covering. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 557–566, 2001.

E. D. Demaine, Y. Huang, C. S. Liao, and K. Sadakane. Canadians should travel randomly. In *Proceedings of the 41st International Colloquium on Automata, Languages and Programming (ICALP)*, volume 8572 of *LNCS*, pages 380–391, 2014.

L. Epstein and A. Levin. Improved randomized results for the interval selection problem. *Theoretical Computer Science*, 411(34–36):3129–3135, 2010.

L. Epstein, Ł. Jeż, J. Sgall, and R. van Stee. Online scheduling of jobs with fixed start times on related machines. In *Proceedings of the 16th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 134–145, 2013.

U. Faigle and W. M. Nawjin. Note on scheduling intervals online. *Discrete Applied Mathematics*, 58(1):13–17, 1995.

U. Feige and J. Vondrák. Approximation algorithms for allocation problems: Improving the factor of $1 - 1/e$. In *Proceedings of the 47th Annual IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 667–676, 2006.

A. Fiat and G. J. Woeginger, editors. *Online Algorithms: The State of the Art*. Springer, 1998.

L. Fleischer, M. X. Goemans, V. S. Mirrokni, and M. Sviridenko. Tight approximation algorithms for maximum general assignment problems. *Mathematics of Operations Research*, 36(3):416–431, 2011.

S. P. Y. Fung, C. K. Poon, and F. Zheng. Online interval scheduling: Randomized and multiprocessor cases. *Journal of Combinatorial Optimization*, 16(3):248–262, 2008.

S. P. Y. Fung, C. K. Poon, and F. Zheng. Improved randomized online scheduling of unit length intervals and jobs. In *Proceeding of the 6th Workshop on Approximation and Online Algorithms (WAOA)*, pages 53–66, 2009.

S. P. Y. Fung, C. K. Poon, and D. K. W. Yung. On-line scheduling of equal-length intervals on parallel machines. *Information Processing Letters*, 112 (10):376–379, 2012.

M. R. Garey and D. S. Johnson. *Computers and Intractability (A Guide to the Theory of NP-Completeness)*. W.H. Freeman and Company, New York, 1979.

M. X. Goemans and D. P. Williamson. New 3/4-approximation algorithms for the maximum satisfiability problem. *SIAM Journal of Discrete Mathematics*, 7:656–666, 1994.

M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1988.

U. I. Gupta, D. T. Lee, and J. Y.-T. Leung. An optimal solution for the channel-assignment problem. *IEEE Transactions on Computers*, 28(11): 807–810, 1979.

W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(3):13–30, 1963.

A. J. Hoffman and J. B. Kruskal. Integral boundary points of convex polyhedra. In Kuhn and Tucker (1956), pages 223–246.

Y. Huang and C. S. LiaoShou. The canadian traveller problem revisited. In *Proceedings of the 23rd International Symposium on Algorithms and Computation (ISAAC)*, pages 352–361, 2012.

K. Jansen and R. Solis-Oba. An asymptotic fully polynomial time approximation scheme for bin covering. *Theoretical Computer Science*, 306:543–551, 2003.

H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

A. W. J. Kolen, J. K. Lenstra, C. Papadimitriou, and F. C. R. Spieksma. Interval scheduling: A survey. *Naval Research Logistics*, 54:530–543, 2007.

B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms.* Springer, 2007.

M. Y. Kovalyov, C. T. Ng, and T. C. E. Cheng. Fixed interval scheduling: Models, applications, computational complexity and algorithms. *European Journal of Operational Research*, 178(2):331–342, 2007.

S. O. Krumke and H. Noltemeier. *Graphentheoretische Konzepte und Algorithmen.* Teubner, 2005.

S. O. Krumke and C. Thielen. Minimum cost flows with minimum quantities. *Information Processing Letters*, 111(11):533–537, 2011.

S. O. Krumke and C. Thielen. Erratum to "Minimum cost flows with minimum quantities" (Krumke and Thielen, 2011). *Information Processing Letters*, 112(13):523–524, 2012.

S. O. Krumke and C. Thielen. The generalized assignment problem with minimum quantities. *European Journal of Operational Research*, 228(1): 46–55, 2013.

S. O. Krumke, C. Thielen, and S. Westphal. Interval scheduling on related machines. *Computers and Operations Research*, 38(12):1836–1844, 2011.

H. W. Kuhn and A. J. Tucker, editors. *Linear Inequalities and Related Systems.* Princeton University Press, 1956.

C. S. Liao and Y. Huang. The covering canadian traveller problem. *Theoretical Computer Science*, 530:80–88, 2014.

R. J. Lipton and A. Tomkins. Online interval scheduling. In *Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 302–311, 1994.

S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations.* John Wiley & Sons, Chichester, 1990.

K. Mehlhorn and M. Ziegelmann. Resource constained shortest paths. In *Proceedings of the 8th Annual European Symposium on Algorithms (ESA)*, pages 326–337, 2000.

M. Minoux. A class of combinatorial problems with polynomially solvable large scale set covering/partitioning relaxations. *RAIRO Recherche Opérationelle*, 21(2):105–136, 1987.

M. Mitzenmacher and E. Upfal. *Probability and Computing – Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, 2005.

H. Miyazawa and T. Erlebach. An improved randomized online algorithm for a weighted interval selection problem. *Journal of Scheduling*, 7(4):293–311, 2004.

G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* Wiley-Interscience, New York, 1988.

Z. Nutov, I. Beniaminy, and R. Yuster. A $(1 - 1/e)$-approximation algorithm for the maximum generalized assignment problem with fixed profits. *Operations Research Letters*, 34:283–288, 2006.

C. Papadimitriou. *Computational Complexity.* Addison Wesley, 1993.

C. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84(1):127–150, 1991.

D. W. Pentico. Assignment problems: A golden anniversary survey. *European Journal of Operational Research*, 176(2):774–793, 2007.

K. Pruhs and C. Stein. How to schedule when you have to buy your energy. In *Proceedings of the 13th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 352–365, 2010.

A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, Chichester, 1998.

H. G. Seedig. Network flow optimization with minimum quantities. In *Operations Research Proceedings 2010: Selected Papers of the Annual International Conference of the German Operations Research Society*, pages 295–300. Springer, 2011.

S. S. Seiden. Randomized online interval scheduling. *Operations Research Letters*, 22(4-5):171–177, 1998.

D. B. Shmoys and É. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62:461–474, 1993.

D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

B. Su and Y. Xu. Online recoverable canadian traveller problem. In *Proceedings of the International Conference on Management Science and Engineering*, pages 633–639, 2004.

B. Su, Y. Xu, P. Xiao, and L. Tian. A risk-reward competitive analysis for the recoverable canadian traveller problem. In *Proceedings of the 2nd international Conference on Combinatorial Optimization and Applications*, pages 417–426, 2008.

C. Thielen and S. Westphal. Complexity and approximability of the maximum flow problem with minimum quantities. *Networks*, 2013.

V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.

S. Westphal. A note on the k-canadian traveller problem. *Information Processing Letters*, 106(3):87–89, 2008.

D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

G. J. Woeginger. Online scheduling of jobs with fixed start and end times. *Theoretical Computer Science*, 130(1):5–16, 1994.

Y. Xu, M. Hu, B. Su, B. Zhu, and Z. Zhu. The canadian traveller problem and its competitive analysis. *Journal of Combinatorial Optimization*, 18(2): 195–205, 2009.

A. C. C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proceedings of the 18th Annual IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 222–227, 1977.

H. Zhang and Y. Xu. The k-canadian travelers problem with communication. In *Proceedings of the Frontiers in Algorithmics and Algorithmic Aspects in Information and Management - Joint International Conference, FAW-AAIM*, pages 17–28, 2011.

X. Zhu, Q. Yuan, A. Garcia-Diaz, and L. Dong. Minimal-cost network flow problems with variable lower bounds on arc flows. *Computers and Operations Research*, 38(8):1210–1218, 2011.

Curriculum Vitae

Persönliche Daten

Name	Marco Bender
Geburtsdatum	20. Mai 1988
Geburtort	Simmern/Hunsrück
Staatsangehörigkeit	deutsch
Wohnsitz	Göttingen

Akademischer Werdegang

01/2012 – 04/2015	Promotionsstudium, Fakultät für Mathematik und Informatik, Georg-August-Universität Göttingen
04/2007 – 11/2011	Studium der Mathematik (Abschluss: Diplom), Technische Universität Kaiserslautern
07/2010 – 11/2010	Auslandssemester, University of Auckland, Neuseeland
03/2006	Abitur, Herzog-Johann-Gymnasium, Simmern

Akademische Tätigkeiten

seit 04/2014	Wissenschaftlicher Mitarbeiter, Institut für Angewandte Stochastik und Operations Research, Technische Universität Clausthal
01/2012 – 03/2014	Wissenschaftlicher Mitarbeiter und Stipendiat, Institut für Numerische und Angewandte Mathematik, Georg-August-Universität Göttingen
10/2008 – 11/2011	Wissenschaftliche Hilfskraft, Fachbereich Mathematik, Technische Universität Kaiserslautern