Aspects of Online Routing and Scheduling

Vom Fachbereich Mathematik
der Technischen Universität Kaiserslautern
zur Verleihung des akademischen Grades
Doktor der Naturwissenschaften
(Doctor rerum naturalium, Dr. rer. nat.)
genehmigte Dissertation

vorgelegt von

Stephan Westphal

aus Oldenburg (Oldb.)
31. August 2007

Gutachter: Prof. Dr. Sven O. Krumke
Prof. Dr. Tjark Vredeveld
Datum der Disputation: 17. Dezember 2007

D 386

to Katrin, Luk, and Liv

Contents

Chapter 1

Introduction

Online Routing and Scheduling

The success of each enterprise depends fundamentally on how efficiently the own operating facilities are used. That applies to a factory, in which cars are produced, exactly the same way as to a hotel chain or a shipping company. The production processes in the factory must be arranged in such a way that they are as economical as possible. Additionally, a multitude of dependencies between the single processes have to be taken into account. Also deadlines and limited capacities play a role. The same applies to the shipping company. Here, the planner must employ his truck fleet in a way, that the profit earned is as high as possible. Apart from the deadlines that have to be met and the distances which have to be driven there are a lot of further important restrictions. Even the owner of some cottages wants to operate his "facilities" at full capacity, and must provide a strategy which helps him to achieve this goal.

Plenty of problems, which focus on the optimal assignment of activities to scarce resources over time, can be tackled with techniques from Scheduling theory. This theory deals with the optimal assignment of jobs to machines. In doing so, there is a multitude of restrictions which can be taken into account. Jobs might have to wait for the completion of others, they might be allowed to be interrupted and continued later on. A job's processing time can depend on the machine it is processed on, and jobs can have deadlines, which mark the final time when they are allowed to be finished. Furthermore, "optimality" can be defined in different ways. One could possibly ask to minimize the completion time of the last job (the *makespan*) or the maximum amount of jobs to be finished on time. There is a huge variety of possibilities how a scheduling problem might look like, and researchers examined many of them. They found

efficient algorithms for some of them and showed that others are unlikely to be solvable in reasonable time.

In classical scheduling the planner knows the complete problem. He knows all jobs and all machines, with all their properties and requirements. In practice, however, decisions often have to be made without knowing all these circumstances. For example, consider the owner of some cottages, who is asked to rent one of the cottages for a specific time to a customer. As he does not know, what offers are coming up in the future, he has to decide with incomplete knowledge about this request.

Problems of this kind, where decisions have to be made without complete knowledge of the input data, are called *online problems*. In this thesis, we examine some online scheduling and vehicle routing problems. We develop algorithms for them, estimate their quality and make statements how good algorithms of that kind can be.

Outline and Main Results

This section is intended to give an overview of this thesis and to provide an outline of how the material is organized. In Chapter 2, we start with some basic terminology used throughout this thesis and provide a formal introduction to Online Optimization.

We start our analysis in Chapter 3, in which we consider the problem of scheduling a maximum profit selection of jobs on m identical machines. Jobs arrive online one by one and each job is specified by its start and end time. The goal is to determine a non-preemptive schedule which maximizes the profit of the scheduled jobs, where the profit of a job is equal to its length. Upon arrival of a new job, an online algorithm must decide whether to accept the job or not. If the job is accepted, the online algorithm must be able to reorganize its already existing schedule such that the new job can be processed together with all previously admitted jobs, however, the algorithm need not specify on which machine the job will eventually be run. We provide competitive algorithms and lower bounds on the competitive ratio for deterministic and randomized algorithms against an oblivious adversary. Our lower bound results essentially match (up to small constants factors) the competitive ratios achieved by our algorithms. As the greedy algorithm showed to have the best empirical performance, we analyzed its worst case behavior and calculated lower bounds on its average case competitivity.

In Chapter 4 we are given m identical machines which are available for a planning horizon from $[0, T]$ and a set of n orders $J = \{j_1, \ldots, j_n\}$. Each order j_i is characterized by its release time $r_i \in [0, T]$, its processing time $b_i \geq 0$, the time slack $\delta_i \geq 0$, and the profit $p_i \geq 0$ which is obtained, if the order is accepted and produced. The time slack δ_i indicates the latest time $r_i + \delta_i$ at which j_i can be started such that it is done until its deadline $r_i + b_i + \delta_i$ when the product must be manufactured.

In Chapter 5 one is given a finite ground set F of different types of items. The items can be ordered in boxes each of which have capacity m at cost m. Then, requests arrive. A request r is can be satisfied by up to k different multisets of (not necessarily different) items. The goal is, to choose these multisets in a way, that the total number of boxes ordered is minimum. We discuss approaches for the offline problem, and compare their performance on randomly generated data. We give a lower bound for deterministic online algorithms and propose an easy algorithm matching this bound.

In Chapter 6 we consider the problem of scheduling jobs on m related machines with precedence constraints such that the makespan is minimized. We focus on the case, where each machine is represented by an agent, who tells us the (not necessarily correct) speed of his machine. Based on these data, we have to generate a schedule and have to pay the agent according to some payment scheme. The goal is to provide a payment scheme, such that it is the best choice for each agent to tell the truth, and additionally that the schedule obtained this way is best possible. For this special version of makespan optimization, we propose a $\mathcal{O}(m^{2/3})$-approximation.

In Chapter 7 we investigate a real-world large-scale vehicle dispatching problem with strict real-time requirements, posed by our cooperation partner, the German Automobile Association (ADAC). Service units starting at their current positions are to serve at most k requests without returning to their home positions, where k is a given number. As Krumke et al. showed that this problem is $\mathcal{NP}$-complete, even for $k = 2$ (see [KSVWar]), we present the polynomial time $(2k-1)$-approximation algorithm MATCHDISPATCH, where again k denotes the maximal number of requests served by a single service unit. For the special case, where k equals the total number of requests, we provide a $(2-1/k)$-approximation which works similar to the Double-Tree-Algorithm for the metric TSP. In order to get a notion of the average case performance of MATCHDISPATCH, we compare its solutions with the exact optimal solutions, which we obtained by deploying an integer programming formulation. This way, we see that the solutions found by MATCHDISPATCH are circa 17% more expensive than the optimal solutions. When we apply 2-Opt to the solutions obtained by MATCHDISPATCH we find even better solutions which cost about 5%

more than the optimum. Generally speaking, for those instances for which it is computationally too demanding to compute the optimal solution, MATCHDISPATCH with 2-Opt provides an attractive alternative to get a hand on almost optimal solutions in very short time.

In Chapter 8 we consider the online problem k-CTP, which is the problem to guide a vehicle from some site s to some site t on a road map given by a graph $G = (V, E)$ in which up to k (unknown) edges are blocked by avalanches. An online algorithm learns from a blocked edge when reaching one of its endpoints. Thus, it might have to change its route to the target t up to k times. We show that no deterministic online algorithm can achieve a competitive ratio smaller than $2k+1$ and give an easy algorithm which matches this lower bound. Furthermore, we show that randomization can not improve the competitive ratio substantially, by establishing a lower bound of $k + 1$ for the competitivity of randomized online algorithms against an oblivious adversary.

In Chapter 9 we consider k-server routing problems (also known as "dial-a-ride-problems"), where k servers move in a metric space and must visit specified points or carry objects from sources to destinations. In the online version requests arrive online while the servers are traveling. Two classical objective functions are to minimize the makespan, i.e., the time when the last server has completed its tour, (k-Traveling Salesman Problem, k-TSP) and to minimize the sum of completion times (k-Traveling Repairman Problem, k-TRP). We reduce a number of gaps between the obtained competitive ratios and the corresponding lower bounds by providing new lower bounds for randomized algorithms. The most dramatic improvement is in the lower bound of $\frac{4e-5}{2e-3} \approx 2.4104$ for k-DARP (the k-TRP when objects need to be carried) to 3 which is currently also the best lower bound for deterministic algorithms.

Acknowledgments

I would like to express my gratitude to everyone who contributed to this thesis. First of all, I am most indebted to my supervisor Professor Sven O. Krumke for all the support, numerous inspiring discussions, and, last but not least, the pleasant atmosphere when working together. Without his never-ending ideas and expertise, this project would never have been accomplished. I also owe my sincere gratitude to my wife Katrin and my son Luk, who supported me the whole time in their very special ways. Finally, I wish to thank all my colleagues of the AG Optimization for all the nice time spent during lunch breaks and lots of coffee.

Credits

The results reported in Chapter 3 were obtained jointly with Sven O. Krumke and Rob van Stee.

The results reported in Chapter 4 were obtained jointly with Sven O. Krumke and Alfred Taudes.

The results reported in Chapter 5 were obtained jointly with Sven O. Krumke and Sandra Gutierrez.

The results reported in Chapter 6 were obtained jointly with Rob van Stee and Sven O. Krumke and Anne Schwahn.

The results reported in Chapter 9 were obtained jointly with Sven O. Krumke and Irene Fink.

I am extremely grateful to all the colleagues mentioned above for their valuable time and for letting me include our joint results in this thesis.

Chapter 2

Preliminaries

In this chapter, we outline based on [KN05] and [KV00] some basic definitions used throughout this thesis. We describe some graph theory, network flows, and some basic definitions considering online optimization. We assume some basic knowledge of complexity theory and linear optimization. For these fundamentals the reader is referred to [GJ79], [Chv83], and especially regarding integer programming to [Sch86] and [NW88].

2.1 Basic notation

We will denote by $\mathbb{R}$ ($\mathbb{Q}$, $\mathbb{Z}$) the real (rational, integer) numbers. The sets $\mathbb{R}_+$ ($\mathbb{Q}_+$, $\mathbb{Z}_+$) stand for the non-negative real (rational, integer) numbers. We denote the set of positive integer numbers without zero by $\mathbb{N} = \mathbb{Z}_+\backslash\{0\}$. For some $n \in \mathbb{N}$, we define by $\mathbb{K}^n$ the set of vectors with n components from $\mathbb{K}$. The transposition of a vector x is x^T. With $\log_a$ we denote the *logarithm* to the basis of a. For $a = 2$ we omit the basis, and by ln, we denote the *natural logarithm*.

2.2 Graph Theory

Formally, a *directed graph* is a quadruple $G = (V, A, \alpha, \omega)$ consisting of a nonempty set V, called the *nodes* (or vertices), a set A, called the *arcs*, and two relations of *incidence* $\alpha : A \to V$ and $\omega : A \to V$ that associate with each arc $a \in A$ its *tail* $\alpha(a)$ and its *head* $\omega(a)$. Usually we just write $G = (V, A)$ and assume that the incidence relation is given implicitly in A. For a given graph G,

define $V(G)$ and $A(G)$ the corresponding sets of vertices resp. arcs. In order to abbreviate the notation we define $n := |V(G)|$ and $m := |A(G)|$. In this thesis, we assume all graphs to have finite sets of vertices and arcs.

An arc $a \in A$ is called a *loop*, if $\alpha(a) = \omega(a)$. Two arcs $a, a' \in A$ are called *parallel*, if $\alpha(a) = \alpha(a')$ and $\omega(a) = \omega(a')$. A graph G is called *simple*, if it does not contain any loops or parallel arcs. For $v \in V$ the set of *outgoing arcs* is denoted as $\delta^+(v) := \{a \in A : \alpha(a) = v\}$, whereas $\delta^-(v) := \{a \in A : \omega(a) = v\}$ describes the set of *ingoing arcs* of v. The *indegree* and the *outdegree* of a node v is defined as $g^-(v) := |\delta^-(v)|$ resp. $g^+(v) := |\delta^+(v)|$. The nodes $u, v \in V$ are called *adjacent* (*neighbors*), if there is some $a \in A$ such that $u = \alpha(a)$ and $v = \omega(a)$.

A *path* in G is a finite sequence $P = (v_0, a_1, v_1, ..., a_k, v_k)$ of nodes $v_0, ..., v_k \in V$ and arcs $a_1, ..., a_k \in A$ such that $\alpha(a_i) = v_{i-1}$ and $\omega(a_i) = v_i$ for $i = 1, \ldots, k$. The *length* of P corresponds to the number k of arcs in P. If $v_0 = v_k$ and $k \geq 1$ we call P a *cycle*. The nodes $\alpha(P) := v_0$ and $\omega(P) := v_k$ are named the *tail* resp. the *head* of P.

With $tr(P) := (v_0, v_1, \ldots, v_k)$ we denote the *trace* of the path P. A path is called *simple* if $a_i \neq a_j$ for $i \neq j$. If it is simple and none of the nodes appears more than once - apart from the case that the first and the last vertex coincide-, it is called *elementary*.

A graph $\tilde{G} = (\tilde{V}, \tilde{A})$ is a *subgraph* of $G = (V, A)$ if $\tilde{V} \subseteq V$ and $\tilde{A} \subseteq A$. A graph $G = (V, A)$ is said to be *connected* if there is a path between any two nodes. A *tree* is a connected subgraph with no cycles. A *spanning tree* of G is a subgraph of G which has the same set of nodes as G and is a tree.

A *cut* (X, Y) in a graph $G = (V, A)$ is a partition of the vertices in nonempty subsets such that $V = X \cup Y$, $X \cap Y = \emptyset$, $X \neq \emptyset$, and $Y \neq \emptyset$.

An *undirected graph* is a triple $G = (V, E, \gamma)$ consisting of a nonempty set of nodes V, a set of edges E and a relation of *incidence* $\gamma : E \to V \times V$ that associates with each edge one or two nodes, called its *ends*. Usually we just write $G = (V, E)$ and assume that the incidence relation is given implicitly in E. The definitions stated above for directed graphs carry over to undirected graphs.

A *clique* is a subgraph of an undirected graph $G = (V, E)$, in which all pairs of vertices are connected by an edge. A graph G whose vertex set forms a clique is said to be *complete*. A clique of maximum cardinality in G is called a *maximum clique*. A set of nodes in a graph G is said to be an *independent set* if there is no edge connecting any of its nodes. An independent set of maximum cardinality is denoted as *maximum independent set*.

Network Flows

We have a digraph $G = (V, A)$ with edge capacities $u : E(G) \to \mathbb{R}_+$, two specified vertices s (*the source*) and t (the *sink*). The quadruple (G, u, s, t) is called a *network*. A *flow* is a function $f : E(G) \to \mathbb{R}_+$ with $f(e) \leq u(e)$ for all $e \in E(G)$. We say that f satisfies the *flow conservation rule* at vertex v if

$$\sum_{e \in \delta^-(v)} f(e) = \sum_{e \in \delta^+(v)} f(e)$$

For a given network (G, u, s, t), an *s-t-flow* is a flow satisfying the flow conservation rule at all vertices except s and t. We define the *value* of an $s-t$-flow f by

$$\text{value}(f) := \sum_{e \in \delta^+(s)} f(e) - \sum_{e \in \delta^-(s)} f(e)$$

Given a digraph $G = (V, A)$ with edge capacities $u : E(G) \to \mathbb{R}_+$, and values $b : V(G) \to \mathbb{R}$ with $\sum_{v \in V(G)} b(v) = 0$, a b-*flow* in (G, u) is a function $f : E(G) \to \mathbb{R}_+$ with $f(e) \leq u(e)$ for all $e \in E(G)$ and $\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) = b(v)$ for all $v \in V(G)$. $b(v)$ is called the balance of vertex v. $|b(v)|$ is called the *supply* if $b(v) > 0$ resp. the *demand* of v otherwise. Vertices v with $b(v) > 0$ are called sources, those with $b(v) < 0$ sinks.

The MINIMUM COST FLOW PROBLEM is defined as the problem of finding for a given digraph G with edge capacities $u : E(G) \to \mathbb{R}_+$, and numbers $b : V(G) \to \mathbb{R}$ a b-flow with minimum weight respective to a weight function $c : E(G) \to \mathbb{R}$ such that $c(f) := \sum_{e \in E(G)} f(e)c(e)$ is minimum or to decide that no b-flow exists.

2.3 Optimization Problems

In this section, we settle some basic definitions and notation about optimization problems and their approximability.

Definition 2.1 (Optimization Problem) An *instance* of an optimization problem is given by a set of feasible solutions X and a *cost* function $c : X \to \mathbb{R}$. A solution $x^* \in X$ is an optimal solution to the corresponding *minimization problem* if and only if $c(x^*) \leq c(x)$ for all $x \in X$. Analogously, it solves the

corresponding *maximization problem* optimally for the case that $c(x^*) \geq c(x)$ for all $x \in X$.

Let I be an instance of a given optimization problem. The minimum possible length of a reasonable binary representation of I is called its *encoding length*. An algorithm which finds an optimal solution for each instance I in time polynomial in its encoding length is termed a *polynomial time algorithm*. It is also said to be *efficient*.

For most of the problems we deal with in this thesis, there exist no efficient algorithms unless $\mathcal{P} = \mathcal{NP}$. In this case, we look for polynomial time approximations of the optimum:

Definition 2.2 An polynomial time algorithm ALG for a minimization problem is called a c-*approximation*, if it obtains a solution with objective value ALG (I) for every instance I of the problem, such that

$$\text{ALG}\,(I) \leq c \cdot \text{OPT}\,(I),$$

with OPT (I) being the objective value of an optimal solution of I. Analogously, a c-approximation for a maximization problem provides a solution with

$$c \cdot \text{ALG}\,(I) \geq \text{OPT}\,(I).$$

2.4 Online Optimization

In general, traditional optimization techniques assume complete knowledge of all data of a problem instance. However, in reality it is unlikely that all information necessary to define a problem instance is available beforehand. Decisions may have to be made before complete information is available. This observation has motivated the research on *online optimization*. An algorithm is called *online* if it makes a decision (computes a partial solution) whenever a new piece of data requests an action.

In online optimization the input is modeled as a finite request sequence $\sigma = (r_1, r_2, \dots)$ which must be served and which is revealed step by step to an online algorithm. How this is done exactly, depends on the specific online paradigm. For the scope of this thesis we will use the so called sequence model.

Let ALG be an *online algorithm*. In the *sequence model* requests must be served in the order of their occurrence. More precisely, when serving request r_j, ALG does not have any knowledge of requests r_i with $i > j$ (or the total number

of requests). When request r_j is presented it must be served by ALG according to the specific rules of the problem. The decision by ALG of how to serve r_j is irrevocable. Only after r_j has been served, the next request r_{j+1} becomes known to ALG. The profit obtained by ALG is denoted as ALG (σ), whereas OPT (σ) is the objective value of the optimal solution.

A *deterministic online algorithm* ALG has to behave the same way on instances which look identical. This means that if the first k requests of two instances are identical, it has to make the same decisions on both of these instances for the first k requests.

Competitive analysis has become a standard way of measuring the quality of online algorithms. Here, the objective value of the solutions obtained by an online algorithm is compared with the best possible solutions. The competitivity asks, how much profit is lost in the worst case due to the online algorithm's restricted knowledge of the input data. A precise definition of competitivity for minimization problems is given within the following definition:

Definition 2.3 (Competitivity for Deterministic Algorithms) A deterministic online algorithm ALG is said to be c-*competitive*, if there exists some constant α such that for every request sequence σ:

$$\text{ALG}\,(\sigma) \leq c \cdot \text{OPT}\,(\sigma) + \alpha$$

For $\alpha = 0$, ALG is said to be *strictly* c-*competitive*. The *competitivity* of ALG is the infimum over all c, such that ALG is c-competitive.

This measure can be seen as the value of a game between an online player trying to choose the best algorithm and a malicious adversary, who selects sequences, on which the chosen algorithm performs poorly. In the case of deterministic online algorithms, the adversary knows all of the algorithm's decisions in advance and explois this knowledge to find sequences on which the chosen algorithm performs worst.

In order to be less predictable, the online player could base its decisions on random events. For example, consider the cottage problem, introduced in Chapter 1. The owner of the cottages, could toss a coin whenever being asked to rent one of his empty houses to someone. Such a *randomized online algorithm* RALG is a probability distribution over deterministic online algorithms ALG_x (x may be thought of as the coin tosses of the algorithm RALG). We cannot compute the competitive ratio as above, as the outcome ALG (σ) is a random variable depending on the coin tosses. Thus, we need a different measure for the quality of randomized online algorithms. How good a randomized

algorithm can be, depends on how much the adversary knows about the actions performed by the online algorithm when processing a sequence, and how much he can react to them. In this thesis we only consider the *oblivious adversary*, which knows the probability distribution used by the randomized online algorithm. Based on this knowledge it chooses the entire request sequence in advance.

Definition 2.4 (Competitivity for Randomized Algorithms) A randomized online algorithm RALG is said to be c-*competitive* against an oblivious adversary, if there exists some constant α such that for every request sequence σ:

$$\mathbb{E}\left[\text{RALG}\left(\sigma\right)\right] \leq c \cdot \text{OPT}\left(\sigma\right) + \alpha$$

For $\alpha = 0$, RALG is said to be *strictly* c-*competitive*

To obtain a lower bound on the competitiveness of any randomized online algorithm, we must provide for each algorithm a worst case sequence and compute the expected cost of the algorithm on that instance. This proves to be much more difficult for randomized than for deterministic online algorithms. For this reason, we apply a variant of Yao's Principle [Yao77] from game theory to make the construction of randomized lower bounds easier.

Let X be a probability distribution over input sequences $\Sigma = \{\, \sigma_x : x \in \mathcal{X} \,\}$. We denote the expected cost of the deterministic algorithm ALG according to the distribution X on Σ by $\mathbb{E}_X\left[\text{ALG}\left(\sigma_x\right)\right]$. Yao's principle can now be stated as follows.

Theorem 2.5 (Yao's principle) *Let* $\{\, \text{ALG}_y : y \in \mathcal{Y} \,\}$ *denote the set of deterministic online algorithms for an online minimization problem. If* X *is a distribution over input sequences* $\{\, \sigma_x : x \in \mathcal{X} \,\}$ *such that*

$$\inf_{y \in \mathcal{Y}} \mathbb{E}_X\left[\text{ALG}_y(\sigma_x)\right] \geq \bar{c}\, \mathbb{E}_X\left[\text{OPT}\left(\sigma_x\right)\right]$$

for some real number $\bar{c} \geq 1$*, then* $\bar{c}$ *is a lower bound on the competitive ratio of any randomized algorithm against an oblivious adversary.*

Proof. See [BEY98, MR95]). □

These definitions and theorems carry over to maximization problems analogously.

Chapter 3

An Online Job Admission Problem

We consider the problem of scheduling a maximum profit selection of jobs on m identical machines. Jobs arrive online one by one and each job is specified by its start and end time. The goal is to determine a non-preemptive schedule which maximizes the profit of the scheduled jobs, where the profit of a job is equal to its length. Upon arrival of a new job, an online algorithm must decide whether to accept the job ("admit the job") or not. If the job is accepted, the online algorithm must be able to reorganize its already existing schedule such that the new job can be processed together with all previously admitted jobs, however, the algorithm need not specify on which machine the job will eventually be run.

We provide competitive algorithms and lower bounds on the competitive ratio for deterministic and randomized algorithms against an oblivious adversary. Our lower bound results essentially match (up to small constants factors) the competitive ratios achieved by our algorithms.

3.1 Introduction

A situation which many of us know: You try to book a cottage in your favorite holiday location for the weekend but the overly friendly person on the phone tells you that they simply can not satisfy your request. If you had called just five minutes earlier, everything would have been fine, but now there is allegedly nothing available. You doubt that this is true. Are they just rejecting your booking request because you just wanted to stay three days and not four? How do they work at all? Better: How *should* they organize their bookings? This must be some easy piece of mathematics!

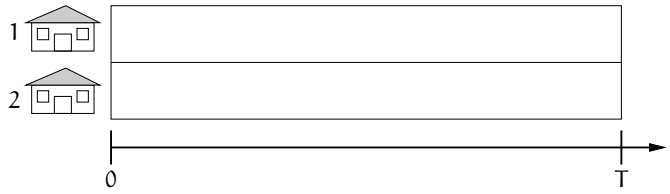

Figure 3.1: Empty booking table for the cottage-rental problem with two (identical) cottages.

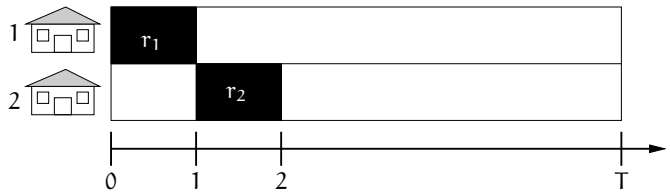

Figure 3.2: The first two customers have been booked into the plan.

You sit down and put yourself in the position of the owner of two identical cottages. The holiday season (which we assume without loss of generality to be the time interval $[0, T]$ where $T \gg 1$) is still in the future and you are awaiting for people to make reservations (Figure 3.1). Naturally, we can assume that the profit you make for a request of length l is l units of money.

A few moments later, the first customer r_1 calls and requests a cottage for the time interval $[0, 1]$. Clearly, we can promise her cottage 1 which gives us a profit of $1-0 = 1$. The next customer r_2 requests a cottage in the interval $[1, 2]$. Both cottages are available then, so we accept the booking request and schedule cottage 2 for her (Figure 3.2). This increases our profit again by one unit. Then, the next customer calls and wishes to get a cottage from 0 to 2. Yikes, we do not have a cottage available during that whole period!

But, let us think one moment. Both cottages are essentially identical and we have not promised r_2 a specific cottage, but only *a* cottage. So, we can simply move his booking from cottage 2 to cottage 1 and we can accommodate the request from r_3 (Figure 3.2). That was not too difficult, after all, was it? And our total profit has risen to $1 + 1 + 2 = 4$.

While we sit back relaxed and content, two more customers call each of which wants to book a cottage for the whole holiday season $[0, T]$. These could be called "ideal customers" in the sense that each of them allows us to fully rent out a cottage at maximum profit. But what is this? No matter how we reorganize our schedule, we can not accept any of them. If we only had known earlier that they would call, we could have rejected the requests of r_1, r_2 and

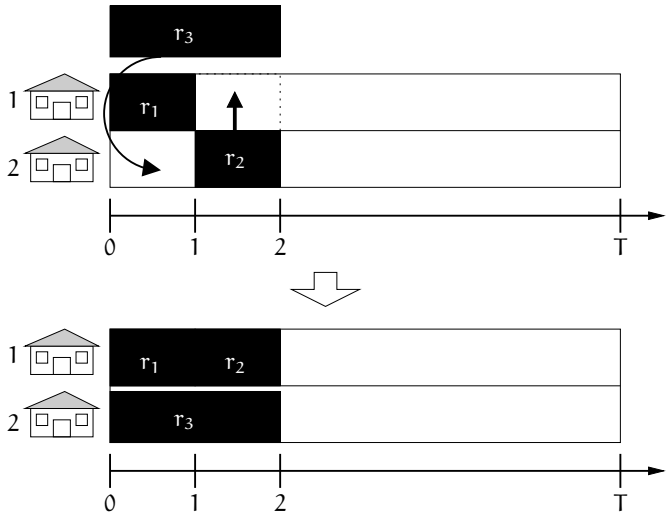

Figure 3.3: By moving customer r_2 from cottage 2 to cottage 1 the new customer r_3 can still be scheduled.

r_3 and made a profit of 2T by accepting the "ideal customers" which book the whole season instead of a lousy 4 units. On the other hand, if we reject r_1, r_2 and r_3 and the two "ideal customers" do not call then we have an empty schedule and no profit which is worse than the profit of 4 that we wince at right now. Maybe the cottage rental problem is not so easy?

We have just discovered the *online aspect* of the cottage rental problem. We are facing incomplete information, and even if every time a new request becomes known we compute a new "optimal" schedule this does not necessarily lead to an overall optimal solution.

The remainder of this paper is intended to shed some light onto the online cottage rental and related problems.

3.2 Problem Definition and Preliminaries

The cottage rental problem is a special case of the job admission problem, denoted by OJA, studied in this chapter. We are given m machines, a time horizon of T time units, and a sequence of jobs $\sigma = r_1, \ldots, r_n$, which are released one by one. Each of these jobs r_j has a fixed start time a_j and end time $b_j > a_j$, and each job needs to be accepted or rejected before we move to the next one. We assume that time has been scaled in such a way that $\min_j(b_j - a_j) = 1$. This assumption is justified for instance in the cottage rental application where the

minimum rental period is a single day (our algorithms still work if $\min_j(b_j - a_j)$ or a positive lower bound for this quantity is known in advance). The goal is to select jobs to be processed such that the sum of the lengths of the accepted jobs is maximized and there exists a feasible non-preemptive assignment of jobs to machines, i.e., such that at any moment in time each machine processes at most one job.

3.2.1 Our Results

In Section 3.4 we develop a general lower bound for the competitive ratio of randomized algorithms for the *job admission problem* (OJA). Specifically, we give a lower bound of $\frac{1}{2}(\log T + 2)$ on the competitive ratio of any randomized algorithm against an oblivious adversary, where T is the time horizon.

In Section 3.5.1 we present a first simple greedy-type deterministic $2\Delta_\sigma + 1$-competitive algorithm GREEDY, where $\Delta_\sigma = \max_{r_i, r_j \in \sigma} \frac{b_i - a_i}{b_j - a_j} = \max_{r_i \in \sigma}(b_i - a_i)$ is the maximum ratio of the profit of two jobs.[1] This simple algorithm forms the basis of the improved algorithm C-GREEDY which we present in the following Section 3.5.3. The main competitiveness result is given in Section 3.5.3, where we give a deterministic algorithm C-GREEDY that matches our lower bound from Section 3.4 up to constant factors for the case $m \geq \lceil \log T \rceil$. Moreover, we show that for $m \leq \lceil \log T \rceil$ our algorithm C-GREEDY provides a competitive ratio of $2m(\sqrt[m]{T} + 1) \leq 2 \log T(\sqrt[m]{T} + 1)$, which is also optimal up to a constant factor.

3.2.2 Previous Work

Several variations on the online job admission problem studied in this chapter have been considered in the literature. OJA is related to the problem of scheduling equal-length jobs on parallel machines, where the jobs have release times and deadlines and the goal is to maximize the number of jobs completed. Baruah et al. [BHS01] showed that a greedy-type algorithm is 2-competitive for this problem (where jobs arrive over time), a lower bound of $4/3$ for the competitive ratio of randomized algorithms was given by Goldman et al. [GPS00]. Chrobak et al. [CJST04] provided a barely random algorithm with competitive ratio $5/3$. The corresponding offline problem can be solved in polynomial time [Bap99] (see also notes in [CJST04]).

[1] Recall that time has been scaled in such a way that $\min_j(b_j - a_j) = 1$.

Van Stee and La Poutré [SP01] considered the problem of partial servicing of online jobs. Here, jobs arrive over time to be rejected or accepted, after which they must start immediately. The algorithm can choose to serve some jobs only partially, and the goal is as here to maximize the profit. The problem is different from ours in its notion of time (new requests cannot appear in the past) and because of the option of serving jobs partially. However, it turns out that several ideas from [SP01] can be used also to give good algorithms for the current problem.

The problem OJA in this chapter can also be seen as a generalized version of online interval scheduling [LT94], where only one machine is available. However, in the paper [LT94], jobs arrive over time instead of one by one. Also, there is no pre-specified time horizon. As mentioned at the beginning, another similar problem which has been studied is seat reservations on trains [BL99, BKN04]. Here passengers arrive online, specifying their desired connection, and need to be assigned a seat immediately. Differences to that paper are that in making seat reservations, it is assumed that an algorithm is not allowed to reject any passenger for whom there is still room in the train, and they furthermore assume that the seat (in our case: machine) has to be assigned immediately upon request. (The paper [BKN04] considers a slightly relaxed case where each passenger may change seats a fixed number of times during the trip.)

Finally, OJA can also be seen as a call admission problem in an optical network [AAF+01, GK05, KP02]. In our case the network is simply a line. The main difference to optical call-admission on the line is the profit model. For call-admission one assumes that each job has a uniform value[2], independent of its length. It seems that this changes the flavor of the problem substantially, since rejecting a large job does not lose you more than rejecting a short job, and generally short jobs are easier to schedule.

3.3 The Offline Problem

In this section we show briefly how the offline problem corresponding to OJA can be solved efficiently. Given a set of jobs J consider a directed graph $G = (V, A)$ with the following nodes: a source s, a sink t and for every job $r_j = [a_j, b_j] \in J$ two nodes u_j, v_j. Thus, $V := \cup_{r_j \in J}\{u_j, v_j\} \cup \{s, t\}$.

[2]The value may depend on the bandwidth of the call but not on its length, which is the path used to route the call.

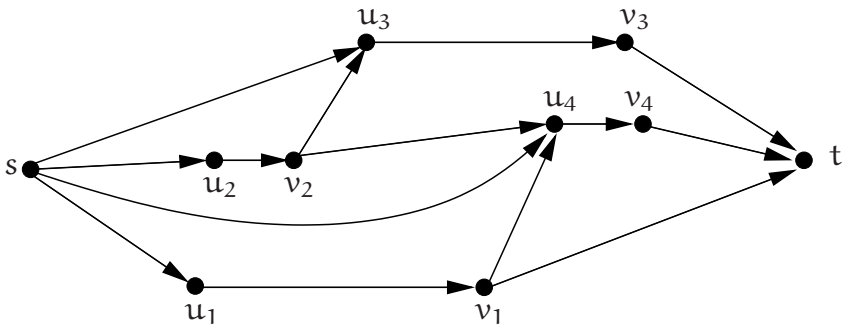

Figure 3.4: A graph G corresponding to a set of jobs $J = \{r_1, \ldots, r_4\}$ with $a_1 < a_2 < b_2 < a_3 < b_1 < a_4 < b_3 < b_4$

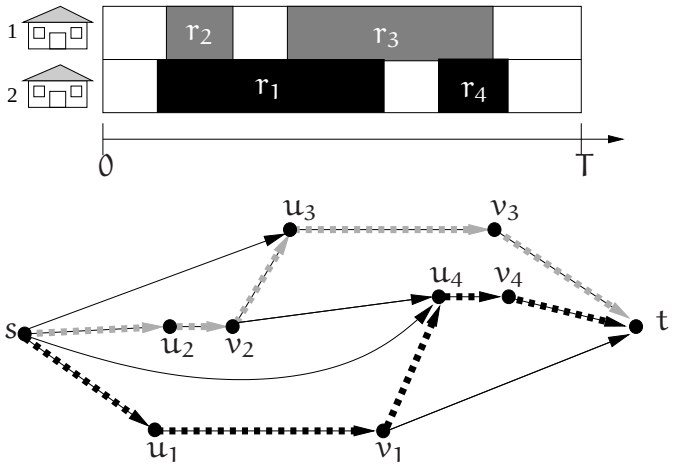

Figure 3.5: Each set of jobs assigned to a single machine in a feasible solution corresponds to an s-t-path in the network G.

For all $r_j \in J$ we have an arc (u_j, v_j) with cost $a_j - b_j$, and the arcs (s, u_j) and (v_j, t) with cost 0. The arc (u_j, v_j) corresponds to the situation where job r_j is accepted. For all $r_i, r_j \in J$ with $b_i \leq a_j$ we introduce an additional arc (v_i, u_j) with cost 0 representing the possibility that r_j can be scheduled directly after r_i on the same machine. All arcs have unit capacity. An example for such a network is shown in Figure 3.4.

Let f be an integral minimum cost flow of value m in G from s to t. Such a flow can be computed efficiently by standard techniques, see e.g. [AMO93]. We claim that an optimal solution of the job admission problem is given by accepting job r_j if and only if the flow value $f(u_j, v_j)$ on arc (u_j, v_j) is nonzero (that is, it is one by our choice of the capacities).

In fact, every feasible schedule of jobs on m machines gives rise to m edge-disjoint paths in G from s to t, one for each machine (cf. Figure 3.5). The profit obtained on the machine equals the negative of the cost of the corresponding path.

Conversely, by flow decomposition [AMO93] we can decompose every s-t-flow of value m in G into m disjoint s-t-paths (since the graph G is acyclic, there can not be cycles in the flow decomposition). Since every s-t-path corresponds to a set of jobs which can be scheduled on a single machine, these paths specify an assignment of the jobs to the specific machines.

3.4 Lower Bounds

Let us first consider the question how well a deterministic algorithm can perform in terms of competitiveness. We first start with deterministic algorithms. To this end, let us consider the situation we had in the introduction once more: we scheduled three small jobs but then could not accommodate the two long jobs (which span the whole interval $[0, T]$) any more.

We are given m machines and the time interval $[0, T]$ for scheduling jobs. The basic idea of our lower bound construction is the following. Let ALG be some arbitrary deterministic online algorithm. We first present m small jobs of length 1 each for the interval $[0, 1]$. If the online algorithm accepts all of the jobs, it gets a profit of $m \cdot 1 = m$ and we will then present m large jobs of length T each for the interval $[0, T]$. Thus, the optimal profit is mT and we can force a ratio of T between the optimal and the online profit. What makes the argument slightly more complicated is the fact that ALG need not accept *all* of the jobs of length 1, so it might have some empty space on the machines which can be used to accommodate the long jobs.

Theorem 3.1 *No deterministic algorithm can achieve a competitive ratio smaller than* $\frac{1}{2}(\log T + 2)$ *for the* OJA.

Proof. Assume that ALG is a c-competitive algorithm for the OJA. The first part σ_0 of the adversial sequence consists of m jobs of unit size starting at time 0 and ending at time 1. Let $q(0)$ be the number of requests from σ_0 accepted by ALG . Since $\text{OPT}(\sigma_0) = m$ and ALG is c-competitive, we must have that $q(0) \geq m/c$.

We will now continue to present blocks of m requests each of length 2^i for $i = 1, \ldots, 2^{\log T}$. Each request starts at time 0. Let σ_i denote the subsequence formed by the requests of length 2^i. Our total input sequence σ is thus $\sigma =$

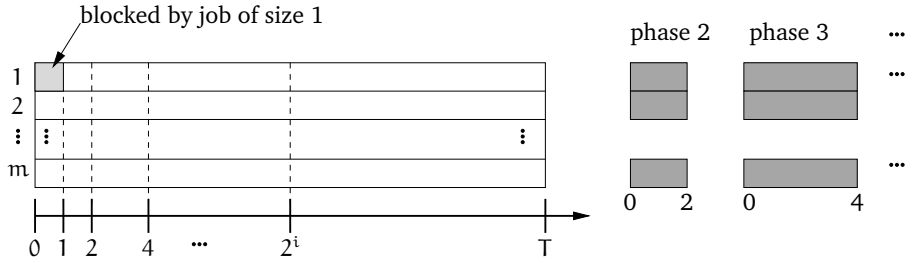

Figure 3.6: Once a job is accepted, it blocks a machine for all future requests.

$\sigma_0\sigma_1 \ldots \sigma_{2^{\log T}}$. Observe that each accepted request blocks a machine for future requests (cf. Figure 3.6).

As we have seen before after σ_0 the online algorithm must have accepted $q(0) \geq m/c$ jobs and thus blocked $q(0)$ machines for future requests. Once the requests in σ_1 have been presented, the optimal solution is to reject all request in σ_0 and accept all the jobs from σ_1 of size 2. Thus, we have $\textsc{Opt}(\sigma_0\sigma_1) = 2m$. In order to achieve a competitive ratio of c the profit obtained by ALG must satisfy:

$$q(0) \cdot 1 + q(1) \cdot 2 \geq \frac{\textsc{Opt}(\sigma_0\sigma_1)}{c} = \frac{2m}{c}, \tag{3.1}$$

where $q(1)$ denotes the number of requests accepted by ALG from the subsequence σ_1. We have seen above that $q(0) \geq m/c$. Since all requests from σ_i for $i \geq 1$ give more profit than the unit size jobs from σ_0 we can assume that ALG accepts *exactly* m/c jobs from σ_0. Using $q(0) = m/c$ in (3.1) results in

$$q(1) \geq \frac{1}{2}\left(\frac{2m}{c} - \frac{m}{c}\right) = \frac{m}{2c}.$$

By the same argument as above we can assume that $q(1) = \frac{m}{2c}$, since this leaves ALG with more space for future jobs which are more profitable than the small ones already seen. We will now show by induction on i that in fact the number $q(i)$ of jobs accepted by ALG from σ_i satisfies $q(i) = \frac{m}{2c}$ for $i \geq 1$. The claim has already been established for $i = 1$. Let us assume that we know that $q(0) = m/c$ and $q(1) = \cdots = q(i-1) = m/2c$. We have $\textsc{Opt}(\sigma_0 \ldots \sigma_i) = m2^i$.

By the fact that ALG is assumed to be c-competitive we have:

$$\begin{aligned}\frac{\text{OPT}(\sigma_0 \dots \sigma_i)}{c} = \frac{2^i m}{c} &\leq \left(\sum_{j=0}^{i-1} 2^j \cdot q(j)\right) + 2^i \cdot q(i) \\ &= \frac{m}{c} + \left(\sum_{j=1}^{i-1} 2^j \cdot \frac{m}{2c}\right) + 2^i \cdot q(i) \\ &= \frac{m}{c}(1 + 2^{i-1} - 1) + 2^i \cdot q(i) \\ &= \frac{2^{i-1} m}{c} + 2^i \cdot q(i)\end{aligned}$$

Solving for $q(i)$ yields $q(i) \geq \frac{m}{2c}$ and by the now familiar argument from above we get that $q(i) = \frac{m}{2c}$. This completes the inductive step.

We are now in the position to establish the lower bound on the competitiveness c of ALG . Observe that the total number of machines blocked by ALG after phase i is $\sum_{j=0}^{i} q(i)$. Since the total number of machines is m we get

$$m \geq \sum_{j=0}^{\log T} q(j) = q(0) + \sum_{j=1}^{\log T} q(j) = \frac{m}{c} + \frac{m}{2c} \log T = \frac{1}{c} m (1 + \frac{1}{2} \log T).$$

This gives us $c \geq 1 + \frac{1}{2} \log T = \frac{1}{2}(\log T + 2)$ as claimed. □

We now extend our lower bound result to randomized algorithms against an oblivious adversary. The difficulty lies in the fact that it is not clear how a *generic randomized algorithm* RALG looks like. Using the same line of arguments as in Theorem 3.1 we see that RALG only needs to accept m/c unit size jobs *on expectation* and not with probability one. Thus, in order to establish our bound, we make use of the following version of Yao's principle:

Theorem 3.2 (Yao's Principle) *Let* G *be an online optimization problem, let* ALG *be any online randomized algorithm for* G *and let* c_{ALG} *be the competitive ratio of* ALG *against an oblivious adversary. Let* $p(i)$ *be any probability distribution over request sequences. Then*

$$c_{\text{ALG}} \geq \max \left\{ \min_j \frac{\mathbb{E}_{p(i)}[\text{OPT}(\sigma_i)]}{\mathbb{E}_{p(i)}[\text{ALG}_j(\sigma_i)]}, \min_j \frac{1}{\mathbb{E}_{p(i)}\left[\frac{\text{ALG}_j(\sigma_i)}{\text{OPT}(\sigma_i)}\right]} \right\}. \tag{3.2}$$

Proof. See [BEY98, BEY99]. □

Theorem 3.3 *Any randomized algorithm for* OJA *has a competitive ratio no smaller than* $\frac{1}{2}(\log T + 2)$ *against an oblivious adversary.*

Proof. For $i = 0, \ldots, \log T$, consider the sequence σ_i which for each $0 \leq j \leq i$ contains m jobs of length 2^j (thus, σ_i specifies a total of $(i+1)m$ jobs). The jobs in σ_i will be given in increasing order of length, and all requests have a start time of 0. Clearly, for each machine at most one of these jobs can be contained in any schedule and we have $\textsc{Opt}\,(\sigma_i) = m2^i$.

We make use of the second bound in (3.2) to derive the lower bound on the competitive ratio of randomized algorithms against an oblivious adversary. Specifically, we give a distribution $p(i)$ over the request sequences σ_i such that for any deterministic algorithm ALG we have

$$\mathbb{E}_{p(i)}\left[\frac{\textsc{Alg}\,(\sigma_i)}{\textsc{Opt}\,(\sigma_i)}\right] \leq \frac{2}{\log T + 2}.$$

Using Yao's principle from Theorem 3.2 above then yields the desired lower bound of $1/(\frac{2}{\log T+2}) = \frac{1}{2}(\log T + 2)$.

Let $q(j)$ be the number of jobs of length 2^j accepted by a given deterministic algorithm ALG , when given any sequence σ_i where $i \geq j$. Since until the point in time when the requests of length 2^j are given these σ_i are identical, ALG has to make the same decision, how many of these jobs to accept and therefore $q(j)$ has to be identical for all those σ_i.

Thus, when processing σ_i the ratio of the profits by ALG and OPT is

$$\frac{\textsc{Alg}\,(\sigma_i)}{\textsc{Opt}\,(\sigma_i)} = \frac{\sum_{j=0}^{i} q(j) \cdot 2^j}{m \cdot 2^i}. \tag{3.3}$$

We now derive a probability distribution p over the sequences σ_i such that for each deterministic algorithm ALG we have

$$\mathbb{E}_{p(j)}\left[\frac{\sum_{j=0}^{i} q(j) \cdot 2^j}{m \cdot 2^i}\right] \leq \frac{2}{\log T + 2}.$$

As afore mentioned by using Yao's principle the bound then follows.

Let $p(i)$ to be the probability that σ_i occurs. Then, the expected value of the profit ratio can be computed by:

$$\begin{aligned}
\mathbb{E}_{p(i)}\left[\frac{\textsc{Alg}\,(\sigma_i)}{\textsc{Opt}\,(\sigma_i)}\right] &= \sum_{i=0}^{\log(T)} p(i) \cdot \frac{\textsc{Alg}\,(\sigma_i)}{\textsc{Opt}\,(\sigma_i)} = \sum_{i=0}^{\log T} p(i) \cdot \frac{\sum_{j=0}^{i} q(j) \cdot 2^j}{m \cdot 2^i} \\
&= \sum_{i=0}^{\log T} p(i) \sum_{j=0}^{i} \frac{2^{j-i}}{m} \cdot q(j) = \sum_{j=0}^{\log T} \sum_{i=j}^{\log T} \frac{2^{j-i} p(i)}{m} \cdot q(j)
\end{aligned}$$

Observe that, given a distribution $p(i)$ on the instances σ_i, all deterministic algorithms only differ in the number of jobs they accept of each of the given length classes of jobs. Thus, we can find the deterministic algorithm with the largest expected profit ratio by solving the following integer linear program:

$$\begin{aligned}
\text{(IP1)} \qquad \max \quad & \sum_{j=0}^{\log T} \sum_{i=j}^{\log T} \frac{2^{j-i} p(i)}{m} \quad q(j) \\
\text{s.t.} \quad & \sum_{j=0}^{\log T} q(j) \leq m \\
& q(j) \geq 0, q(j) \in \mathbb{Z} \qquad \text{for all } j = 0, \ldots, \log T
\end{aligned}$$

To obtain an upper bound for the optimal solution of this problem it suffices to find a feasible solution of the dual of its linear relaxation, which is given by:

$$\begin{aligned}
\text{(LP1)} \qquad \min \quad & m \cdot y \\
\text{s.t.} \quad & y \geq \sum_{i=j}^{\log T} \frac{2^{j-i} p(i)}{m} \qquad \text{for all } j = 0, \ldots, \log T \\
& y \geq 0
\end{aligned}$$

Observe that the dual (LP1) has only a single variable. Thus, we can easily compute its optimal solution:

$$\begin{aligned}
& \min\{my : y \geq \sum_{i=j}^{\log T} \frac{2^{j-i} p(i)}{m}, j = 0, \ldots, \log T\} \\
= \quad & \max_{j=0,\ldots,\log T} \{m \cdot \sum_{i=j}^{\log T} \frac{2^{j-i} p(i)}{m}\} = \max_{j=0,\ldots,\log T} \{\sum_{i=j}^{\log T} 2^{j-i} p(i)\}.
\end{aligned}$$

The quality of the bound obtained this way depends on the applied distribution p. The distribution which yields the best bound can be found by solving

another linear program:

$$
\begin{array}{lll}
\text{(LP2)} \quad \min & y & \\
\text{s.t.} & \sum_{i=j}^{\log T} 2^{j-i} p(i) \le y & \text{for all } j = 0, \ldots, \log T \\
& \sum_{i=0}^{\log T} p(i) = 1 & \\
& p(i) \ge 0 & \text{for all } i = 0, \ldots, \log T \\
& y \ge 0 &
\end{array}
$$

The optimum is attained for $p(i) := \frac{1}{\log T+2}$ for $i = 0, \ldots, \log T - 1$ and $p(\log T) = \frac{2}{\log T+2}$, which can be easily seen by using the Fundamental Theorem of Linear Programming (see e.g. [Chv83]) and the fact that p as given above is a basic solution. Thus, we have

$$
\begin{aligned}
\max_{j=0,\ldots,\log T} \{\sum_{i=j}^{\log T} 2^{j-i} p(i)\} &= \max_{j=0,\ldots,\log T} \{\left(\sum_{i=j}^{\log T} \frac{2^{j-i}}{\log T + 2} \right) + \frac{2^{j-\log T}}{\log T + 2}\} \\
&= \max_{j=0,\ldots,\log T} \{\frac{1}{\log T + 2} \left(\frac{2^j}{T} + \sum_{i=j}^{\log T} 2^{j-i} \right)\} \\
&= \max_{j=0,\ldots,\log T} \{\frac{1}{\log T + 2} \left(\frac{2^j}{T} + 2^j \sum_{i=j}^{\log T} \frac{1}{2^i} \right)\} \\
&= \max_{j=0,\ldots,\log T} \{\frac{1}{\log T + 2} \left(\frac{2^j}{T} + 2^j \cdot 2 \cdot \left(\frac{1}{2^j} - \frac{1}{2^{\log T+1}} \right) \right)\} \\
&= \max_{j=0,\ldots,\log T} \{\frac{1}{\log T + 2} \left(\frac{2^j}{T} + 2 - \frac{2^j}{T} \right)\} = \frac{2}{\log T + 2}
\end{aligned}
$$

This completes the proof. □

3.5 Competitive Algorithms

In this section we present competitive algorithms for the OJA. The basis of our algorithms is provided by a simple greedy-type algorithm GREEDY which we analyze in Section 3.5.1. This algorithm works acceptably well if all jobs

have approximately the same length. In Section 3.5.2 we show how to use randomization in order to turn GREEDY into a competitive algorithm CRS-GREEDY for jobs of substantially different sizes. In Section 3.5.3 we essentially derandomize CRS-GREEDY and obtain the same competitiveness bounds by means of deterministic algorithms.

3.5.1 A Greedy-Type Deterministic Algorithm

Let GREEDY be the algorithm, which accepts a new job r_j as long as there exists a schedule which contains all previously accepted jobs and r_j. For a given input sequence σ, define its *length ratio* as $\Delta_\sigma := \max_{r_i, r_j \in \sigma} \frac{b_i - a_i}{b_j - a_j} = \max_{r_i \in \sigma}(b_i - a_i)$, i.e., the maximum ratio of two job durations in σ. The algorithm GREEDY does not require that $\min_j(b_j - a_j) = 1$. Our first goal is to establish that GREEDY is $2\Delta_\sigma + 1$-competitive. In the remainder of this section, we will consider a fixed input sequence σ and simply write Δ for Δ_σ.

Let σ_G be the set of jobs, which are accepted by GREEDY and σ_{OPT} be the set of jobs, which are accepted by an optimal offline algorithm. We denote by $X := \sigma_G \cap \sigma_{OPT}$ be the set of jobs which are accepted by both of these algorithms, $Y := \sigma_G \setminus \sigma_{OPT}$ be the set of all the jobs accepted only by GREEDY and $Z := \sigma_{OPT} \setminus \sigma_G$ the set of jobs only accepted by the optimal offline-algorithm.

Consider the schedule that GREEDY outputs. Consider the machines one by one, and on each machine, consider the jobs on it from left to right. Denote the jobs on machine j by $1, \ldots, i_j$, their start times by a_i and finish times by b_i ($i = 1, \ldots, i_j$). Whenever $a_{i+1} - b_i > 2\Delta$, we say that the interval $[b_i + \Delta, a_{i+1} - \Delta]$ is a *gap*. If this happens on machine j, we say that the gap is of type j.

Lemma 3.4 *Every job in Z has an empty intersection with every gap.*

Proof. Suppose there is a job in Z that has nonzero intersection with some gap of type j. This job could be placed entirely on the machine j, without overlapping the existing jobs on that machine, by the definition of a gap (since its length is at most Δ). So GREEDY would have accepted this job, a contradiction. □

Theorem 3.5 *For an input sequence σ with length ratio Δ,* GREEDY *achieves a competitive ratio of $2\Delta + 1$.*

Proof. Take a machine j. Consider an interval between two gaps of type j (or an interval until the first gap / after the last gap / the entire interval $[0, T]$, if

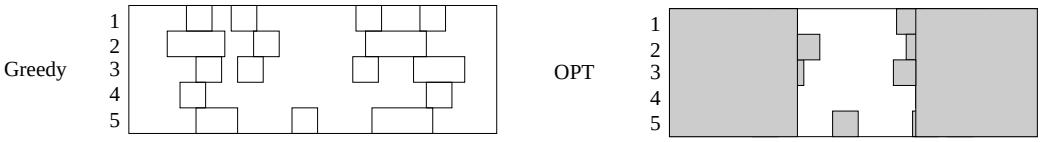

Figure 3.7: An example schedule of GREEDY for an input where $\Delta = 3$. There are five machines, time is on the horizontal axis. All the jobs served by OPT that GREEDY does not serve must be within the shaded areas. Since there is a gap on machine 4, the jobs between the large shaded areas are either both served by GREEDY and OPT , or only by GREEDY.

there are no gaps on machine j). Call such an interval a non-gap-interval. On machine j, there can be at most Δ idle time at the start and at the end of a non-gap-interval. If this is not true, the gap would have been defined differently. Thus on machine j, some job starts within time Δ of any gap, and after that job finishes, each time within time 2Δ a new job starts, until the next gap appears (at most Δ after the last job completes) or the end of the schedule is reached. Since each job has length at least 1, this means that on machine j, within each non-gap-interval, at least $1/(1 + 2\Delta)$ of the time some job is running in the schedule of GREEDY. This reasoning holds for any machine $j = 1, \ldots, m$. For future calculations, we now say simply that GREEDY is running a job of density ("height") $1/(1 + 2\Delta)$ *at all times* within each non-gap-interval. This does not increase the overall profit of GREEDY and simplifies the comparison to OPT .

On the other hand, in an optimal solution, by Lemma 3.4 no jobs in Z can be running at any time during gaps. Now consider the intervals between two gaps of *any* type in order of increasing starting time. Call these intervals "allowed intervals". We find that all jobs in Z are run only during allowed intervals. However, on each machine, an allowed interval I is a subinterval of a non-gap-interval, so on each machine GREEDY earns (running this job of density $1/(1 + 2\Delta)$) at least $1/(1 + 2\Delta)$ of the length of I. So in total, during I it earns at least $m/(1 + 2\Delta)$ times the length of I, and of course OPT earns at most m times the length of I during I.

Finally, consider a gap G. The only jobs that OPT has accepted and that overlap (partially) with G are the jobs in X that GREEDY is also running, by Lemma 3.4. However, we have modified the GREEDY-schedule by spreading each job out over a non-gap-interval. Thus for a job that runs for t units of time during a gap G, we have that GREEDY earns at least $t/(1 + 2\Delta)$ during G, and OPT clearly earns at most t during G.

This concludes the proof. An illustration is given in Figure 3.7. □

3.5.2 An Algorithm Based on Classify and Randomly Select

In this section we show that the GREEDY algorithm from above can be used to obtain a randomized algorithm CRS-GREEDY with competitive ratio $5\lceil\log(T)\rceil$ by applying the classify and randomly select-paradigm [ABFR94].

Assume again that the minimum length of an interval is $\min_{r_j\in\sigma}(b_j - a_j) = 1$. We divide the possible input requests into $N := \lceil\log T\rceil$ disjoint classes $C_1, \ldots, C_N$, with $j \in C_i$ if and only if $2^{i-1} \leq b_j - a_j < 2^i$. The algorithm CRS-GREEDY chooses class C_i with probability $\frac{1}{N}$. Then, when processing a sequence σ the algorithm ignores all requests not in class C_i and uses GREEDY to process the requests in class C_i.

For $i = 1, \ldots, N$ let $\sigma_i := \sigma \cap C_i$ and OPT_i denote the total profit of jobs from class C_i accepted by OPT. If GREEDY processes σ_i for some i, it achieves a competitive ratio of 5, since $\Delta_{\sigma_i} \leq \frac{2^i}{2^{i-1}} = 2$. Since there is a probability of $\frac{1}{N}$ that the algorithm picks the class which contributes the biggest part to the optimal solution we can estimate the expected value of the machine time obtained by CRS-GREEDY as follows:

$$\begin{aligned}\mathbb{E}[\text{CRS-GREEDY}(\sigma)] &= \sum_{i=1}^{N}\frac{1}{N}\cdot\text{GREEDY}(\sigma_i) \geq \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2\Delta_{\sigma_i}+1}\text{OPT}(\sigma_i)\\ &\geq \frac{1}{5N}\sum_{i=1}^{N}\text{OPT}(\sigma_i) \geq \frac{1}{5N}\sum_{i=1}^{N}\text{OPT}_i = \frac{1}{5N}\text{OPT}(\sigma).\end{aligned}$$

Thus, CRS-GREEDY achieves a competitive ratio of $5N = 5\lceil\log T\rceil$.

We remark here that the above algorithm can be modified easily for the case that $\min_j(b_j - a_j) = \varepsilon \neq 1$ is known in advance and then provides a competitive ratio of $5\lceil\log T/\varepsilon\rceil$.

3.5.3 An Improved Deterministic Algorithm

We will now use GREEDY and ideas from the classify-and-select paradigm to obtain a deterministic algorithm which achieves an improved competitiveness. As in the previous section we first assume that $\min_j(b_j - a_j) = 1$.

Lemma 3.6 *Suppose that we are given a sequence of jobs σ. For $1 \leq k \leq m$ let $\text{OPT}^{(k)}(\sigma)$ denote the optimal offline profit achievable using k machines (so that $\text{OPT}(\sigma) = \text{OPT}^{(m)}(\sigma)$). Then,*

$$\frac{k}{m}\cdot\text{OPT}^{(m)}(\sigma) \leq \text{OPT}^{(k)}(\sigma) \leq \text{OPT}^{(m)}(\sigma).$$

Proof. Given an optimal solution for m machines, a feasible solution can be obtained by accepting the jobs scheduled on the k machines with the highest profits. Thus, $\text{OPT}^{(k)}(\sigma) \geq \frac{k}{m}\text{OPT}^{(m)}(\sigma)$. The second inequality is trivial. □

Similar to the randomized algorithm CRS-GREEDY, the improved deterministic algorithm C-GREEDY divides the jobs into classes. How this is done, depends on the specific relation between the number m of machines and the time horizon T.

Instances with $m \geq \lceil \log T \rceil$

In order to simplify the presentation, we first assume that the time horizon $T = 2^k$ is a power of two. In this case, C-GREEDY reserves exactly $\lfloor m/\log T \rfloor$ machines for each of the classes C_i. For each of the $k = \log T$ classes it uses an instantiation of GREEDY to process the jobs.

Lemma 3.7 *If $T = 2^k$ and $m = k \cdot t$ for some $k, t \in \mathbb{Z}^+$, then* C-GREEDY *is $5 \log T$ competitive.*

Proof. Similar as in Lemma 3.6 let $\text{OPT}^{(t)}$ and $\text{GREEDY}^{(t)}$ be the respective algorithms which schedule jobs on $t = m/\log T$ machines instead of on m machines. Let OPT_i be the profit of OPT obtained by jobs in class C_i. Then

$$\begin{aligned}
\text{OPT}(\sigma) &= \sum_{i=1}^{\log T} \text{OPT}_i \leq \sum_{i=1}^{\log T} \text{OPT}^{(m)}(\sigma_i) \\
&\overset{\text{Lemma 3.6}}{\leq} \sum_{i=1}^{\log T} \log T \cdot \text{OPT}^{(t)}(\sigma_i) \qquad (3.4) \\
&\overset{\text{Theorem 3.5}}{\leq} \log T \sum_{i=1}^{\log T} (2\Delta_{J_i} + 1)\text{GREEDY}^{(t)}(\sigma_i) \\
&\overset{\Delta_{C_i} \leq 2}{\leq} 5 \log T \sum_{i=1}^{\log T} \text{GREEDY}^{(t)}(\sigma_i) = 5 \log T \cdot \text{C-GREEDY}(\sigma).
\end{aligned}$$

□

Lemma 3.8 *If $T = 2^k$ and $m \geq \log T$ for some $k \in \mathbb{Z}^+$, then* C-GREEDY *is $10 \log T$ competitive.*

Proof. For $m \geq \log T$ we have $2\lfloor \frac{m}{\log T} \rfloor \geq \frac{m}{\log T}$. Thus, we get

$$\text{OPT}(\sigma_i) \leq \frac{m}{\lfloor \frac{m}{\log T} \rfloor} \text{OPT}^{(t)}(\sigma_i) \leq 2 \log T \cdot \text{OPT}^{(t)}(\sigma_i). \tag{3.5}$$

Using the computation from Lemma 3.7, but applying (3.5) instead of Lemma 3.6 in (3.4) gives the desired bound on the competitive ratio. □

We finally extend our result to the general case where T is not a power of two and m is not an integer multiple of $\log T$.

If T is not a power of two, then C-GREEDY simply rounds up T to the next power $2^k \geq T$ of two, so that $2^{k-1} \leq T < 2^k$. Since every instance with given T can be seen as an instance with time horizon 2^k we obtain the following result by applying Lemma 3.8:

Theorem 3.9 *For $m \geq \lceil \log T \rceil$, the algorithm* C-GREEDY *is $10(\lceil \log T \rceil) \leq 10(\log T + 1)$-competitive.* □

Instances with $m < \lceil \log T \rceil$

If $m < \lceil \log T \rceil$, then C-GREEDY uses a different partition of the machines by using a different classification of the jobs. For $j = 1, \ldots, m$ machine j is only allowed to accept jobs with length between $T^{\frac{j-1}{m}}$ and $T^{\frac{j}{m}}$. This way we obtain m classes where for each class i the ratio of the longest and the smallest possible jobs is $\Delta_i = \sqrt[m]{T}$.

Theorem 3.10 *For $m \leq \lceil \log T \rceil$,* C-GREEDY *has a competitive ratio of $2m(\sqrt[m]{T} + 1)$.*

Proof. Analogously to the two preceding proofs, we can upper bound the optimal offline profit as:

$$\begin{aligned}
\textsc{Opt}\,(\sigma) &\leq \sum_{i=1}^{m} \textsc{Opt}^{(m)}(\sigma_i) \\
&\overset{\text{Lemma 3.6}}{\leq} m \sum_{i=1}^{m} \textsc{Opt}^{(1)}(\sigma_i) \\
&\overset{\text{Theorem 3.5}}{\leq} m \sum_{i=1}^{m} 2(\Delta_i + 1)\textsc{greedy}^{(1)}(\sigma_i) \\
&\overset{\Delta_i \leq \sqrt[m]{T}}{\leq} m \cdot 2(\sqrt[m]{T} + 1) \sum_{i=1}^{m} \textsc{greedy}^{(1)}(\sigma_i) \\
&= m \cdot 2(\sqrt[m]{T} + 1) \cdot \textsc{c-greedy}(\sigma).
\end{aligned}$$

□

The input sequence from Lemma 4 of [SP01] can be adapted for the current problem by letting all jobs have the same starting time (since jobs no longer arrive over time but in a list). This gives a lower bound of $m(\sqrt[m]{T} - 1)$ for any online algorithm. This means that the algorithm C-GREEDY is optimal up to a factor of slightly more than 2.

We note that the deterministic algorithm C-GREEDY can be modified to handle that case that $\min_{r_j \in \sigma}(b_j - a_j) \neq 1$ but this quantity (or a lower bound $\varepsilon > 0$ for it) is known in advance. In this case in all competitiveness bounds T is replaced by T/ε in the expressions.

3.6 Semi-Online with non-increasing job sizes

In this section, we examine the competitive ratios which can be achieved by randomized and deterministic algorithms, if there is some more knowledge given about future jobs. A common way to do so, is to assume that the jobs arrive in order of non-increasing job lengths ([SSW98], [EF01]).

3.6.1 Lower Bounds for Deterministic Algorithms

Let ALG be a deterministic c-competitive algorithm. Then, when being presented σ_1, which contains m jobs of length 1 starting at time 1, it has to accept

at least $\frac{m}{c}$ jobs. Thus, when being presented σ_2, containing all of the jobs of σ_1 and m additional jobs of length 1 starting at time $\epsilon/2$, m jobs of length 1, starting at time $2-\epsilon/2$ and m jobs of length $1-\epsilon$, starting at time $1+\epsilon/2$, the algorithm can make a profit of at most

$$\frac{m}{c} \cdot 1 + \left(m - \frac{m}{c}\right) \cdot (3-\epsilon).$$

whereas the optimal algorithm achieves a profit $(3-\epsilon) \cdot m$. Since the algorithm is c-competitive, it must hold that

$$\frac{\frac{m}{c} \cdot 1 + \left(m - \frac{m}{c}\right) \cdot (3-\epsilon)}{(3-\epsilon) \cdot m} \geq \frac{1}{c} \quad \text{resp.} \quad c \geq \frac{5-2\epsilon}{3-\epsilon}$$

Since ϵ can be chosen arbitrarily small, there is no deterministic algorithm with competitivity less than $5/3$. Randomization cannot help here, as the following theorem shows.

Theorem 3.11 *Any randomized algorithm for* OJA *with non-increasing job lengths has a competitive ratio no smaller than* $5/3$.

Proof. We show this bound by applying Yao. Let σ_1, σ_2 be defined as in the proof for the deterministic bound above, p be the probability that the examined sequence σ equals σ_1 and let $1-p$ be the probability that σ equals σ_2. It suffices to consider the deterministic algorithms, which accept k jobs from σ_1 and then as many as possible from $\sigma_2 \setminus \sigma_1$. Figure 3.8 shows such a scheduling.

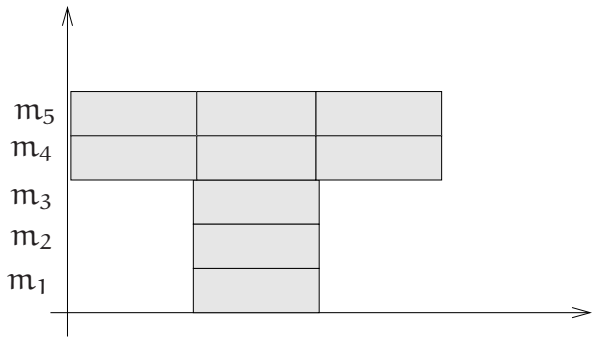

Figure 3.8: Example for five machines with $k = 3$

The expected value of the competitivity of ALG $^{(k)}$ is:

$$\begin{aligned}
\mathbb{E}\left[\frac{c(\text{ALG}^{(k)}(\sigma))}{c(\text{OPT}(\sigma))}\right] &= p \cdot \frac{c(\text{ALG}^{(k)}(\sigma_1))}{c(\text{OPT}(\sigma_1))} + (1-p) \cdot \frac{c(\text{ALG}^{(k)}(\sigma_2))}{c(\text{OPT}(\sigma_2))} \\
&= p \cdot \frac{k}{m} + (1-p) \cdot \frac{k + (3-\epsilon) \cdot (m-k)}{(3-\epsilon) \cdot m} \\
&= p \cdot \frac{k}{m} + (1-p) \cdot \left(\frac{k}{(3-\epsilon) \cdot m} + 1 - \frac{k}{m}\right) \\
&= p \cdot \frac{k}{m} + \left(\frac{k}{(3-\epsilon) \cdot m} + 1 - \frac{k}{m}\right) \\
&\qquad - p \cdot \left(\frac{k}{(3-\epsilon) \cdot m} + 1 - \frac{k}{m}\right) \\
&= \frac{k}{(3-\epsilon) \cdot m} + 1 - \frac{k}{m} - \left(\frac{k}{(3-\epsilon) \cdot m} + 1 - \frac{2k}{m}\right) \cdot p \\
&= \frac{k}{(3-\epsilon) \cdot m} + 1 - \frac{k}{m} - \frac{pk}{(3-\epsilon) \cdot m} - p + \frac{2pk}{m} \\
&= 1 - p + \left(\frac{1}{(3-\epsilon) \cdot m} - \frac{1}{m} - \frac{p}{(3-\epsilon) \cdot m} + \frac{2p}{m}\right) \cdot k
\end{aligned}$$

Since the expected value is a linear function of k, the maximum is attained for $k = 0$ if the slope is negative and for $k = m$ if the slope is positive. Thus, it suffices to consider the competitivities of $\text{ALG}^{(0)}$ and $\text{ALG}^{(k)}$ which are given by

$$\mathbb{E}\left[\frac{c(\text{ALG}^{(0)}(\sigma))}{c(\text{OPT}(\sigma))}\right] = 1 - p$$

and

$$\begin{aligned}
\mathbb{E}\left[\frac{c(\text{ALG}^{(m)}(\sigma))}{c(\text{OPT}(\sigma))}\right] &= 1 - p + \left(\frac{1}{(3-\epsilon) \cdot m} - \frac{1}{m} - \frac{p}{(3-\epsilon) \cdot m} + \frac{2p}{m}\right) \cdot m \\
&= 1 - p + \left(\frac{1}{3-\epsilon} - 1 - \frac{p}{3-\epsilon} + 2p\right) \\
&= \frac{1 - p + (3-\epsilon)p}{3-\epsilon} = \frac{1 + (2-\epsilon)p}{3-\epsilon}.
\end{aligned}$$

So, for given p and ϵ we have a lower bound of $\max\{1-p, \frac{1+(2-\epsilon)p}{3-\epsilon}\}$. The minimum is attained for

$$\begin{aligned} 1-p &= \frac{1+(2-\epsilon)p}{3-\epsilon} \\ 3-\epsilon-(3-\epsilon)p &= 1+(2-\epsilon)p \\ 2-\epsilon &= (5-2\epsilon)p \\ p &= \frac{2-\epsilon}{5-2\epsilon} \end{aligned}$$

The expected value for this p is

$$1-p = 1-\frac{2-\epsilon}{5-2\epsilon} = \frac{5-2\epsilon-2+\epsilon}{5-2\epsilon} = \frac{3-\epsilon}{5-2\epsilon} \overset{\epsilon\to 0}{\searrow} 0.6$$

Since ϵ can be chosen arbitrarily small, we have a lower bound of $\frac{5}{3}$ on the competitivity of every randomized algorithm. □

3.6.2 Deterministic Algorithms

First of all, we have a look at the Greedy algorithm. Because of the non-increasing job lengths it cannot be fooled as easily as before, since it is not possible anymore, to block jobs which have a higher profit.

Theorem 3.12 *On every sequence σ of jobs with non-increasing job lengths, the Greedy algorithm is 3-competitive.*

Proof. This can be seen by comparing a schedule obtained by Greedy with an optimal schedule. As every job accepted by Greedy can block at most three jobs and since these cannot yield a higher profit, the ratio of 3 follows. □

We try to find a better algorithm than greedy, which means that we want the competitivity to be less than 3. Therefore, we consider the Algorithm 1 which accepts a job as long as for a time period not longer than a fraction of c of its running time, there are more than half of the machines occupied. In the following, we derive lower bounds for different values of c.

Lemma 3.13 GREEDY $-$ c *is at least $2+c$-competitive.*

Proof. Consider the sequence $\sigma = \{j_1, \ldots, j_{5m}\}$ with

- $a_{j_i} = 1$,$b_{j_i} = 2$ for $i = 1, \ldots m$,

Algorithm 1 Greedy-c

Input: a sequence $\sigma = (j_1, \ldots, j_n)$, c with $0 \leq c \leq 1$
Output: a subset $J \subseteq \{j_1, \ldots, j_n\}$
$J := \emptyset$
for $i := 1$ to n **do**
  Let $I := \{t \in [a_{j_i}, b_{j_i}] : |\{k \in J : a_k \leq t \leq b_k\}| \geq m/2\}$
  if $|I| \leq c \cdot (b_{j_i} - a_{j_i})$ **then**
    $J := J \cup \{j_i\}$
  end if
end for
Return J

- $a_{j_i} = c$,$b_{j_i} = 1 + c$ for $i = m + 1, \ldots 2m$,
- $a_{j_i} = \epsilon$,$b_{j_i} = 1 + \epsilon$ for $i = 2m + 1, \ldots 3m$,
- $a_{j_i} = 1 + c - \epsilon$,$b_{j_i} = 2 + c - \epsilon$ for $i = 3m + 1, \ldots 4m$,
- $a_{j_i} = 1 + \epsilon$,$b_{j_i} = 1 + c - \epsilon$ for $i = 4m + 1, \ldots 5m$,

The optimal algorithm chooses $j_{2m+1}, \ldots, j_{5m}$ whereas GREEDY $-$ c accepts $j_1, \ldots, j_{\lceil m/2 \rceil}$ and $j_{m+1}, \ldots, j_{m+\lfloor m/2 \rfloor}$ as one can see in Figure 3.9.

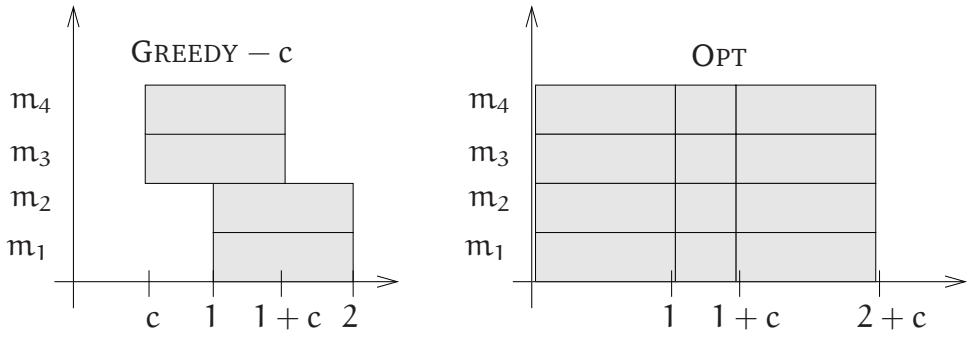

Figure 3.9: Solution of GREEDY $-$ c resp. OPT

Thus, the competitivity on σ is

$$\frac{\text{OPT}}{\text{GREEDY} - c} = \frac{m \cdot (1 + c - 2\epsilon + 1)}{m} = 2 + c - 2\epsilon.$$

Since ϵ can be chosen arbitrarily small, the competitivity cannot be better than $2 + c$. □

Lemma 3.14 *For* $c \leq \frac{1}{2}$ GREEDY $- c$ *is not better than* $6 - 4c$*-competitive.*

Proof. Consider m to be even and the sequence $\sigma = \{j_1, \ldots, j_{4m}\}$ with

- $a_{j_i} = 1$,$b_{j_i} = 2$ for $i = 1, \ldots m$,
- $a_{j_i} = c + \epsilon$,$b_{j_i} = 1 + c$ for $i = m + 1, \ldots 2m$,
- $a_{j_i} = 2 - c$,$b_{j_i} = 3 - c - \epsilon$ for $i = 2m + 1, \ldots 3m$,
- $a_{j_i} = 1 + c$,$b_{j_i} = 2 - c$ for $i = 3m + 1, \ldots 4m$.

The optimal algorithm chooses $j_{m+1}, \ldots, j_{4m}$ whereas GREEDY $- \frac{1}{2}$ accepts $j_1, \ldots, j_{m/2}$ as one can see in Figure 3.10.

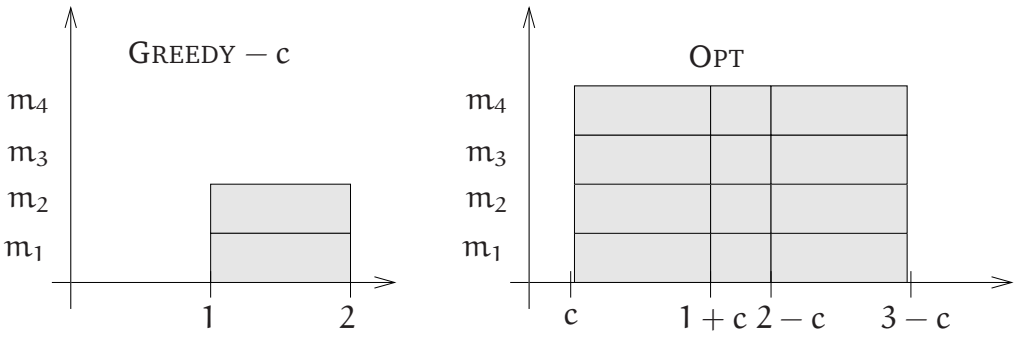

Figure 3.10: Solution of GREEDY $- c$ resp. OPT

Thus the competitivity on σ is

$$\frac{\text{OPT}}{\text{GREEDY} - c} = \frac{m \cdot (3 - 2c - 2\epsilon)}{m/2} = 6 - 4c - 4\epsilon$$

for ϵ arbitrarily small. □

Lemma 3.15 *For* $c > \frac{1}{2}$ GREEDY $- c$ *is not better than* $2/c$*-competitive.*

Proof. Consider m to be even and the sequence $\sigma = \{j_1, \ldots, j_{3m}\}$ with

- $a_{j_i} = 1$,$b_{j_i} = 2$ for $i = 1, \ldots m$,
- $a_{j_i} = 1.5 - 0.5/c + \epsilon$,$b_{j_i} = 1.5$ for $i = m + 1, \ldots 2m$,
- $a_{j_i} = 1.5$,$b_{j_i} = 1.5 + 0.5/c - \epsilon$ for $i = 2m + 1, \ldots 3m$,

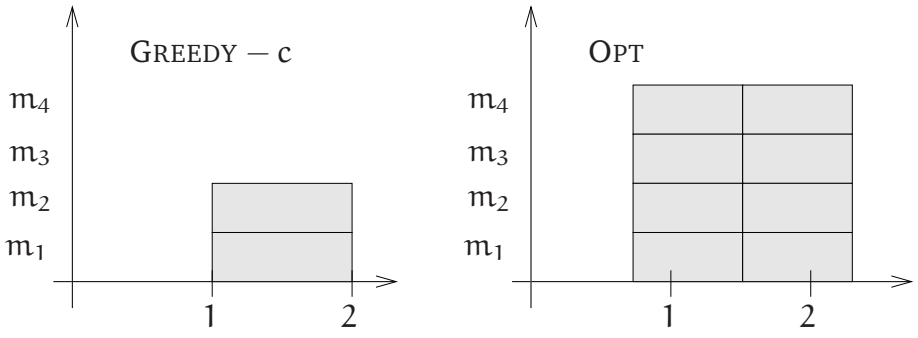

Figure 3.11: Solution of GREEDY − c resp. OPT

The optimal algorithm chooses $j_{m+1}, \ldots, j_{3m}$ whereas GREEDY $-\frac{1}{2}$ accepts $j_1, \ldots, j_{m/2}$ as one can see in Figure 3.11.

Thus the competitivity on σ is

$$\frac{\text{OPT}}{\text{GREEDY} - c} = \frac{m \cdot (0.5/c + 0.5/c - 2\epsilon}{m/2} = 2/c - 4\epsilon$$

for ϵ arbitrarily small. □

The minimum possible competitivity ratios derived from the three preceding lemmas are plotted in Figure 3.12. One can see that GREEDY − c is worse than GREEDY for $c < 2/3$ and cannot have a performance ratio better than $1 + \sqrt{3} > 2,73$ in general.

3.7 Experimental Results

In order to get a feeling for the practical quality of the algorithms presented, we implemented them and tested their competitivity for different numbers of jobs n, machines m, machine times T and maximal job lengths max_l. The jobs' start- and end-times were chosen by first sampling the length l of each job uniformly from the integers between 1 and max_l and choosing the start time uniformly from the integers between 0 and $T - l$ afterwards.

In Table 3.1 we present the simulation results with each row representing 1000 randomly generated instances. The left part states the parameter setting, whilst in the right half the average-case competitivities achieved by GREEDY , CRS-GREEDY , C-GREEDY and GREEDY on semi-online instances are given with the corresponding sample standard deviations.

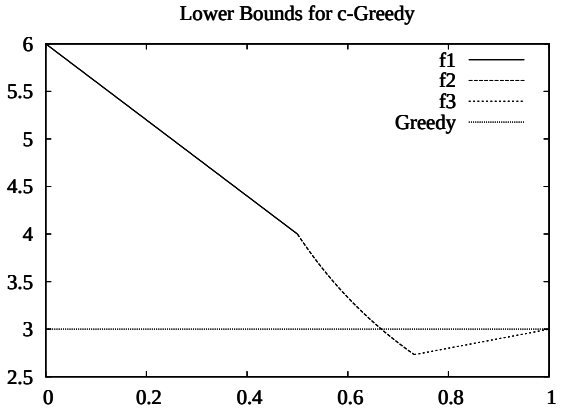

Figure 3.12: Minimum possible competitivity for GREEDY − c compared to GREEDY

Iter.	n	m	T	max_l	c_G[%]	$\mathbb{E}[c_{CRS}]$ [%]	c_{class}[%]	c_{Gs}[%]
1000	50	2	64	10	74.3 ± 5.7	24.5 ± 1.7	69.4 ± 6.8	89.9 ± 3.9
1000	50	2	64	30	73 ± 8	27.9 ± 2.5	60.1 ± 7.6	91 ± 4.5
1000	50	2	64	50	75.1 ± 9.5	34.9 ± 3	55 ± 8.1	88.2 ± 4.9
1000	50	2	64	64	74.5 ± 11	32.9 ± 3.4	51.6 ± 8.8	99.6 ± 1
1000	50	5	64	10	76.4 ± 4.8	18.5 ± 0.9	40.7 ± 4.4	93.6 ± 2.8
1000	50	5	64	30	69.5 ± 5.6	22.5 ± 1.6	46.3 ± 5.1	91.1 ± 3.3
1000	50	5	64	50	73.1 ± 7.1	29.5 ± 2	43.5 ± 4.8	89.3 ± 3.6
1000	50	5	64	64	72.8 ± 8.5	27.5 ± 2.3	40.6 ± 5.4	98.9 ± 1.3
1000	50	10	64	10	93.7 ± 4	16.7 ± 0.1	40.2 ± 4.7	99.6 ± 0.9
1000	50	10	64	30	68.5 ± 4.9	18.2 ± 1.2	22.7 ± 2.5	91.7 ± 2.4
1000	50	10	64	50	71.6 ± 6.1	26.2 ± 1.4	26.1 ± 2.9	91.6 ± 2.9
1000	50	10	64	64	71.2 ± 7.1	23.8 ± 1.6	23.4 ± 2.9	97.7 ± 1.6
1000	50	15	64	10	99.6 ± 1.3	16.7 ± 0	64.1 ± 5.9	100 ± 0
1000	50	15	64	30	70.9 ± 4.6	16.6 ± 0.9	30.2 ± 3.3	92.7 ± 2.3
1000	50	15	64	50	71.5 ± 5.5	24.2 ± 1.3	33 ± 2.8	93.4 ± 2.3
1000	50	15	64	64	71.3 ± 6.1	21.8 ± 1.6	28.9 ± 3	97.1 ± 1.6
1000	100	2	64	10	75.7 ± 4.6	29.6 ± 1.6	71.9 ± 5.4	89.4 ± 3
1000	100	2	64	30	76.3 ± 6.7	33 ± 2.3	67.2 ± 6.2	91.9 ± 3.2
1000	100	2	64	50	77.3 ± 7.6	39.2 ± 2.7	62 ± 7.4	88.8 ± 4.8
1000	100	2	64	64	78.5 ± 8.9	38.5 ± 3	59.6 ± 7.6	99.7 ± 0.6
1000	100	5	64	10	70.4 ± 3.6	22.6 ± 0.9	38.3 ± 2.7	88.5 ± 2.1
1000	100	5	64	30	72 ± 5	26.8 ± 1.4	51.1 ± 5	92 ± 2.4
1000	100	5	64	50	74.3 ± 5.8	33.7 ± 1.8	47.9 ± 4.4	88.2 ± 3.3
1000	100	5	64	64	75.9 ± 7.3	32.5 ± 2.1	46.3 ± 5.2	99.4 ± 0.7

Iter.	n	m	T	$\max_l$	c_G[%]	$\mathbb{E}[c_{CRS}]$ [%]	c_{class}[%]	c_{Gs}[%]
1000	100	10	64	10	73.2 ± 3.2	17.7 ± 0.5	27.6 ± 2.3	92.7 ± 2.3
1000	100	10	64	30	68.3 ± 4	22.4 ± 1.1	23.9 ± 2.1	92 ± 1.8
1000	100	10	64	50	71.7 ± 4.9	29.2 ± 1.4	27.7 ± 2.4	88.4 ± 2.5
1000	100	10	64	64	73.4 ± 5.7	27.7 ± 1.6	26.1 ± 2.6	98.6 ± 1
1000	100	15	64	10	85.8 ± 3.4	16.7 ± 0.1	43.4 ± 3.4	98.7 ± 1.4
1000	100	15	64	30	66.7 ± 3.5	19.7 ± 1	29.5 ± 2	91.7 ± 1.6
1000	100	15	64	50	70.6 ± 4.4	26.9 ± 1.1	34.6 ± 2.4	89.2 ± 2.2
1000	100	15	64	64	72 ± 5.1	25.3 ± 1.2	32 ± 2.7	97.9 ± 1.2
1000	50	2	128	10	79.3 ± 5.2	17.7 ± 1.1	45.3 ± 4.2	93.9 ± 3.2
1000	50	2	128	30	71.9 ± 7.1	20.1 ± 1.6	63.2 ± 6.6	89 ± 4.4
1000	50	2	128	50	71.6 ± 7.9	25.2 ± 2	56.3 ± 7.2	89.5 ± 4.8
1000	50	2	128	70	72.3 ± 8.9	30.7 ± 2.5	52.5 ± 7.5	88.8 ± 6.3
1000	50	2	128	90	73 ± 9.1	30.2 ± 2.5	49.9 ± 7.4	87.1 ± 6.2
1000	50	5	128	10	91.7 ± 3.9	14.5 ± 0.3	54.7 ± 5.4	98.8 ± 1.4
1000	50	5	128	30	69.2 ± 5.1	15.5 ± 1.1	37.4 ± 3.9	89.5 ± 3
1000	50	5	128	50	68.5 ± 5.7	20.4 ± 1.3	37.2 ± 6.1	90.1 ± 3.3
1000	50	5	128	70	69.4 ± 6.4	24.6 ± 2.5	40.1 ± 5.2	89.8 ± 5
1000	50	5	128	90	70.9 ± 7	25.8 ± 1.7	38.3 ± 4.7	87.4 ± 4.3
1000	50	10	128	10	99.9 ± 0.7	14.3 ± 0	57.6 ± 5.8	100 ± 0
1000	50	10	128	30	77.8 ± 4.9	14 ± 0.5	26.1 ± 3.4	95.7 ± 2.7
1000	50	10	128	50	68.2 ± 4.8	16.3 ± 0.9	25 ± 2.8	90.7 ± 2.4
1000	50	10	128	70	68 ± 5.4	18.6 ± 1.7	28 ± 2.8	91.7 ± 3.3
1000	50	10	128	90	70.1 ± 5.7	22.9 ± 1.2	26.9 ± 2.8	90 ± 3.2
1000	50	15	128	10	100 ± 0	14.3 ± 0	83.4 ± 5.4	100 ± 0
1000	50	15	128	30	92.7 ± 4.2	14.2 ± 0.2	42.1 ± 5.3	99.8 ± 0.8
1000	50	15	128	50	73.7 ± 4.5	14.6 ± 0.5	34.9 ± 3.8	93.6 ± 2.8
1000	50	15	128	70	69.3 ± 4.7	16.1 ± 1.2	36.9 ± 3.3	93.6 ± 2.1
1000	50	15	128	90	70.7 ± 5.3	20.7 ± 1.3	34.6 ± 2.9	93.4 ± 2.5
1000	100	2	128	10	74.1 ± 4.1	21 ± 1	40.1 ± 2.8	89.7 ± 2.7
1000	100	2	128	30	72.8 ± 5.7	23.8 ± 1.5	67.7 ± 5.7	89.4 ± 3.4
1000	100	2	128	50	74.2 ± 6.3	29.6 ± 1.9	63 ± 5.9	90.4 ± 4
1000	100	2	128	70	74.9 ± 7.2	34.9 ± 2.1	59.5 ± 6.5	90.9 ± 4.1
1000	100	2	128	90	75.9 ± 7.8	34.2 ± 2.3	56.8 ± 6.7	89.2 ± 5.1
1000	100	5	128	10	76.5 ± 3.4	15.7 ± 0.5	42.1 ± 3.1	93.6 ± 2
1000	100	5	128	30	68.3 ± 4.1	18.5 ± 0.9	38.4 ± 3	88.1 ± 2.2
1000	100	5	128	50	70.1 ± 4.7	24.2 ± 1.2	42.6 ± 5	90.2 ± 2.7
1000	100	5	128	70	71.4 ± 5.4	29.8 ± 1.5	44.3 ± 5.1	91.1 ± 3.4
1000	100	5	128	90	72.5 ± 5.8	29.3 ± 1.6	41.8 ± 4.4	87.6 ± 4.1
1000	100	10	128	10	94.1 ± 2.8	14.3 ± 0.1	40.5 ± 3.4	99.6 ± 0.6
1000	100	10	128	30	66 ± 3.3	14.8 ± 0.7	21.8 ± 1.8	87.4 ± 1.9
1000	100	10	128	50	66.7 ± 3.7	20.2 ± 0.9	25.2 ± 2.1	90 ± 2
1000	100	10	128	70	68.1 ± 4.4	24.8 ± 1.8	28.8 ± 2.2	91.1 ± 3

Iter.	n	m	T	$\max_l$	c_G[%]	$\mathbb{E}[c_{CRS}]$ [%]	c_{class}[%]	c_{Gs}[%]
1000	100	10	128	90	70 ± 4.7	25.5 ± 1.2	28.1 ± 2.3	87.2 ± 3.1
1000	100	15	128	10	99.7 ± 0.8	14.3 ± 0	64.6 ± 4.3	100 ± 0
1000	100	15	128	30	68.9 ± 3.1	13.6 ± 0.5	29.2 ± 2.4	90.4 ± 2.4
1000	100	15	128	50	65.8 ± 3.4	17.8 ± 0.7	31.6 ± 2.1	89.4 ± 1.6
1000	100	15	128	70	66.8 ± 3.9	20.6 ± 1.6	36.1 ± 2.3	91.4 ± 2.7
1000	100	15	128	90	68.8 ± 4.3	23.4 ± 0.9	35.1 ± 2.3	87.6 ± 2.6

Table 3.1: Competitivity of the given algorithms under uniformly distributed job lengths

Regardless of the different parameter settings, the greedy algorithm performed best. It achieved on average a competitivity of 74.76 %, while the expected value of CRS-GREEDY was 23.12 % and C-GREEDY could make 42.68 % of the optimal solutions' profit. The last column shows the effect of ordering the jobs within the sequences in non-increasing order of their size. Due to Theorem 3.12 we know that GREEDY is 3-competitive on instances of that kind. In fact, its average case competitiveness seems to be much better than that, as it achieved on average 92.49 % of the profit of the optimal solutions. Although, we have seen that its worst case behavior is poor, we have to admit that it seems to be a good choice for practice.

3.8 Preliminary Conclusions

Going back to our initial story, we have discovered organizing the bookings for cottages is not a trivial task, at least not, if booking requests arrive online. The competitive algorithms presented in this chapter all work by classifying customers according to the length of the desired booking interval and then treating each "customer class" separately. In fact, the lower bounds tell us that such a classification makes sense if we want to handle worst case scenarios.

So, if your holiday location works in a competitive way, the initial suspicion that our request was rejected just because we asked for three days instead of four may be justified. On the other hand the theoretical lower bounds are somewhat discouraging. Even randomization does not help.

Even more discouraging are the simulation results presented in Section 3.7. They show that the proposed algorithms do not provide a good capacity utilization in the average case, whereas the T-competitive greedy algorithm seems to reveal an average-case competitiveness of circa 75 %. As a matter of fact, they

show that despite the worst case behavior we should focus on the average-case as well.

3.9 Average Case Analysis

In this section, we analyze the expected average-case performance of the algorithms presented in the preceding sections and we compare them. We restrict ourselves to two special cases. In Subsection 3.9.1 we take a look at instances where all jobs start at the same time. We will see that under the given distribution the average-case performance of the algorithms differs completely from their worst case behavior, which matches the simulation results presented in Section 3.7. As these results confirm the suspicion that the greedy algorithm is a good choice, we examine its expected performance for one machine in Subsection 3.9.2 more closely and compute bounds for up to 15 jobs. Observe that these expected values also imply bounds for instances with n jobs and $m > 1$ machines, as their expected load is at least as high as the expected load of greedy when processing $\lfloor n/m \rfloor$ jobs on one machine.

3.9.1 Equal Start Times

We restrict ourselves on instances where all the jobs are starting at time 0. This means that exactly one job can be scheduled on each machine. The job lengths X_i are uniformly distributed random variables between 0 and T. Since the greedy algorithm is always optimal for instances with $n \leq m$ we only consider instances with $n > m$.

Definition 3.16 If $X_1, ..., X_n$ is a random sample from an absolutely continuous population. $X_{1:n} \leq X_{2:n} \leq \cdots \leq X_{n:n}$ is called the *order statistic* obtained by arranging the preceding random sample in increasing order of magnitude.

Theorem 3.17 *If $X_1, ..., X_n$ is an i.i.d. sample from the uniform distribution on the interval $[0, T]$, the expected value of $X_{k:n}$ is*

$$\mathbb{E}[X_{k:n}] = \frac{k \cdot T}{n+1}$$

Proof. see [ABN92] □

Due to the fact that the optimal algorithm chooses the jobs corresponding to $X_{n-m+1:n}, \ldots, X_{n:n}$, the expected profit of the optimal algorithm is:

$$\mathbb{E}[\textsc{Opt}] = \mathbb{E}\left[\sum_{i=n-m+1}^{n} X_{i:n}\right] = \sum_{i=n-m+1}^{n} \mathbb{E}[X_{i:n}] = \sum_{i=n-m+1}^{n} \frac{i \cdot T}{n+1}$$
$$= \frac{T}{n+1} \sum_{i=n-m+1}^{n} i = \frac{T}{n+1} \cdot m \cdot \frac{2n-m+1}{2} = \frac{mT(2n-m+1)}{2(n+1)}$$

The profit of the greedy algorithm is:

$$\mathbb{E}[\textsc{Greedy}] = \mathbb{E}\left[\sum_{i=1}^{m} X_i\right] = \sum_{i=1}^{m} \mathbb{E}[X_i] = \sum_{i=1}^{m} \frac{T}{2} = \frac{mT}{2}$$

For the expected profit achieved by

$$\frac{\mathbb{E}[\textsc{Opt}]}{\mathbb{E}[\textsc{Greedy}]} = \frac{\frac{mT(2n-m+1)}{2(n+1)}}{\frac{mT}{2}} = \frac{2n-m+1}{n+1} = 1 + \frac{n-m}{n+1}$$
$$= 1 + \frac{n+1-1-m}{n+1} = 2 - \frac{m+1}{n+1} \overset{n>m}{<} 2 - \alpha^{-1}$$

with $\alpha := n/m$ being the average load on each machine.

Consider the algorithm C-GREEDYpresented in Section 3.5.3. Recall that this algorithm reserves $\frac{m}{\log T}$ machines for each of the classes of jobs $J_1, \ldots, J_{\log T}$ with $j \in J_i$ if and only if $2^{i-1} \leq b_j - a_j < 2^i$. Since the job lengths are uniformly distributed, the expected length of a job in class J_i is $\frac{2^{i-1}+2^i}{2} = 3 \cdot 2^{i-2}$. We can derive an upper bound on the profit of C-GREEDYby assuming that after processing the sequence all machines are occupied. This means that for each class J_i we have $\frac{m}{\log T}$ machines processing jobs with an expected length of $3 \cdot 2^{i-2}$. So, the expected profit achieved by this algorithm can be estimated by

$$\mathbb{E}[\text{C-GREEDY}] \leq \sum_{j=1}^{\log T} \left(\frac{m}{\log T} \cdot 3 \cdot 2^{j-2}\right) = \frac{3m}{2 \log T} \cdot \sum_{j=0}^{\log T - 1} 2^j$$
$$= \frac{3m}{2 \log T} \cdot (2^{\log T} - 1) = \frac{3m}{2 \log T} \cdot (T - 1)$$

This means that the comparison of C-GREEDYand GREEDY reads as follows:

$$\frac{\mathbb{E}\,[\text{GREEDY}\,]}{\mathbb{E}\,[\text{C-GREEDY}]} \geq \frac{m \cdot (1+\frac{T}{2})}{\frac{3m}{2\log T} \cdot (T-1)} = \frac{2\log T(1+\frac{T}{2})}{3 \cdot (T-1)}$$
$$= \frac{\log T(2+T)}{3 \cdot (T-1)} > \frac{\log T}{3}$$

When only considering instances with all jobs starting at the same time, we see that the average profit obtained by C-GREEDYis much worse than the average profit obtained by GREEDY even though it works much better in the worst case.

3.9.2 The Average Case Performance of Greedy

We consider the case of one machine with operating time 1 and assume that the job lengths l_i for all jobs $i = 1, \ldots n$ are uniformly distributed between 0 and 1. As the operating time of the machine is 1 and thus $[a_i, b_i] \subseteq [0, 1]$ for all $i = 1, \ldots, n$, a_i has to be within $[0, 1 - l_i]$. We chose a_i uniformly from this interval.

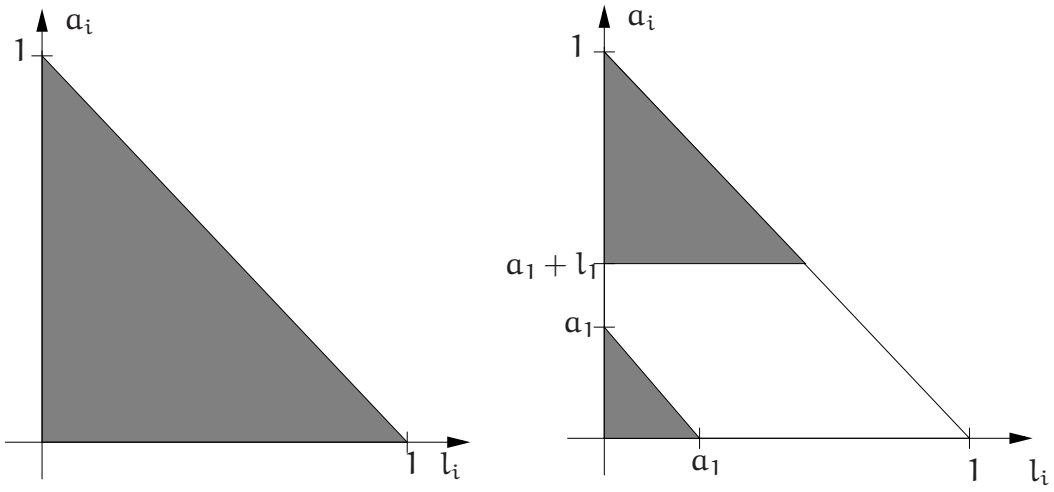

Figure 3.13: Set of possible jobs for $i = 1$ and for $i = 2$

After the first job has been accepted, there is only room remaining for additional jobs in the intervals $[0, a_1]$ and $[b_1, 1]$. In Figure 3.13 there is an illustration of the feasible jobs for $i = 1, 2$. Here, every possible job corresponds to a point in the triangle with side length 1. The regions of this triangle which correspond to jobs which fit into a current schedule are shaded. At the beginning, there is no job in the schedule which might block future jobs, which is the reason that all jobs are admissible and thus the whole triangle is shaded. After the acceptance of the first job r_1, the feasible region reduces to two smaller triangles. Observe that in the case of $a_1 = 0$ or $a_1 + l_1 = 1$ one of the remaining triangles has side length 0. Analogously, after having accepted k jobs the set of feasible solutions corresponds to up to $k + 1$ shaded triangles (see for example Figure 3.14).

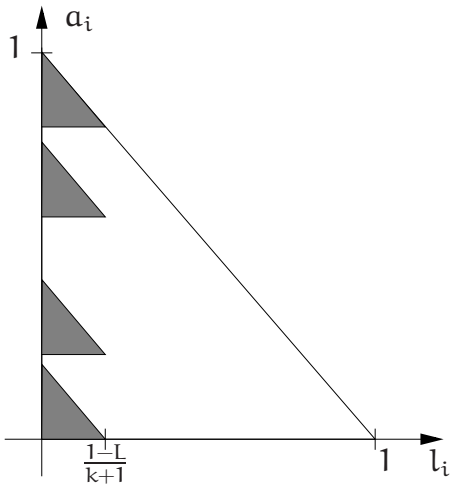

Figure 3.14: Set of admissible jobs after having accepted k jobs with total length L

In the following, we focus on one of those small triangles which emerge during the execution of greedy, and we compute the probability that future requests which correspond to points in this triangle do fit into the schedule after the acceptance of the first of these requests (which is admitted, as our algorithm is greedy). This probability will be used to state a formulation of the expected value of all these requests, which come up and can be scheduled in this triangle. As a special case, we derive the expected value for the whole interval $[0, 1]$.

Let the triangle we choose from have side length b and let (a, l) be the start point and the length of the first job. Like mentioned above, after the acceptance of this job, there are two new triangles describing the requests which do not

block this first request. Let p_1 and p_2 be the probabilities that a job chosen in the future is within the triangle below a respectively the triangle above $a + l$.

$$\begin{aligned} p_1 &= \int_0^a \frac{a-x}{b-x}dx = \int_0^a 1 - \frac{b-a}{b-x}dx = [x + (b-a)\ln(b-x)]_0^a \\ &= a + (b-a)(\ln(b-a) - \ln(b)) = a + (b-a)\ln(1-\frac{a}{b}) \end{aligned}$$

$$\begin{aligned} p_2 &= \int_0^{b-a-l} \frac{b-a-l-x}{b-x}dx = \int_0^{b-a-l} 1 - \frac{a+l}{b-x}dx \\ &= [x + (a+l)\ln(b-x)]_0^{b-a-l} = b - a - l + (a+l)(\ln(a+l) - \ln(b)) \end{aligned}$$

Since a is chosen uniformly between 0 and $b - l$, these probabilities itself are random variables with density functions $p_1(x, l)$ and $p_2(x, l)$.

$$\begin{aligned} p_1(x,l) &= a + (b-a)(\ln(b-a) - \ln(b)) \\ &= (b-l)x + (b-(b-l)x)(\ln(b-(b-l)x) - \ln(b)) \qquad \forall x, l \in [0,1] \end{aligned}$$

$$p_2(x,l) = 1 - (1-l)x - l + ((1-l)x + l)\ln((1-l)x + l) \qquad \forall x, l \in [0,1]$$

Due to the way the start- and endpoints are sampled and for algebraic reasons we know that

$$p_1(x,l) = p_2(1-x,l) \qquad \forall x, l \in [0,1]. \tag{3.6}$$

Let $E(n, b)$ be the expected value of the greedy algorithm when processing n requests in a triangle with side length b. In the following, we abbreviate $p_1(x, l)$ and $p_2(x, l)$ by p_1 resp. p_2.

Lemma 3.18 *For all* $l \in (0, 1)$ *we have* $E(i, l) = E(i, 1) \cdot l$

Proof. When regarding a triangle with side length l instead of 1, all lengths and starting-points are scaled down by a factor of l. For this reason, the expected value is just reduced by the same factor. □

$$
\begin{aligned}
E(n,1) =& \frac{1}{2} + \int_0^1 \int_0^1 \sum_{i,j:i+j<n} \left(\binom{n-1}{i} \binom{n-1-i}{j} p_1^i \cdot p_2^j \cdot (1-p_1-p_2)^{n-1-i-j} \right. \\
& \left. \cdot (E(i,(1-l)\cdot x) + E(j,(1-l)\cdot x + l)) \right) dx\, dl \\
=& \frac{1}{2} + \int_0^1 \int_0^1 \sum_{i=0}^{n-1} \left(\binom{n-1}{i} p_1^i \cdot E(i,(1-l)\cdot x) \right. \\
& \left. \cdot \sum_{j=0}^{n-1-i} \left(\binom{n-1-i}{j} \cdot p_2^j \cdot (1-p_1-p_2)^{n-1-i-j} \right) \right) dx\, dl \\
& + \int_0^1 \int_0^1 \sum_{j=0}^{n-1} \left(\binom{n-1}{j} p_2^j \cdot E(j,(1-l)\cdot x + l) \right. \\
& \left. \cdot \sum_{i=0}^{n-1-j} \binom{n-1-j}{i} p_1^i \cdot (1-p_1-p_2)^{n-1-i-j} \right) dx\, dl \\
=& \frac{1}{2} + \int_0^1 \int_0^1 \sum_{i=0}^{n-1} \left(\binom{n-1}{i} p_1^i (1-p_1)^{n-1-i} \cdot E(i,(1-l)\cdot x) \right) dx\, dl \qquad (3.7) \\
& + \int_0^1 \int_0^1 \sum_{j=0}^{n-1} \left(\binom{n-1}{j} p_2^j \cdot (1-p_2)^{n-1-j} \cdot E(j,(1-l)\cdot x + l) \right) dx\, dl \qquad (3.8)
\end{aligned}
$$

The integrand in (3.7) gives the value of the expected load which will be achieved in the lower triangle for given l and x. Thus, the integral gives the expected load within this triangle. Analogously, (3.8) gives the expected load within the upper triangle. Due to the way start- and endpoints are sampled these expected values coincide. This is also implied by Equation (3.6). The integral in (3.7) can be simplified by

$$\int_0^1 \int_0^1 \sum_{i=0}^{n-1} \left(\binom{n-1}{i} p_1^i (1-p_1)^{n-1-i} \cdot E(i,(1-l)\cdot x) \right) dx\, dl$$

$$\overset{\text{Lem.3.18}}{=} \int_0^1 \int_0^1 \sum_{i=0}^{n-1} \left(\binom{n-1}{i} p_1^i (1-p_1)^{n-1-i} \cdot (1-l)\cdot x \cdot E(i,1) \right) dx\, dl$$

$$= \sum_{i=0}^{n-1} \left(\binom{n-1}{i} E(i,1) \int_0^1 \int_0^1 p_1^i (1-p_1)^{n-1-i} \cdot (1-l)\cdot x dx\, dl \right)$$

$$= \sum_{i=0}^{n-1} (\binom{n-1}{i} E(i,1) \int_0^1 \int_0^1 ((1-l)x + (1-(1-l)x)\ln(1-(1-l)x))^i$$

$$(1-((1-l)x + (1-(1-l)x)\ln(1-(1-l)x)))^{n-1-i} \cdot (1-l)\cdot x dx\, dl)$$

$$\overset{a:=(1-l)x}{=} \sum_{i=0}^{n-1} (\binom{n-1}{i} E(i,1)$$

$$\underbrace{\int_0^1 \int_0^1 (a+(1-a)\ln(1-a))^i (1-(a+(1-a)\ln(1-a)))^{n-1-i} \cdot \frac{a}{1-l}\, da\, dl)}_{=:h(i,n)}$$

Thus, we derive a recursive formulation for $n > 0$ by

$$E(n,1) = \frac{1}{2} + 2 \cdot \sum_{i=0}^{n-1} \left(E(i,1) \binom{n-1}{i} \cdot h(i,n) \right).$$

For $n \in \{0,\ldots 15\}$, this gives the expected load as displayed in Table 3.2. As the optimal algorithm cannot attain a higher load than 1, we can also use $E(n,1)$ as a lower bound for the expected competitivity of the greedy algorithm. In order to check the quality of this bound, we have obtained approximate average competitive ratios by sampling instances according to the given distribution and computing the loads achieved by the optimal algorithm and the greedy algorithm. Every row in Table 3.2 corresponds to 1000 samples. Observe that the expected values $E(n,1)$ approximate the simulation results quite well.

Moreover, these expected values also imply bounds for instances with n jobs and $m > 1$ machines, as their expected load is at least as high as the expected load of greedy when processing $\lfloor n/m \rfloor$ jobs on one machine.

n	$E(n,1)$	load derived by Simulation
0	0.000	1
1	0.500	0.807
2	0.543	0.722
3	0.572	0.706
4	0.593	0.689
5	0.611	0.713
6	0.625	0.689
7	0.636	0.689
8	0.647	0.689
9	0.656	0.693
10	0.663	0.686
11	0.670	0.705
12	0.677	0.697
13	0.683	0.709
14	0.688	0.716
15	0.693	0.704

Table 3.2: Expected competitivity of greedy algorithm on one machine under uniformly distributed job length

3.9.3 An Algorithm which Performs Good in the Average Case if the Number of Jobs is known Beforehand

Even though we have seen in the preceding subsections that the greedy algorithm reveals a good average-case behavior, it does not seem to be clever to accept a low-profit job when it is reasonable that plenty of even better jobs are about to come up. If the algorithm expects a high number n of jobs to arrive, it makes sense, only to accept high profit requests, whereas a lower threshold is more suitable if the number of unpublished jobs is low, too. Hence, if n is known beforehand, the greedy algorithm should accept only jobs which incur at least a certain profit of α, for some properly chosen α.

In this section, we analyze the average-case performance of this modified greedy algorithm which we call GREEDY$-\alpha$ and propose such a threshold which depends on n and is close to optimal. Thereby, we assume the same distribution on the lengths and starting-points of the jobs as in Subsection 3.9.2.

Theorem 3.19 (Chernoff-Bound) *Let $X_1, X_2, \ldots, X_n$ be a sequence of n independent Bernoulli-trials with $\Pr[X_i = 1] = p$ and $\Pr[X_i = 0] = 1 - p$. Then pn is the expected number of successes $(X_i = 1)$ of the experiment.*

For all $\delta \in [0, 1]$

$$\Pr\left[\sum X_i \leq (1-\delta) \cdot pn\right] \leq \exp\left(-\frac{\delta^2}{2} pn\right) \tag{3.9}$$

Proof. See for example [MU05] □

We consider the greedy algorithm which skips all jobs of length $< \alpha$ and schedules all the other jobs greedily. So, every job accepted, has an expected length of $\alpha + \frac{(1-\alpha)}{2} = \frac{(1+\alpha)}{2}$.

Lemma 3.20 *For $\alpha \in [0, \frac{n-m+1}{n}]$ we can estimate the expected profit of* GREEDY$-\alpha$ *by:*

$$\mathbb{E}\left[\text{GREEDY} - \alpha\right] \geq \left(1 - \exp\left(-\frac{((1-\alpha)n - m + 1)^2}{2(1-\alpha)n}\right)\right) \cdot m \cdot \frac{(1+\alpha)}{2}$$

Proof. We want to find an upper bound on the probability that less than m jobs have the desired length of at least α and are accepted. Observe that every set of m jobs can be scheduled at the same time as they can be put on different machines.

For $i = 1, \dots n$ let $Y_i = 1$ if the length of job i is greater or equal α and $Y_i = 0$ otherwise. Hence, $\Pr[Y_i = 1] = 1 - \alpha$ and $\mathbb{E}\left[\sum_{i=1}^{n} Y_i\right] = n(1-\alpha)$.

For

$$\delta := 1 - \frac{m-1}{(1-\alpha)n} \overset{m \geq 1}{\leq} 1$$

we can restate the probability that less than m jobs are accepted by

$$\Pr\left[\sum Y_i \leq m-1\right] = \Pr\left[\sum Y_i \leq (1-\delta)\cdot(1-\alpha)n\right]$$

As $\delta \geq 0$ iff $\alpha \leq 1 - \frac{m-1}{n}$, we can apply (3.9) for all $\alpha \leq 1 - \frac{m-1}{n}$

$$\begin{aligned}
\Pr\left[\sum Y_i \leq m-1\right] &= \Pr\left[\sum Y_i \leq (1-\delta)\cdot(1-\alpha)n\right] \\
&\leq \exp\left(-\frac{\left(1-\frac{m-1}{(1-\alpha)n}\right)^2}{2}(1-\alpha)n\right) \\
&= \exp\left(-\frac{\left(\frac{(1-\alpha)n-m+1}{(1-\alpha)n}\right)^2}{2}(1-\alpha)n\right) \\
&= \exp\left(-\frac{((1-\alpha)n-m+1)^2}{2(1-\alpha)n}\right)
\end{aligned}$$

We can estimate the expected profit by:

$$\begin{aligned}
\mathbb{E}[\textsc{Greedy}-\alpha] &\geq \left(1-\Pr\left[\sum Y_i \leq m-1\right]\right)\cdot m \cdot \frac{(1+\alpha)}{2} \\
&\geq \left(1-\exp\left(-\frac{((1-\alpha)n-m+1)^2}{2(1-\alpha)n}\right)\right)\cdot m\cdot\frac{(1+\alpha)}{2} \\
&= \left(1-\exp\left(-\frac{n}{2}+\frac{n}{2}\alpha+m-1-\frac{(m-1)^2}{2(1-\alpha)n}\right)\right)\cdot m\cdot\frac{(1+\alpha)}{2}
\end{aligned}$$

□

We choose α' in dependence of m and estimate its expected profit:

$$\alpha' := \frac{2}{n}\left(\ln\left(\frac{2n}{n^2-4(m-1)^2+2n}\right) - (m-1) + \frac{n}{2} + \frac{(m-1)^2}{2n}\right). \quad (3.10)$$

For this specific choice of α, we can estimate the terms from the expected values lower bound by:

$$\begin{aligned}
&\left(1-\exp\left(\ln\left(\frac{2n}{n^2-4(m-1)^2+2n}\right) + \frac{(m-1)^2}{2n} - \frac{(m-1)^2}{2(1-\alpha')n}\right)\right)\\
&\overset{\alpha'>0}{>} \left(1-\exp\left(\ln\left(\frac{2n}{n^2-4(m-1)^2+2n}\right) + \frac{(m-1)^2}{2n} - \frac{(m-1)^2}{2n}\right)\right)\\
&= \left(1-\frac{2n}{n^2-4(m-1)^2+2n}\right)
\end{aligned}$$

and

$$\begin{aligned}
\frac{(1+\alpha')}{2} &= \frac{(1+\frac{2}{n}\left(\ln\left(\frac{2n}{n^2-4(m-1)^2+2n}\right) - (m-1) + \frac{n}{2} + \frac{(m-1)^2}{2n}\right))}{2}\\
&= \frac{\left(1+\frac{2}{n}\ln\left(\frac{2n}{n^2-4(m-1)^2+2n}\right) - \frac{2}{n}(m-1) + 1 + \frac{2}{n}\frac{(m-1)^2}{2n}\right)}{2}\\
&= \left(1+\frac{1}{n}\ln\left(\frac{2n}{n^2-4(m-1)^2+2n}\right) - \frac{m-1}{n} + \frac{(m-1)^2}{2n^2}\right).
\end{aligned}$$

As the optimal algorithm cannot have a capacity utilization better than one on each machine we have $\mathbb{E}[\textsc{Opt}] \leq m$. Together with the estimations stated above, we can now estimate the ratio of the expected value of $\textsc{Greedy} - \alpha'$ as

$$\begin{aligned}
\frac{\mathbb{E}[\textsc{Greedy} - \alpha']}{\mathbb{E}[\textsc{Opt}]} \geq &\left(1-\frac{2n}{n^2-4(m-1)^2+2n}\right)\\
&\cdot\left(1+\frac{1}{n}\ln\left(\frac{2n}{n^2-4(m-1)^2+2n}\right) - \frac{m-1}{n} + \frac{(m-1)^2}{2n^2}\right)
\end{aligned}$$

Figures 3.15, 3.16 and 3.17 illustrate the quality of this bound. For higher numbers of jobs we see that the algorithm almost works as good as the optimal algorithm. This intuition is settled by the following lemma:

Lemma 3.21 *The expected competitivity of* GREEDY $-\ \alpha'$ *converges to one for* $n \to \infty$.

Proof.

$$\lim_{n\to\infty} \frac{\mathbb{E}\left[\text{GREEDY} - \alpha'\right]}{\mathbb{E}\left[\text{OPT}\right]} \geq (1-0)\cdot(1+0-0+0) = 1$$

□

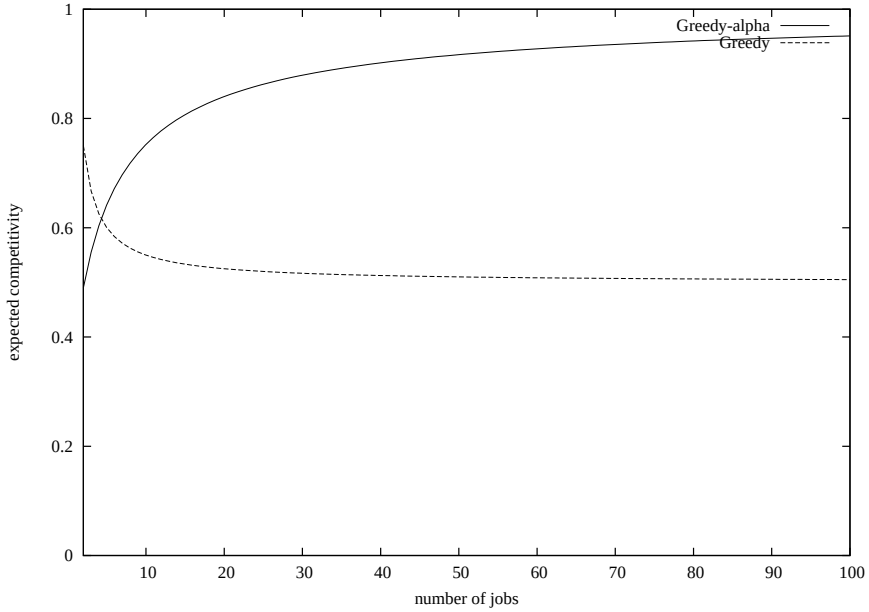

Figure 3.15: Lower bound for expected competitivities of Greedy-α' and Greedy on one machine

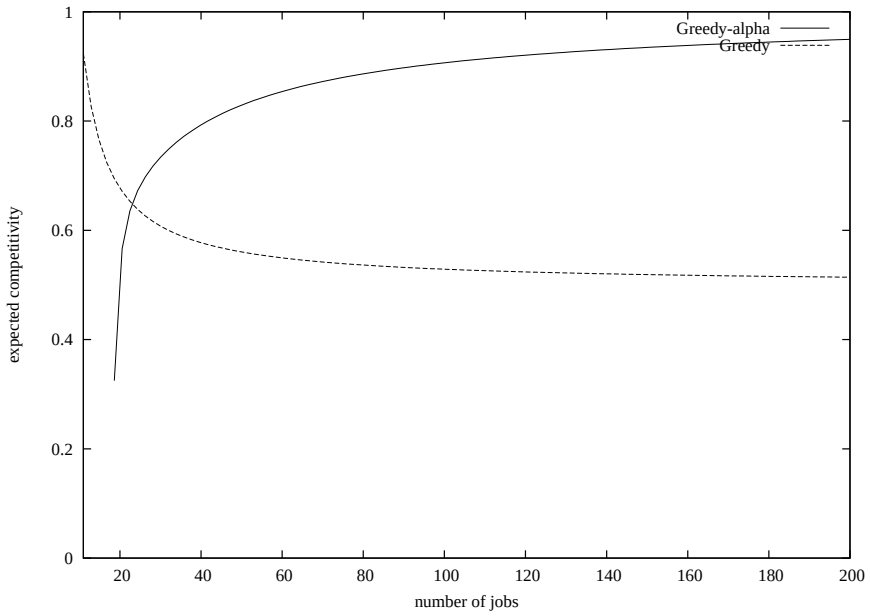

Figure 3.16: Lower bound for expected competitivities of Greedy-α' and Greedy on ten machines

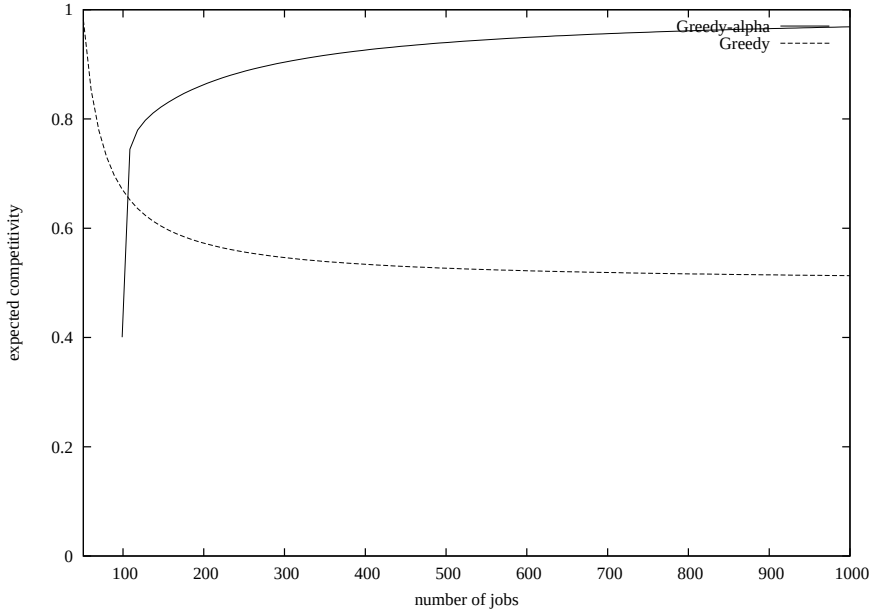

Figure 3.17: Lower bound for expected competitivities of Greedy-α' and Greedy on 50 machines

Chapter 4

Revenue Management for Scheduling Problems

4.1 Introduction

In this chapter we consider a problem which is similar to the one discussed in the preceding chapter. Here, the jobs are not committed to a given start time, but can be started within a time interval beginning with the release time of the job. Moreover, we allow arbitrary profits which do not depend on the lengths of the jobs anymore.

We are given m identical machines which are available for a planning horizon from $[0, T]$ and a set of n orders $J = \{j_1, \ldots, j_n\}$. Each order j_i is characterized by its release time $r_i \in [0, T]$, its processing time $b_i \geq 0$, a time slack $\delta_i \geq 0$, and the profit $p_i \geq 0$ which is obtained, if the order is accepted and produced. The time slack δ_i indicates the latest time $r_i + \delta_i$ at which j_i can be started such that it is done until its deadline $r_i + b_i + \delta_i$ when the product must be manufactured.

Each order j_i which is accepted must be scheduled without preemption on one of the m machines. In the offline version of the problem, all orders are known at time 0. In the practically more relevant online version at time t only orders with release time at most t are known to an online algorithm. Upon release of a new order j_i at time r_i an online algorithm must decide immediately whether to accept or to reject the order. If the order is accepted, the algorithm must provide a feasible schedule for all accepted orders which is consistent with previous decisions, that is, at all times $t' < t$, the new schedule must coincide with all previously computed schedules.

The problem is a generalization of the problem of scheduling equal-length jobs on parallel machines, where the jobs have release times and deadlines and the goal is to maximize the number of jobs completed in time. Baruah et al. [BHS01] showed that a greedy-type algorithm is 2-competitive for this problem with $m = 1$, equal profits and nondecreasing absolute deadlines for slacks over time (i.e. $r_i + b_i + \delta_i \leq r_k + b_k + \delta_k$ for all j_i, j_k with $r_i < r_k$).

For $m = 1$, equal job lengths and equal profits, Goldman et al. [GPS00] give a lower bounds of $4/3$ for the competitive ratio of randomized algorithms. They also show that no deterministic algorithm can better than 2-competitive in this setting and show that the greedy algorithm matches this bound. We generalize this result in Theorem 4.7 by showing that the greedy algorithm is $(1 + p_{max}/p_{min})$-competitive for p_{max} and p_{min} being upper resp. lower bounds for the profits of the jobs.

Ball and Queyranne analyzed the problem of online booking [BQ06] and found lower bounds and competitive algorithms for the problem with different kinds of profits. As two jobs which do not block each other can be processed on the same machine, our problem is a generalization of the problem studied by Ball and Queyranne.

In Section 4.2 we consider the hardness of the offline problem and propose an integer linear programming approach to find optimal solutions. Focusing on the case where all jobs have the same length, we derive competitive deterministic and randomized algorithms in Section 4.3 for the online problem with all jobs having the same length. For these algorithms and their analysis in Subsections 4.3.2 and 4.3.3, we reuse some of the ideas which were already applied in Subsections 3.5.2 and 3.5.3. Finally, we evaluate the empirical performance of the proposed algorithms in Section 4.4.

4.2 The Offline Problem

If $\delta_i = 0$ for each order j_i, the problem can be solved optimally by network flow techniques, as this problem is equivalent to the job-admission problem from the previous chapter. In general, this problem is $\mathcal{NP}$-hard, as we show in Subsection 4.2.1. In Section 4.2.2 we provide an integer linear programming formulation, which can be applied to all of the other cases for which there are optimal algorithms with polynomial running time not available.

4.2.1 Hardness

Lemma 4.1 *The problem of computing an optimal subset of jobs to be accepted is $\mathcal{NP}$-hard for each fixed $m \in \mathbb{N}$.*

Proof. We first consider the case $m = 2$ which we handle by a reduction from the PARTITION-Problem. Given an instance of PARTITION specified by n items with sizes $s_1, \ldots, s_n$ and $B = \sum_{i=1}^{n} s_i/2$ we proceed as follows: We set $T = B$ and for each item i we build an order j_i with $r_i = 0$, $b_i = s_i$,$p_i = s_i$, and $\delta_i = B - s_i$. So each item has a deadline of $T = B$ and we can accept all jobs if and only if we can partition the items into two sets where the sizes of all the items in each set sum up to B.

The case $m = 1$ is handled by the following idea: For a given instance of MAXIMUMSUBSETSUM specified by n numbers with sizes $s_1, \ldots, s_n$ and $B \in \mathbb{N}$, we have $T = B$ and for each of the n items i we introduce n orders $j_1, \ldots, j_n$ analogously to the case $m = 2$. This way an optimal subset of jobs corresponds to an optimal solution of the given SUBSETSUM problem.

Analogously, we can show $\mathcal{NP}$-hardness for $m \geq 3$ by reduction from BIN-PACKING. In this case, every machine corresponds to a bin, and the question whether all items fit into m bins boils down to the question whether the corresponding jobs can be scheduled on m machines. □

As preemption does not matter in the above constructions, we conclude that the problem would still be $\mathcal{NP}$-hard even if we allowed preemption.

4.2.2 An Integer Linear Programming Approach

As the offline algorithm knows all the requests beforehand, he can choose an appropriate discretization of $[0, T]$ such that the offline problem can be solved by the following time-indexed integer linear program:

$$
\begin{aligned}
\max \quad & \sum_{i=1}^{n} p_i y_i & & \\
& \sum_{i=1}^{n} x_{i,t} \leq m, & t \in [0,T] \qquad & (4.1a)\\
& \sum_{t=0}^{T} x_{i,t} = y_i b_i, & j_i \in J \qquad & (4.1b)\\
& x_{i,t} \geq x_{i,t-1} - z_{i,t}, & j_i \in J, t \in [0,T] \qquad & (4.1c)\\
& \sum_{t=0}^{T} z_{i,t} \leq y_i, & j_i \in J \qquad & (4.1d)\\
& x_{i,t} = 0 & j_i \in J, t \notin [r_i, r_i + b_i + \delta_i - 1] \qquad & (4.1e)\\
& x_{i,t}, z_{i,t} \in \{0,1\}, & j_i \in J, t \in [0,T] & \\
& y_i \in \{0,1\}, & j_i \in J &
\end{aligned}
$$

Here, the binary variable $y_i = 1$ if and only if order j_i is accepted and $x_{i,t} = 1$ if and only if job j_i is being processed at time t. $z_{i,t} = 1$ indicates that time t is the last time job j_i is being worked at.

Condition (4.1a) ensures that at any time at most m jobs are being processed, equality (4.1b) states that an accepted order j_i needs to be processed for exactly b_i units of time. Conditions (4.1c) and (4.1d) ensure that the time slot reserved for an accepted job is in fact an interval and (4.1e) guarantees that a job is processed neither before its release time nor after its deadline.

Lemma 4.2 *Any feasible solution* (x, y, z) *for the IP (4.1) corresponds to a feasible schedule which can be computed from* (x, y, z) *in linear time.*

Proof. We have seen that for each accepted order j_i the feasible solution (x, y, z) provides an interval $I_i = [\alpha_i, \beta_i]$ in which j_i is being processed. We consider the interval graph G with vertex set consisting of all intervals of accepted jobs. By condition (4.1a) the maximum clique size of G is at most m which by perfectness of interval graphs implies that G can be colored (in linear time) by at most m colors. Clearly, each such coloring of the intervals with at most m colors corresponds to an assignment of the jobs to machines which together with the intervals gives a feasible schedule. □

Cutting Planes

In order to improve the formulation stated above, we give some cutting planes. When a job i is processed at some time t, then due to its length b_i, it must be done by time $t + b_i$, thus:

$$x_{i,t} + x_{i,t+b_i} \leq 1 \qquad j_i \in J, t \in [r_i, r_i + \delta_i] \tag{4.2}$$

In order to show, that this valid inequality is a cutting plane, we consider an instance consisting of two machines and three jobs j_1, j_2, j_3, with $r_i = 0$, $\delta_i = 1$, $p_i = 1$, and $b_i = 2$ for $i = 1, 2, 3$. We choose $T = 3$ and $\{1, 2, 3\}$ as a discretization of $[0, 3]$. This means that $x_{i,t}$ is supposed to be 1 if j_i is processed during the whole interval $[t - 1, t]$. As we cannot schedule more than one job on a machine, and we have two machines available, the optimal solution can be achieved by scheduling one job on each machine, obtaining a profit of 2.

Consider the feasible fractional solution of (4.1a)-(4.1e) with $y_i = 1$ and $x_{i,t} = 2/3$ for $i, t = 1, 2, 3$, which yields a profit of 3 and is thus an optimal solution of the linear relaxation of the given IP. Due to $x_{i,1} + x_{i,3} = 4/3 > 1$, this solution violates (4.2) and we can call Inequality (4.2) a cutting plane, as it cuts of this solution.

As $z_{i,t} = 1$ if and only if t is the last time slot of j_i's processing time, we know that $x_{i,t} = 1$ for the last b_i time slots before. This is expressed by the following valid inequality:

$$x_{i,t'} \geq z_{i,t} \qquad j_i \in J, t' \in [t - b_i + 1, t]. \tag{4.3}$$

On the other hand, $x_{i,t} = 0$ for all other t, which is enforced by the following valid inequality:

$$x_{i,t'} \leq 1 - z_{i,t} \qquad j_i \in J, t' \notin [t - b_i + 1, t] \tag{4.4}$$

According to (4.1a)-(4.1e) and (4.2), a job j_1 of length 2 can be processed in $[0, 4]$ in a way such that $x_{1,1} = x_{1,4} = 2/3$ and $x_{1,2} = x_{1,3} = 1/3$. Since $z_{1,3} = 2/3$ and $x_{1,1} = 2/3$ this solution violates (4.4), and it disobeys (4.3), due to $x_{1,2} = x_{1,3} = 1/3 < 2/3 = z_{1,3}$. For this reason we can call the valid inequalities (4.3) and (4.4) cutting planes for the polyhedron defined by (4.1a)-(4.1e) and (4.2).

These cutting planes proved to be useful in the computational experiments presented in Section 4.4, as they decreased the time spent on solving the offline problems drastically.

4.3 Online Algorithms

We only consider online algorithms for the case that all of the jobs have unit size.

For seat reservation problems, which came up in the airline industry (see e.g. [MR99]), algorithms based on *Partitioned Protection Level Policies* and *Nested Protection Level Policies* proved to be useful. Both of these approaches are based on the idea of dividing the jobs into classes $C_1, \ldots, C_k$ such that all of the elements of C_i have a higher profit than the elements of C_k for $i > k$ and assigning the resources to the classes. Protection Level Policies try to allocate the requests to a resource of the same class. If this is not possible, Partitioned Protection Level Policies reject the request, while a Nested Protection Level Policies tries to assign the request to one of the resources meant for jobs with less profit. Inspired by the randomized algorithm CRS-GREEDY which is based on the classify and randomly select-paradigm [ABFR94], we adopt Protection Level Policies in Subsection 4.3.3 and 4.3.4 to our problem and derive the algorithms C-GREEDYand NESTED-GREEDY, which apply a greedy algorithm as a subroutine.

In order to obtain a lower bound for the competitive ratio of online algorithms, we refer to Chapter 3:

Theorem 4.3 *If all jobs have unit size, any randomized algorithm has a competitive ratio no smaller than* $\frac{1}{2}(\log(p_{max}/p_{min}) + 2)$ *against an oblivious adversary.*

Proof. The proof of Theorem 3.3 carries directly over to this problem, as the instances considered there can be implemented in our setting. For this reason, we need to define for each job in the instances considered in the proof of Theorem 3.3 a job j with $r_j = 0$, $\delta_j = 0$, and the profit attaining the value of the job's length in the instance of OJA. This way, we have the same property that each machine can process at most one job and the profits obtained are identical. Since $p_{max} = T$ and $p_{min} = 1$, the proof of the claimed lower bound can be given along the same lines. □

4.3.1 The Greedy algorithm

In this subsection we analyze the competitivity of a greedy-type algorithm, which schedules all jobs as early as possible, and rejects those for which it is not possible to find a feasible schedule containing them.

For a given solution $\tilde{J} \subseteq J$ and $j_i \in \tilde{J}$ define t_i to be the starting time of j_i.

Lemma 4.4 *If all jobs have unit size and the same slack δ, then there exists an optimal solution $\tilde{J} \subseteq J$, such that $t_i \leq t_k$ for all $j_i, j_k \in \tilde{J}$ with $r_i \leq r_k$.*

Proof. We establish the claim by proving that two jobs j_i, j_k with $t_i \leq t_k$ and $r_i > r_k$ are allowed to change places (see Figure 4.1).

As $t_k \geq t_i \geq r_i$ and $t_k \leq r_k + \delta \leq r_i + \delta$, it is allowed to schedule job j_i at time t_k. On the other hand job j_k can take the place of j_i, since $t_i \geq r_i > r_k$ and $t_i \leq t_k \leq r_k + \delta$.

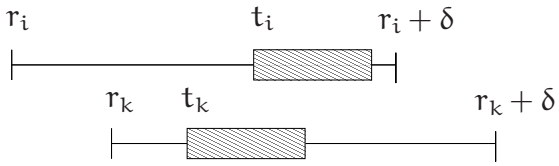

Figure 4.1: Jobs j_i and j_k are allowed to change their start times

By applying these swaps one by one an optimal solution with the desired property can be created. □

Theorem 4.5 *If all jobs have unit size and the same slack δ and profit p, then scheduling jobs greedily in the order of increasing release times gives an optimal solution.*

Proof. For a given sequence of jobs σ let $\text{OPT} = (j_1, j_2, \ldots j_k)$ be the jobs scheduled by the optimal algorithm and $\text{GREEDY} = (i_1, i_2, \ldots, i_h)$ be the jobs scheduled by the greedy algorithm, with both sequences ordered by the release times of their elements. For $j \in \text{OPT}$ let t_j^o be the start time of j in the schedule of OPT and define t_j^g the start time of j in the schedule obtained by the greedy algorithm for all $j \in \text{GREEDY}$. Let $p := \min\{l : j_l \neq i_l \text{ or } t_{j_l}^o \neq t_{i_l}^g\}$ be the index of the first job which is not treated by both algorithms in the same way.

Because of Lemma 4.4 we can assume that the optimal solution starts its accepted jobs in order of their release times. Observe that GREEDY has the same property. Additionally, we choose OPT to be an optimal solution with maximum p.

Case 1: Suppose $j_p = i_p$ but the job is started by the algorithms at different times. Then, because of the way the greedy algorithm works, we know that $t_{i_p}^g < t_{j_p}^o$. In this case, we obtain another optimal solution OPT $'$, by starting i_p earlier. With $t_{j_p}^o := t_{i_p}^g$ the latter jobs $j_{p+1}, \ldots, j_k$ can still be scheduled and, as this start time fits to the greedy solution, it fits for the first $p-1$ jobs of the optimal solution as well. As OPT $'$ still schedules with increasing release times,

we have found a solution with higher p. This contradicts the maximality of OPT with respect to p.

Case 2: Suppose $j_p \neq i_p$. As OPT is optimal, there is a conflict between i_p and j_p. We claim that the optimal solution can schedule i_p at time $t^g_{i_p}$ instead of j_p. This way we obtain a schedule OPT′ which contradicts the maximality of OPT with respect to p.

As we have $r_{i_p} \leq r_{j_p}$ and thus $t^g_{i_p} \leq t^o_{j_p}$ (GREEDY always starts as soon as possible), the optimal algorithm could have started i_p instead of j_p at time $t^g_{i_p}$. It remains to show that it is not possible that the optimal algorithm accepted i_p but started it later. In this case there was some $l > p$ such that $j_l = i_p$ and as nondecreasing release times imply nondecreasing start times in OPT , we have:

$$t^o_{i_p} = t^o_{j_l} \geq t^o_{j_{l-1}} \geq \cdots \geq t^o_{j_p}$$

Since the greedy algorithm accepted i_p, we have $r_{i_p} \leq r_{j_p}$, thus $t^o_{i_p} \leq t^o_{j_p}$ and consequently

$$t^o_{i_p} = t^o_{j_l} = t^o_{j_{l-1}} = \cdots = t^o_{j_p}$$

So, all of these jobs start at the same time, which means that i_p can be seen as the next job that the optimal algorithm accepts, which means that $j_p = i_p$. This case is treated above as Case 1. □

Lemma 4.6 *If all jobs have unit size, arbitrary slacks δ_i and profits between p_{min} and p_{max}, then scheduling jobs greedily in the order of increasing release times is not better than $1 + \frac{p_{max}}{p_{min}}$.*

Proof. Consider the sequence consisting of m low-profit jobs with release time 0 and $\delta = 1.5$ and m high-profit jobs with release time 0.5 and $\delta = 0$. Greedy accepts all of the low-profit jobs and starts them immediately at time 0, while the optimal algorithm starts all of them at time 1.5. This way all of the high-profit jobs can be scheduled and the optimal algorithm achieves a profit of $m \cdot p_{max} + m \cdot p_{min}$, while greedy makes a profit of $m \cdot p_{min}$. The lower bound obtained reads

$$\frac{m \cdot p_{max} + m \cdot p_{min}}{m \cdot p_{min}} = 1 + \frac{p_{max}}{p_{min}}$$

□

Theorem 4.7 *If all jobs have unit size, arbitrary slacks δ_i and profits between p_{min} and p_{max}, then scheduling jobs greedily in order of increasing release times is $1 + \frac{p_{max}}{p_{min}}$-competitive.*

Proof. For a given sequence of jobs σ let OPT be the set of jobs scheduled by the optimal algorithm and GREEDY be the set of jobs scheduled by the greedy algorithm. For $j \in$ OPT let t_j^o be the start time of j in the schedule of OPT and define t_j^g the start time of j in the schedule obtained by the greedy algorithm for all $j \in$ GREEDY. Consider the mapping $g :$ GREEDY $\rightarrow$ OPT which is defined by Algorithm 2

Algorithm 2 mapping

```
1: Input: GREEDY , OPT
2: Output: a function g mapping GREEDY on OPT
3: for all j ∈ GREEDY do
4:    if j ∈ OPT then
5:       g(j) := {j}
6:       OPT := OPT \ {j}
7:    else
8:       g(j) := ∅
9:    end if
10: end for
11: for all j ∈ GREEDY do
12:    Choose k ∈ OPT with t_k^o minimal from all jobs in OPT overlapping j
13:    if k exists then
14:       g(j) := g(j) ∪ {k}
15:       OPT := OPT \ {k}
16:    end if
17: end for
18: return g
```

We claim that every $j \in$ OPT is mapped by g. For this reason, suppose there was $j \in$ OPT which is not the picture of any $l \in$ GREEDY. Then, there are m jobs $j_1, \ldots j_m \in$ GREEDY which block every possible start time of j. Otherwise, GREEDY would have accepted j. This means $t_{j_i} < r_j$ and $t_{j_i} > r_j + \delta - 1$. We can assume that $\delta_j < 1$ as there cannot exist a job with $t_{j_i} < r_j$ and $t_{j_i} > r_j + \delta - 1 \geq r_j$. As $j \notin g(j_i)$ for all $i = 1, \ldots, m$ there have to be m jobs $k_1, \ldots, k_m \in$ OPT $\setminus$ GREEDY which start not later than j. If one of them started later, g would have mapped j to one of the j_i instead. Thus, $t_{k_i} \leq t_j$ for $i = 1, \ldots, m$ and consequently $t_{k_l} \leq t_j - 1 \leq r_j + \delta - 1 < r_j$ for some l.

Now, we can apply the same argumentation to k_l as to j. There have to be m jobs $p_1, \ldots, p_m \in$ GREEDY which blocked the acceptance of k_l. So, all of these jobs start before r_{k_l} and end after $r_{k_l} + \delta$. It is important to mention that these jobs are different from the jobs which block j. Suppose there was a job $\tilde{j}$ which

blocks j and k_l, then it has to be started before the release time of k_l and it ends after the last possible start time of j. So, we need $t_{\tilde{j}} < r_{k_l} \leq t_{k_l} \leq t_j - 1 \leq r_j$ and $t_{\tilde{j}} > r_j + \delta$ which is contradiction. So, as all of the p_i are not mapped to k_l, there have to m more jobs in OPT which start not later than k_l. Like above, one of them starts not later than $t_{k_l} - 1$ and so on.

This way we obtain an endless sequence of jobs, which contradicts the fact that the input sequence in finite. Thus, all elements of OPT are mapped by g and we have $\text{OPT} = \cup_{j \in \text{GREEDY}} g(j)$. Now, we can estimate the value of the optimal solution.

$$\begin{aligned}
\sum_{j \in \text{OPT}} p_j &= \sum_{j \in \text{GREEDY}} f(g(j)) \\
&\leq \sum_{j \in \text{GREEDY}} p_j + \frac{p_{max}}{p_{min}} p_j \\
&= \sum_{j \in \text{GREEDY}} p_j \left(1 + \frac{p_{max}}{p_{min}}\right) \\
&= \left(1 + \frac{p_{max}}{p_{min}}\right) \sum_{j \in \text{GREEDY}} p_j
\end{aligned}$$

□

4.3.2 An Algorithm Based on Classify and Randomly Select

Assume now we are given a lower and an upper bound bound p_{min} and p_{max} for the profits of the elements of σ. Given some $\Delta > 1$, we divide the possible input requests into $N := \lceil \log_\Delta T \rceil$ disjoint classes $C_1, \ldots, C_N$, with $j \in C_i$ if and only if $p_{min} \cdot \Delta^{i-1} \leq p_j < p_{min} \cdot \Delta^i$ for $i = 1, \ldots, N$. The algorithm CRS-GREEDY chooses class C_i with probability $\frac{1}{N}$ Then, when processing a sequence σ the algorithm ignores all requests not in class C_i and uses GREEDY to process the requests in class C_i.

For $i = 1, \ldots, N$ let $\sigma_i := \sigma \cap C_i$ and OPT_i denote the total profit of jobs from class C_i accepted by OPT . If GREEDY processes σ_i for some i, it achieves a competitive ratio of $1 + \Delta$ by Theorem 4.7, since $\Delta_{\sigma_i} \leq \Delta^i / \Delta^{i-1} = \Delta$. Since there is a probability of $\frac{1}{N}$ that the algorithm picks the class which contributes the biggest part to the optimal solution we can estimate the expected value of

the profit obtained by CRS-GREEDY as follows:

$$\begin{aligned}\mathbb{E}\left[\text{CRS-GREEDY}(\sigma)\right] &= \sum_{i=1}^{N} \frac{1}{N} \cdot \text{GREEDY}\left(\sigma_i\right) \geq \frac{1}{N} \sum_{i=1}^{N} \frac{1}{1+\Delta} \text{OPT}\left(\sigma_i\right) \\ &\geq \frac{1}{(1+\Delta)N} \sum_{i=1}^{N} \text{OPT}\left(\sigma_i\right) \geq \frac{1}{(1+\Delta)N} \sum_{i=1}^{N} \text{OPT}_i \\ &= \frac{1}{(1+\Delta)N} \text{OPT}\left(\sigma\right).\end{aligned}$$

Thus, CRS-GREEDY achieves a competitive ratio of

$$\begin{aligned}(1+\Delta)\,N = (1+\Delta)\left\lceil \log_\Delta \frac{p_{max}}{p_{min}} \right\rceil &\leq (1+\Delta)\left(\log_\Delta \frac{p_{max}}{p_{min}} + 1\right) \\ &= \frac{1+\Delta}{\ln \Delta}\left(\ln \frac{p_{max}}{p_{min}} + \ln \Delta\right)\end{aligned}$$

The first term is smallest for the Δ' which minimizes

$$\frac{1+\Delta}{\ln \Delta}$$

Looking at the derivatives, we see that

$$1 + 1/\Delta' = \ln \Delta' \qquad \text{and equivalently} \qquad \Delta' = \frac{1+\Delta'}{\ln \Delta'}. \tag{4.5}$$

So, for choosing Δ as Δ' we obtain a competitivity of

$$\begin{aligned}\frac{1+\Delta'}{\ln \Delta'}(\ln \frac{p_{max}}{p_{min}} + \ln \Delta') &\overset{\text{Eq.(4.5)}}{=} \Delta'\left(\ln \frac{p_{max}}{p_{min}} + \ln \Delta'\right) \\ &\overset{\Delta' \leq 3.59113}{\leq} 3.59113\left(\ln \frac{p_{max}}{p_{min}} + 1.27847\right)\end{aligned}$$

Thus, together with Theorem 4.3 we can state the following theorem:

Theorem 4.8 *If all jobs have unit size, arbitrary slacks δ_i and profits between p_{min} and p_{max}, then* CRS-GREEDY *is $3.59113\left(\ln \frac{p_{max}}{p_{min}} + 1.27847\right)$-competitive, which is optimal up to a constant factor.*

4.3.3 Partitioned Protection Level Policies

We will now use GREEDY and ideas from the classify-and-select paradigm to obtain a deterministic algorithm which achieves an improved competitiveness.

As it applies for the problem considered in this chapter as well, we restate Theorem 3.6 from the preceding chapter:

Lemma 4.9 *Suppose that we are given a sequence of jobs* σ. *For* $1 \leq t \leq m$ *let* $\text{OPT}^{(t)}(\sigma)$ *denote the optimal offline profit achievable using* t *machines (so that* $\text{OPT}(\sigma) = \text{OPT}^{(m)}(\sigma)$*). Then,*

$$\frac{t}{m} \cdot \text{OPT}^{(m)}(\sigma) \leq \text{OPT}^{(t)}(\sigma) \leq \text{OPT}^{(m)}(\sigma).$$

Instances with $m \geq \log_{\Delta'}(p_{max}/p_{min})$

Similar to the randomized algorithm CRS-GREEDY, the improved deterministic algorithm C-GREEDY divides the jobs into $k := \lceil \log_{\Delta'} p_{max}/p_{min} \rceil$ classes with class C_i holding all the jobs with profit between $p_{min} \cdot \Delta'^{i-1}$ and $p_{min} \cdot \Delta'^{i}$. The algorithm reserves exactly $t := \lfloor m/k \rfloor$ machines for each of the k classes C_i and uses an instantiation of GREEDY to process the jobs on each of the classes. The next Lemma shows that the competitive ratio of this algorithm is only a factor of $1 + 1/t$ away from the competitive ratio of CRS-GREEDY :

Lemma 4.10 *For* $m \geq \log_{\Delta'} \frac{p_{max}}{p_{min}}$ C-GREEDY *is* $\left(1 + \frac{1}{t}\right)\left(\ln \frac{p_{max}}{p_{min}} + \ln \Delta'\right)\Delta'$*- competitive.*

Proof. Similar as in Lemma 4.9 let $\text{OPT}^{(t)}$ and $\text{GREEDY}^{(t)}$ be the respective algorithms which schedule jobs on t machines instead of on m machines. Let OPT_i be the profit of OPT obtained by jobs in class C_i.

Then with

$$\frac{m/k}{\lfloor m/k \rfloor} \leq \frac{t+1}{t} = 1 + \frac{1}{t} \qquad \Rightarrow \qquad \frac{m}{t} \leq k \cdot \left(1 + \frac{1}{t}\right) \tag{4.6}$$

we can estimate the profit of the optimal algorithm by

$$\begin{aligned}
\textsc{Opt}(\sigma) &= \sum_{i=1}^{k} \textsc{Opt}_i \leq \sum_{i=1}^{k} \textsc{Opt}^{(m)}(\sigma_i) \\
&\overset{\text{Lemma 4.9}}{\leq} \sum_{i=1}^{k} \frac{m}{t} \cdot \textsc{Opt}^{(t)}(\sigma_i) \\
&\overset{\text{Eq.(4.6)}}{\leq} \sum_{i=1}^{k} k \cdot \left(1 + \frac{1}{t}\right) \cdot \textsc{Opt}^{(t)}(\sigma_i) \\
&\overset{\text{Theorem 4.7}}{\leq} k \cdot \left(1 + \frac{1}{t}\right) \sum_{i=1}^{k} (\Delta + 1)\textsc{Greedy}^{(t)}(\sigma_i) \\
&= k \cdot \left(1 + \frac{1}{t}\right) (\Delta' + 1) \sum_{i=1}^{k} \textsc{Greedy}^{(t)}(\sigma_i) \\
&= k \cdot \left(1 + \frac{1}{t}\right) (\Delta' + 1) \cdot \textsc{c-greedy}(\sigma).
\end{aligned}$$

With

$$k = \lceil \log_{\Delta'}(p_{max}/p_{min}) \rceil \leq \log_{\Delta'} \frac{p_{max}}{p_{min}} + 1 = \frac{\ln p_{max}/p_{min} + \ln \Delta'}{\ln \Delta'}$$

we obtain the claimed competitivity of

$$\begin{aligned}
k \cdot \left(1 + \frac{1}{t}\right)(\Delta' + 1) \leq\ & \left(1 + \tfrac{1}{t}\right)\left(\ln \tfrac{p_{max}}{p_{min}} + \ln \Delta'\right) \tfrac{\Delta'+1}{\ln \Delta'} \\
\overset{\text{Eq.(4.5)}}{=}\ & \left(1 + \tfrac{1}{t}\right)\left(\ln \tfrac{p_{max}}{p_{min}} + \ln \Delta'\right) \Delta'.
\end{aligned}$$

□

Observe that $1 + 1/t \leq 2$ for $m \geq \log_{\Delta'} \frac{p_{max}}{p_{min}}$.

Instances with $m < \log_{\Delta'}(p_{max}/p_{min})$

In this case we cannot apply c-greedy as stated above, since t would be less than one, which means that there are more classes of jobs than machines.

Lemma 4.11 *For* $m < \log_{\Delta'} \frac{p_{max}}{p_{min}}$ c-greedy *is* $m \cdot \left(\sqrt[m]{p_{max}/p_{min}} + 1\right)$ *- competitive.*

Proof. In this case, we define $\Delta := \sqrt[m]{p_{max}/p_{min}}$ and analogously to the preceding Lemma the competitivity of this algorithm can be estimated as:

$$\begin{aligned}
\textsc{Opt}(\sigma) &= \sum_{i=1}^{m} \textsc{Opt}_i \leq \sum_{i=1}^{m} \textsc{Opt}^{(m)}(\sigma_i) \\
&\overset{\text{Lemma 4.9}}{\leq} \sum_{i=1}^{m} m \cdot \textsc{Opt}^{(1)}(\sigma_i) \\
&\overset{\text{Theorem 4.7}}{\leq} m \cdot \sum_{i=1}^{m} (\Delta + 1)\textsc{Greedy}^{(1)}(\sigma_i) \\
&= m \cdot (\Delta' + 1) \sum_{i=1}^{k} \textsc{Greedy}^{(1)}(\sigma_i) \\
&= m \cdot (\sqrt[m]{p_{max}/p_{min}} + 1) \cdot \textsc{c-greedy}(\sigma).
\end{aligned}$$

□

4.3.4 Nested Protection Level Policies

Suppose, C-GREEDY processes a sequence σ with all jobs having the same profit. In this case, the algorithm schedules jobs only on the t machines of the corresponding class and rejects all the others. As this behavior is unsatisfactory from the practitioner's point of view, we derive another algorithm NESTED-GREEDY which has a competitivity not worse than C-GREEDY but overcomes this deficit.

The difference between C-GREEDY and NESTED-GREEDY is that before rejecting a job $j \in C_i$, NESTED-GREEDY tries to schedule it on one of the machines meant for jobs of class C_{i-1}. If it cannot be scheduled on one of those machines, it is handed to the next class and so on. If j could not be scheduled on any of the machines considered, it is rejected.

Observe that the profit realized by a solution found by GREEDY on some sequence σ cannot become smaller if an additional job j with higher profit is added to σ. As j can displace at most one of the other jobs, and the replaced job has a smaller profit, the overall profit can only become larger.

Like C-GREEDY, NESTED-GREEDY uses an instantiation of GREEDY to process the jobs of each of the classes C_i. But in contrast to the other algorithm, NESTED-GREEDY enlarges each class C_i by those jobs belonging to $C_{i+1}, C_{i+1}, \ldots, C_k$ which could not be scheduled on any of the corresponding

machines. Because of the observation made enough, we see that the profit obtained by NESTED-GREEDY is not smaller than the profit of C-GREEDY. Consequently, the competitive ratios proven in Subsection 4.3.3 apply for NESTED-GREEDY as well

4.4 Experimental Results

Despite the worst case behavior of the online algorithms studied above, we would like to get an impression of their practical quality. Thus, we implemented them and tested their competitivity for different numbers of jobs n and machines m. The machine time is 100 and each job has length 10, the jobs' start times are sampled uniformly from the interval of all possible start times $[1, 70]$. δ_i is chosen uniformly between 0 and 20 and the profit of job i is calculated by $p_i := \alpha - \beta \cdot \delta_i$ with $\alpha := 100$ and $\beta := 5$. This linear profit function reflects the experience that jobs requiring a high operational availability come along with a higher profit, which is a common assumption in revenue management [TR05].

Release times and slack times are chosen in a way such that they can be sampled independently. Suppose the release time r_i was greater than 70, then δ_i could not attain 20 anymore, due to the job length and the machine time of 100. As a consequence jobs with release times greater than 70 would have smaller δ_i and thus larger p_i.

According to the definition of C-GREEDY and NESTED-GREEDY given in the previous section, it can happen that not all of the machines are assigned to classes. In contrast to their definition, we have implemented these algorithms such that these $m - \lfloor m/k \rfloor$ machines left over are assigned to the classes meant for the highest profit jobs. Each of these classes received an additional machine this way. This is allowed, since this modification does not harm the competitivities proven for C-GREEDY and NESTED-GREEDY.

In Table 4.1 we present the simulation results with each row representing a set of randomly generated instances. The left part states the parameter setting, whilst in the right half the average-case competitivities achieved by GREEDY CRS-GREEDY and C-GREEDY are given with the corresponding sample standard deviations.

Iter.	n	m	c_G[%]	$\mathbb{E}[c_{CRS}]$ [%]	c_{class}[%]	c_{nested}[%]
20	30	2	54.2 ± 9.2	36.3 ± 6.9	44.8 ± 5.8	59.2 ± 7.5
20	30	5	89.6 ± 5.3	35.5 ± 6.4	68.5 ± 9.4	93 ± 4.4
20	30	10	100 ± 0	35 ± 5.1	93.1 ± 6.5	100 ± 0

Iter.	n	m	c_G[%]	$\mathbb{E}[c_{CRS}]$ [%]	c_{class}[%]	c_{nested}[%]
20	30	15	100 ± 0	36.7 ± 6.8	98.9 ± 2.3	100 ± 0
20	40	2	47.5 ± 5.4	33.2 ± 3.7	38.1 ± 6	52 ± 5.1
20	40	5	77.4 ± 5.5	34.6 ± 4.9	58.6 ± 9.4	77.9 ± 4.4
20	40	10	99.7 ± 1.2	36.7 ± 6.8	90.5 ± 5.6	100 ± 0
20	40	15	100 ± 0	37.5 ± 7.4	97.2 ± 3.4	100 ± 0
20	50	2	39.8 ± 6	34.5 ± 2.9	38.5 ± 3.8	47.2 ± 5.8
20	50	5	67.9 ± 6	34.3 ± 5.1	51.9 ± 6.2	69.9 ± 4.4
20	50	10	96.9 ± 3.3	34 ± 3.8	78.4 ± 8.8	98.8 ± 2.9
20	50	15	100 ± 0	35.8 ± 6.1	90.6 ± 7.4	100 ± 0
20	60	2	38.2 ± 4.5	34.5 ± 2.4	35.1 ± 3.7	44.2 ± 4.4
20	60	5	58.3 ± 5.2	34.8 ± 5.5	49.9 ± 4.6	65 ± 4.4
20	60	10	92.7 ± 5.2	34.1 ± 3.8	69.7 ± 5.5	94.1 ± 5.1
20	60	15	100 ± 0	33.3 ± 0	79.5 ± 6.2	100 ± 0
20	70	2	33.7 ± 7.7	36.2 ± 2.8	34 ± 5	40.6 ± 6.6
20	70	5	49.4 ± 4.8	34.2 ± 4.8	48.8 ± 5.6	57.6 ± 4
20	70	10	89 ± 4.9	33.8 ± 3.9	63.8 ± 4.3	90.7 ± 3.4
20	70	15	99.6 ± 0.9	33.3 ± 0	76 ± 6.5	100 ± 0
20	80	2	32.8 ± 6.4	33.3 ± 3.5	33.1 ± 3.1	39.2 ± 4.9
20	80	5	44.5 ± 4.4	32.9 ± 1.7	44.6 ± 2.8	53.8 ± 3.1
20	80	10	80.7 ± 3.6	32.7 ± 0.7	56.6 ± 4.5	82.9 ± 3.7
20	80	15	99.2 ± 1.1	33.3 ± 0	66.3 ± 5.8	99.7 ± 0.8
20	90	2	29.1 ± 4	33.6 ± 1.3	30.4 ± 3.2	35.6 ± 3.2
20	90	5	42.7 ± 5	33.6 ± 3.4	44.2 ± 3.1	51.9 ± 3.3
20	90	10	73.8 ± 4.9	32.7 ± 1	51 ± 5.3	76.2 ± 3.8
20	90	15	96.1 ± 2.3	34.1 ± 3.8	61.3 ± 5.9	97.9 ± 2.7
20	100	2	28.8 ± 3.8	32.6 ± 1.8	32.4 ± 3.4	37.2 ± 3.9
20	100	5	40.7 ± 5.3	32 ± 2.6	42.9 ± 2.6	49.8 ± 2.9
20	100	10	66.5 ± 3.3	33.1 ± 1.1	49.4 ± 3.2	69.2 ± 3.1
20	100	15	93 ± 4	33.1 ± 0.7	56.6 ± 5.8	94.4 ± 4.1
20	110	2	28.7 ± 5.7	31.9 ± 2.1	32 ± 2.9	36.4 ± 5.2
20	110	5	37.1 ± 3.4	32.3 ± 2.2	41.2 ± 2.6	47.3 ± 2.3
20	110	10	61.3 ± 3.9	33.6 ± 1.1	47.8 ± 2.5	63.4 ± 3.7
20	110	15	88.3 ± 3.5	33.1 ± 0.6	53.4 ± 4.3	90 ± 3.1
20	120	2	28 ± 3.7	32.7 ± 2.1	30.4 ± 3.1	35 ± 3.9
20	120	5	35.6 ± 4.9	32.6 ± 1.5	40.9 ± 2.1	45.9 ± 3.3
20	120	10	55.9 ± 3.9	33.2 ± 1	45.5 ± 2.5	59.3 ± 2.8
20	120	15	81.9 ± 2.7	33.2 ± 0.4	49.2 ± 4.5	83.1 ± 3.1
20	130	2	27.5 ± 5.1	31.4 ± 1.7	30 ± 2.6	34.9 ± 4.2
20	130	5	34.1 ± 3.4	32.5 ± 2.1	40.2 ± 2.2	45.6 ± 2.4

Iter.	n	m	c_G[%]	$\mathbb{E}[c_{CRS}]$ [%]	c_{class}[%]	c_{nested}[%]
20	130	10	51.3 ± 3.3	33 ± 1.2	43.8 ± 2.4	55.6 ± 2.9
20	130	15	74.8 ± 3.1	33.1 ± 0.8	47.9 ± 3.8	77 ± 3.9
20	140	2	27.2 ± 3.9	31 ± 1.6	29.8 ± 2.5	34.2 ± 3.1
20	140	5	33.2 ± 2.6	32.5 ± 1.3	39.5 ± 1.4	44.3 ± 2.4
20	140	10	48.6 ± 2.8	32.6 ± 1.9	42.5 ± 1.9	54 ± 2.1
20	140	15	69.7 ± 3.7	33.3 ± 0.8	45 ± 3.5	72.1 ± 3.5
20	150	2	26.4 ± 4.2	30.8 ± 1.6	29.3 ± 2.2	33.5 ± 3.3
20	150	5	32.8 ± 4.2	32.1 ± 2.1	38.9 ± 2.1	43.1 ± 2.8
20	150	10	45 ± 2.3	32.5 ± 2	41.5 ± 1.6	51.2 ± 2
20	150	15	65.1 ± 2.9	33.2 ± 0.6	42.5 ± 2.9	67.8 ± 2.3

Table 4.1: Experimental competitivity of GREEDY , CRS-GREEDY C-GREEDY and NESTED-GREEDY

In order to display the dependency of the algorithms' competitivities to the load n/m of the instances considered, Figure 4.2 shows how the monitored competitivities of the algorithms decrease in dependence of the average number of jobs each machine is faced with. Every entry of Table 4.1 corresponds to one points in Figure 4.2.

The disability of C-GREEDY to accept high-profit jobs, when all of the corresponding machines are already occupied, is resolved by the NESTED-GREEDY, as this algorithm is able to schedule these jobs on other machines. By the use of this modification, the resulting algorithm NESTED-GREEDY is able to compute solution of a quality level comparable to those obtained by GREEDY. In fact, most of the solutions produced by NESTED-GREEDY are slightly better than the solutions of GREEDY.

In consideration of Table 4.1 and Figure 4.2 it appears that NESTED-GREEDY outperforms GREEDY not only in the worst case but also in the average case.

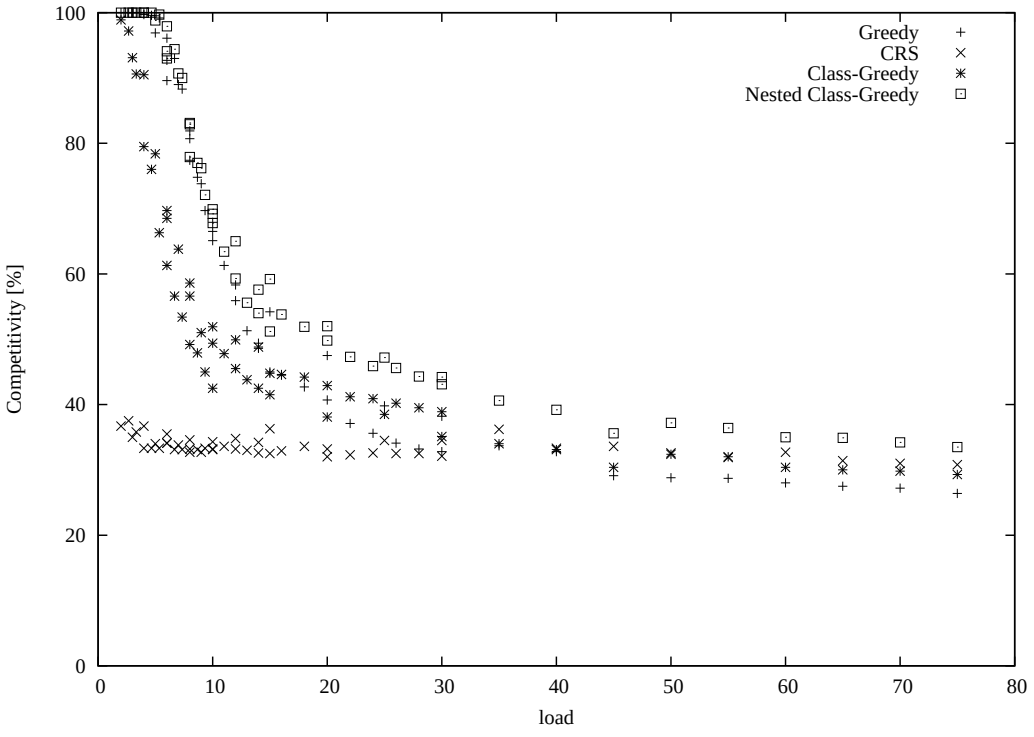

Figure 4.2: Experimental competitivity in relationship to the system load

Chapter 5

On a Product Serving Problem

5.1 Introduction

Consider a florist who is asked to create a bouquet of flowers for a special occasion (e.g. a wedding). In most of the cases this request is not defined precisely. The customer has only a rough idea, what he expects the bunch of flowers to look like, and so there are several possibilities to satisfy the client's wish. The florist has several flowers in stock, and he can reorder flowers in boxes of fixed size, say m flowers. As a result of the *configuration* of flowers he picks, he might have to reorder some boxes. For the purpose of a high profit he tries to keep the number of boxes ordered low.

Besides this application there are many real-life problems, mainly in industries or businesses where the final product to be sold is the result of the simple combination of raw material (i.e., meals in restaurants, mixed-nut bins, among others). In all of these cases we are given a ground set of items called F, a set of configurations C, and a set of products P. Every configuration $c \in C$ is a multiset made up of (not necessarily different) elements of F, and every product $p \in P$ consists of a subset of C. In the example given above, F corresponds to the different kinds of flowers, every $p \in P$ describes a request for a bunch of flowers which can be satisfied by one of the possible configurations $c \in p$ reflecting the different possibilities to compose a bouquet of flowers meeting the customer's requirements.

At first we take a look at the offline problem in Section 5.2, which corresponds to the case where the florist is given a list of orders and has to decide how to satisfy all of the jobs. In this case, all of the clients' requests can be taken into account simultaneously.

In Section 5.3 we consider the online problem, which reflects the case of a florist standing in his shop and serving customers one by one. When serving a customer, he can only base his decisions upon the number of flowers he has in stock, and the possible configurations of flowers that are allowed to serve the current request, as he does not know, what the upcoming clients are asking for.

Motivated by the application sketched above, we look at the problem when each request r consists of up to k configurations $\{c_1, c_2, \ldots, c_l\}$ with each of these configurations being sets of items. So, in order to satisfy r we must pick *all* the elements of at least one configuration $c_i \in r$. We assume for simplicity that initially there are no items in stock.

5.2 Offline Results

We focus on the offline variant of the Product Serving Problem:

Definition 5.1 (k-PRODUCT-SERVING(k-PS)**)** One is given a finite ground set F of different types of items. The items can be ordered in boxes each of which have capacity m at cost m. Then, requests arrive. A request r is a set $\{c_1, c_2, \ldots, c_l\}$ of configurations with $l \leq k$, where $k \in \mathbb{N}$ is a fixed constant and c_i being multisets of (not necessarily different) items. In order to serve request r, *all* the elements of at least one $c_i \in r$ have to be picked, thus reducing the inventory in the corresponding boxes by the amount being asked for. The goal is to serve all requests at minimum cost.

For $k = 1$ there is exactly one feasible configuration for each product. Thus, there is exactly one solution for every instance of 1-PRODUCT-SERVING. As this case is trivial, we focus on k-PRODUCT-SERVING for $k \geq 2$.

Lemma 5.2 k-PRODUCT-SERVING *is* NP*-hard for each fixed* $k \geq 2$.

Proof. Consider the BIN-PACKING problem, where one is given a finite set U of objects, a size $s(u) \in Z^+$ for each $u \in U$, a positive integer k and a positive integer bin capacity b. The question is to decide whether there exists a partition of U into disjoint sets $U_1, U_2, \ldots, U_k$ such that the sum of the sizes of the objects in each U_i is b or less.

We define an instance of k-PRODUCT-SERVING with each bin corresponding to a box of size $b + 1$ and each object $u \in U$ being represented by a request, asking for $s(u)$ items of the same kind:

For each bin, we introduce one kind of item and one request, which asks for one item of this kind. Thus, at least one box has to be ordered for every $f \in F$.

As every object $u \in U$ can be put into any of the k bins, we add a request which asks for $s(u)$ items of the same kind for every $u \in U$. This means that for every u there are k different configurations representing the k different bins. As the instance of BIN-PACKING allows every box to contain objects of a total size of b, and one unit of every box is already used by a request not representing an object, we choose the capacity of the boxes to be $b+1$. Since there is a solution for this instance of k-PRODUCT-SERVING consisting only of k orderings if and only if there is a solution to the corresponding instance of BIN-PACKING, the claim follows. □

5.2.1 A Logarithmic Approximation for $|c| = 1$ for all $c \in C$

At first we look at the restriction of the problem sketched above, in which every configuration holds exactly one element. In terms of the example stated in the introduction of this chapter, every customer asks for a single flower, and the florist is free to choose one from up to k different types.

We will use a simple greedy-type algorithm for obtaining an approximation. Let P denote the set of products that need to be covered. In each iteration we buy that type of items (at cost m for restocking the whole box) which appears in the most unserved products. We delete the served products from P and continue.

Suppose that overall there are t iterations. Let ϕ_j denote the number of yet unserved products *after* iteration j. Then, $\phi_0 = |P|$ and $\phi_{t-1} \geq 1$, since we do t iterations and have not stopped after the $t-1$st iteration.

Fix some iteration j and let f_j be the item box chosen in this iteration. We claim that

$$\frac{m}{\phi_{j-1} - \phi_j} \leq \frac{\text{OPT}}{\phi_{j-1}}. \tag{5.1}$$

To see this, consider the optimum solution consisting of OPT/m item inventory upstockings. Using all the items from OPT covers all ϕ_{j-1} unserved products and costs OPT . Thus, on average the cost to profit ratio of a item upstocking in OPT is OPT/ϕ_{j-1} and, thus, there must exist a box which has ratio no more than OPT/ϕ_{j-1}.

Let r_j denote the number of products covered by the box chosen in iteration j. Then, for the potential we have

$$\phi_j = \phi_{j-1} - r_j.$$

Since $r_j = \phi_{j-1} - \phi_j$ we obtain from (5.1) that

$$\phi_j \leq \left(1 - \frac{m}{OPT}\right) \phi_{j-1}.$$

Thus, by induction

$$\phi_{t-1} \leq \phi_0 \left(1 - \frac{m}{OPT}\right)^{t-1}.$$

Taking natural logarithms on both sides and simplifying using the estimate $\ln(1-\tau) \leq -\tau$, we obtain

$$t - 1 \leq \frac{OPT}{m} \ln\left(\frac{\phi_0}{\phi_{t-1}}\right).$$

Recall that $\phi_{t-1} \geq 1$ and $\phi_0 = |P|$, so we have a total of

$$t \leq 1 + \frac{OPT}{m} \cdot \ln|P|$$

iterations which means that the approximation ratio is no greater than $1 + \ln|P|$ and we can state the following theorem:

Theorem 5.3 *For instances with* $|c| = 1$ *for all* $c \in C$, k-PRODUCT-SERVING *can be approximated by a factor of* $(1 + \ln|P|)$ *in linear time, that is* $\mathcal{O}(|F| + |P| + l)$ *with* $l := \sum_{r \in P} |r|$ *denoting the number of links between* F *and* P.

Proof. The claimed approximation ratio is proven above. It remains to show that the algorithm runs in linear time. In our implementation in Algorithm 3, we use a vector A of lists, with $A[i]$ containing the items which are able to satisfy i different products. As there are $|P|$ different products, all items are stored in the lists $A[0], A[1], \ldots, A[|P|]$. Additionally, we introduce $n[f]$ stores in which list f is stored, i.e. $n[f] = i$ if and only if $f \in A[i]$ For all $f \in F$, $p[f]$ is a list holding the products which can be satisfied by f. In each iteration of Algorithm 3, an item f which can satisfy the maximum number of items is chosen. Then, up to m products are chosen from $p[f]$. Let r be one of the products chosen this way. As r is satisfied this way, all other items $f' \in r$ cannot satisfy this request anymore, and A, n and p have to be updated.

The initialization takes time $\mathcal{O}(|P| + m)$ for p, and $\mathcal{O}(|F| + l)$ for b, n, and A. As the second while-loop is entered at most once for every $r \in P$ and the list-operations can be implemented in constant time, we can conclude that the for-loop starting in line 15 runs in time $\mathcal{O}(|P| + l)$. In total, the implementation stated in Algorithm 3 runs in time $\mathcal{O}(|P| + |F| + l)$, which proves the claim. □

Algorithm 3 Greedy-Heuristic for one item configurations

```
1: Input: An instance of defined by sets F, P
2: Output: A vector b
3: // Initialize p, time O(|P| + l)
4: for all r ∈ P do
5:    for all f ∈ r do
6:       Add r to p[f]
7:    end for
8: end for
9: // Initialize b, n, A, time O(|F| + l)
10: for all f ∈ F do
11:    b[f] := 0
12:    Set n[f] to length of p[f]
13:    Add f to A[n[f]]
14: end for
15: for i = |P| to 1 do
16:    while A[i] not empty do
17:       Let f be the first element of A[i]
18:       b[f] := b[f] + 1
19:       // Satisfy m products by buying a box of type f
20:       cntr := 1
21:       while cntr ≤ m and p[f] not empty do
22:          cntr := cntr + 1
23:          Let r be the first element of p[f]
24:          // Remove links to r
25:          for all f′ ∈ r do
26:             i := n[f′]
27:             Delete f′ from A[i]
28:             Add f′ to A[i − 1]
29:             n[f′] := n[f′] − 1
30:             Delete r from p[f′]
31:          end for
32:       end while
33:    end while
34: end for
35: return b
```

5.2.2 Approaches for "large" m

We restrict ourselves to the problem, when m is so large that for each set of configurations chosen, it suffices to buy at most one box of every kind of item. This way, the problem boils down to finding the least cardinality subset F' of F, such that all requests can be satisfied only making use of elements of F'. We discuss several approaches to solve this problem, compare their performance on randomly generated data, and give a logarithmic approximation.

Definition 5.4 (MINCARDINALITY-PRODUCT-SERVING) Given requests P, find a minimum cardinality subset $F' \subseteq F$, such that every product $p \in P$ contains a configuration $c \in p$ with $f \in F'$ for all $f \in c$.

MINCARDINALITY-PRODUCT-SERVING corresponds to HITTINGSET, if all configurations contain exactly one element. Hence, we can follow that it is $\mathcal{NP}$-hard.

Greedy

A greedy algorithm could work the following way: In each step choose a yet unsatisfied requests and satisfy it at minimum cost. This procedure is displayed in Algorithm 4.

By considering the following example, we can see that this algorithm can perform arbitrarily bad. For $n \in \mathbb{N}$ consider the instance consisting of the items $x_{i,j}, y_i, z_i$ with $i, j \in \{1, \ldots, n\}$ and n products $p_1, \ldots, p_n$, with each p_i consisting of the two configurations $\{x_{i,1}, x_{i,2}, \ldots, x_{i,n}\}$ and $\{y_1, y_2, \ldots, y_n, z_i\}$ for $i \in \{1, \ldots, n\}$. For every product, the Algorithm 4 buys all the items of the first configuration, whereas the optimal algorithm buys the items of the second configuration. Thus, the greedy algorithm is $n^2/(2n) = n/2$-approximate on this instance.

Integer Programming Formulations and Lagrangean Relaxation of the Subproblem

MINCARDINALITY-PRODUCT-SERVING can be solved by solving the following integer linear program:

Algorithm 4 Greedy-Heuristic

```
1: Input: An instance of defined by sets F, C, P
2: Output: A feasible solution F′ of MinCardinality-Product-Serving
3: // Set of items bought
4: F′ := ∅
5: // Set of satisfied configurations
6: C′ := ∅
7: for all r ∈ P do
8:    // If request not satisfied yet
9:    if r ∩ C′ = ∅ then
10:      // Choose cheapest configuration c*
11:      c* := argmin_{c∈r} |c \ F′|
12:      // Buy missing items to satisfy c*
13:      F′ := F′ ∪ c*
14:      // Update C′
15:      C′ := {c ∈ C : c ⊆ F′}
16:   end if
17: end for
18: return F′
```

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i \in F} x_i & & & \\
\text{subject to} \quad & \sum_{c \in r} y_c \geq 1 & & \forall r \in P & (5.2)\\
& x_i \geq y_c & & \forall r \in P, c \in r, i \in c & (5.3)\\
& x_i, y_c \in \{0,1\} & & \forall i \in F, r \in P, c \in r &
\end{aligned}
$$

In this formulation $x_i = 1$ if and only if $i \in F'$ and $y_c = 1$ if c is satisfied. Inequality 5.2 ensures that for each product at least one of its configurations is chosen and Inequality 5.3 guarantees that configuration c can only be chosen if all of its items are elements of F'.

We relax the first constraint and introduce for every $r \in P$ a Lagrangean multiplier $\lambda_r \geq 0$. This leads to subproblem (LP_λ):

$$\begin{aligned}
\text{minimize} \quad & \sum_{i \in F} x_i + \sum_{r \in P} \lambda_r \left(1 - \sum_{c \in r} y_c\right) = \sum_{i \in F} x_i - \sum_{r \in P, c \in r} \lambda_r \cdot y_c + \sum_{r \in P} \lambda_r \\
\text{subject to} \quad & x_i \geq y_c && \forall r \in P, c \in r, i \in c \\
& x_i, y_c \in \{0, 1\} && \forall i \in F, r \in P, c \in r
\end{aligned}$$

The remaining problem is a special case of the provisioning problem and can be formulated as a Minimum Cut problem which can be solved in polynomial time (see [Law76] chapter 4).

Thus, we can approximate the optimal solution by solving the Lagrangean dual making use of the steepest descent method [JS04]. For each λ, we obtain a solution which covers some of the products. As the remaining polyhedron is integer, we know that the highest objective value of the Lagrangean relaxation corresponds to the value of the linear relaxation.

The other products are satisfied in a greedy fashion: For each unsatisfied product choose a configuration which can be satisfied in the cheapest way and buy the additional items needed. A full description of the method applied here is given in Algorithm 5.

The empirical performance of Algorithm 5 is monitored in Table 5.1. The first three columns state the number of products, configurations and items. The instances are sampled in the following way: Let $X_{f,c}$ for $f \in F, c \in C$ be independent Bernoulli random variables with parameter p_1 and $X_{f,c} = 1$ if and only if f is part of configuration c. Analogously, let $Y_{c,p}$ for $c \in C, p \in P$ be independent Bernoulli random variables with parameter p_2 indicating whether c is an appropriate configuration for p. The probabilities p_1 and p_2 are chosen in a way such that the E_c is the expected number of feasible configurations for each product and and that E_f denotes the expected number of items in a configuration.

The right part of Table 5.1 states the average approximation ratios and their standard deviation, for three different algorithms. Algorithm 5 outperforms the other two approaches. We see that rounding up the fractional solutions of the LP-relaxation does not produce cost-efficient integer solutions. The Greedy algorithm chooses in every iteration one product by buying the least number of items necessary. This way, it achieves competitive approximation ratios, but is outperformed by Algorithm 5 on most of the instances.

Algorithm 5 Lagrangean-Greedy-Heuristic

Input: An instance of defined by sets F, C, P
Output: A feasible solution F' of MINCARDINALITY-PRODUCT-SERVING
Compute objective value z_{LP} of linear relaxation
for all $r \in P$ **do**
  $\lambda_r := 0$
end for
repeat
  Compute objective value z_λ of optimal solution $(\tilde{x}, \tilde{y})$ of LP_λ
  // Compute gradient of Lagrangean function at λ
  for $r \in P$ **do**
    $d_r := 1 - \sum_{c \in r} \tilde{y}_c$
  end for
  // Compute step-length
  $s := (z_{LP} - z) / \sum_{r \in P} |d_r|$
  for $r \in P$ **do**
    $\lambda_r := \lambda_r + d_r \cdot s$
  end for
until $z_{LP} - z_\lambda < 0.001$
for all $r \in P$ **do**
  if $\sum_{c \in r} \tilde{y_c} < 1$ **then**
    // Choose cheapest configuration c^*
    $c^* := \text{argmin}_{c \in r} \sum_{f \in c} (1 - \tilde{x}_f)$
    // Buy missing items to satisfy c^*
    for all $f \in c^*$ **do**
      $\tilde{x}_f := 1$
    end for
    for all $c \in C$ **do**
      $\tilde{y}_c := \min_{f \in c} \tilde{x}_f$
    end for
  end if
end for
$F' := \{f \in F : \tilde{x}_f = 1\}$
return F'

$\|P\|$	$\|C\|$	$\|F\|$	E_c	E_f	LP-rounding [%]	Greedy [%]	Lag.-Heu. [%]	#inst.
40	50	500	10	12	377.37 ± 73.05	39.35 ± 16.84	21.55 ± 12.68	110
40	50	500	4	5	30.16 ± 33.1	18.79 ± 9.59	4.12 ± 5.36	108
40	50	500	8	9	240.71 ± 78.64	36.98 ± 15.71	17.49 ± 9.97	110
60	50	200	10	10	260.1 ± 36.25	32.04 ± 13	24.82 ± 12.07	300
60	50	200	10	12	213.4 ± 27.67	30.45 ± 12.03	26.43 ± 11.19	300
60	50	200	12	12	265.64 ± 33.41	33.18 ± 13.29	29.42 ± 12.88	300
60	50	200	2	3	0.85 ± 3.01	7.54 ± 5.16	0.24 ± 0.96	300
60	50	200	4	4	40.62 ± 32.43	18.71 ± 9.28	7.37 ± 6.28	300
60	50	200	6	6	161.44 ± 43.17	26.36 ± 10.63	15.93 ± 8.13	300
60	50	200	8	8	229.43 ± 34.84	30.82 ± 11.53	20.62 ± 9.18	300

Table 5.1: Performance of different algorithms on randomly generated data.

Applying Results for more General SAT-Problems

MINCARDINALITY-PRODUCT-SERVING can be seen as a generalized satisfiability problem, where each "clause" is a disjunction of conjunctions, e.g. $(x_1 \wedge x_2 \wedge x_3) \vee (x_4 \wedge x_5 \wedge x_6) \vee (x_1 \wedge x_3 \wedge x_6)$ with x_i being true if and only if $i \in F'$. In our problem, there are no negated literals. It might make sense to allow negated literals in order to apply results for more general SAT-problems. But, in general such problems are hard to approximate, which we illustrate by a reduction from MINIMUM INDEPENDENT DOMINATING SET, which cannot be approximated within $|V|^{1-\varepsilon}$ for any $\varepsilon > 0$ in polynomial time, unless $\mathcal{P} = \mathcal{NP}$ (see [Hal93]).

Definition 5.5 (Minimum Independent Dominating Set) For a given graph $G = (V, E)$, the goal of the MINIMUM INDEPENDENT DOMINATING SET problem is to find an independent dominating set of minimum cardinality for G, i.e., a subset $V' \subseteq V$ such that for all $u \in V \setminus V'$ there is a $v \in V'$ for which $(u, v) \in E$, and such that no two vertices in V' are joined by an edge in E.

Let $G = (V, E)$ be an arbitrary graph with n vertices and m edges. For every $i \in V$ we have a literal x_i indicating if i is part of the independent dominating set. For every $i \in V$ with neighbors $(j_1, \ldots j_{n_i})$ we have a clause

$$\left((x_i \wedge \bar{x}_{j_1} \wedge \bar{x}_{j_2} \wedge \cdots \wedge \bar{x}_{j_i}) \vee (\bar{x}_i \wedge x_{j_1}) \vee (\bar{x}_i \wedge x_{j_2}) \vee \cdots \vee (\bar{x}_i \wedge x_{j_{n_i}})\right)$$

The first configuration expresses the possibility that i is member of V', which means that all nodes adjacent to i cannot be part of V'. Otherwise,

one of i's neighbors has to be part of V' which is expressed by the other configurations.

Every feasible truth assignment with minimum $\sum_{i \in V} x_i$ corresponds to a solution of MINIMUM INDEPENDENT DOMINATING SET

Transformation of Clauses

Since we can transform the clauses of the given form to clauses in "standard form" by de Morgan's rules, e.g.

$$\begin{aligned}(x_1 \wedge x_2 \wedge x_3) \vee (x_4 \wedge x_5 \wedge x_6) \Leftrightarrow \\ (x_1 \vee x_4) \wedge (x_1 \vee x_5) \wedge (x_1 \vee x_6) \\ \wedge (x_2 \vee x_4) \wedge (x_2 \vee x_5) \wedge (x_2 \vee x_6) \\ \wedge (x_3 \vee x_4) \wedge (x_3 \vee x_5) \wedge (x_3 \vee x_6)\end{aligned}$$

obtaining $M := \sum_{r \in P} \prod_{j \in r} c'_j \leq \sum_{r \in P} |F|^{|r|}$ "regular clauses" with $c'_j \leq |c_j| \leq |F|$ being the number of different items corresponding to configuration j.

Then, MINCARDINALITY-PRODUCT-SERVING can be modeled as a hitting set problem and we can approximate it by $1 + \ln(M) \leq 1 + \ln(|P||F|^k) = 1 + \ln(|P|) + k \ln(|F|)$ (see [Joh74]).

5.2.3 The General Case

In this subsection we exploit the approximation stated in Subsection 5.2.2 in order to approximate k-PRODUCT-SERVING.

Lemma 5.6 *If for each product every configuration has the same amount of items and we have a c-approximation* ALG *with running time $\mathcal{O}(r)$ for* MINCARDINALITY-PRODUCT-SERVING*, we can derive a $(c+1)$-approximation* ALG *$'$ for the generalized problem with running time $\mathcal{O}(r + |F| + |P|)$.*

Proof. Apply ALG to find a c-approximation h for the minimum number of different items necessary to serve all requests. Then we know that the optimal algorithm requires at least h/c different kinds of items. Let T be the truth assignment corresponding to the solution of ALG on the given instance and $F' \subseteq F$ be the corresponding set of items which are needed due to ALG in order to serve all requests. There is at least one possible configuration for

each product which can be served by the chosen boxes. We pick one arbitrary configuration for each product. This way, we can determine, how many items have to be ordered and call this value $b(f)$ for every item $f \in F$. Clearly, $b(f) = 0$ for $f \in F \setminus F'$. Let $F_1 := \{f \in F' : b(f) \leq m\}$ be the set of items for which ALG $'$ only buys a single box. The cost of ALG $'$ can be estimated as

$$\begin{aligned}
\text{ALG}\,' &= \sum_{f \in F'} m \cdot \lceil \frac{b(f)}{m} \rceil = \sum_{f \in F_1} m \cdot \lceil \frac{b(f)}{m} \rceil + \sum_{f \in F' \setminus F_1} m \cdot \lceil \frac{b(f)}{m} \rceil \\
&\leq |F_1| \cdot m + \sum_{f \in F' \setminus F_1} (b(f) + m) \\
&= |F'| \cdot m + \sum_{f \in F' \setminus F_1} b(f) \\
&= h \cdot m + \sum_{f \in F' \setminus F_1} b(f) \\
&\leq c \cdot \text{OPT} + \sum_{f \in F'} b(f) \qquad (5.4) \\
&\leq c \cdot \text{OPT} + \text{OPT} = (c+1) \cdot \text{OPT} \qquad (5.5)
\end{aligned}$$

Inequation (5.4) holds, since the optimal algorithm requires at least h/c different kinds of items, which means that it has to buy at least h/c different boxes with total cost hm/c. As every configuration of a product asks for the same amount of items, the total number of items required is always the same. For this reason, $\text{OPT} \geq \sum_{f \in F'} b(f)$ which means that (5.5) is true.

Since the algorithm needs to apply ALG once and the rest of the assignment can be done in linear time, the given running time estimation follows. □

Therefore we have a $2+\ln(|P|)+k\ln(|F|) = \mathcal{O}(\ln(|P|)+\ln(|F|))$-approximation for k-PRODUCT-SERVING.

5.3 Online Results

In this section, we examine the online version of k-PRODUCT-SERVING. In this version an instance consists of a sequence of n products $\sigma = (p_1, p_2, \ldots p_n)$. Whenever a request p_i is being released, it has to be decided by which of its configurations $c \in p_i$ it is satisfied and consequently for which of the items new boxes have to be bought. The following Lemma gives a lower bound on

the competitivity of deterministic online algorithms which holds even for the special case considered in Subsection 5.2.1.

Lemma 5.7 *If no additive constant is allowed, no algorithm can be better than* m*-competitive*

Proof. Let $|F| = 2m - 1$. The online algorithm is being offered a request r_1 consisting of m different items F_1. Since there are no items in stock, one of the boxes has to be ordered. Let f_1 be the box chosen by the algorithm. Then the algorithm is being offered a request r_2 consisting of the items which had just been offered but not bought and a new one which was not asked for yet. This procedure is repeated m times and the algorithm always has to order a box which results in overall costs of m^2 for the online algorithm. Since F_1 contains m different items and only one item has been removed within every step, there has to be at least one item f which was asked for in every of the m requests. Thus, the optimal offline algorithm can get along with only a single order of cost m. This results in competitivity of at least m for every deterministic algorithm. □

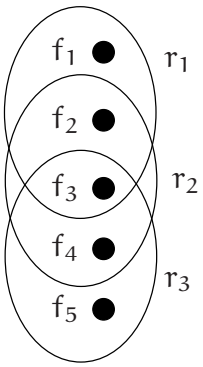

Figure 5.1: Example for $m = 3$

See Figure 5.1, which is the case for $m = 3$, where ALG buys boxes of f_1, f_2 and f_3 whereas the optimal algorithm only buys a box of f_3.

For a given sequence of requests $\sigma = (p_1, p_2, \ldots p_n)$, define $l_\sigma := \sum_{i=1}^{n} \min_{c \in p_i} |c|$ to be the minimum number of items necessary to serve σ. As the optimal algorithm has to reorder at least $\lceil l_\sigma / m \rceil$ times, we can estimate its cost by $\text{OPT} \geq \lceil l_\sigma / m \rceil \cdot m \geq l_\sigma$. We consider the algorithm GREEDY that always chooses for every product a least cardinality configuration and only reorders whenever the stock for a specific item is empty.

Lemma 5.8 *The algorithm* GREEDY *that chooses for each product* $p_i \in \sigma$ *a configuration* $c \in p_i$ *of minimum cardinality is* m*-competitive.*

Proof.

$$\frac{\text{GREEDY}}{\text{OPT}} \leq \frac{l_\sigma \cdot m}{\lceil \frac{l_\sigma}{m} \rceil \cdot m} \leq m.$$

□

Lemmas 5.7 and 5.8 prove that the greedy algorithm stated above provides the best competitive ratio possible for deterministic algorithms. In the remainder of this chapter we will show that its competitivity can be even better for special cases.

As this algorithm has at most $m-1$ items of every kind in stock, we have $\text{GREEDY} \leq l_\sigma + |F| \cdot (m-1)$, and we can estimate its competitivity in the following way:

$$\frac{\text{GREEDY}}{\text{OPT}} \leq \frac{l_\sigma + |F| \cdot (m-1)}{\lceil \frac{l_\sigma}{m} \rceil \cdot m} \leq \frac{l_\sigma + |F| \cdot (m-1)}{l_\sigma} = 1 + \frac{|F| \cdot (m-1)}{l_\sigma} \tag{5.6}$$

Hence, for $\epsilon > 0$ the greedy algorithm is $1+\epsilon$-competitive for any sequence with $l_\sigma \geq \frac{|F| \cdot (m-1)}{\epsilon}$. Equation (5.6) can be restated as

$$\text{GREEDY} \leq 1 \cdot \text{OPT} + |F| \cdot (m-1)$$

which shows that the greedy algorithm is 1−competitive if we allow an additive constant of $|F| \cdot (m-1)$.

Chapter 6

A Monotone Approximation Algorithm for Scheduling with Precedence Constraints

6.1 Introduction

Internet users and service providers act selfishly and spontaneously, without an authority that monitors and regulates network operation in order to achieve some social optimum such as minimum total delay. An interesting and topical question is how much performance is lost because of this. This generates new algorithmic problems, in which we investigate the cost of the lack of coordination, as opposed to the lack of information (online algorithms) or the lack of unbounded computational resources (approximation algorithms).

There has been a large amount of previous research into approximation and online algorithms for a wide variety of computational problems, but most of this research has focused on developing good algorithms for problems under the implicit assumption that the algorithm can make definitive decisions which are always carried out. On the internet, this assumption is no longer valid, since there is no central controlling agency. To solve problems which occur, e.g., to utilize bandwidth efficiently (according to some measure), we now not only need to deal with an allocation problem which might be hard enough to solve in itself, but also with the fact that the entities that we are dealing with (e.g. agents that wish to move traffic from one point to the other) do not necessarily follow our orders but instead are much more likely to act selfishly in an attempt to optimize their private return (e.g. minimize their latency).

Mechanism design is a classical area of research with many results. Typically, the fundamental idea of mechanism design is to design a game in such a way that truth telling is a dominant strategy for the agents: it maximizes the profit for each agent individually. That is, each agent has some private data that we have no way of finding out, but by designing our game properly we can induce them to tell us what that is (out of well-understood self-interest), thus allowing us to optimize some objective while relying on the truthfulness of the data that we have. This is done by introducing *side payments* for the agents. In a way, we reward them (at some cost to us) for telling us the truth. The role of the mechanism is to collect the claimed private data (bids), and based on these bids to provide a solution that optimizes the desired objective, and hand out payments to the agents. The agents know the mechanism and are computationally unbounded in maximizing their utility.

The seminal paper of Archer and Tardos [AT01] considered the general problem of one-parameter agents. The class of one-parameter agents contain problems where any agent i has a private value t_i and his valuation function has the form $w_i \cdot t_i$, where w_i is the work assigned to agent i. Each agent makes a bid depending on its private value and the mechanism, and each agent wants to maximize its own profit. The paper [AT01] shows that in order to achieve a truthful mechanism for such problems, it is necessary and sufficient to design a *monotone* approximation algorithm. An algorithm is monotone if for every agent, the amount of work assigned to it does not increase if its bid increases. More formally, an algorithm is monotone if given two vectors of length m, b, b' which represent a set of m bids, which differ only in one component i, i.e., $b_i > b'_i$, and for $j \neq i$, $b_j = b'_j$, then the total size of the jobs (the work) that machine i gets from the algorithm if the bid vector is b is never higher than if the bid vector is b'.

Using this result, monotone (and therefore truthful) approximation algorithms were designed for several classical problems, like scheduling on related machines to minimize the makespan, where the bid of a machine is the inverse of its speed [AT01, Arc04, APPP04, AAS05, Kov05], shortest path [AT02, ESS04], set cover and facility location games [DMV03], and combinatorial auctions [LOS99, MN02, APTT03].

6.2 Problem Definition and Preliminaries

In this chapter, we consider the problem of scheduling jobs in a multiprocessor setting where there are precedence constraints between tasks, and where the

performance measure is the makespan, the time when the last task finishes. We denote the number of processors by m and the number of jobs by n. This is a classic scheduling problem considered by Graham in his seminal paper [Gra66] where he showed that list scheduling produces a $(2-\frac{1}{m})$-approximate solution on identical machines. We consider the version where the machines are related: each machine has a speed at which it runs, which does not depend on the job being run.

Denote the size of job j by p_j ($j = 1,\ldots,n$). Denote the speed of machine i by s_i ($i = 1,\ldots,m$). In our model, each machine belongs to a selfish user. The private value (t_i) of user i is equal to $1/s_i$, that is, the cost of doing one unit of work. The load on machine i, L_i, is the total size of the jobs assigned to machine i divided by s_i. The profit of user i is $P_i - L_i$, where P_i is the payment to user i by the payment scheme defined by Archer and Tardos [AT01].

Our goal is to minimize $\max_i L_i$. This problem is $\mathcal{NP}$-complete in the strong sense [GJ79] even on identical machines and without precedence constraints.

6.2.1 Previous Results (Non-Selfish Machines)

Jaffe [Jaf80] presented an algorithm for this problem with approximation ratio of $\mathcal{O}(\sqrt{m})$. This was later improved to $\mathcal{O}(\log m)$, first by Chudak and Shmoys using a linear programming relaxation [CS99] and then by Chekuri and Bender with a combinatorial algorithm [CB01], using a new and more involved lower bound for the optimal makespan.

6.2.2 Our Result

We present a monotone approximation algorithm based on Jaffe [Jaf80] which achieves an approximation ratio of $\mathcal{O}(m^{2/3})$. Throughout the chapter, we assume that the jobs are sorted in order of non-increasing size ($p_1 \geq p_2 \geq \cdots \geq p_n$), and the machines are sorted in a fixed order of non-decreasing bids (i.e. non-increasing speeds, assuming the machine agents are truthful, $s_1 \geq s_2 \geq \cdots \geq s_m$).

6.3 Algorithm

Our algorithm works as follows. For simplicity of presentation, we assume that $m^{2/3}$ is an integer.

1. For a given bid vector $b = (b_1, \ldots, b_m)$, normalize such that the largest speed (smallest bid) is 1.

2. Ignore all machines with speed less than α, where $\alpha < 1$ is a parameter to be fixed later.

3. If at most $m^{2/3}$ machines remain, assign all jobs to the fastest machine. Extend the partial ordering given by the precedence constraints to a complete ordering and run the jobs in this order.

4. If $i > m^{2/3}$ machines remain, consider all the schedules produced by List Scheduling on $m^{2/3}+1, \ldots, i$ *identical* machines and use the schedule that minimizes the **maximum load** (i.e. **not** necessarily the makespan!). For this schedule, reorder the job loads such that the ith largest load ends up on the ith fastest machine according to the bids.

6.4 Analysis

Theorem 6.1 *This algorithm is monotone.*

Proof. If a machine that receives no load becomes slower (increases its bid), it will still receive zero load. If it becomes faster, it might get some load, whereas previously it did not get any load.

If a machine which is not the fastest but which receives some load becomes slower, it will move down in the speed ranking of the machines, progressively getting less and less load until finally it gets zero load. If such a machine becomes faster, it moves up in the speed ranking until it is the fastest and receives the most load.

If the machine which is already fastest becomes even faster, this might lead to some other machines being dropped from consideration. If *only* the fastest machine remains (that is, there are at most $m^{2/3}$ machines with speed at least α times the maximum speed), it clearly gets more load than before, because it now gets all the load.

Otherwise, the *largest load* (which is the load on the fastest machine, that we are considering) does not decrease, because our algorithm considers all options of using $m^{2/3+1}, \ldots, i$ machines where i is the number of machines that are not ignored with the old speeds. Thus if the largest load is smaller with the new amount of machines, we would have used this amount of machines earlier even though we had more machines available.

Conversely, if the fastest machine becomes slower, then if it previously had all the load there is nothing to show. Else, similar to before we find that the maximum load (which is what is assigned to the fastest machine) can only decrease, because our algorithm checks at least as much possibilities than before. We can use this reasoning until the fastest machine becomes the second fastest machine, and then we can use the reasoning above (for other machines which become slower). □

Theorem 6.2 *For the right choice of α, this algorithm has an approximation ratio of $\mathcal{O}(m^{2/3})$.*

Proof. Scale the job sizes such that the optimal makespan is 1. If our algorithm uses only one machine, then the last $m - m^{2/3}$ machines all have speed at most α. Thus the optimal load on the first $m^{2/3}$ machines is at most 1, and the optimal load on the remaining machines is at most α. Furthermore, there are no gaps in the schedule produced by our algorithm. Thus it has a makespan of at most

$$m^{2/3} + (m - m^{2/3})\alpha, \tag{6.1}$$

compared to an optimal makespan of 1.

If our algorithm uses i machines, we have that the makespan on *identical* machines is at most $1 + (m-1)/i$ times optimal according to Graham [Gra66]. This expression is maximized for $i = m^{2/3} + 1$, which is the smallest value of i that our algorithm uses (besides 1).

Since the algorithm uses machines of speeds at least $\alpha < 1$ instead of machines of speed 1, the actual makespan is at most

$$x = \frac{1}{\alpha}\left(1 + \frac{m-1}{m^{2/3}+1}\right) \tag{6.2}$$

times the optimal makespan on identical machines *of speed 1*, and therefore certainly at most x times the optimal makespan on the actual (slower) machines. We have

$$\lim_{m\to\infty} x = m^{1/3}/\alpha. \tag{6.3}$$

Balancing (6.1) and (6.3) for α, we find an approximation ratio of $\phi m^{2/3}$ for $\alpha = 1/(\phi m^{1/3})$ and $m \to \infty$, where $\phi = 1.618\ldots$ □

6.5 Open Questions

An obvious open question is to improve this approximation ratio. However, finding a better approximation ratio in the context of selfish machines does

not seem easy. In particular, the approach of Chudak and Shmoys [CS99] does not seem suitable because we do not know how the output of the linear programming relaxation changes when the speeds of the machines change. On the other hand, the lower bound introduced by Chekuri and Bender [CB01] is a complicated formula of the speeds, for which it is also not easy to analyze the change when one of these speeds changes.

Chapter 7

VDP with Limited Tour Length

7.1 Introduction

The German Automobile Club ADAC (*Allgemeiner Deutscher Automobil-Club*) maintains a heterogeneous fleet of service vehicles in order to assist people whose cars break down on their way. Service requests arrive online and are handled in five help centers spread over Germany. The goal is to provide low operational costs and a good quality of service. The general dispatching problem at ADAC is *online* (i.e., decisions have to be made on the basis of incomplete data). In this chapter we are concerned with the solution of the following *offline-subproblem* (snapshot-optimization): compute an optimal dispatch for all currently known requests subject to all operational constraints. The real-world instances to solve are large-scale exhibiting up to 700 requests and 200 units.

The offline subproblem is currently used at ADAC for an automated dispatching system on the basis of cost-reoptimization. This means that a current dispatch is maintained, which contains all known yet unserved requests and which is near optimal on the basis of the current data; whenever a unit becomes idle its next request is read from the current dispatch; at each event (new request, finished service, etc.) the dispatch is updated by a reoptimization run.

A feasible current dispatch for all known requests and available service vehicles is a partition of the requests into tours for units and contractors such that each request is in exactly one tour and each unit drives exactly one tour so that the cost function is minimized. Cost contributions come from driving costs for units, fixed service costs per requests for contractors, and a strictly convex

lateness cost for the violation of soft time windows at each request (currently quadratic). The latter cost structure is chosen so as to avoid large individual waiting times for customers. For details we refer to [KRT02a, HKR05, KRT02b].

Under high load situations, some of these requirements are relaxed for the benefit of a higher number of events served per time unit. In contrast to the general case, in which lots of different cost parameters have to be taken into account, only the total distance driven is of interest.

As the serving of a request takes approximately 20 minutes, it does not make sense to consider tours which are too long. For example a customer which is planned to be served in the sixth place could wait for about two hours until his car is fixed. But within this time, it is very likely that the whole plan has been changed, because of other requests which came up and changed the whole dispatching problem to such a great extent that the customer might be served by a different unit at a different time. Situations like this happen quite often, as we are facing an online problem. For this reason we impose an upper bound k on the number of events served per unit.

In the next section, the exact setting of the problem and the notation used in this chapter are introduced.

7.2 Problem Definition and Preliminaries

We are given a finite set of service vehicles (or units) U, a set of requests (or events) E and a metric distance function $d : (U \cup E) \times (U \cup E) \to \mathbb{R}^+$. A *tour* consists of a unit $u \in U$ and a sequence of requests $(e_{u,1}, e_{u,2}, \ldots, e_{u,h(u)})$, which are visited by vehicle u in the given order. We will denote such a tour by a vector $\big(u, e_{u,1}, e_{u,2}, \ldots, e_{u,h(u)}\big)$. The cost $c(e_i, e_j)$ of driving from e_i to e_j corresponds to the metric distance $d(e_i, e_j)$ between the two points.

Definition 7.1 *(Vehicle Dispatching Problem,* VDP-*k)*
Given requests E, units U, costs c as above and a number $k \in \mathbb{N}$ such that $|E| \leq k|U|$, the vehicle dispatching problem VDP-k consists of finding a tour $\big(u, e_{u,1}, e_{u,2}, \ldots, e_{u,h(u)}\big)$ for each unit $u \in U$ which serves $h(u) \leq k$ requests, such that each request is served in exactly one tour and such that the total cost of the tours is minimized.

7.2.1 Previous Work

This problem is related to *metric multi-depot vehicle routing problems (MDVRP)* and to *metric k-customer vehicle routing problems*. In a multi-depot vehicle routing problem a fleet of vehicles located at more than one depot are to serve locally dispersed customers such that the vehicles return to one of the depots and the transportation costs are minimized. The difference to our problem is that we do not want our service units to return to their home positions, since by the time the assigned requests are served, new requests have arrived that are to be assigned to service units in the next iteration. The multi-depot vehicle routing problem has been shown to be $\mathcal{NP}$-hard for more than one depot [BCG87].

In the metric k-*customer vehicle routing problem(k-VRP)* all vehicles are based at one depot and are required to serve at most k customers each such that the transportation costs are minimized. It is known that the metric 2-customer vehicle routing problem is polynomially solvable, since it can be transformed to a minimum matching problem, whereas for $k \geq 3$ the metric k-VRP is $\mathcal{NP}$-hard, which was shown by Haimovich and Rinnooy Kan [HRK85].

For our problem, we know that serving at most one customer is easy. Again the problem can be transformed to a matching problem and solved efficiently. In contrast, Dischke [Dis04] showed that the problem becomes $\mathcal{NP}$-hard for $k \geq 3$ and Krumke et al. [KSVWar] showed later on that the same applies for the case $k = 2$.

7.2.2 Our Results

In Section 7.3 we take a look at the algorithm BESTINSERTION which has been used so far and give lower bound of $2^{|U|-1} + 1$ on its approximation ratio. As this algorithm behaves poorly in the worst case we present in Section 7.4 the algorithm MATCH-DISPATCH and prove that it is a $(2k-1)$-approximation for the metric VDP-k with running time $\mathcal{O}(n^3)$. For the case that there is no restriction on the amount of requests served per unit, which means that k equals the total number of requests, we provide in Section 7.5 a $2 - 1/|k|$-approximation which works similar to the Double-Tree-Algorithm for the metric TSP. In Section 7.6 we state an integer linear program based on an arc-based network flow problem with budget constraints which solves VDP-k exactly. Additionally we give a cutting plane with corresponding heuristic separation algorithm which improves the formulation and thus accelerates the speed of solving the problem. Finally, in Section 7.7 we have a look at the quality of the solutions

given by MATCH-DISPATCH and discuss our computational experiments and the numerical results obtained thereby.

7.3 BestInsertion and a Lower Bound on its Approximation Ratio

So far, solutions for this problem have been computed by a heuristic called BESTINSERTION. This heuristic starts with an empty tour for each unit. The requests are considered one by one and each request is added to a tour in a way that the overall costs increase by the smallest amount and no tour covers more than k events. This algorithm cannot guarantee a satisfactory approximation ratio as one can see by the following lemma:

Lemma 7.2 BESTINSERTION *is not better than* $2^{|U|-1}+1$*-approximate.*

Proof. We consider an example, where all units and requests are located on the real line. Let the units $U = \{u_1, u_2, \ldots, u_m\}$ be on the positions $-1-\epsilon, 1, 2, 4, \ldots, 2^{m-2}$ with $\epsilon > 0$ be arbitrarily small. Furthermore, there are k requests at position 0 and for $i = 0, 1, 2, \ldots, m-2$ there are additional k requests at position 2^i. If the requests are considered by BESTINSERTION in ascending order of their coordinates, then the first k requests are located at position 0 and the algorithm constructs a tour, which covers all of these requests by u_2. The next k requests are at position 1, and the algorithm assigns them to u_3 for a total cost of 1 and so on. Finally, there are k requests left at position 2^{m-2} which forces the only remaining unit u_1 to serve these requests incurring costs of $2^{m-2}+1+\epsilon$. Overall, this routing costs $2\cdot 2^{m-2}+1+\epsilon = 2^{|U|-1}+1+\epsilon$ whereas the optimal routing would have incurred costs of $1+\epsilon$ (see Figures 7.1 and 7.2).

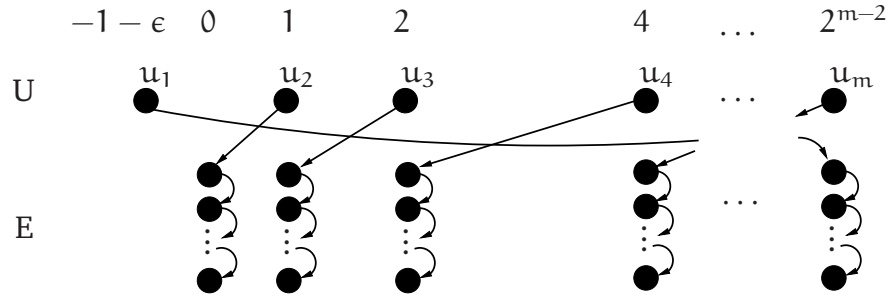

Figure 7.1: The solution obtained by BEST-INSERTION

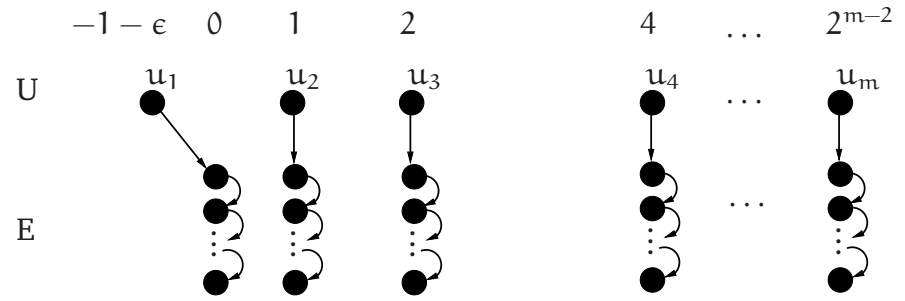

Figure 7.2: The optimal solution

□

Corollary 7.3 BESTINSERTION *is not better than* $0.5 \cdot \sqrt[k]{2}^{|E|} + 1$*-approximate.*

Proof. Follows directly from Lemma 7.2 and $k \geq |E|/|U|$. □

For this reason, we look for an algorithm, which guarantees a better approximation ratio.

7.4 A $(2k-1)$-Approximation

We define the auxiliary graph $G = (V, A)$ with $V = E \cup U \cup \{s, t\}$ and $A = A_1 \cup A_2 \cup A_3$, with $A_1 := \{s\} \times E$, $A_2 := E \times U$, $A_3 := U \times \{t\}$. Define capacities $u : A \to \mathbb{R}$ with $u(a) := k$ for all $a \in A_1$ and $u(a) := 1$ for all $a \in A_2 \cup A_3$. Define costs $c : A \to \mathbb{R}$ with $c((u, e)) := d(u, e)$ for all $a \in A_2$ and $c(a) = 0$ for all $a \in A_1 \cup A_3$. We can find an integral maximal $s-t$–flow of minimum cost in $\mathcal{O}(n^3)$ time. This flow corresponds to an assignment of units to events such that every request is assigned to at most one unit and that each unit is assigned to at most k requests. For all $u \in U$ let $E_u := \{e_1, e_2, \ldots, e_h\} \subseteq E$ be the set of requests assigned to u ordered such that $d(o_u, e_i) \leq d(o_u, e_{i+1}) i = 0, \ldots, h-1$. Our heuristic assigns these requests in the given order to u.

Let $M := \{(o_u, e) | \forall u \in U, e \in E_u\} \subseteq A_2$ be the set of arcs corresponding to the assignment chosen by MATCH-DISPATCH, $D \subseteq A$ be the set of arcs corresponding to the solution obtained thereby and $O \subseteq A$ be the set of arcs chosen by the optimal solution. Then, we can state the following propositions:

Lemma 7.4

$$c(D) \leq (2 - \frac{1}{k}) \cdot c(M)$$

Algorithm 6 Match-Dispatch

1: **Input:** A set of units U, a set of requests E, a metric weight function $c : (U \cup E) \times (U \cup E) \to \mathbb{R}^+$.
2: **Output:** a set of tours $\mathcal{T}$
3: $\mathcal{T} := \emptyset$
4: Construct auxiliary graph G
5: Compute maximal $s-t$–flow of minimum cost
6: **for all** $u \in U$ **do**
7: Let $E_u := \{e_1, e_2, \ldots, e_h\}$ be the set of requests assigned to u
8: $\mathcal{T} := \mathcal{T} \cup (u, e_1, e_2, \ldots, e_h)$
9: **end for**
10: **return** $\mathcal{T}$

Proof. For all $u \in U$ let $E_u = (e_1, \ldots, e_{h(u)})$ be the events covered by u in the given order. Recall that $h(u) \leq k$.

$$\begin{aligned}
c(D) &= \sum_{u \in U} \left(d((o_u, e_1)) + \sum_{i=2}^{h(u)} d((e_{i-1}, e_i)) \right) \\
&\leq \sum_{u \in U} \left(d((o_u, e_1)) + \sum_{i=2}^{h(u)} c((e_{i-1}, o_u)) + c((o_u, e_i)) \right) \\
&= \sum_{u \in U} \left(d((o_u, e_1)) + \sum_{i=2}^{h(u)} d((o_u, e_{i-1})) + \sum_{i=2}^{h(u)} d((o_u, e_i)) \right) \\
&= \sum_{u \in U} \left(2 \cdot (\sum_{i=1}^{h(u)-1} d((o_u, e_i))) + d((o_u, e_{h(u)})) \right) \\
&= 2 \cdot c_{OPT}(M) - \sum_{u \in U} \left(d((o_u, e_h)) \right) \\
&\leq (2 - \frac{1}{k}) \cdot c(M)
\end{aligned}$$

□

Lemma 7.5

$$k \cdot c(O) \geq c(M)$$

Proof. For all $u \in U$ let $E_u = (e_1, \ldots, e_h)$ be the events covered by u in the

given order. Recall that $h \leq k$.

$$\begin{aligned} k \cdot c(O) &= k \cdot \left(\sum_{u \in U} d((o_u, e_1)) + \sum_{i=2}^{h} d((e_{i-1}, e_i)) \right) \\ &\geq \cdot \sum_{u \in U} \sum_{j=1}^{|E_u|} \left(d((o_u, e_1)) + \sum_{i=2}^{j} d((e_{i-1}, e_i)) \right) \\ &\geq \cdot \sum_{u \in U} \sum_{j=1}^{|E_u|} d((o_u, e_{u,j})) \\ &\geq c(M) \end{aligned}$$

□

Observation 7.6 $k \cdot c(O) = c(M)$ *can only hold if for all* $u \in U$ *the distances between the elements of* E_u *are zero.*

Theorem 7.7 MATCH-DISPATCH *is a* $2k-1-$ *approximation of the metric VDP-*k.

Proof.

$$\frac{c(D)}{c(O)} \leq \frac{(2 - \frac{1}{k}) \cdot c(M)}{\frac{1}{k} \cdot c(M)} = 2k - 1$$

□

Lemma 7.8 *For* $k = 2$ *the Approximation ratio of Theorem 7.7 is tight.*

Proof. Consider the example shown in Figure 7.4. The heuristic assigns the events e_1, e_2 to u_1 and e_3, e_4 to u_2 causing costs of $2 \cdot 3 = 6$, whereas an optimal solution would assign the events e_1, e_3 to u_1 and e_2, e_4 to u_2 causing costs of $2 \cdot (1 + \epsilon) = 2 + 2 \cdot \epsilon$. For $\epsilon \to 0$ the approximation rate converges to 3. □

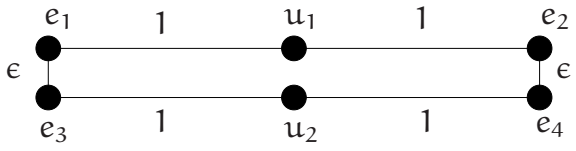

Figure 7.3: The graph for the proof of Theorem 7.8

Observe that this example works for the special case of Euclidean distances, too. Lemma 7.8 can be generalized for arbitrary k and therefore it is not possible to analyze it more tightly.

Lemma 7.9 *The approximation ratio of Theorem 7.7 is tight for arbitrary* k.

Proof. Let $U := \{u_i | i = 1, \ldots, k\}$ be the set of units and $E := \{e_{i,j} | i, j = 1, \ldots, k\}$ the set of requests. Let c be the metric closure induced by the weight function c', with $c'(u_i, e_{i,j}) = 1$ and $c'(e_{j,i}, e_{j+1,i}) = \epsilon$ for all $j = 1, \ldots k-1$, $i = 1, \ldots k$ (see Figure 7.4). Analogously to the proof of Lemma 7.8 MATCH-DISPATCH chooses to assign $e_{i,j}$ to u_i for all $i, j = 1, \ldots k$ incurring costs of $k \cdot (2 \cdot k - 1)$ whereas it is possible to get hand on a better solution by assigning $e_{i,j}$ to u_j for all $i, j = 1, \ldots k$ incurring costs of $k \cdot (1 + (k-1) \cdot \epsilon)$. Thus, MATCH-DISPATCH is not better than $(2k-1)$-approximate on arbitrary metric systems. □

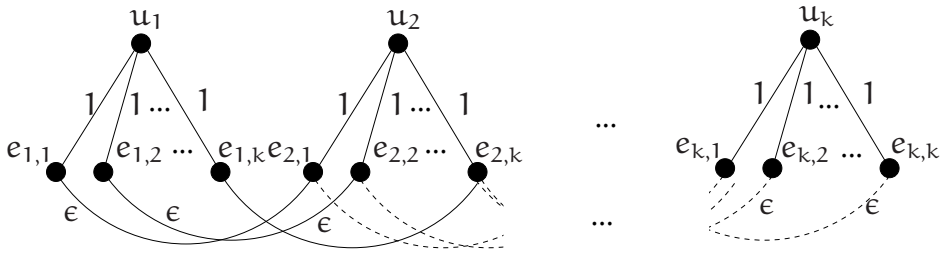

Figure 7.4: The graph for the proof of Lemma 7.9

7.5 A $(2-1/|k|)$-Approximation for the Metric Case with $k = |E|$

In this case, we can provide another algorithm derived from the double-tree approximation for the metric TSP, which is much better than the one of the previous section.

Therefore, we construct an undirected simple graph $G = (V := U \cup E, R := V \times V)$ with a cost function $c : A \to \mathbb{R}^+$ with $c(i,j) = 0$ if $i, j \in U$ and $c(i,j) := d(i,j)$ else. After computing a minimum spanning tree T in G, we remove the edges $U \times U$ obtaining $|U|$ connected components T_u with exactly one element $u \in U$ in each of them. These connected components form the tours. For each u let $v_u \in T_u$ be the event with the maximum distance from o_u within T_u. Let T'_u be the Graph obtained by doubling the edges of T_u. Find an Eulerian tour S in T'_u such that v_u is the last event being served for the first time within this tour. Since all of the other events have already been served u can stop at v_u.

Algorithm 7 Tree-Dispatch

Input: A set of units U, a set of requests E, a metric weight function $c : (U \cup E) \times (U \cup E) \to \mathbb{R}^+$.
Output: a set of tours $\mathcal{T}$
$\mathcal{T} := \emptyset$
$G = (V := U \cup E, R := V \times V)$
for all $u_i, u_j \in U$ **do**
  $c((u_i, u_j)) := 0$
end for
Compute minimum spanning tree T in G w.r.t c
$A := A \setminus U \times U$
for all $u \in U$ **do**
  Let T_u be the connected component of T with $u \in T_u$
  Let v_u be the event with maximum distance from o_u within T_u
  Find Eulerian Tour S_u in T'_u with v_u being the last element served
  Let P_{v_u} be the simple $o_u - v_u$-path in T_u
  $\mathcal{T} := \mathcal{T} \cup (S_u \setminus P_{v_u})$
end for
Return $\mathcal{T}$

For Algorithm 7 we need to show that:

Lemma 7.10 *It is always possible to find an Eulerian tour S_u in T'_u such that v_u is the last event being served for the first time within this tour.*

Proof. Since c is a nonnegative function, we can assume v_u to be a leaf. Let P_{v_u} be the simple $o_u - v_u$-path in T_u. We can find an Eulerian tour S_u by applying a DFS to T_u and whenever there is a decision to be made which vertex to visit next we rather pick one which is not an element of P_{v_u} than one which is an element of P_{v_u}. This way, we will have reached all other nodes in T_u prior to v_u. □

Lemma 7.11 *For all $u \in U$ it holds that*

$$c(P_{v_u}) \geq \frac{1}{|T_u|} c(T_u)$$

Proof.

$$c(T_u) \leq \sum_{v \in T_u} c(P_v) \leq \sum_{v \in T_u} c(P_{v_u}) \leq |T_u| \cdot c(P_{v_u})$$

□

Let $O \subseteq R$ be the set of arcs chosen by the optimal solution.

Lemma 7.12

$$c(T) \leq c(O)$$

Proof. By adding $|U|-1$ edges to O such that all elements of $|U|$ are connected, we can express O as a spanning tree in G. Since T is a minimum spanning tree the proposition follows directly. □

Theorem 7.13 *For* $k = |E|$ TREE-DISPATCH *is a* $2 - \frac{1}{|E|}$ *approximation of the metric VDP-*k

Proof. Due to Theorems 7.12 we can conclude:

$$\begin{aligned}
\frac{c(\mathcal{T})}{c(O)} &\leq \frac{\sum_{u \in U} c(S_u) - c(P_{v_u})}{c(T)} \\
&\overset{\text{Lem.7.11}}{\leq} \frac{\sum_{u \in U} 2 \cdot c(T_u) - \frac{1}{|T_u|} c(T_u)}{c(T)} \\
&\leq \frac{(2 - \frac{1}{|E|}) \sum_{u \in U} c(T_u)}{c(T)} \\
&= 2 - \frac{1}{|E|}
\end{aligned}$$

□

7.6 An Exact Algorithm for the VDP with Limited Tour Length

In this section, we are going to solve the problem presented in the preceding section exactly. For this reason we are going to present an integer programming formulation in Section 7.6.1. In the remainder of the chapter, we give some cutting planes in order to improve the polyhedral description. Additionally, we propose in Section 7.6.2 algorithms to separate them and show the results of some experimental computations which demonstrate the effectiveness of the constraints generated this way.

7.6.1 An Integer Programming Formulation

We consider this problem as a MinCostFlow-Problem in a time expanded network (Figure 7.5). This network consists of the nodes $G = (V' \cup V'', A)$ with $V' := \{u_1, u_2, \ldots, u_m\}$ representing the units and $V'' := \{e_{i,l} : i = 1, \ldots, n, l = 1, \ldots, k\}$ consisting of k copies of each element e_i of E. G holds an arc from u_i to $e_{j,1}$ at cost $d((o_{u_i}, e_j))$ and arcs from $e_{i,l}$ to $e_{j,l+1}$ for $l = 1, \ldots, k-1$ at cost $d((e_i, e_j))$. All arcs have capacity 1.

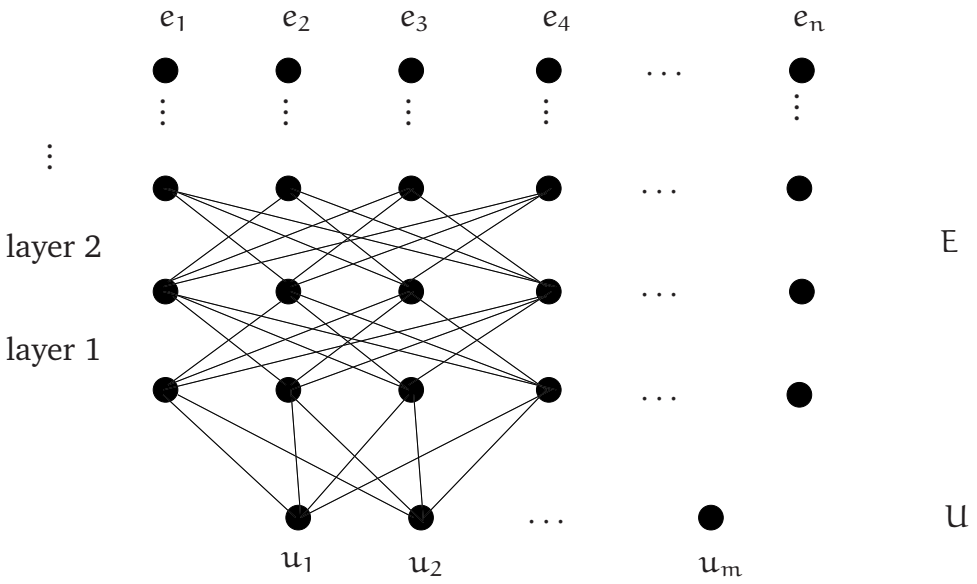

Figure 7.5: Graph for MinCostFlow representation of VDP-k

Obviously, there are no paths in this network consisting of more than k arcs and each path from u_i to $e_{j,l}$ can be seen as a tour driven by unit u_i with e_j being the lth request served. A set of m such paths provides feasible solution of VDP-k, if all of these paths start at different u_i and for all $e_j \in E$ there is exactly one of its k copies $e_{j,l}$ covered by one of these paths. This last property is hard to ensure by applying standard MinCostFlow techniques. Thus, we apply an integer programming formulation of the just described network flow problem and add an additional constraint to guarantee that each request is served exactly once:

$$\text{minimize} \quad \sum_{u\in U, i\in E} d((o_u, e_i)) \cdot y_{u,i} + \sum_{i,j\in E, l\in L} d((e_i, e_j)) \cdot x_{i,l,j} \tag{7.1}$$

$$\text{subject to} \quad \sum_{i\in E} y_{u,i} = 1 \qquad \forall u \in U \tag{7.2}$$

$$\sum_{u\in U} y_{u,i} - \sum_{j\in E} x_{i,1,j} = 0 \qquad \forall i \in E \tag{7.3}$$

$$\sum_{j\in E} x_{j,l-1,i} - \sum_{h\in E} x_{i,l,h} = 0 \qquad \forall l \in L : l > 1, i \in E \tag{7.4}$$

$$\sum_{u\in U} y_{u,i} + \sum_{j\in E, l\in L} x_{j,l,i} = 1 \qquad \forall i \in E \tag{7.5}$$

$$x_{i,l,j}, y_{u,i} \in \{0,1\} \qquad \forall i,j \in E, l \in L, u \in U \tag{7.6}$$

Here, the binary variable $y_{u,i} = 1$ if and only if there is unit of flow on arc $(u, e_{i,1})$, which corresponds to vehicle u driving directly to request i. Analogously, $x_{i,l,j} = 1$ if and only if there is one unit of flow on arc $(e_{i,l}, e_{j,l+1})$ resp. there is one unit driving directly from Request $i \in E$ to request $j \in E$ and i has been the lth request which has been served along the corresponding tour. $L := 1, \ldots k-1$ is the index set of the layers of arcs between the nodes corresponding to the events. Constraints (7.2) to (7.4) maintain the balance of the corresponding flow and constraint (7.5) guarantees that every request is served exactly once. The objective function (7.1) ensures that the optimal solution of the IP corresponds to a set of paths of minimum cost.

7.6.2 Improving the Formulation

In this subsection, we give better formulations for the set of integral solutions described by the IP above. Observe that for $|U| = 1$ the problem is a variant of the TSP. Thus, we cannot hope to find an ideal formulation for VDP-k as this implied an ideal formulation for TSP as well.

Lemma 7.14 *If $d(i,j) > 0$ for all $i,j \in E$, the optimal solution of the linear relaxation of (7.1) to (7.6) is not integral.*

Proof. We prove the claim by constructing a feasible fractional solution $(\tilde{x}, \tilde{y})$ of the linear relaxation of (7.1) to (7.6) which has an objective value not greater than the optimal solution. Furthermore, we give a sufficient condition for instances on which the objective value of $(\tilde{x}, \tilde{y})$ is strictly smaller.

For an arbitrary instance of VDP-k consider the solution generated by MATCHDISPATCH. In the first step the algorithm computes an assignment of events to units, such that at most k events are mapped to each unit and that the total distance from the events to their chosen units is minimized. Recall that in section 7.4 the arcs corresponding to this assignment in the appropriate graph are called M and their cost is denoted as $c(M)$ whereas the cost of the optimal solution is named $c(O)$.

Consider the following fractional solution of the corresponding IP with $\tilde{y}_{u,i} = 1/k$ if and only if $(u,i) \in M$ and $\tilde{x}_{i,l,i} = 1/(|L|+1) = 1/k$ for all $i \in E, l \in L$, all other entries of $(\tilde{x},\tilde{y})$ are set to 0. A visualization of such a solution is shown in Figure 7.6. Observe that the objective value of this solution is

$$c(\tilde{x},\tilde{y}) = \frac{1}{k} \cdot c(M) \overset{\text{Lem.7.5}}{\leq} c(O)$$

Recall that according to Observation 7.6 the equality in Lemma 7.5 does only hold if for all $u \in U$ the distances between the elements in E_u are zero. Consequently, in all other cases, the objective value of $(\tilde{x},\tilde{y})$ is strictly smaller than the optimal integer solution. As all the distances attain positive values, we have $c(\tilde{x},\tilde{y}) < c(O)$. As we have found a fractional solution attaining a lower objective value than the optimal integer solution we can conclude that the optimal solution is fractional as well.

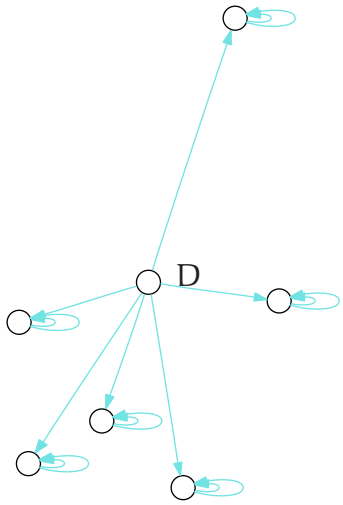

Figure 7.6: An example for the solution of the linear relaxation with all units having their home positions at point D.

□

In order to avoid solutions of that kind we add constraints (7.7):

$$x_{i,l,i} = 0 \quad \forall i \in E, l \in L. \tag{7.7}$$

Obviously, these constraints do not cut off any integer solution, but forbid solutions which possess loops of any kind. As it is possible to construct fractional solutions similar to $(\tilde{x}, \tilde{y})$ for the integer linear program which arises by adding (7.7) to (7.1) to (7.6), we introduce further cutting planes.

Let $S \subseteq E$ be a subset of the requests E. Since every tour covers at most k requests one needs at least $\lceil |S|/k \rceil$ different units to serve all the request in S from which follows that $\lceil |S|/k \rceil$ is a lower bound on the amount of flow entering S. This leads us to the following inequality:

$$\sum_{u \in U, m \in S} y_{u,m} + \sum_{i \notin S, m \in S, l \in L} x_{i,l,m} \geq \left\lceil \frac{|S|}{k} \right\rceil \quad \forall S \subseteq E \tag{7.8}$$

Since we know from inequality (7.5) that the total flow entering each node is exactly 1, we know by summing up over all the elements of S that

$$\sum_{m \in S} \left(\sum_{u \in U} y_{u,m} + \sum_{i \in E, l \in L} x_{i,l,m} \right) = |S|. \tag{7.9}$$

Subtracting Inequality (7.8) from (7.9) yields

$$\sum_{m \in S} \left(\sum_{i \in E, l \in L} x_{i,l,m} \right) - \left(\sum_{i \notin S, m \in S, l \in L} x_{i,l,m} \right) \leq |S| - \left\lceil \frac{|S|}{k} \right\rceil \quad \forall S \subseteq E$$

and finally

$$\sum_{i,j \in S, l \in L} x_{i,l,j} \leq \lfloor |S| \cdot (1 - 1/k) \rfloor \quad \forall S \subseteq E \tag{7.10}$$

This means that within a subset $S \subseteq E$ it is not allowed to have a total flow of more than $\lfloor |S| \cdot (1-1/k) \rfloor$. We call sets which do not obey (7.10) *violating*. Since there is an exponential amount of such inequalities, as the amount of subsets is exponential, we need polynomial time algorithms which find violating sets.

A possible approach would be to compute for all $h \leq |V|$ the set S_h with cardinality h which maximizes the left hand side of (7.10). If the corresponding inequalities are satisfied for all these sets, (7.10) is satisfied for all other subsets of V as well, otherwise the corresponding S_h induces a separating hyperplane.

Let $(\tilde{x}, \tilde{y})$ be a solution of the linear relaxation. The separation problem for (7.10) is solved if we can find in a complete graph with vertices E and edge weight $b([i,j]) := \sum_{l \in L}(\tilde{x}_{i,l,j} + \tilde{x}_{j,l,i})$ on the edge connecting the vertices corresponding to i and j for all $i, j \in E$ a clique of cardinality h and maximum weight with respect to b in polynomial time. Unfortunately, this problem is an instance of the weighted version of the DENSEST K-SUBGRAPH PROBLEM. This problem is $\mathcal{NP}$-hard, which can be seen by reduction from the MAXIMUM CLIQUE PROBLEM [FS97].

It might be possible that our separation problem is not $\mathcal{NP}$-hard, as the values of b obey a special structure, but there is only little hope, since the DENSEST K-SUBGRAPH PROBLEM is $\mathcal{NP}$-hard even for several special cases. For this reason, we will focus in the following on heuristic approaches to find violating sets.

Observe that for $|U| = 1$ the inequalities (7.10) correspond to the subtour inequalities of the VDP.

Lagrangean Relaxation of the DENSEST k-SUBGRAPH problem

Consider the complete graph with edge weights b as defined above. S_h can be found by solving the following integer linear program:

$$\begin{aligned} \text{maximize} \quad & \sum_{e \in E} b_e u_e & & \\ \text{subject to} \quad & \sum_{i \in V} v_i \leq h & & (7.11) \\ & u_e \leq v_i & \forall e \in E, i \in \gamma(e) & \\ & u_e, v_i \in \{0,1\} & \forall e \in E, i \in V & \end{aligned}$$

Here, the binary variables v_i, which correspond to the nodes of the complete graph, are 1 if and only if $i \in S_h$ and u_e is 1 if and only if both of the endpoints of e are in S_h. As there is no polynomial algorithm known which solves this problem we relax the cardinality constraint (7.11) and obtain the Lagrangean relaxation with Lagrangean multiplier c.

$$\begin{aligned} Z_c := \text{maximize} \quad & \sum_{e\in E} b_e u_e - c \sum_{i\in V} v_i \\ \text{subject to} \quad & u_e \le v_i && \forall e \in E, i \in \gamma(e) \\ & u_e, v_i \in \{0,1\} && \forall e \in E, i \in V \end{aligned}$$

This is a special case of the provisioning problem, which can be restated as a MinCut-Problem and hence be solved in polynomial time (see [Law76], chapter 4).

We will use this subproblem to find violating subsets. Suppose the subproblem is solved for some arbitrary c with an objective value of Z_c and variables attaining u', v', and the corresponding set S is not a violating set. Let $w(S) := \sum_{e\in E} b_e u'_e$ be the profit obtained on the edges.

$$w(S) \le \lfloor |S|(1-1/k) \rfloor$$

Furthermore, we have

$$w(S) - c \cdot |S| = Z_c \tag{7.12}$$

Due to optimality of Z_c we have for all $S' \subseteq V$

$$w(S') - c \cdot |S'| \le Z_c$$

Hence, for every $S' \subseteq V$ we have:

$$\begin{aligned} w(S') &\le Z_c + c \cdot |S'| = Z_c + c \cdot |S| - c \cdot |S| + c \cdot |S'| \\ &= w(S) + c(|S'| - |S|) \le \lfloor |S|(1-1/k) \rfloor + c(|S'| - |S|) \\ &\le |S|(1-1/k) + c(|S'| - |S|) = (1 - 1/k - c)|S| + c|S'| \end{aligned} \tag{7.13}$$

Lemma 7.15 *For* $c < 1 - 1/k$ *all* S' *with*

$$|S'| \ge |S| + \frac{1-1/k}{1-1/k-c} \tag{7.14}$$

are not violating.

Proof. Due to inequality (7.13)

$$\begin{aligned} w(S') &\leq |S|(1-1/k)+c(|S'|-|S|) = (1-1/k-c)|S|+c|S'| \\ &\overset{(7.14)}{\leq} (1-1/k-c)\left(|S'|-\frac{1-1/k}{1-1/k-c}\right)+c|S'| \\ &= (1-1/k-c)|S'|-1-1/k+c|S'| = (1-1/k)(|S'|-1) \\ &\leq \lfloor |S'|(1-1/k)\rfloor \end{aligned}$$

Which means that S' is not violating. □

Remark that we have for all $S' \subseteq V$:

$$(|S'|-1)(1-1/k) \leq \lfloor |S'|(1-1/k)\rfloor \leq |S'|(1-1/k)$$

Every set S' corresponds to a linear function with y-intercept $w(S')$ and slope $-|S'|$. Thus, the x-axis is crossed at $\frac{w(S')}{|S'|}$, which is called the *density* of S'.

Lemma 7.16 *If there are no violating subsets then* $Z_{1-1/k} \leq 0$.

Proof. If all $S' \subseteq V$ are non-violating, then:

$$w(S') \leq \lfloor |S'|(1-1/k)\rfloor \leq |S'|(1-1/k)$$

Thus we have the following upper bound on the density:

$$\frac{w(S')}{|S'|} \leq (1-1/k)$$

Let S be the set corresponding to $Z_{1-1/k}$, then

$$Z_{1-1/t} \overset{(7.12)}{=} w(S)-(1-1/k)\cdot|S| \leq |S|(1-1/k)-(1-1/k)\cdot|S| = 0$$

□

As the flow entering each node is at most 1, we have for all $S' \subseteq V$

$$w(S') \leq |S'| \text{ resp. } \frac{w(S')}{|S'|} \leq 1$$

Lemma 7.17 *If S is violating, then it has a density greater than* $1-1/k-\frac{1-/k}{|S|}$

Proof.

$$w(S) > \lfloor |S|(1-1/k) \rfloor \geq (|S|-1)(1-1/k)$$

which means for the density of S that

$$\frac{w(S)}{|S|} > 1 - 1/k - \frac{1-1/k}{|S|}$$

□

Lemmas 7.15 to 7.17 can be applied to find violating constraints. For this reason, one should compute $Z_{1-1/k}$. If it is positive, due to Lemma 7.17 a violating set has been found. After each computation 7.15 gives an upper bound on the cardinality of a violating set and Lemma 7.17 can be applied to find a lower bound for the "important" Lagrangean multipliers. Additionally, some v_i should be fixed in order not to get E as the only set which maximizes Z_c for all values of $c \leq 1 - 1/k$.

A Greedy Approach to Find Violated Constraints

Let $(\tilde{x}, \tilde{y})$ be a solution of the linear relaxation and $b([i,j])$ be defined like above. Then, for any $S \subseteq E$ let the inner flow of S be the sum of all the flow values between elements of S which can be calculated by $\frac{1}{2}\sum_{i,j\in S} b[i,j]$. We present a greedy approach, which works in the following way. Starting with a set S containing a single vertex i and an Inner Flow of 0, S is enlarged in every iteration by the vertex j whose adding to S makes the inner flow of S increase by the biggest amount. After each iteration, it is checked if S obeys (7.10) and if it does not do so, add S to $\mathcal{C}$ which is a container for violating subsets found this way. A formal description of this greedy procedure is given by Algorithm 8. Here, for any S and $j \notin S$ there is a value $Fl[j] := \sum_{i\in S, j\notin S} b[i,j]$ which stores the amount of flow between j and its neighbors in S, which makes it possible to find the best $j \in E \setminus S$ in $\mathcal{O}(|E|)$ time. The overall running time is $\mathcal{O}(|E|^3)$.

7.7 Numerical Simulation Results

For every instances, the position of units and requests where chosen randomly from a rectangle of size 2000×4000 with uniform distribution. The maximal tour length is always defined as $k := \lceil |E|/|U| \rceil$.

Algorithm 8 Greedy-Violation

Input: a set of requests E, a solution $(\tilde{x}, \tilde{y})$ of LP
Output: a collection of violating sets $\mathcal{C}$
$\mathcal{C} := \emptyset$
for all $i \in E$ **do**
  $S := \{i\}$
  $\mathrm{InnerFlow} := 0$
  for all $j \in E$ **do**
    $Fl[j] := b([i, j])$
  end for
  repeat
    Choose $j \in E \setminus S$ with $Fl[j]$ maximal
    $S := S \cup \{j\}$
    $\mathrm{InnerFlow} := \mathrm{InnerFlow} + Fl[j]$
    for all $j \in E \setminus S$ **do**
      $Fl[j] := Fl[j] + b([i, j])$
    end for
    if $\mathrm{InnerFlow} > \lfloor |S| \cdot (1 - 1/k) \rfloor$ **then**
      $\mathcal{C} := \mathcal{C} \cup S$
    end if
  until $S = E$
end for
return $\mathcal{C}$

$\|E\|$	$\|U\|$	LPgap	LPCutgap	IPtime	IPCuttime	#instances
35	10	5.6%	0.5%	6.35 s	0.8 s	20
35	15	3.6%	0.2%	0.56 s	0.17 s	20
35	5	12.6%	2.5%	318.89 s	36.09 s	20
40	10	7.3%	1%	8.24 s	3.88 s	20
40	15	3.8%	0.2%	0.76 s	0.33 s	20
40	20	2.1%	0%	0.15 s	0.04 s	20
40	5	12.5%	2.5%	1331.92 s	172.2 s	19
45	10	7.5%	1.3%	58.4 s	7.47 s	20
45	15	4.4%	0.6%	6.13 s	1.84 s	20
45	20	2.6%	0.2%	0.5 s	0.12 s	20
50	10	8.5%	2.1%	417.07 s	36.19 s	20
50	15	5.8%	0.8%	12.02 s	2.87 s	20
50	20	3.7%	0.3%	3.25 s	0.49 s	20
60	10	9.4%	3.2%	2932.35 s	440.44 s	15
60	15	7.2%	2.1%	261.69 s	35.67 s	18
60	20	4%	0.4%	33.36 s	2.99 s	20

Table 7.1: Performance of cutting planes

The performance of the cutting planes can be seen in Table 7.1. The first two columns denote the number of requests $|E|$ and the number of service units $|U|$. The average values of the integrality gaps of the linear relaxation and the linear relaxation with cutting planes for all instances with the same $|E|$ and $|U|$ is displayed in columns three and four. For ip and lp being the objective function values of the IP resp. its linear relaxation, the integrality gap is defined as $(ip - lp)/ip$. It can be seen that this gap is much smaller, when cutting planes have been applied. Consequently, an IP-solver working with Branch-&Bound can be initialized with a tighter lower bound to work with, and thus obtains optimal IP-solutions in shorter time. This effect can be seen in the fifth and sixth column, where the average amount of time spent by CPLEX solving the IPs without any cutting planes resp. with generated cutting planes is shown. Especially for instances with a high value for k, the application of cutting planes takes effect. Here, the optimal solutions can be obtained in 11% of the time.

Even though the problem can be solved exactly much faster by applying these cutting planes, the high load instances of the ADAC still have too many events and units to be tackled this way exactly. For this reason, we need to have a look at heuristics, which approximate the optimal solution to a good extent.

$\|E\|$	$\|U\|$	k	gap(MD)	gap (MD, 2opt)	#instances
10	5	2	$7.66\% \pm 0.61\%$	$1.84\% \pm 0.13\%$	20
16	8	2	$9.17\% \pm 0.56\%$	$2.37\% \pm 0.14\%$	20
20	10	2	$10.54\% \pm 0.30\%$	$0.86\% \pm 0.02\%$	10
30	15	2	$16.22\% \pm 0.87\%$	$4.35\% \pm 0.16\%$	10
40	20	2	$10.22\% \pm 0.46\%$	$1.64\% \pm 0.03\%$	10
60	30	2	$15.39\% \pm 0.33\%$	$3.76\% \pm 0.05\%$	10
20	5	4	$17.01\% \pm 1.29\%$	$3.61\% \pm 0.14\%$	20
32	8	4	$16.33\% \pm 0.81\%$	$5.95\% \pm 0.12\%$	20
40	10	4	$15.79\% \pm 0.32\%$	$6.23\% \pm 0.18\%$	10
60	15	4	$17.77\% \pm 0.12\%$	$7.24\% \pm 0.11\%$	10
30	5	6	$12.87\% \pm 0.21\%$	$3.28\% \pm 0.08\%$	20
48	8	6	$17.41\% \pm 0.41\%$	$7.73\% \pm 0.27\%$	20
60	10	6	$20.52\% \pm 1.28\%$	$8.13\% \pm 0.05\%$	10

Table 7.2: Performance of MATCHDISPATCH on the plane with $k = 2, 4, 6$

The performance of MATCHDISPATCH on the plane can be seen in Table 7.2.

The first three columns denote the number of requests $|E|$, the number of service units $|U|$ and the tour length k. The last number states the number of instances to which MATCHDISPATCH has been applied and the optimal solution has been computed. This experiments resulted in an average optimality gap of MATCHDISPATCH and its variance which is given in the fourth column. We see that the solutions found by MATCHDISPATCH are circa 17% more expensive than the optimal solutions. When we apply 2-Opt to the solutions obtained by MATCHDISPATCH, we find even better solutions which cost roughly 5% more than the optimum. Generally speaking, for those instances for which it is computationally too demanding to compute the optimal solution, MATCHDISPATCH with 2-Opt provides an attractive alternative to get a hand on almost optimal solutions in very short time.

Additionally, observe that the objective function values of the solutions found here are much better than the worst case competitivity of $2k - 1$.

Chapter 8

A Note on the k-Canadian Traveler Problem

8.1 Introduction

The shortest path problem is a well-studied problem in combinatorial optimization. Given an undirected graph $G = (V, E)$ with two nodes s and t and a cost function $d\colon E \to \mathbb{R}^+$ representing the time it takes to traverse the edges, one seeks to determine a shortest path from s to t (with respect to d). We consider the following online variant of the problem, in which some of the edges of G are blocked, and an online algorithm only learns from the blocking of an edge when reaching one of its endpoints. Whenever a blocked edge which is part of the planned route is reached, it cannot be passed, and for this reason the remaining path has to be changed. This problem is called the Canadian Traveler Problem (CTP) and has been introduced by Papadimitriou and Yannakakis [PY91].

Papadimitriou and Yannakakis [PY91] showed that it is PSPACE-complete to find an online algorithm with a bounded competitive ratio. If there is a parameter k given which bounds the number of blocked edges from above, the resulting problem is called k-Canadian Traveler Problem (k-CTP). Bar-Noy and Schieber [BNS91] studied the k-CTP and several other variants of the CTP, but they did not consider the problem from a competitive analysis point of view. Instead, they consider the worst case criterion (see [BDB94] for details) which aims at a strategy where the maximum cost is minimized.

To the best of our knowledge, there is no work concerning the competitive ratio of the k-CTP. A deterministic online algorithm ALG for k-CTP is c-

competitive, if for any input σ the total length $\text{ALG}(\sigma)$ of the s-t-path produced by ALG on input σ is at most $c\,\text{OPT}(\sigma)$, where $\text{OPT}(\sigma)$ is the length of a shortest s-t-path in G with the blocked edges removed. We show in this chapter that for every deterministic online algorithm, there is an instance of k-CTP where it cannot be better than $(2k+1)$-competitive. Additionally, we provide an easy deterministic algorithm which matches this bound. Thus, the given lower bound is sharp and our simple algorithm is optimal in terms of competitive analysis. Furthermore, we show a lower bound of $k+1$ for randomized algorithms against an oblivious adversary by applying Yao's principle.

8.2 Tight Competitiveness Bounds

Lemma 8.1 *There is no deterministic online algorithm with competitive ratio less than* $2k+1$.

Proof.

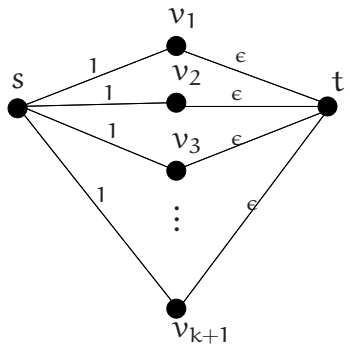

Figure 8.1: The graph G for the proof of the lower bound

Consider the graph displayed in Figure 8.1. Every deterministic algorithm corresponds to a permutation, which describes in which order the paths (s, v_i, t) for $i = 1, \ldots, k+1$ are being searched for a way to get from s to t. For each of these algorithms consider the instance, where the only passable way is the last one tried and (v_i, t) is blocked for all other i. Thus, the competitivity cannot be better than $\frac{k\cdot(1+1)+1+\epsilon}{1+\epsilon} = \frac{2k+1+\epsilon}{1+\epsilon}$ Since ϵ can attain any positive value, the lemma follows. □

In the following we provide an easy algorithm BACKTRACK which matches the above lower bound for arbitrary undirected graphs $G = (V, E)$. Consider the algorithm which tries to get from s to t by taking the shortest s-t-path P_1

(ties being broken arbitrarily). If one of the edges $e_1 \in P_1$ is being blocked, the vehicle drives back from its current location (which is an endpoint of e_1) to s and then tries to get to t by taking a shortest path P_2 in $G = (V, E_1)$ with $E_1 := E \setminus \{e_1\}$. If this path turns out to be impassable again because of some edge $e_2 \in P_2$, then the vehicle backtracks again to s and looks for the shortest path P_3 in $G = (V, E_2)$ where $E_2 := E \setminus \{e_1, e_2\}$ and so on.

Observe that the vehicle backtracks at most k times, since there are no more than k edges which are impassable simultaneously. Thus, the algorithm ends up with a path P_q where $q \leq k+1$. Since $E_1 \supset E_2 \supset \cdots \supset E_q$ we have $d(P_1) \leq d(P_2) \leq \cdots \leq d(P_q)$. Thus, the overall distance driven is less or equal to

$$\sum_{i=1}^{q-1} 2d(P_i) + d(P_q) \leq (2q-1)d(P_q) \leq (2k+1)d(P_q).$$

As the offline algorithm knows all the blocked edges in advance, it chooses P_q (or another path of the same length) to get from s to t. Thus, $\textsc{Opt}(\sigma) = d(P_q)$ and BACKTRACK is $(2k+1)$-competitive. Combining this result with Lemma 8.1 we get:

Theorem 8.2 *The simple algorithm* BACKTRACK *is* $(2k+1)$*-competitive for the* k*-CTP and this is best-possible.* □

We continue to derive a lower bound for randomized algorithms.

Lemma 8.3 *There is no randomized online algorithm against an oblivious adversary with competitive ratio less than* $k+1$.

Proof. Consider again the graph shown in Figure 8.1. We choose $i \in \{1, \ldots, k+1\}$ uniformly at random, block all edges (v_j, t) with $j \neq i$ and leave all other edges intact. So, only the path (s, v_i, t) is passable at cost $1+\epsilon$ and the expected optimal offline cost is $1+\epsilon$. Moreover, with probability $1/(k+1)$ an arbitrary deterministic online algorithm finds the path (s, v_i, t) on the ℓth trial for $\ell = 1, \ldots, k+1$. If the algorithm is successful on its ℓth try, it incurs a cost of $2\ell - 1 + \epsilon$ and, hence, its expected cost is at least

$$\begin{aligned}\frac{1}{k+1}\sum_{\ell=1}^{k+1} 2\ell - 1 + \epsilon &= \frac{1}{k+1} \cdot \big((k+1)(k+2) - (k+1) + (k+1)\epsilon\big) \\ &= k+1+\epsilon.\end{aligned}$$

This proves that for any deterministic online algorithm its expected cost (with respect to the distribution given on the inputs) is at least $k+1+\epsilon$, while we

have seen that the expected optimal cost is $1 + \epsilon$. The claim of the lemma now follows from Yao's Principle (Theorem 2.5). □

Chapter 9

Lower Bounds for Online k-Server Routing Problems

9.1 Introduction

In a k-server routing problem, k-servers move in a metric space in order to serve requests. In the Traveling Salesman Problem (TSP) and the Traveling Repairman Problem (TRP) requests are simply points in the space and a request at point x is served if at least one of the servers visits x after the request has been released. In the Dial-a-Ride version (DARP) of the Problems, each request r_j specifies a source a_j and a target b_j between which an object has to be transported. In the setting considered in this paper, the server can carry at most one object at a time and preemption is not allowed, that is, once and object has been picked up it must be delivered to its destination without intermediate droppings.

We are concerned with *online* server routing problems. All servers start in a designated point 0 of the metric space, called the *origin* at time 0, and travel at most at unit speed. Requests are released over time while the servers are traveling. Thus, a request r_j is specified by a triple $r_j = (t_j, a_j, b_j)$, where $t_j \geq 0$ is the time when the request becomes released and a_j and b_j denote the source and destination of the corresponding object. The completion time of request r_j is the time when the object has been delivered at b_j. In the TSP and TRP we have $a_j = b_j$ for all requests r_j, such that in order to serve a request at $a_j = b_j$ it suffices to visit point a_j by (at least) one server at some time $t \geq t_j$.

An online algorithm does not know about the existence of a request before its release time. It must base its decisions solely on past information. This

online model allows the servers to wait. However, waiting yields an increase in the completion times of the points still to be served. Decisions are revocable as long as they have not been executed. A common way to evaluate the quality of online algorithms is *competitive analysis* [BEY98]: An algorithm is called c*-competitive* if its cost on any input sequence is at most c times the cost of an optimal offline algorithm. For a randomized algorithm against an oblivious adversary, one considers the expected cost.

We derive lower bounds for three online k-server routing problems:

k-**TSP in** $\mathbb{R}^2$ The metric space is the Euclidean plane $\mathbb{R}^2$ and the objective is to minimize the time when the last request is served ("minimize the makespan").

k-**TRP in** $\mathbb{R}$ The metric space is the real line $\mathbb{R}$ and the objective is to minimize the sum of completion times.

k-**DARP** General metric spaces are allowed, and some objects may have to be carried (i.e. $a_j \neq b_j$ for some requests r_j); the objective is to minimize the sum of completion times.

The basic method for deriving the lower bounds is Yao's principle (Theorem 2.5).

9.1.1 Previous Work

In [FS01] Feuerstein and Stougie presented the first competitive algorithms for the 1-TRP and the 1-DARPon the real line. The competitive ratios obtained were 9 and 15. In the same paper lower bounds of $1+\sqrt{2}$ and 3 on the competitive ratio of any deterministic algorithm for the TRP and the DARP, respectively, were proved. This left a large gap between lower and upper bounds.

The bounds were subsequently improved in [KdPPS03, KdPPS06], where a 5.8285-competitive deterministic and a 3.8738-competitive randomized algorithm for the 1-DARP in general metric spaces was given. Moreover, the lower bound for the 1-DARP was raised to 2.4104.

The study of the case $k > 1$ for both the k-TSP and the k-TRP in specific metric spaces was essentially initiated in [BS06], where competitive algorithms and lower bounds were given (see also Table 9.1). It turned out that the gaps between the lower and upper bounds increased for $k > 1$. In general metric spaces, the best deterministic competitive algorithm for the k-TSP achieves a

Problem	Upper Bounds	Previous Best Lower Bound	New Lower Bound
k-TSP in $\mathbb{R}^2$	$1+\sqrt{2}$ *[BS06]* (deterministic)	$5/4$ *[BS06]*	$1+\frac{1}{3-2/k}$
k-TRP in $\mathbb{R}$	$1+\mathcal{O}(\frac{\log k}{k})$ *[BS06]* (deterministic)	$1+\frac{1}{2k}$ *[BS06]*	$1+\frac{1}{2k-1}$
k-DARP	$(1+\sqrt{2})^2 \approx 5.8285$ *[BS06]* (deterministic) $\frac{2+\sqrt{2}}{\ln(1+\sqrt{2})} \approx 3.8738$ *[KdPPS03, KdPPS06]* (randomized for $k=1$)	$\frac{4e-5}{2e-3} \approx 2.4104$ *[KdPPS03]* (for $k=1$)	3 (for *all* $k \geq 1$)

Table 9.1: Upper and lower bounds for randomized algorithms for k-server routing problems

competitive ratio of 2 [AKR00], which matches the lower bound for $k = 1$ from [AFL+01]. As shown in [BS06], for the real line the best competitive ratio for the k-TSP and k-TRP is $1+\mathcal{O}(k/\log k)$.

9.1.2 Our Results

We provide new lower bounds for randomized algorithms for k-server routing problems. Table 9.1 shows our new lower bounds in comparison to the previously best results. Our improvements for the k-TSP in $\mathbb{R}^2$ from $5/4$ to essentially $4/3$ and for the k-TRP in $\mathbb{R}$ from $1+1/(2k)$ to $1+1/(2k-1)$ may be considered to be minor. However, for the k-DARP the increase of the lower bound from 2.4104 to 3 is a major leap, in particular, since the best deterministic lower bound for the problem from [FS01] remains at 3.

As mentioned before, our basic method for deriving the lower bounds is Yao's Principle. While for the k-TSP and k-TRP we use a discrete probability distribution on the input sequences, for the k-DARP we use a method explained in [Sei00] to compute a suitable distribution once our ground set of request sequences has been fixed.

9.2 k-TSP on the Plane

In our lower bound construction we use a (seemingly) more general problem, in which each request r_j located at some point a_j also has a processing time $p_j \geq 0$ (this processing time can be divided among the servers processing the request). The request is completed, once one or more servers have visited the point a_j for a total of at least p_j time units. As shown in [BS06], such "long" requests can be emulated in the Euclidean plane with arbitrary precision by giving a high enough number of requests packed inside an arbitrarily small square around a_j.

Theorem 9.1 *The competitive ratio of any randomized online algorithm for the* k-TSP *on the plane is at least* $1 + \frac{1}{3-2/k}$.

Proof. The adversary gives a request at time 1 with processing time p, in a point x which is drawn uniformly at random from $\{(a, 0), (-a, 0)\}$, for some $a \leq 1$. As the optimal algorithm, has its servers already positioned at x, when the request appears, it can let all of them work on the job at the same time. As they can share the workload, the expected makespan of the optimal algorithm is $\mathbb{E}[\textsc{Opt}(\sigma)] = 1 + p/k$. Analogously to the proof of Theorem 5.1 in [BS06] by Bonifaci and Stougie, we can estimate the expected cost of any deterministic online algorithm by

$$\mathbb{E}[\textsc{Alg}(\sigma)] \geq 1 + p/k + 1/2 d((a, 0), (-a, 0)) = 1 + p/k + a. \tag{9.1}$$

for the case, that p is chosen big enough such that all request contribute to the serving of the request. Hence, we have

$$\frac{\mathbb{E}[\textsc{Alg}(\sigma)]}{\mathbb{E}[\textsc{Opt}(\sigma)]} \geq \frac{1 + p/k + a}{1 + p/k} = 1 + \frac{a}{1 + p/k}.$$

For Equation (9.1) to hold, we require all servers to contribute to serving the request. In the worst case, at time 1, $k - 1$ servers are located at x and one server at -1. The $k - 1$ servers need time $\frac{p}{k-1}$ to serve the request and the kth server needs time $1 + a$ to arrive at x, that is to assure that all servers will contribute to serving the request, we require

$$\frac{p}{k-1} \geq 1 + a \;\Leftrightarrow\; a \leq \frac{p}{k-1} - 1.$$

Hence, we have to solve the NLP to find the best possible bound:

$$\begin{aligned} \max_{p,\alpha} \quad & 1 + \frac{\alpha}{1+p/k} \\ \text{s.t} \quad & \alpha \leq \frac{p}{k-1} - 1 \\ & \alpha \leq 1 \end{aligned}$$

which is equivalent to maximizing

$$1 + \frac{\min\{1, \frac{p}{k-1} - 1\}}{1+p/k}. \tag{9.2}$$

If the minimum in (9.2) equals $\frac{p}{k-1} - 1$, that is $\frac{p}{k-1} - 1 \leq 1$ ($\Leftrightarrow p \leq 2k-2$), we obtain

$$\frac{\mathbb{E}[\textsc{Alg}(\sigma]}{\mathbb{E}[\textsc{Opt}(\sigma)]} \geq 1 + \frac{\frac{p}{k-1} - 1}{1+p/k}.$$

As this therm increases with p, we choose $p = 2k-2$ in order to get the best possible bound for this case.

$$\frac{\mathbb{E}[\textsc{Alg}(\sigma)]}{\mathbb{E}[\textsc{Opt}(\sigma)]} \geq 1 + \frac{1}{3-2/k}.$$

In the case where $\frac{p}{2k-2} \geq 1$,

$$\frac{\mathbb{E}[\textsc{Alg}(\sigma]}{\mathbb{E}[\textsc{Opt}(\sigma)]} \geq 1 + \frac{1}{1+p/k} \overset{p=2k-2}{=} 1 + \frac{1}{3-2/k}.$$

Hence, by Yao's Principle, for $p = 2k-2$ ($\Rightarrow \alpha = 1$) a lower bound for the competitive ratio of $1 + \frac{1}{3-2/k}$ is obtained.

□

9.3 k-TRP on the Real Line

Theorem 9.2 *The competitive ratio of any randomized online algorithm for the* k*-TRP for any* $k \geq 2$ *on the real line is at least* $1 + \frac{1}{2k-1}$.

Proof. At time 1, the adversary gives a single request in a point drawn uniformly at random from the set $R := \{-1, -\frac{k-2}{k-1}, .., -\frac{1}{k-1}, 0, \frac{1}{k-1}, ..., \frac{k-2}{k-1}, 1\}$, that

is the set consisting of $2k+1$ points spread evenly in the interval [-1,1]. Let the location of request point j be denoted by r_j such that $r_j < r_k$ for $j < k$. We have that $\mathbb{E}[\text{OPT}(\sigma)] = 1$. Hence, by Yao's Principle it remains to show that $\mathbb{E}[\text{ALG}(\sigma)] \geq 1 + \frac{1}{2k-1}$. That is, we have to show that the best deterministic algorithm has an expected cost of at least $1 + \frac{1}{2k-1}$.

We claim that there exists a best deterministic online algorithm, which has all the requests located on the request points at time 1. Let ALG be an optimal deterministic online algorithm. Let s_i be the location of server i at time 1 and define $R_i \subseteq R$ to be the set of request points which are closest to this server at time 1 (see Figure 9.1). If some $r \in R$ could be part of two different subsets, choose one of them arbitrarily.

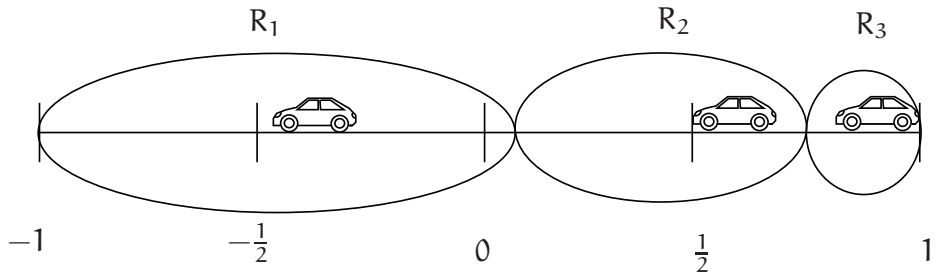

Figure 9.1: Example for $k = 3$

When the request appears, ALG will send the server to the request with minimum distance. Thus, the expected value can be computed as follows:

$$\mathbb{E}[\text{ALG}(\sigma)] = 1 + \sum_{i=1}^{k} \sum_{r \in R_i} \left(\frac{1}{2k-1} \cdot d(r, s_i) \right) = 1 + \frac{1}{2k-1} \sum_{i=1}^{k} \sum_{r \in R_i} d(r, s_i)$$

We claim, that there is an optimal algorithm which has all the servers located on request points. Suppose, there was a server i, which is not located at a request point at time 1. Let $R_i^- \subseteq R_i$ be the request assigned to this server which are located on the left of it, and let R_i^+ be the number of requests which are located on the right side of i. If $R_i^- > R_i^+$, one can obtain a better expected value by moving i to the left by $\epsilon > 0$ units, as all requests on the left can be served at cost $d(r, s_i) - \epsilon$ and all requests on the right can be served at cost $d(r, s_i) + \epsilon$, which means for the sum of all these requests:

$$\begin{aligned}
&\sum_{r\in R_i^-}(d(r,s_i)-\epsilon)+\sum_{r\in R_i^+}(d(r,s_i)+\epsilon)\\
&=\sum_{r\in R_i^-}d(r,s_i)-|R_i^-|\cdot\epsilon+\sum_{r\in R_i^+}d(r,s_i)+|R_i^+|\cdot\epsilon\\
&=\sum_{r\in R_i^-}d(r,s_i)+\sum_{r\in R_i^+}d(r,s_i)+\underbrace{(|R_i^+|-|R_i^-|)}_{<0}\cdot\epsilon\\
&<\sum_{r\in R_i^-}d(r,s_i)+\sum_{r\in R_i^+}d(r,s_i)=\sum_{r\in R_i}d(r,s_i)
\end{aligned}$$

Along the same lines, we obtain a better expected value by moving the server to the right, if $|R_i^+| > |R_i^-|$. These two cases contradict the optimality of ALG . If $|R_i^+| = |R_i^-|$, we can move the server to the right or to the left without changing the expected value. This allows us to move it to the next request point. Therefore, we can conclude that there is always a best deterministic online algorithm which locates all servers on request points at time 1.

Since there are $2k-1$ request points and k servers, the probability that a request point is chosen, where a server is located is $k/(2k-1)$. Otherwise, a server has to move to the request point, where the request appears and, as the distance between the request points is $1/(k-1)$, the server has to move at least $\frac{1}{k-1}$ units. For this reason, we can estimate the expected value as:

$$\mathbb{E}\left[\text{ALG}\left(\sigma\right)\right] \geq 1+\frac{k}{2k-1}\cdot 0+\frac{k-1}{2k-1}\cdot\frac{1}{k-1}=1+\frac{1}{2k-1}$$

□

9.4 k-DARP

Theorem 9.3 *The competitive ratio of any randomized online algorithm for the k-DARP for any $k \geq 1$ in general metric spaces is at least 3.*

Proof. As metric space, we choose a star consisting of $2k+1$ rays named A and $B_1, ..., B_{2k}$. At time 0, one request σ_1 from o to 1_A is given. With probability q, there are no further requests. With probability $1-q$, at time $2y$, on k of the $2k$ "B-rays" the adversary gives respectively m requests in $2y_{B_{i_1}}, ..., 2y_{B_{i_k}}$, $i_1 \neq i_2 ... \neq i_k \in \{1, ..., 2k\}$, where $y \in [\nu, 1]$ is chosen according to some

probability density function $p(y)$ which satisfies $q + \int_\nu^1 p(y)dy = 1$. $\nu \in [0,1]$ is chosen by the adversary in such a way that the competitive ratio is maximal. The probability that m of these requests are given in $2y_{B_i}$ is $\frac{1-q}{2k}$ for each $i \in \{1,...,2k\}$.

An offline algorithm pays 1 with probability q and $1 + (4 + 2km)y$ if the additional $k \cdot m$ requests arrive. Hence, the expected optimal offline cost is

$$\mathbb{E}\left[\textsc{Opt}\,(\sigma)\right] = q + \int_\nu^1 (1 + (4 + 2km)y)p(y)dy. \tag{9.3}$$

To calculate the expected cost of an online algorithm $\textsc{Alg}_x$, let $2x$ be the time when one of the servers starts serving σ_1, unless the $k \cdot m$ requests arrive before time $2x$. Wlog $x \leq 1$, since if no further requests arrive, the algorithm knows that there will be no further requests. As in the previous proofs, one can show, that the servers of the best deterministic online algorithm will wait in the origin until time $2x$ (respectively $2y$ if $x \geq y$).

With probability q, the online cost is $2x + 1$. If $x \geq y$, $\textsc{Alg}_x$ first serves the $k \cdot m$ requests before serving σ_1. In this case, it pays $1 + (6 + 4km)y$. Otherwise, if $x < y$, the online algorithm first has to serve σ_1 before it can start serving the last m requests. Wlog let s_1 be the server serving σ_1. Then, if s_1 is back in the origin after having served σ_1 before $s_2, ..., s_k$ are back in o after having served respectively m of the $k \cdot m$ requests, $\textsc{Alg}_x$ pays $(2x+1)+(k-1)4my+(2x+2+2y)m$, otherwise, the online algorithm's cost is $(2x+1)+(k-1)4my+(6y+2y)m$. Hence, the online algorithm pays $(2x + 1) + 4my + (\min\{2x + 2, 6y\} + 2y)m \geq (2x + 1) + 4my + (2x + 2y + 2y)m = 1 + (2 + 2m)x + 8my$. Therefore, we have

$$\begin{aligned}\mathbb{E}\left[\textsc{Alg}_x(\sigma)\right] &\geq (1 + 2x)q + \int_\nu^x (1 + (6 + 4km)y)p(y)dy \\ &\quad + \int_x^1 (1 + (2 + 2m)x + 4km \cdot y)p(y)dy \\ &=: \phi(x).\end{aligned}$$

To maximize the expected cost of any deterministic online algorithm, we wish to choose q and $p(y)$ such that $\min_{x\in[0,1]} \phi(x)$ is maximized. To this end, we use a heuristic approach where we choose q and $p(y)$ such that $\phi(x)$ is constant for all $x \in [0,1]$. Then we have that $\frac{\partial^j \phi}{\partial x^j} = 0 \ \forall j \geq 1$. Differentiating, we find that

$$\frac{\partial \phi}{\partial x} = 2q + (4 - 2m)xp(x) + (2 + 2m)\int_x^1 p(y)dy$$

$$\frac{\partial^2 \phi}{\partial x^2} = (4-2m)xp'(x) + (2-4m)p(x).$$

Since $\frac{\partial^2 \phi}{\partial x^2} = 0$, we obtain the differential equation

$$\frac{p'(x)}{p(x)} = \frac{2m-1}{(2-m)x}$$

which has solutions of the form

$$p(x) = x^{\frac{2m-1}{2-m}} \cdot c.$$

Using $q + \int_\nu^1 p(y)dy = 1$, we obtain

$$c = \frac{1-q}{\frac{2-m}{m+1}(1-\nu^{\frac{m+1}{2-m}})}$$

and hence

$$p(x) = \frac{(1-q)(m+1)}{(2-m)(1-\nu^{\frac{m+1}{2-m}})} x^{\frac{2m-1}{2-m}}.$$

Plugging $p(x)$ in $\phi(x)$, we find that

$$\begin{aligned}\phi(x) = (2x+1)q + \frac{(1-q)(3+4km+4km^2+(6+6m)x)}{3(1-\nu^{\frac{m+1}{2-m}})} \\ - \frac{(1-q)(\nu^{\frac{m+1}{2-m}}(3+(4km^2+4km+6m+6)\nu))}{3(1-\nu^{\frac{m+1}{2-m}})}.\end{aligned} \tag{9.4}$$

Since we wanted to choose q and $p(y)$ such that $\phi(x)$ is constant for all $x \in [0,1]$, we must have that $\frac{\partial \phi(x)}{\partial x} = 0$, that is

$$2q + \frac{1-q}{1-\nu^{\frac{m+1}{2-m}}}(2+2m) = 0$$

and therefore

$$q = \frac{m+1}{m+\nu^{\frac{m+1}{2-m}}}.$$

Now, the expected cost of the optimal offline algorithm and $\textsc{Alg}_x$ can be determined by plugging the obtained results for q and $p(y)$ in (9.3) and (9.4):

$$\mathbb{E}[\textsc{Alg}_x(\sigma)] \geq \frac{3m-4km-4km^2+\nu^{\frac{m+1}{2-m}}(3+(4km^2+4km6m+6)\nu)}{3(\nu^{\frac{m+1}{2-m}}+m)}$$

$$\mathbb{E}\left[\textsc{Opt}\ (\sigma)\right] = \frac{-4 - m - 2km - 2km^2 + \nu^{\frac{m+1}{2-m}}(3 + (2km^2 + 2km + 4m + 4)\nu)}{3(\nu^{\frac{m+1}{2-m}} + m)}.$$

We conclude that

$$\begin{aligned}\frac{\mathbb{E}\left[\textsc{Alg}_{\ x}(\sigma)\right]}{\mathbb{E}\left[\textsc{Opt}\ (\sigma)\right]} &\geq \frac{-5m - 8m^2 + \nu^{\frac{m+1}{2-m}}(3 + (8m^2 + 14m + 6)\nu)}{-4 - 5m - 4m^2 + \nu^{\frac{m+1}{2-m}}(3 + (4m^2 + 8m + 4)\nu)} \\ &\overset{m\to\infty}{\to} \frac{3 - 4k \cdot \log(\nu)}{1 - 2k \cdot \log(\nu)}.\end{aligned}$$

Hence the desired result is obtained as $\nu \to 1$.

□

Bibliography

[AAF+01] B. Awerbuch, Y. Azar, A. Fiat, S. Leonardi, and A. Rosén. On-line competitive algorithms for call admission in optical networks. *Algorithmica*, 31:29–43, 2001.

[AAS05] N. Andelman, Y. Azar, , and M. Sorani. Truthful approximation mechanisms for scheduling selfish related machines. *Proc. of 22nd International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 69–82, 2005.

[ABFR94] B. Awerbuch, Y. Bartal, A. Fiat, and A. Rosén. Competitive non-preemptive call control. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 312–320, 1994.

[ABN92] B. C. Arnold, N. Balakrishnan, and H. N. Nagaraja. *A First Course in Order Statistics*. Wiley, New York, 1992.

[AFL+01] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo. Algorithms for the on-line traveling salesman. *Algorithmica*, 29(4):560–581, 2001.

[AKR00] N. Ascheuer, S. O. Krumke, and J. Rambau. Online dial-a-ride problems: Minimizing the completion time. In *Proceedings of the 17th International Symposium on Theoretical Aspects of Computer Science*, volume 1770 of *Lecture Notes in Computer Science*, pages 639–650. Springer, 2000.

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Networks Flows*. Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[APPP04] V. Auletta, R. De Prisco, P. Penna, and G. Persiano. Deterministic truthful approximation mechanisms for scheduling related machines. *Proc. of 21st International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 608–619, 2004.

[APTT03] A. Archer, C. Papadimitriou, K. Talwar, and E. Tardos. An approximate truthful mechanism for combinatorial auctions with single parameter agents. *Proc. of 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 205–214, 2003.

[Arc04] A. Archer. Mechanisms for discrete optimization with rational agents. *PhD thesis, Cornell University*, 2004.

[AT01] A. Archer and E. Tardos. Truthful mechanisms for one-parameter agents. *Proc. 42nd Annual Symposium on Foundations of Computer Science*, pages 482–491, 2001.

[AT02] A. Archer and E. Tardos. Frugal path mechanisms. *Proc. of 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 991–999, 2002.

[Bap99] P. Baptiste. Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine with equal processing times. *Journal of Scheduling*, 2:245–252, 1999.

[BCG87] A. A. Bertossi, P. Carraresi, and G. Gallo. On some matching problems arising in vehicle scheduling models. *Networks*, 11, 1987.

[BDB94] S. Ben-David and A. Borodin. A new measure for the study of online algorithms. *Algoritmica*, 11, 1994, 1994.

[BEY98] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[BEY99] A. Borodin and R. El-Yaniv. On randomization in on-line computation. *Information and Computation*, 150(2):244–267, 1999.

[BHS01] S. K. Baruah, J. Haritsa, and N. Sharma. On-line scheduling to maximize task completions. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 39:65–78, 2001.

[BKN04] J. Boyar, S. Krarup, and M.N. Nielsen. Seat reservation allowing seat changes. *Journal of Algorithms*, 52:169–192, 2004.

[BL99] J. Boyar and K. S. Larsen. The seat reservation problem. *Algorithmica*, 25:403–417, 1999.

[BNS91] A. Bar-Noy and B. Schieber. The canadian traveller problem. In *SODA '91: Proceedings of the second annual ACM-SIAM symposium on Discrete algorithms*, pages 261–270, Philadelphia, PA, USA, 1991. Society for Industrial and Applied Mathematics.

[BQ06] M. O. Ball and M. Queyranne. Towards robust revenue management: Competitive analysis of online booking. *Social Science Research Network Electronic Paper Collection*, 2006.

[BS06] V. Bonifaci and L. Stougie. Online k-server routing problems. In *Proceedings of the 4th Workshop on on Approximation and Online Algorithms*, Lecture Notes in Computer Science, 2006.

[CB01] C. Chekuri and M. A. Bender. An efficient approximation algorithm for minimizing makespan on uniformly related machines. *Journal of Algorithms*, 41:212–224, 2001.

[Chv83] V. Chvátal. *Linear programming*. A series of books in the mathematical sciences. New York - San Francisco: W. H. Freeman and Company, 1983.

[CJST04] M. Chrobak, W. Jawor, J. Sgall, and T. Tichy. Online scheduling of equal-length jobs: Randomization and restarts help. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming*, volume 3142 of *Lecture Notes in Computer Science*, pages 358–370. Springer, 2004.

[CS99] F. A. Chudak and D. B. Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. *Journal of Algorithms*, 30(2):323–343, 1999.

[Dis04] I. Dischke. Disposition von Einsatzfahrzeugen: Startheuristiken, Branching-Regeln und Rundungstechniken. Diplomarbeit, TU Berlin, 2004.

[DMV03] N. R. Devanur, M. Mihail, and V. V. Vazirani. Strategyproof cost-sharing mechanisms for set cover and facility location games. *ACM Conference on E-commerce*, pages 108–114, 2003.

[EF01] L. Epstein and L. M. Favrholdt. Optimal preemptive semi-online scheduling to minimize makespan on two related machines. Technical Report PP-2001-18, 2 2001.

[ESS04] E. Elkind, A. Sahaiand, and K. Steiglitz. Frugality in path auctions. Proc. of 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA):701–709, 2004.

[FS97] U. Feige and M. Seltser. On the densest k-subgraph problems. Technical report, Jerusalem, Israel, Israel, 1997.

[FS01] E. Feuerstein and L. Stougie. On-line single server dial-a-ride problems. *Theoretical Computer Science*, 268(1):91–105, 2001.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability: A guide to the theory of NP-Completeness*. W.H. Freeman, New York, 1979.

[GK05] E. Gassner and S. O. Krumke. Deterministic online optical call admission revisited. In *Proceedings of the 3rd Workshop on on Approximation and Online Algorithms*, Lecture Notes in Computer Science. Springer, 2005. To appear.

[GPS00] S. A. Goldman, J. Parwatikar, and S. Suri. Online scheduling with hard deadlines. *Journal of Algorithms*, 34:370–389, 2000.

[Gra66] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical J.*, 45:1563–1581, 1966.

[Hal93] M. M. Halldorsson. Approximating the minimum maximal independence number. *Inform. Process. Lett.*, 46:169–172, 1993.

[HKR05] B. Hiller, S. O. Krumke, and J. Rambau. Reoptimization gaps versus model errors in online-dispatching of service units. *Discrete Applied Mathematics*, 2005. A preliminary version appeared in the Proceedings of the Latin-American Conference on Combinatorics, Graphs and Algorithms, 2004.

[HRK85] M. Haimovich and A. H. G. Rinnooy Kan. Bounds and heuristics for capacitated routing problems. *Mathematics of Operations Research*, 10, 1985.

[Jaf80] J. M. Jaffe. Efficient scheduling of tasks without full use of processor resources. *Theoretical Computer Science*, 12:1–17, 1980.

[Joh74] D. S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. System Sci.*, 9:256–278, 1974.

[JS04] F. Jarre and J. Stoer. *Optimierung*. Springer, Berlin Heidelberg New York, 2004.

[KdPPS03] S. O. Krumke, W. E. de Paepe, D. Poensgen, and L. Stougie. News from the online traveling repairman. *Theoretical Computer Science*, 295(1–3):279–294, 2003. A preliminary version appeared in the

Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science, 2001, vol. 2136 of Lecture Notes in Computer Science.

[KdPPS06] S. O. Krumke, W. E. de Paepe, D. Poensgen, and L. Stougie. Erratum to News from the online traveling repairman. *Theoretical Computer Science*, 352(1-3):347–348, 2006.

[KN05] S. O. Krumke and H. Noltemeier. *Graphentheorische Konzepte und Algorithmen*. B.G. Teubner,, 2005.

[Kov05] A. Kovacs. Fast monotone 3-approximation algorithm for scheduling related machines. *Proc. of 13th Annual European Symposium on Algorithms (ESA)*, pages 616–627, 2005.

[KP02] S. O. Krumke and D. Poensgen. Online call admission in optical networks with larger wavelength demands. In *Proceedings of the 28th International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 2573 of *Lecture Notes in Computer Science*, pages 333–344. Springer, 2002.

[KRT02a] S. O. Krumke, J. Rambau, and L. M. Torres. Online dispatching of automobile in real-time. ZIB-Report 02-18, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 2002.

[KRT02b] S. O. Krumke, J. Rambau, and L. M. Torres. Real-time dispatching of guided and unguided automobile service units with soft time windows. In *Proceedings of the 10th Annual European Symposium on Algorithms*, volume 2461 of *Lecture Notes in Computer Science*, pages 637–648. Springer, 2002.

[KSVWar] S. O. Krumke, S. Saliba, T. Vredeveld, and S. Westphal. Approximation algorithms for a vehicle routing problem. *Mathematical Methods of Operations Research*, to appear.

[KV00] B. Korte and J. Vygen. *Combinatorial Optimization*. Springer, Berlin, Heidelberg, New York, 2000.

[Law76] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, 1976.

[LOS99] D. J. Lehmann, L. O'Callaghan, and Y. Shoham. Truth revelation in rapid, approximately efficient combinatorial auctions. *ACM Conference on Electronic Commerce*, pages 96–102, 1999.

[LT94] R. J. Lipton and A. Tomkins. Online interval scheduling. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 302–311, 1994.

[MN02] A. Mu'alem and N. Nisan. Truthful approximation mechanisms for restricted combinatorial auctions. *Proc. of the 18th National Conference on Artificial Intelligence and 14th Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI)*, pages 379–384, 2002.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[MR99] J. I. McGill and G. J. Van Ryzin. Revenue management: Research overview and prospects. *Transportation Science*, 33(2):233–256, 1999.

[MU05] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA, 2005.

[NW88] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, Inc., 1988.

[PY91] C. H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theor. Comput. Sci.*, 84(1):127–150, 1991.

[Sch86] A. Schrijver. *Theory of linear and integer programming*. Wiley-Interscience Series in Discrete Mathematics. A Wiley-Interscience Publication. Chichester: John Wiley & Sons Ltd., 1986.

[Sei00] S. Seiden. A guessing game and randomized online algorithms. In *Proceedings of the 32nd Annual ACM Symposium on the Theory of Computing*, pages 592–601, 2000.

[SP01] R. van Stee and J. A. La Poutré. Partial servicing of on-line jobs. *Journal of Scheduling*, 4(6):379–396, 2001.

[SSW98] S. Seiden, J. Sgall, and G. J. Woeginger. Semi-online scheduling with decreasing job sizes. Technical Report KAM-DIMATIA Series 98-410, 1998.

[TR05] K. T. Talluri and G. Van Ryzin. *The Theory and Practice of Revenue Management*. Springer, New York, 2005.

[Yao77] A. C. C. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proceedings of the 18th Annual IEEE Symposium on the Foundations of Computer Science*, pages 222–227, 1977.

Curriculum Vitae

06/1996	Graduation (Abitur) at Altes Gymnasium Oldenburg
07/1996 - 07/1997	Civilian service at Malteser Hilfsdienst e.V. Oldenburg
10/1997 - 04/2004	Studies in Applied Mathematics (Wirtschaftsmathematik) at Technische Universität Berlin Specialization in Discrete Mathematics/Optimization
02/2000 - 12/2000	Part-Time Employee at PSI AG Berlin
01/2001 - 04/2003	Part-Time Employee at front2back AG Berlin
06/2003 - 05/2004	Student Assistant at Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB)
04/2004	Master's degree in Applied Mathematics (german diploma)
since 06/2004	Teaching Assistant at Technische Universität Kaiserslautern, Department of Mathematics, AG Optimization

Wissenschaftlicher und beruflicher Werdegang

06/1996	Abitur am Alten Gymnasium Oldenburg
07/1996 - 07/1997	Zivildienst beim Malteser Hilfsdienst e.V. Oldenburg
10/1997 - 04/2004	Studium der Wirtschaftsmathematik an der Technischen Universität Berlin mit den Schwerpunkten “Algorithmisch Diskrete Mathematik”, “Datenbanken und Informationssysteme” und “Logistik”
02/2000 - 12/2000	Studentischer Mitarbeiter bei der PSI AG Berlin
01/2001 - 04/2003	Studentischer Mitarbeiter bei der front2back AG Berlin
06/2003 - 05/2004	Studentischer Mitarbeiter beim Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB)
04/2004	Diplom in Wirtschaftsmathematik
seit 06/2004	Wissenschaftlicher Mitarbeiter in der Arbeitsgruppe Optimierung des Fachbereichs Mathematik der Technischen Universität Kaiserslautern