Policies for Online Target Date Assignment Problems:

Competitive Analysis versus Expected Performance

Diplomarbeit

bei
Prof. Dr. Martin Grötschel
Prof. Dr. Jörg Rambau

Dezember 2005

vorgelegt von

STEFAN HEINZ

Technischen Universität Berlin
Fachbereich II: Mathematik und Naturwissenschaften
Institut für Mathematik
Studiengang Wirtschaftsmathematik

Die selbständige und eigenhändige Anfertigung versichere ich an Eides statt.

Berlin, Dezember 2005

Stefan Heinz

Acknowledgments

Although my name is the only one appearing on the title page, several people have been directly and indirectly involved in the creation of this thesis.

First of all, I want to thank my supervisors Prof. Dr. Martin Grötschel and Prof. Dr. Jörg Rambau. Thanks to Prof. Dr. Martin Grötschel for supervising this thesis, for being an outstanding teacher and motivator, and for providing an excellent working environment at the *Konrad-Zuse-Zentrum für Informationstechnik Berlin*. Working at this institute in the last two years was very instructive. Specially, the Friday morning research sessions with Nicole Megow, Andreas Tuchscherer, and Dr. Tjark Vredeveld inspired me a lot. It was always a pleasure for me to work with them. A great portion of the results presented in this thesis were derived from these research sessions.

Relating to this diploma thesis I especially thank Andreas Tuchscherer, Benjamin Hiller, and Prof. Dr. Jörg Rambau for reading the preliminary versions, their professional advice, their helpful suggestions, and for answering any question I had. Special thanks to Andreas Tuchscherer for reading all chapters several times, for giving great suggestions to improve the presentation of this thesis, and for teaching me the programming language `C++`.

Last, but not least, many thanks to my parents for always supporting and encouraging me throughout the years.

Thanks to all of you!

Berlin, December 2005 *Stefan Heinz*

Contents

1

Introduction

In this diploma thesis we introduce a general framework for a new class of online problems featuring a two-stage decision process, the *Online Target Date Assignment Problem*. We present general online algorithms for instances of this problem. These algorithms are analyzed in two different ways. On the on hand, competitive analysis is used to analyze the worst case behavior of these algorithms. On the other hand, we develop a new approach, for approximating the optimal value function of a discounted Markov decision problem locally. This method is used to analyze the expected performance of the presented algorithms in special situations. In the sequel all necessary definitions and used approaches are explained step-by-step.

The outline of this chapter is as follows. In the first section, we briefly describe the customer service application arising at Hermes Technischer Kundendienst and illustrate its important features. In Section 1.2 we generalize this application to the class of Online Target Date Assignment Problems, discuss different versions of it, and present the instances of Online Target Date Assignment Problems considered in this work. A brief overview on previous work is given in Section 1.3. This chapter is closed with Section 1.4 in which we present the contribution of this work and the structure of this thesis.

1.1 The Customer Service Application arising at Hermes Technischer Kundendienst

In this section we introduce a real-life problem arising at Hermes Technischer Kundendienst (HTK) and point out its special features. The company HTK exists on the German market since more than 25 years. Their main business is to repair electronic equipment such as brown[1] and white[2]

[1] Household electrical entertainment appliances such as televisions and music systems. These appliances were traditionally finished with wood. This is now rather rare, but the name has stuck, even for goods that are unlikely ever to have been provided in a wooden case (e.g. camcorders).

[2] Large household appliances such as ovens and refrigerators. These were formerly finished with white enamel. Today they are often colored but the name has also stuck.

goods. Depending on the size of the broken equipment, HTK developed two different types of repair services. For devices such as a coffee maker or vacuum cleaner the company authorizes a delivery service to pick up the broken device or ask the customer for sending it to a repair shop. After the device has been fixed, they bring or send it back. However, if the device is "large" such as a refrigerator or laundry machine, then a service technician has to visit the customer to fix the problem at its location. The second workflow is the one of interest in this work.

1.1.1 The Application

The workflow of interest can be divided into two stages. In a first stage a customer calls in and requests a repair service for a broken device which requires to be fixed at its location. Therefore, a service technician has to visit this customer to repair this device. This service has to be done within a certain time frame (usually within two weeks). The customer must be given the day and possibly a more narrow time window when a service technician will arrive. This arrangement has to be made while the customer is on the phone and without knowledge of future service requests. Moreover, this appointment is irrevocable. By choosing the service day, several side constraints have to be taken into account. For example:

- since the device has to be repaired at its location, it has to be ensured that possibly needed spare parts are available at the day of service;
- it is to guarantee that a service technician is available on the service day which has the qualification to repair the broken device.

However, until the promised service day arrives, the decision which service technician to send and in which order the customer should be visited can be safely deferred. This desicion is made in a second stage.

In the second stage exact schedules and routings of service technicians are computed. In detail, in the end of each day all requests are known which have to be served at the following day. The problem, which now arises, is to design a set of routes for fleets of vehicles (service technicians) for the purpose of serving all requests assigned to the following day at minimum cost. Therefore, the night before gives time to solve a large scale vehicle dispatching problem. Thereby, a lot of side constraints have to be considered. For example:

- each request assigned to the following day has to be served;
- since the qualifications of service technicians differ, it has to be ensured that a schedule for a service technician only contains requests which require skills this service technician has;
- necessary spare parts have to be available and reachable for the service technicians;
- if a customer received a more narrow time window as a day, it has be guaranteed that a service technician visits this customer within the promised time window.

The solution of the vehicle routing problem determines the schedules and routings for each service technician for the next day.

1.1.2 Special Features of this Application

The customer service application exhibits a two-stage decision process. In a first stage, an appointment with a customer has to made immediately and irrevocably while he is on the phone and without knowledge of future service requests. In the second stage process, exact schedules and routings of service technicians for a fixed day are computed.

In particular, the first stage is an *online assignment problem* since assignment decisions have to be made only with the information revealed so far which characterizes an *online problem*. The second stage is an *offline optimization problem*, since all requests are known which have to be served at the next days in such a way that an optimality criterion is satisfied, this is, to serve all requests at a minimum cost.

Furthermore, this two-stage decision process has the following cost structure. At the point where a customer has to receive an appointment, it is not known what each possible assignment decision will cost. This is the case since in general future service requests are unknown and therefore, the input instances for the second stage optimization problem are incomplete. The actual cost for assignment decisions are not known until the second stage optimization problem is solved with a complete input instance. In general, the assignment decisions in the first stage influence the overall cost because they determine the input and thus the optimal costs of the second stage optimization problem.

1.2 Considered Problem Setting and its Versions

Many problems encountered in real-life involve a two-state decision process. In a first stage an arising request has to be assigned immediately and irrevocably to a resource before the next request is revealed, while in a second stage process, certain "sub-instances", these are instances formed by requests assigned to a particular resource, can be solved to optimality later.

In this work the resource which has to be assigned to requests is a target date, a date at which the service should take place. This leads to the class of *Online Target Date Assignment Problem* (ONLINETDAP) which are a generalized version of the real-world problem arising at HTK. Requests for the ONLINETDAP become known at certain dates. An *online algorithm* has to assign a target date to each arising request, this means, immediately after a request is revealed and without knowledge about future requests. The assignment decision specifies the date on which a request has to be processed, for instance on which day a service technician comes by to fix the broken laundry machine. The cost at a target date is given by the *downstream cost*, the optimal cost of processing all requests assigned to this particular date w. r. t. some fixed downstream offline optimization problem, for example, the cost of an optimal dispatch for service technicians.

There are different versions of this problem setting. If all arising requests are known in advance, both stages can be integrated and solved offline to obtain an overall optimal solution, even in many practical applications. However, if for each request the first assignment decision has to be made immediately after this request is released and without knowledge about future requests, the situation changes. In this case, the first stage has be handled as on online assignment problem. As long as each request has to be assigned to a future target date the second stage can still be processed offline. However, if the release date of a request is itself also a feasible target date, even the second stage is an online problem. It would be also conceivable, that all requests on a single date might be collected, and the target date for these requests are chosen and communicated at the end of this date. This is a relaxation of the online requirement of the assignment decision.

In this thesis we considers the version where in the first stage each request has to be assigned immediately after it is released and without knowledge about future requests, that is, the requests have to be processed in an online fashion. The second stage can be carried out offline since only future target dates can be assigned to each request. For this version we presents algorithms for the ONLINETDAP independently of the particular downstream problem, when the overall objective is to minimize either the sum or the maximum of all downstream costs. As the first basic examples, we analyze these algorithms for the particular academic downstream problems of bin-packing and non-preemptive scheduling on identical and parallel machines.

1.3 Related Work

In a joint work with Sven O. Krumke, Nicole Megow, Jörg Rambau, Andreas Tuchscherer, and Tjark Vredeveld we introduced in [HKM+05] the novel concept of an Online Target Date Assignment Problem as a general framework for online problems which exhibit a two-stage decision process. To the best of our knowledge there was no previous work down which covers online problems with this nature for the case where no stochastic information about future requests is available. However, the subproblems are studied in the past.

There is a lot of research going on in the field online optimization. Important results from this field are presented in the book of Fiat and Woeginger [FW98] as well as in the book of Borodin and El-Yaniv [BEY98]. A survey on classical competitive analysis for online algorithms is given by Albers in [Alb03].

The considered academic downstream problems bin-packing and non-preemptive scheduling on identical and parallel machines are well known and considered in many different versions. Coffman, Garey, and Johnson [CGJ96] published a survey which reviews the literature on worst-case and average-case behavior of approximation algorithms for one-dimensional bin-packing. The book of Leung [Leu04] provides a coverage on the recent and advanced topics on scheduling.

The downstream optimization problem arising at HTK is large scale vehicle dispatching problem. These optimization problems are well studied in the past. The book of Toth and Vigo [TV02] gives a review over this class of optimization problems.

In the case where stochastic assumptions are considered for future requests, many stochastic models, for example, Markov decision processes [Put94], are unsolvable for practical problem sizes. In a joint work with Volker Kaibel, Matthias Peinhardt, Jörg Rambau, and Andreas Tuchscherer we present in [HKP+05] a method for approximating the optimal value function of large scale discounted Markov decision problem locally. The runtime complexity of this approach and the number of touched states does not depend on the number of states of the whole system. There is a sparse sampling algorithm for large scale Markov decision processes with performance guarantees and runtime complexity similar to the ones that we state [KMN02]. Furthermore, neuro-dynamic programming [Ber01] and similar ideas [SP04] lack the ability to produce guarantees for the approximation quality.

1.4 Contribution and Outline

The contribution of this thesis is the following. First of all, we provide a mathematical framework for general ONLINETDAPs. Furthermore, we analyze instantiation of ONLINETDAPs where the overall objective is to minimize either the sum or the maximum of all downstream costs. Within the ONLINETDAP framework, our results are online algorithms and lower and upper bounds on their performance guarantee, this is, the competitive ratio, for the particular downstream problems bin-packing and parallel-machine scheduling. These results are partly form joint work [HKM+05] and presented in more details as in this paper. We also present the method introduced in [HKP+05] to approximate the optimal value function of a discounted Markov decision problem locally. This approach is based on the classical linear programming formulation for discounted Markov decision problems. We use this method to evaluate the expected performance of some online algorithms applied to associated Markov decision processes of instances of ONLINETDAPs.

For ONLINETDAPs with the overall objective to minimize the sum of all downstream costs, we call them MIN-TOTAL ONLINETDAPs, we present in particular the general online algorithm PACKTOGETHERORDELAY (PTD). It turns out that this algorithm is competitive for specific settings of the academic downstream problems bin-packing and parallel-machine scheduling. Furthermore, we introduce the online algorithm PACKFIRSTORDELAY (PFD) which seems to be even more promising for this problem setting than PTD. However, it is not possible to state this conjecture in any satisfying result for the competitive ratio of PFD. Therefore, we use our new method for approximating the optimal value function of a discounted Markov decision problem locally to analyze these two algorithms w. r. t. their expected performance. This shows that in the case of the downstream problem bin-packing the conjecture that PFD has a better performs than PTD seems to be true. However, in the case of parallel-machine scheduling the computational results do not support this conjecture any more.

If the overall objective function is to minimize the maximum downstream cost over all target date, we call this problem MIN-MAX ONLINETDAPs, we state the general online algorithm BALANCE (BAL). This algorithm is competitive for certain problem setting of the considered downstream problem bin-packing and parallel-machine scheduling.

Furthermore, we observe for both objective functions that special profiles for the considered downstream problem, as for example, bounded number of bins or unbounded number of machines per target date, lead to trivial problems or prevent any deterministic online algorithm from achieving a constant competitive ratio.

The outline of this thesis is as follows. Chapter 2 introduces briefly the field online optimization as well as the field stochastic dynamic optimization. Standard analysis tools of these fields are presented which are useful to evaluate instantiations of ONLINETDAPs. Chapter 3 is devoted to the mathematical problem formulation of ONLINETDAPs. Furthermore, elementary ONLINETDAPs are introduced and analyzed using competitive analysis. Our new approach from the field stochastic dynamic optimization is presented in Chapter 4. In Chapter 5 we use this new method to present computational results for two online algorithms PTD and PFD applied to associated Markov decision processes of instances of MIN-TOTAL ONLINETDAPs. Chapter 6 is devoted to a summary, conclusion, and an outlook. Finally, the appendices include a short overview of mathematical symbols and notations used in this thesis, a summary in German, and additional computational results.

2

Analysis Methods for Optimization with Dynamic Input Data

In order to analyze the Online Target Date Assignment Problem there are different possible approaches. This chapter presents the two analysis methods used in this thesis to analyze policies for Online Target Date Assignment Problems.

This chapter is organized as follows. The first two sections provide a basic overview on Online Optimization and Stochastic Dynamic Optimization. Both methods deal with planning under uncertainty, but still differ in the assumptions they make. Section 2.3 closes this chapter by reviewing both approaches. This shows why we use these methods to analyze instances of Online Target Date Assignment Problem.

2.1 Online Optimization

Classical optimization approaches assume complete knowledge about all problem data in advance. These problems are also called *offline optimization problems*. The assumption concerning complete information applies for certain applications. However, for the customer service application described in Section 1.1 this assumption is obviously not satisfied.

Online optimization problems are a special class of optimization problems where the input instances are not given completely in advance. Instead, an instance arises step-by-step and decisions have to be made based only on the information revealed so far. Each decision leads to a cost or profit and the task is to minimize the total cost or to maximize the gained profit. In this work only minimization problems are considered. Therefore, all definitions in this section refer to minimization problems. However, the definitions can easily be adapted for maximization problems.

Most online optimization problems can be formalized as a request-answer game which was introduced in [BDBK$^+$94].

Definition 2.1 (Request-Answer Game)
A *request-answer game* consists of a request set $\mathcal{R}$, a nonempty and finite answer set $\mathcal{A}$, and cost functions $\text{cost}_n : \mathcal{R}^n \times \mathcal{A}^n \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ for $n \in \mathbb{N}$. Let $\mathcal{C}$ denote the union of the functions cost_n over all $n \in \mathbb{N}$. An instance is given by a request sequence $\sigma = r_1, r_2, \ldots, r_n$ of $n \in \mathbb{N}$ requests from $\mathcal{R}$. The task is to find an answer sequence $(a_1, a_2, \ldots, a_n) \in \mathcal{A}^n$ such that the cost $\text{cost}_n(r_1, r_2, \ldots, r_n, a_1, a_2, \ldots, a_n)$ is minimized. A request-answer game is given by the triple $(\mathcal{R}, \mathcal{A}, \mathcal{C})$. △

A request-answer game itself does not define an online optimization problem since no restriction is made on the way the answers have to be given. In the online setting an *online algorithm* has compute the answers for a given request sequence. An online algorithm has to serve a request right after it arises according to the specific rules of a request-answer game.

Definition 2.2 (Deterministic Online Algorithm)
Let $(\mathcal{R}, \mathcal{A}, \mathcal{C})$ be a request-answer game. A *deterministic online algorithm* ALG is a sequence of functions $f_1, f_2, \ldots$, where $f_i : \mathcal{R}^i \to \mathcal{A}$. If $\sigma = r_1, r_2, \ldots, r_n$ is a sequence of $n \in \mathbb{N}$ requests from $\mathcal{R}$, then the output of ALG for this sequence is

$$\mathsf{ALG}[\sigma] = (a_1, a_2, \ldots, a_n) \in \mathcal{A}^n, \quad \text{where } a_i = f_i(r_1, r_2, \ldots, r_i).$$

The cost incurred by ALG on σ is denoted by $\mathsf{ALG}(\sigma)$ and defined as

$$\mathsf{ALG}(\sigma) = \text{cost}_n(\sigma, \mathsf{ALG}[\sigma]).$$

△

Note that the answer a_i may only depend on the requests $r_1, r_2, \ldots, r_i$ for $i = 1, 2, \ldots, n$. Therefore, the definition of a deterministic online algorithm meets the requirement that such algorithms have to make decisions based only on partial information.

Besides the class of deterministic online algorithms there exists the class of *randomized online algorithms*. These algorithms use a probability distribution over a set of deterministic online algorithms to generate an answer for a given request. Therefore, the answer sequence as well as the cost are random variables.

Definition 2.3 (Randomized Online Algorithm)
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). △

Note that the definition points out that every deterministic online algorithm is a randomized online algorithm with probability 1 on a certain outcome. Hence, the class of deterministic online algorithms is included in the class of randomized online algorithms.

Online algorithms provide for each sequence of requests an answer sequence which comes along with a cost. The task is to generate an answer sequence that minimizes this cost. The standard

technique for analyzing the performance of an online algorithm is competitive analysis. This method measures the performance of an online algorithm against an *optimal offline algorithm*. An optimal offline algorithm has access to the complete input instance in advance and serves it at a minimum cost, called *optimal offline cost*.

Definition 2.4 (Optimal Offline Cost)
Let $(\mathcal{R}, \mathcal{A}, \mathcal{C})$ be a request-answer game and $\sigma = r_1, r_2, \ldots, r_n$ a sequence of $n \in \mathbb{N}$ requests from $\mathcal{R}$. Then the *optimal offline cost* is defined as

$$\mathsf{OPT}(\sigma) = \min\{\mathrm{cost}_n(\sigma, a) \mid a \in \mathcal{A}^n\}.$$

△

Using competitive analysis the performance of a deterministic online algorithm is measured as follows.

Definition 2.5 (Competitive Deterministic Online Algorithm)
Let $(\mathcal{R}, \mathcal{A}, \mathcal{C})$ be a request-answer game and $c \geq 1$ a real number. A deterministic online algorithm ALG is called *c-competitive* if

$$\mathsf{ALG}(\sigma) \leq c \cdot \mathsf{OPT}(\sigma)$$

holds for any request sequence σ. The *competitive ratio* of ALG is the infimum over all c such that ALG is c-competitive. △

Note that the definition does not make any restriction on the computational complexity a deterministic online algorithm has. The only scarce resource in competitive analysis is information.

Competitive analysis is a worst case analysis for online algorithms since the performance guarantee must hold for each request sequence. Moreover, competitive analysis can be seen as a game between the online algorithm and a malicious adversary. The malicious adversary tries to generate a request sequence such that the online algorithm performs as "bad" as possible compared to the optimal offline cost. In doing so, the malicious adversary has knowledge about the algorithm. That is, he knows for any request sequence all answers of a deterministic online algorithms in advance.

The answer sequence as well as the cost of a randomized online algorithm are random variables. Therefore, the competitive ratio of a randomized online algorithm depends on the amount of information an adversary has access to. In the standard adversary model, the adversary has knowledge about the probability distribution a randomized online algorithm uses but does not know the exact outcome for each request sequence. Hence, an adversary has to choose an entire request sequence before an online algorithm starts processing the chosen sequence. Such an adversary is called *oblivious adversary* in the literature.

Definition 2.6 (Oblivious Adversary)
An *oblivious adversary* has to generate the entire request sequence in advance based only on the description of the randomized online algorithm but before any request is served by the randomized online algorithm. △

As mentioned before the definition of the competitive ratio of a randomized online algorithm depends on the kind of adversary. For the purpose of introducing competitive analysis, we restrict ourselves to an oblivious adversary which is the weakest of those introduced in [BDBK+94].

Definition 2.7 (Competitive Randomized Online Algorithm)
Let $(\mathcal{R}, \mathcal{A}, \mathcal{C})$ be a request-answer game and $c \geq 1$ a real number. A randomized online algorithm RALG with a probability distribution X over a set $\{\mathsf{ALG}_x\}$ of deterministic online algorithms is said to be *c-competitive* against the oblivious adversary if

$$\mathbb{E}[\mathsf{ALG}_x(\sigma)] \leq c \cdot \mathsf{OPT}(\sigma)$$

holds for each sequence σ. Here the expression $\mathbb{E}[\mathsf{ALG}_x(\sigma)]$ denotes the expectation with respect to the probability distribution X over $\{\mathsf{ALG}_x\}$ which defines RALG. The *competitive ratio* of RALG is the infimum over all c such that RALG is c-competitive against the oblivious adversary. △

The above definition reduces to Definition 2.5 in the case of a deterministic online algorithm. Since the adversaries are not as powerful as in the deterministic case, randomized online algorithms usually provide a better competitive ratio than deterministic online algorithms.

Of course, lower bounds on the competitive ratio of online algorithms are also of interest. In order to obtain such a lower bound for an online algorithm a request sequence has to be constructed where this algorithm performs "bad" compared to the optimal offline cost. Besides a lower bound on the competitive ratio of a certain online algorithm it is also of interest to find a lower bound which holds for any online algorithm of the considered online optimization problem. In the deterministic case it is comparatively easy to find suitable request sequences. Since the cost of a randomized online algorithm is a random variable, it can be difficult to bound the competitive ratio from below. In such cases Yao's Principle is an approach to find lower bounds on the competitive ratio of any randomized online algorithms for the considered online problem (see [BEY98, Chapter 8]).

This thesis focuses only on deterministic online algorithms. Therefore, we are not going further into the theory of randomized online algorithms. For more details see [BDBK+94, BEY98, MR95].

2.2 Stochastic Dynamic Optimization

The field Stochastic Dynamic Optimization deals with optimization problems over stochastic processes. Therefore, such problems consist of a stochastic process and an optimality criterion. The *sequential decision model* is a common framework for these optimization problems. A decision maker observes the current state of a system. Based on this state, a control from a set of feasible controls has to be chosen. The control choice produces two results: the decision maker receives an immediate cost or profit, and the system evolves to a new state according to a probability distribution determined by the control choice.

This thesis focuses on a sequential decision model in discrete time over an infinite time horizon where the set of feasible controls, the costs, and the transition probabilities depend only on the

current state and chosen control but not on states occupied and controls chosen in the past. This model is called *Markov decision process* or *stationary discrete-time dynamic system*. Moreover, only problems are considered where a decision leads to a cost.

Before a formal definition for a Markov decision process is given, we want to point out that in this section random variables are denoted by the capital letter X.

Definition 2.8 (Markov Decision Process)
A *Markov decision process* is a collection of objects (S, C, P, cost) where

- S is a finite set of states labeled by integers $i = 1, 2, \ldots, N$ ($S = \{1, 2, \ldots, N\}$);
- C is a finite set of controls where $C(i) \subseteq C$ denotes the set of feasible controls for each state $i \in S$;
- P is a transition probability matrix with $p_{ij} : C(i) \to [0, 1]$ for $i, j \in S$ where $p_{ij}(u)$ gives the probability that the system evolves to state j if in state i the control u is applied and
$$\sum_{j \in S} p_{ij}(u) = 1 \quad \forall\, i \in S,\ u \in C(i);$$
- $\text{cost} : S \times C \times S \to \mathbb{R}_{\geq 0}$ is a cost function where $\text{cost}(i, u, j)$ determines the incurred cost if the system evolves from state i to state j using the control u. The cost value $\text{cost}(i, u, j)$ is only defined if $u \in C(i)$. △

Note that a Markov decision process is a *stationary process*, that is, the feasible controls for a state $i \in S$, the transition probability matrix, and the the cost function do not depend on the stage of the process.

A *policy* for a Markov decision process defines the control to choose for each state of the process. This decision may only depend on the current state of the system and the stage of the process.

Definition 2.9 (Policy)
A *policy* for a Markov decision process (S, C, P, cost) is a sequence $\pi = \mu_1, \mu_2, \ldots$ of function $\mu_k : S \to C$ for $k = 1, 2, \ldots$. The policy π is called *admissible* for the Markov decision process (S, C, P, cost) if $\mu_k(i) \in C(i)$ for $k = 1, 2, \ldots$ and all $i \in S$. The set of all admissible policy for a Markov decision process is denoted by Π. △

An admissible policy of the form $\pi = \mu, \mu, \ldots$ is called *stationary* since the control chosen by the policy π for a state does not depend on the stage of the process. A stationary policy $\pi = \mu, \mu, \ldots$ is denoted by μ.

For a given Markov decision process (S, C, P, cost) and an admissible policy $\pi = \mu_1, \mu_2, \ldots$ the *total cost* for an initial state $i \in S$ is a random variable

$$X_\pi(i) = \left[\sum_{k=0}^{\infty} \text{cost}(X_k, \mu_k(X_k), X_{k+1}) \mid X_0 = i \right].$$

In this term X_k is the random variable specifying the state of the system at stage $k \in \mathbb{N}_0$. Since the total costs for admissible policies are random variables, it necessitates a method for comparing these random variables to classify the policies for a state $i \in S$. There are several approaches to generate a stochastic order on random variables. In this thesis only the expectation is used to create for each state $i \in S$ a stochastic order on the total costs for admissible policies. For other examples see [Put94]. Moreover, a discount factor $\alpha \in (0,1)$ is used to take into account that future costs matter to us less than the same costs incurred at the present time. That means, a unit cost at stage k is worth only α^k. Therefore, the *expected total discounted cost* for a given initial state i of a Markov decision process (S, C, P, cost) and an admissible policy $\pi = \mu_1, \mu_2, \ldots$ is defined as

$$J_\pi(i) = \mathbb{E}\left[\sum_{k=0}^{\infty} \alpha^k \,\text{cost}(X_k, \mu_k(X_k), X_{k+1}) \mid X_0 = i\right] \quad \text{with} \quad \alpha \in (0,1).$$

This expectation defines for each state $i \in S$ a stochastic order on the random variables for the total discounted costs of admissible policies. On this basis the policies for the given Markov decision process are comparable if the expectations exist. This leads to the optimality criterion considered in this work. For other optimality criteria see [Ber01, Put94].

Since the state set and the control set are finite, there exists a constant $M \in \mathbb{R}_{\geq 0}$ which bounds the cost function of a Markov decision process (S, C, P, cost) from above, that is, $0 \leq \text{cost}(i,u,j) \leq M$ for all $i, j \in S$ and $u \in C(i)$. Therefore, the expected total discounted cost of an admissible policy π exists for any initial state $i \in S$ since $\alpha \in (0,1)$ and

$$\begin{aligned} J_\pi(i) &= \mathbb{E}\left[\sum_{k=0}^{\infty} \alpha^k \text{cost}(X_k, \mu_k(X_k), X_{k+1}) \mid X_0 = i\right] \\ &\leq \mathbb{E}\left[\sum_{k=0}^{\infty} \alpha^k M\right] \\ &\leq \mathbb{E}\left[\frac{M}{1-\alpha}\right] \\ &= \frac{M}{1-\alpha}. \end{aligned}$$

Moreover, this inequality proves that for any admissible policy π and initial state $i \in S$ the expected total discounted cost is bounded from above by

$$J_\pi(i) \leq M/(1-\alpha). \tag{2.1}$$

Each decision leads to an immediate cost. For a given Markov decision process (S, C, P, cost) the *expected stage cost* when control $u \in C(i)$ is applied at state $i \in S$ is given by

$$\overline{\text{cost}}(i,u) = \sum_{j \in S} p_{ij}(u)\,\text{cost}(i,u,j) \quad \forall\, i \in S,\ u \in C(i).$$

Since the state space is finite and the cost function is bounded from above, it follows that the expected stage costs exist. Note that these costs are greater than or equal to zero for all $i \in S$ and $u \in C(i)$.

A Markov decision process together with the optimality criterion minimizing the expected total discounted cost is said to be a *discounted Markov decision problem* (see [Put94]).

Definition 2.10 (Discounted Markov Decision Problem)
A *discounted Markov decision problem*, or briefly an MDP, consists of a *discount factor* $\alpha \in (0,1)$ and a *Markov decision process* (S, C, P, cost). The task is to find for each $i \in S$ an admissible policy π^* which minimizes the expected total discounted cost, that is,

$$J_{\pi^*}(i) = \min_{\pi \in \Pi} J_\pi(i).$$

This defines the *optimal value function* $J^* : S \to \mathbb{R}_{\geq 0}$ with $J^*(i) = J_{\pi^*}(i)$. The policy π^* is called an *optimal policy*. △

Note that in general an optimal policy π^* may depend on the initial state. However, in our case a policy exists that is not only optimal for each initial state, but also a *stationary policy*. This follows from the theorem below which provides main results for discounted Markov decision problems. Therefore, the task of an MDP is reduced to find a stationary policy μ^* which minimizes the expected total discounted cost for any state $i \in S$. Since the state space S is finite ($S = \{1, 2, \ldots, N\}$), a stationary policy μ can be seen as vector with N elements from C where the i-th element gives the control to apply at the state with label i. Therefore, $\mu \in C^N$ and

$$J^*(i) = J_{\mu^*}(i) = \min_{\mu \in C^N} J_\mu(i) \quad \forall i \in S.$$

Definition 2.11 (Optimal Control)
Consider an MDP with state space S and control set C. A control $u \in C(i)$ is called *optimal* for a given state $i \in S$ if an optimal policy μ^* exists for the given MDP with $\mu^*(i) = u$.

The following theorem taken from the book of Bertsekas [Ber01] provides a collection of important results for discounted Markov decision problems. For the proof and more details see [Ber01, Volume I, Chapter 7].

Theorem 2.12 (Known Facts). *The following statements hold for an* MDP *with discount factor* $\alpha \in (0,1)$ *and Markov decision process* (S, C, P, cost) *where* $S = \{1, 2, \ldots, N\}$*:*

(a) Given arbitrary real numbers $J_0(1), J_0(2), \ldots, J_0(N)$*, the sequence* $J_k(i)$ *generated by the iteration*

$$J_{k+1}(i) = \min_{u \in C(i)} \{\overline{\mathrm{cost}}(i,u) + \alpha \sum_{j \in S} p_{ij}(u) J_k(j)\}, \quad \forall\, i \in S, \tag{2.2}$$

converges to the optimal cost $J^*(i)$ *for each* $i \in S$*.*

(b) The optimal costs $J^*(1), J^*(2), \ldots, J^*(N)$ satisfy Bellman's equation

$$J^*(i) = \min_{u \in C(i)} \{\overline{\text{cost}}(i,u) + \alpha \sum_{j \in S} p_{ij}(u) J^*(j)\} \quad \forall\, i \in S \tag{2.3}$$

and they are the unique solution of this equation.

(c) For any stationary policy μ the costs $J_\mu(1), J_\mu(2), \ldots, J_\mu(N)$ are the unique solution of the equations

$$J_\mu(i) = \overline{\text{cost}}(i, \mu(i)) + \alpha \sum_{j \in S} p_{ij}(\mu(i)) J_\mu(j) \quad \forall\, i \in S. \tag{2.4}$$

Furthermore, given arbitrary real numbers $J_0(1), J_0(2), \ldots, J_0(N)$, the sequence $J_k(i)$ generated by the iteration

$$J_{k+1}(i) = \overline{\text{cost}}(i, \mu(i)) + \alpha \sum_{j \in S} p_{ij}(\mu(i)) J_k(j), \quad \forall\, i \in S,$$

converges to the cost $J_\mu(i)$ for each $i \in S$.

(d) A stationary policy μ is optimal if and only if for every state $i \in S$, the control $\mu(i)$ attains the minimum in Bellman's equation (2.3)*.*

Remark. The iteration (2.2) $J_k \mapsto J_{k+1}$ is contractive with contraction constant α, that is, for any two vectors $J_k, \tilde{J}_k \in \mathbb{R}^N$ follows $\|J_{k+1} - \tilde{J}_{k+1}\|_\infty \leq \alpha \|J_k - \tilde{J}_k\|_\infty$.

Theorem 2.12 provides the basics for the standard techniques *value iteration*, *policy iteration*, and *linear programming* to compute the optimal value function J^*. These techniques are briefly introduced below. For more details see [Ber01, Put94]. Note that, if the optimal value function J^* is known, an optimal policy μ^* is given by Theorem 2.12(d) as:

$$\mu^*(i) \in \operatorname*{argmin}_{u \in C(i)} \{\overline{\text{cost}}(i,u) + \alpha \sum_{j \in S} p_{ij}(u) J^*(j)\} \quad \forall\, i \in S.$$

In particular, the standard techniques can also be used to compute an optimal policy.

Value Iteration. The value iteration algorithm is based on the iteration equation (2.2)

$$J_{k+1}(i) = \min_{u \in C(i)} \{\overline{\text{cost}}(i,u) + \alpha \sum_{j \in S} p_{ij}(u) J_k(j)\}, \quad \forall\, i \in S.$$

Starting with arbitrary real numbers $J_0(1), J_0(2), \ldots, J_0(N)$ this update step is done successively. Theorem 2.12(a) states that this iteration converges to the optimal cost $J^*(i)$ for each $i \in S$. Generally, value iteration requires an infinite number of iterations. There are approaches which use some error bounds to strengthen the value iteration algorithm (see [Ber01]).

Policy Iteration. The idea of policy iteration is to start with an arbitrary stationary policy μ^0 and generate an improving sequence of stationary policies $\mu^1, \mu^2, \ldots$, that is, $J_{\mu^{k+1}}(i) \leq J_{\mu^k}(i)$ for all $i \in S$ and $k \in \mathbb{N}_0$. The process stops if $J_{\mu^{k+1}}(i) = J_{\mu^k}(i)$ for all $i \in S$. The improving sequence is generated as follows:

$$\mu^{k+1}(i) \in \operatorname*{argmin}_{u \in C(i)} \{\overline{\text{cost}}(i,u) + \alpha \sum_{j \in S} p_{ij}(u) J_{\mu^k}(j)\} \quad \forall\, i \in S. \tag{2.5}$$

Each iteration phase contains two steps. In the first step of the k-th iteration the linear system (2.4) is used to compute the costs $J_{\mu^k}(1), J_{\mu^k}(2), \ldots, J_{\mu^k}(N)$. In the second step a new stationary policy μ^{k+1} is obtained which satisfies (2.5). It can be shown that this algorithm produces an improving sequence of stationary policies and terminates with an optimal policy (see [Ber01]). There is only a finite number of stationary policies since the state set and the control set are finite. Therefore, the policy iteration ends after a finite number steps since an improving sequence of policies is generated and the sequence stops after no improvement is made.

Linear Programming. It is possible to formulate a linear program to compute the optimal value function J^* and an optimal policy μ^* of a given MDP. If the value iteration starts with an initial vector $J_0 = (J_0(1), J_0(2), \ldots, J_0(N))$ which satisfies

$$J_0(i) \leq \min_{u \in C(i)} \{\overline{\text{cost}}(i,u) + \alpha \sum_{j \in S} J_0(j)\}, \quad \forall\, i \in S$$

($J \equiv 0$ satisfies these inequalities), it can be shown that the generated sequence of vectors $J_1, J_2, \ldots$ is monotonously increasing, that is, $J_k(i) \leq J_{k+1}(i)$ for all $i \in S$ and $k \in \mathbb{N}_0$. Therefore, J^* is the component-wise "largest" J that satisfies the constraints

$$J(i) \leq \overline{\text{cost}}(i,u) + \alpha \sum_{j \in S} p_{ij}(u) J(j) \quad \forall\, i \in S, \forall\, u \in C(i).$$

Hence, the optimal value function J^* can be derived as the optimal solution of the following linear program:

$$\begin{aligned} &\max \sum_{i \in S} J(i) \\ &\text{subject to } J(i) \leq \overline{\text{cost}}(i,u) + \alpha \sum_{j \in S} p_{ij}(u) J(j) \qquad \forall i \in S, \forall u \in C(i) \end{aligned} \tag{2.6}$$

Since $\overline{\text{cost}}(i,u) \geq 0$ for all $i \in S$ and $u \in C(i)$, it follows that $J \equiv 0$ is a feasible solution. Therefore, this linear program is feasible. The optimal solution is also unique since Theorem 2.12(b) states that the optimal value function J^* is unique. Mine and Osaki show in [MO70] that the policy iteration algorithm is closely related to this linear programming approach.

2.3 Reviewing both Analysis Methods

This section reviews both analysis methods introduced in the last two sections. This will show why we use these approaches in this thesis to analyze examples of Online Target Date Assignment Problems.

The field of stochastic dynamic optimization provides frameworks which allow to make an average case analysis as well as a worst case analysis for known policies. It is assumed that a probability distribution concerning further evens is given. In real-world applications such a probability distribution is often gained out of collected statistics in the past. Therefore, it is often undetermined how well the given probability distribution reflects the future. Furthermore, the methods for computing the optimal value function are often computationally infeasible because of the complexity of the problems. It also turned out that the approaches for a worst case analysis yield usually trivial results which are often useless. Therefore, Sleator and Tarjan [ST85] introduced a style of worst case analysis which compares the cost of policy with the optimal cost. This method has come to be known as competitive analysis.

Competitive analysis is a worst case analysis since the performance guarantee for an online algorithm must hold for any possible request sequence. Therefore, a competitive online algorithm gives an answer to the question: How "bad" can this algorithm be? The answer is often very pessimistic since sequences which forces an online algorithm to perform "bad" compared to the optimal offline algorithm are often pathological constructions. This means, these sequences are usually totally unlikely to appear in reality. A reason for this is that in a competitive analysis we typically consider arbitrary request sequences whereas in practice only restricted classes of inputs occur. This drawback partly results since competitive analysis makes no assumption concerning the future, that is, after a request is assigned it is assumed that there is no information about the next request. In real-world applications this assumption is sometimes too strong. Therefore, competitive online algorithms often perform much better in practical use as the competitive ratio may imply. To reflect this better performance a line of research suggested other measures for evaluating online algorithms such as smoothed competitive analysis. The smooth competitive analysis is a hybrid between average case analysis and worst case analysis and tries to explain the good performance of algorithms in practice which have a poor worst case behavior. For more details see [BLMS$^+$03]. On the other hand, competitive analysis does not make any restriction on the computational complexity of an online algorithm. The only scarce resource in this approach is information. This means, a competitive online algorithm does not have to be practical at all since online algorithms usually have to provide a decision under real-time aspects and the competitive analysis does not make any restriction on the algorithm's computational complexity.

In this thesis, competitive analysis is used to detect competitive online algorithm for examples of Online Target Date Assignment Problems. Moreover, for these problems we prove with this approach lower bounds on the competitive ratio for any deterministic online algorithm. The traced algorithms are analyzed concerning their average case behavior with a new method from the field stochastic dynamic optimization which is based on the classical linear programming formulation.

3

The Online Target Date Assignment Problem

This chapter is based on joint work with Sven O. Krumke, Nicole Megow, Jörg Rambau, Andreas Tuchscherer, and Tjark Vredeveld [HKM+05]. In the following we formalize the class of online optimization problems introduced in Section 1.2. We consider elementary examples of these problems. For these we present competitive online algorithms and also prove lower bounds on the competitive ratio of any deterministic online algorithm.

The outline of this chapter is as follows. In Section 3.1 we provide a mathematical model for the considered online problems. Elementary examples are presented in Section 3.2. These examples are analyzed in Section 3.3 and Section 3.4 w. r. t. two different objective functions using competitive analysis.

3.1 The Framework

The customer service application described in Section 1.1 is an example for online problems that feature a two-stage structure. In the first stage requests arise one-by-one and have to be assigned to target dates immediately and irrevocably, that is, the requests are processed in an online fashion. Moreover, a request has to be served at a target date within a certain time window (for instance two weeks). Instances for the second stage offline optimization problem are generated from requests assigned to the same target date. We assume that these instances are solved offline to optimality. This section provides a general framework for online problems of this type, the *Online Target Date Assignment Problem* (ONLINETDAP).

The second stage of an ONLINETDAP is given by an *offline optimization problem*. This means, the complete input instance of the second stage is specified in advance. It is asked to find for a given input instance an optimal solution whenever this instance is feasible, that is, for the given input instance exists a feasible solution. The following definition of an offline optimization problem is from the online optimization lecture notes [KR02].

Definition 3.1 (Offline Optimization Problem)
An *offline optimization problem* (over an alphabet Σ) is a quadruple $(\mathcal{I}, F, K, \mathcal{M})$ where

- $\mathcal{I} \subseteq \Sigma^*$ is the set of all admissible input instances with

$$\Sigma^* = \{x_1 x_2 \dots x_n \mid n \in \mathbb{N} \wedge x_i \in \Sigma \text{ for } i = 1, 2, \dots, n\} \cup \{\emptyset\};$$

- $F(I) \subseteq \Sigma^*$ is the set of all feasible solutions for an input instance $I \in \mathcal{I}$. We call F *feasibility function*;
- $K : \mathcal{I} \times \Sigma^* \to \mathbb{R}_{\geq 0}$ is the *objective function* where $K(I, O)$ denotes the objective value of a feasible solution O for an input instance I. This value is only defined if $O \in F(I)$;
- $\mathcal{M} \in \{\min, \max\}$ specifies the objective sense of the problem. △

Offline optimization problems are not the focus of this thesis. Therefore, these problems are seen as a black box or an oracle which provide for every admissible input instance an optimal solution as output whenever the given input instance is feasible.

The first stage of an ONLINETDAP is an online assignment problem where requests have to be assigned immediately and irrevocably to a target date which represents an offline optimization problem. Therefore, a request r contains a specific offline optimization problem information $I(r)$. Moreover, a request r has a *release date* $t(r)$ and *deadline date* $T(r)$ which are both nonnegative integers with $t(r) < T(r)$.

Definition 3.2 (Request)
A *request* r is a triple $(t(r), T(r), I(r))$ where $t(r) \in \mathbb{N}_0$ is the release date, $T(r) \in \mathbb{N} \cup \{\infty\}$ is the deadline date with $t(r) < T(r)$, and $I(r) \in \mathcal{I}$ is the specific offline optimization problem information of request r for an offline optimization problem $Q = (\mathcal{I}, F, K, \mathcal{M})$. This request r is called *compatible* to Q. △

Note that a request consists of relevant information for the first stage as well as for the second stage. The release date and deadline date are only of interest in the first stage. In contrast, the specific offline optimization problem information is useful for both stages since this specific information may influence the assignment decision and is relevant for the offline optimization problem.

Definition 3.3 (Request Sequence)
Let $\mathcal{R}$ be a request set. A *request sequence* is a sequence $\sigma = r_1, r_2, \dots, r_n$ of $n \in \mathbb{N}$ requests from $\mathcal{R}$ which satisfies $t(r_{i-1}) \leq t(r_i)$ for all $i \in \{2, 3, \dots, n\}$. △

The release date and the deadline date of a request define the set of target dates which are *feasible* for this request. A target date is feasible for a request if this target date is later than the release date and not later than the deadline date.

Definition 3.4 (Feasible Target Date)
Let r be a request with release date $t(r)$ and deadline date $T(r)$. A target date $d \in \mathbb{N}$ for r is called *feasible* if $d \in \{t(r)+1, t(r)+2, \ldots, T(r)\}$. △

Since the first feasible target date for a request r is $d = t(r) + 1$, it follows that at the end of each target date the input instance for the second stage concerning the next target date is known completely. Therefore, the second stage of an ONLINETDAP can be solved offline as considered in this work. It could be also of interest that the release date of a request is itself a feasible target date. In that case, a large portion of the input data for the current date is known. Since requests still can be assigned to the this date, the second stage has to be handled as an online optimization problem. Therefore, both stages have to be processed online. This case is not considered in this thesis.

As mentioned before an ONLINETDAP involves two stages. The first stage is an online assignment problem where requests have to be assigned immediately and irrevocably to a target date. All requests assigned to the same target date form an instance of an offline optimization problem solved in the second stage. Therefore, the associated offline optimization problem is called *downstream problem* of the ONLINETDAP. It is assumed that instances of the downstream problem are solved offline to optimality whenever these instances are feasible. The assignment decisions of the first stage influence the overall cost, since they determine the input instances of the downstream problem, and thus the optimal cost of the downstream optimization problem which affect the overall cost. A formal definition of an ONLINETDAP is given below.

Definition 3.5 (Online Target Date Assignment Problem)
An *Online Target Date Assignment Problem* (ONLINETDAP) is given by a triple $(Q, \mathcal{R}, \mathcal{C})$ where

- Q is an offline optimization problem, called *downstream problem* of the ONLINETDAP;
- $\mathcal{R}$ is the set of all possible requests where each request is compatible to Q and each subset of $\mathcal{R}$ forms an admissible input instance for Q;
- $\mathcal{C}$ is the set of cost functions $\text{cost}_m : \mathcal{R}^m \times \mathbb{N}^m \to \mathbb{R}_{\geq 0}$ for $m = 1, 2, \ldots$.

An instance is given by a request sequence $\sigma = r_1, r_2, \ldots r_n$ of $n \in \mathbb{N}$ requests from $\mathcal{R}$. The task is to assign each request r_i to a feasible target date $d_i \in \mathbb{N}$ for $i = 1, 2, \ldots, n$ such that the cost $\text{cost}_n(\sigma, d_1, d_2, \ldots, d_n)$ is minimized. In doing so, d_i may depend only on the requests $r_1, r_2, \ldots, r_i$ for $i = 1, 2, \ldots, n$. △

Remark. ONLINETDAPs are only of interest if the objective function of the downstream problem influences the cost functions of the given ONLINETDAP. Otherwise, an ONLINETDAP is just a specific request-answer game as defined in Definition 2.1.

The research goal is to find competitive online algorithms for various versions of ONLINETDAPs. For a given ONLINETDAP and a given input instance $\sigma = r_1, r_2, \ldots, r_n$ ($n \in \mathbb{N}$), an

online algorithm ALG has to generate an assignment as output according to Definition 2.2 of a deterministic online algorithm. Therefore,

$$\mathsf{ALG}[\sigma] = (d_1, d_2, \ldots, d_n) \in \mathbb{N}^n$$

where d_i depends only on the requests $r_1, r_2, \ldots, r_i$ for $i = 1, 2, \ldots, n$. The cost incurred by ALG is denoted by $\mathsf{ALG}(\sigma)$ and defined as

$$\mathsf{ALG}(\sigma) = \mathrm{cost}_n(\sigma, \mathsf{ALG}[\sigma]).$$

Since subsequences of requests which are assigned to the same target date are of special interest for the downstream problem, the following often used notation is introduced.

Notation. Let $(d_1, d_2, \ldots, d_n) \in \mathbb{N}^n$ be an assignment for a sequence $\sigma = r_1, r_2, \ldots, r_n$ of $n \in \mathbb{N}$ requests. The subsequence of σ which includes all requests assigned to the target date $d \in \mathbb{N}$ is denoted with σ_d.

The task is to obtain an assignment according to the specific rules of the ONLINETDAP such that the overall cost is minimized. Therefore, an assignment for a request sequence of an ONLINETDAP is called *feasible* if all requests are assigned to feasible target dates and all resulting input instances of the downstream problem are feasible.

Definition 3.6 (Feasible Solution / Feasible Assignment)
Let $(Q, \mathcal{R}, \mathcal{C})$ be an ONLINETDAP, $\sigma = r_1, r_2, \ldots, r_n$ a sequence of $n \in \mathbb{N}$ requests from $\mathcal{R}$, and $(d_1, d_2, \ldots, d_n) \in \mathbb{N}^n$ an assignment for the request sequence σ. This solution (assignment) is called *feasible* for σ and $(Q, \mathcal{R}, \mathcal{C})$ if the following two conditions are satisfied:

(i) Each request of σ is assigned to a feasible target date, that is, $t(r_i) < d_i \leq T(r_i)$ for $i = 1, 2, \ldots, n$;

(ii) The resulting input instances of the downstream problem $Q = (\mathcal{I}, F, K, M)$ are feasible, that is, $\sigma_d \in \mathcal{I}$ and $F(\sigma_d) \neq \emptyset$ if $|\sigma_d| \neq 0$ for $d = 1, 2, \ldots, k$ where $|\sigma_d|$ determines the number of requests the sequence σ_d has and $k = \max\{T(r_i) \mid i = 1, 2, \ldots, n\}$. △

With the definition of a feasible assignment we are able to define a *feasible online algorithm*.

Definition 3.7 (Feasible Online Algorithm)
An online algorithm ALG is called *feasible* for an ONLINETDAP if ALG generates a feasible solution for each request sequence σ whenever a feasible solution for σ exists. △

As mentioned before, it is assumed that any input instance of a downstream problem can be solved to optimality in the case this input instance is feasible. Therefore, the *downcost function* for a downstream problem $Q = (\mathcal{I}, F, K, \mathcal{M})$ is defined as

$$\mathrm{downcost}(I) = \mathcal{M}\{K(I, O) \mid O \in F(I)\} \quad \forall I \in \mathcal{I} : F(I) \neq \emptyset.$$

Note that the downcost function is only defined for input instances which are feasible. Moreover, the downcost function determines for such an input instance the optimal value. This value is called *downstream cost*.

3.2 Elementary ONLINETDAPs

Based on the given cost model, there are two intuitive objectives for an ONLINETDAP. One is to minimize the total downstream cost over all target dates. Another is to minimize the maximum downstream cost over all target dates. Each objective defines a special class of ONLINETDAPs, the MIN-TOTAL ONLINETDAPs and the MIN-MAX ONLINETDAPs. This section introduces these two classes of ONLINETDAPs. Moreover, two elementary downstream problems are considered and presented in this section. One is the classical one-dimensional bin-packing problem and the other is a parallel-machine scheduling problem. These downstream problems and classes of ONLINETDAPs lead to the following instantiations:

- MIN-TOTAL ONLINETDAP w. r. t. bin-packing;
- MIN-TOTAL ONLINETDAP w. r. t. parallel-machine scheduling;
- MIN-MAX ONLINETDAP w. r. t. bin-packing;
- MIN-MAX ONLINETDAP w. r. t. parallel-machine scheduling.

These examples are analyzed using competitive analysis in Section 3.3 and Section 3.4, respectively.

3.2.1 Minimize the Total Downstream Cost (MIN-TOTAL ONLINETDAP).

The focus of optimization problems is often a resource which is consumed, such as money or oil. These resources are called *non-renewable resources* in the literature. A standard task is to achieve a certain global goal with a minimal input of a non-renewable resource. Since in the case of an ONLINETDAP such a resource is consumed on each single target date, the objective to minimize the total usage of this resource can be of interest. This leads to the class of MIN-TOTAL ONLINETDAPs.

Definition 3.8 (MIN-TOTAL ONLINETDAP)
A MIN-TOTAL ONLINETDAP is an ONLINETDAP $(Q, \mathcal{R}, \mathcal{C})$ with an arbitrary downstream problem Q, a request set $\mathcal{R}$ where each request from $\mathcal{R}$ is compatible to Q, and a set $\mathcal{C}$ of cost functions $\mathrm{cost}_m : \mathcal{R}^m \times \mathbb{N}^m \to \mathbb{R}_{\geq 0}$ for $m = 1, 2, \ldots$. These cost functions are defined as

$$\mathrm{cost}_m(\sigma, d_1, d_2, \ldots, d_m) = \sum_{d=1}^{k} \mathrm{downcost}(\sigma_d)$$

where $\sigma = r_1, r_2, \ldots, r_m$ is a sequence of request from $\mathcal{R}$, $(d_1, d_2, \ldots, d_m) \in \mathbb{N}^m$ is a feasible assignment for the request sequence σ, and $k = \max\{T(r_i) \mid i = 1, 2, \ldots, m\}$. Therefore, a MIN-TOTAL ONLINETDAP is given by the pair $(Q, \mathcal{R})$. △

3.2.2 Minimizing the Maximum Downstream Cost (MIN-MAX ONLINETDAP).

Apart from the non-renewable resources there are resources which can be used without losing them afterwards, such as workers or machines. These resources are called *renewable resources*. Since these resources can be very expensive, one aims to find solutions that require a minimal amount of a renewable resource at any point of time. In our case the resource has to be available on each single target date. Therefore, such a local goal has to hold on each target date. Hence, minimizing the maximum usage of a renewable resource over all target dates is a suitable goal. This defines the class of MIN-MAX ONLINETDAPs.

Definition 3.9 (MIN-MAX ONLINETDAP)
A MIN-MAX ONLINETDAP is an ONLINETDAP $(Q, \mathcal{R}, \mathcal{C})$ with an arbitrary downstream problem Q, a request set $\mathcal{R}$ where each request from $\mathcal{R}$ is compatible to Q, and a set $\mathcal{C}$ of cost functions $\text{cost}_m : \mathcal{R}^m \times \mathbb{N}^m \to \mathbb{R}_{\geq 0}$ for $m = 1, 2, \ldots$. These cost functions are defined as

$$\text{cost}_m(\sigma, d_1, d_2, \ldots, d_m) = \max\{\text{downcost}(\sigma_d) \mid d = 1, 2, \ldots, k\}$$

where $\sigma = r_1, r_2, \ldots, r_m$ is a sequence of request from $\mathcal{R}$, $(d_1, d_2, \ldots, d_m) \in \mathbb{N}^m$ is a feasible assignment for the request sequence σ, and $k = \max\{T(r_i) \mid i = 1, 2, \ldots, m\}$. Therefore, a MIN-MAX ONLINETDAP is given by the pair $(Q, \mathcal{R})$. △

3.2.3 Elementary Downstream Problems

This section introduces the two elementary downstream problems which are of interest in this thesis: the bin-packing problem and a parallel-machine scheduling problem.

Bin-Packing Problem

The classical one-dimensional bin-packing problem accepts as input a sequence $L = s_1, s_2, \ldots, s_n$ of $n \in \mathbb{N}$ items each with a size $s_i \in (0, 1]$ for $i = 1, 2, \ldots, n$. It is asked to pack these items into a minimum number of unit-capacity bins. Formally, we are looking for a partition of the index set of L into a minimum number of sets $B_1, B_2, \ldots, B_m$ such that

$$\sum_{i \in B_j} s_i \leq 1 \quad \forall\, j \in \{1, 2, \ldots, m\}.$$

This describes the offline version of bin-packing where the assumption is made that the input sequence is known in advance. Apart from the offline version there is an online version where items arrive one-by-one and have to be packed immediately and irrevocably. The online version is not of interest in this thesis since the downstream problems are solved offline. [CGJ96] provides a survey on approximation algorithms for the classical one-dimensional bin-packing problem including results for the online version as well.

The focus of this work is on two variants of the offline version. One variant considers a bounded number $b \in \mathbb{N}$ of unit-capacity bins. The other assumes an infinite number of unit-capacity bins ($b = \infty$) as in the classical problem setting. Furthermore, the objective function determines the minimum number of unit-capacity bins b^* needed to serve an input sequence if $b^* \leq b$. Otherwise, the given input sequence is infeasible. Hence, a bin-packing problem has one parameter $b \in \mathbb{N} \cup \{\infty\}$ which determines the number of available unit-capacity bins. It follows a (very) formal definition.

Definition 3.10 (Bin-Packing Problem)
A *bin-packing problem* with $b \in \mathbb{N} \cup \{\infty\}$ unit-capacity bins is an offline optimization problem $(\mathcal{I}, F, K, \mathcal{M})$ where $\mathcal{I} = \{s_1, s_2, \ldots, s_n \mid n \in \mathbb{N}_0 \wedge 0 < s_i \leq 1\}$, $F(I)$ is the set of all feasible packings for an $I \in \mathcal{I}$ which do not require more than b unit-capacity bins, K is the objective function which returns for a feasible packing the number of used bins, and the objective sense is to minimize this number ($\mathcal{M} = \min$). △

Note that in the case where the number of available bins is bounded there exist input instances which are infeasible.

Furthermore, in our setting a bin-packing problem is of interest where all items have the same size. If this is the case we mention it separately. Moreover, in this situation the terms a bin is *fully filled* and a bin is *partially filled* are used. A fully filled bin means that this bin contains the maximum number of items (with equal size) it can hold and a partially filled bin is not fully filled. Note that in the case where all items have the same size the bin-packing problem by itself is trivial. However, in connection with the ONLINETDAP framework there are interesting results, as we show later.

Parallel-Machine Scheduling

Scheduling problems appear in many different versions. The basic problem receives a sequences $L = p_1, p_2, \ldots, p_n$ of $n \in \mathbb{N}$ jobs as input where each job has a processing time $p_i > 0$ for $i = 1, 2, \ldots, n$. We want to assign these jobs to $m \in \mathbb{N}$ parallel and identical machines such that the makespan, that is, the latest completion time of a job, is minimized. Scheduling problems arise as offline versions as well as online versions. Again the online versions are not of interest since the downstream problems are solved offline. For more details on online scheduling see [Sga98]. In the offline case the whole input sequence and the processing time of each job of this sequence is known. Formally, we are looking for a partition of the index set of L into m sets $M_1, M_2, \ldots, M_m$ such that

$$\max\{\sum_{i \in M_j} p_i \mid j = 1, 2, \ldots, m\}$$

is minimized.

The scheduling problems which are of interest in this work are from the same design as the bin-packing problems. One is a bounded variant where $m \in \mathbb{N}$ machines are available. The other is

the corresponding unbounded variant where unlimited machines are available ($m = \infty$). Moreover, the scheduling problems we are looking at accept as input any sequence of positive real numbers where each number represents a job with a processing time. The downcost function determines the minimum makespan to serve an input sequence on m parallel and identical machines in a non-preemptive way. This means, each job has to be processed on a machine in one piece. The parallel-machine scheduling problem has one parameter $m \in \mathbb{N} \cup \{\infty\}$ determining the number of parallel and identical machines.

Definition 3.11 (Parallel-Machine Scheduling Problem)
A *parallel-machine scheduling problem* with $m \in \mathbb{N} \cup \{\infty\}$ parallel and identical machines is an offline optimization problem $(\mathcal{I}, F, K, \mathcal{M})$ where $\mathcal{I} = \{p_1, p_2, \ldots, p_n \mid n \in \mathbb{N}_0 \wedge p_i > 0\}$, $F(I)$ is the set of all feasible schedules for an $I \in \mathcal{I}$ on no more than m parallel and identical machines where each job is processed in a non-preemptive way, K is the objective function which returns the makespan of a feasible schedule, and the objective sense is to minimize this makespan ($\mathcal{M} = \min$). △

Note that the scheduling problem provides unlimited capacity independently of the number of available machines m. Therefore, any input instances is feasible.

Again in the setting of ONLINETDAPs a parallel-machine scheduling problem is of interest where all jobs have the same processing time. If this is the case we mention it also separately. Obviously, this scheduling problem by itself is trivial but in connection with the ONLINETDAP framework this setting becomes interesting.

3.3 Competitive Analysis for MIN-TOTAL ONLINETDAPs

The previous section introduced the class of MIN-TOTAL ONLINETDAPs as well as two elementary downstream problems, the bin-packing problem and the parallel-machine scheduling problem. These form elementary MIN-TOTAL ONLINETDAPs. Additionally, we assume in the following that all requests have $\delta \in \mathbb{N} \cup \{\infty\}$ feasible target dates, that is, $T(r) - t(r) = \delta$ for each request r. For the associated MIN-TOTAL ONLINETDAPs this section presents results received by competitive analysis. Most of the results were obtained in joint work with Sven O. Krumke, Nicole Megow, Jörg Rambau, Andreas Tuchscherer and Tjark Vredeveld [HKM+05]. The proofs for these results are more detailed as in [HKM+05].

We assume that all requests have $\delta \in \mathbb{N} \cup \{\infty\}$ feasible target dates. A set of requests which includes only requests with $\delta \in \mathbb{N}$ feasible target dates, that is, $T(r) - t(r) = \delta$, is denoted by $\mathcal{R}_\delta$. Such a request set $\mathcal{R}_\delta$ is called *restricted request set*. In general a request r has a release date $t(r)$ and a deadline date $T(r)$. If a restricted request set $\mathcal{R}_\delta$ with $\delta \in \mathbb{N} \cup \{\infty\}$ is considered, the deadline of each request $r \in \mathcal{R}_\delta$ is given by $T(r) = t(r) + \delta$. Hence, the feasible target dates of a request r are completely described by the release date $t(r)$ and δ. Since in this section we consider only restricted request sets, a request r is a pair $(t(r), I(r))$ instead of a triple $(t(r), T(r), I(r))$.

Consider a restricted request set $\mathcal{R}_\delta$ with $\delta = 1$. In this case each request $r \in \mathcal{R}_\delta$ has only one feasible target date. Therefore, an algorithm which assigns each request to its feasible target date is a feasible online algorithm. Moreover, such an algorithm is 1-competitive since any request sequence has no more than one feasible assignment. Therefore, $\delta > 1$ is assumed.

3.3.1 The General Online Algorithm PACKTOGETHERORDELAY

Before elementary MIN-TOTAL ONLINETDAP examples are analyzed the general algorithm PACK-TOGETHERORDELAY, or briefly PTD, is introduced. Moreover, in the case of MIN-TOTAL ONLINETDAP a general result can be shown for the algorithm PTD if the downstream problem of a MIN-TOTAL ONLINETDAP satisfies some properties. To describe the algorithm PTD the following definition is necessary.

Definition 3.12 (Used Target Date)
A target date is called *used* if a request r exists which has been already assigned to this target date. △

The algorithm PTD can be described verbally as follows. PTD accepts as input a request r. Moreover, the algorithm needs the current assignments to the feasible target dates of this request. The output of PTD is a target date d for the given request r. First of all, the algorithm PTD tries to pack requests together. This means, if for request r feasible and used target dates exist, then the algorithm PTD assigns this request to the earliest of them. Otherwise, the algorithm uses a delay tactic and assigns request r to its deadline target date $T(r)$. Algorithm 1 gives a formal description of PTD. Note that the algorithm PTD only works in general if each request r has a deadline $T(r) \in \mathbb{N}$. Otherwise, the delay strategy of the algorithm is not well defined. Moreover, this algorithm does not compute a feasible assignment in general since the pack together tactic can produce input instances for the downstream problem which are infeasible. In the cases where this algorithm is considered it is ensured that any input instance of the downstream problem is feasible and each request has a deadline.

The following example illustrates the workflow of the algorithm PTD.

Example 3.13 (PTD). Consider an ONLINETDAP with restricted request set $\mathcal{R}_\delta$ where $\delta \in \mathbb{N}$. Moreover, let $\sigma = r_1, r_2, \ldots, r_n$ be a sequence of $n \in \mathbb{N}$ requests from $\mathcal{R}_\delta$ where request r_i has a release date $t(r_i) = i - 1$ for $i = 1, 2, \ldots, n$. Therefore, request r_i has to be assigned to a target date $d \in \{i, i+1, \ldots, i+\delta-1\}$ for $i = 1, 2, \ldots, n$. The algorithm PTD generates the following assignment:

$$\mathsf{PTD}[\sigma] = (a_1, a_2, \ldots, a_n) \in \mathbb{N}^n \quad \text{with} \quad a_i = \left\lceil \frac{i}{\delta} \right\rceil \cdot \delta \quad \text{for } i = 1, 2, \ldots, n.$$

Figure 3.1 illustrates the assignment of PTD for σ. △

Input : A request r and the current assignments $\sigma_{t(r)+1}, \sigma_{t(r)+2}, \ldots, \sigma_{T(r)}$ to the feasible target dates of request r.
Output : An assignment $d \in \mathbb{N}$ for request r.

for $d \leftarrow t(r)+1$ **to** $T(r)$ **do**
  if $\sigma_d \neq \emptyset$ **then**
    // pack together tactic;
    return d;
// delay tactic;
return $d \leftarrow T(r)$;

Algorithm 1: PACKTOGETHERORDELAY (PTD)

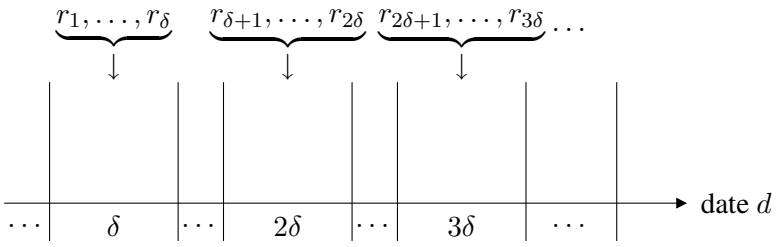

Figure 3.1: Assignment of PTD for the request sequence $\sigma = r_1, r_2, \ldots, r_n$ from Example 3.13.

While studying the algorithm PTD for a MIN-TOTAL ONLINETDAP with restricted request set $\mathcal{R}_\delta$ ($\delta \in \mathbb{N}$), two important observations can be made. On the one hand the used target dates the algorithm PTD produces are at least δ time units apart. On the other hand for any request sequence σ where $D \subseteq \mathbb{N}$ denotes the set of all target dates used by the algorithm PTD to serve σ it follows:

$$\mathsf{PTD}(\sigma) = \sum_{d \in D} \mathsf{PTD}(\sigma_d).$$

The following lemmas prove these two observations since they are needed to show a general result for the algorithm PTD in the current problem setting.

Lemma 3.14. *Consider an* ONLINETDAP *with restricted request set $\mathcal{R}_\delta$ where $\delta \in \mathbb{N}$. For a given sequence $\sigma = r_1, r_2, \ldots, r_n$ of $n \in \mathbb{N}$ requests from $\mathcal{R}_\delta$ let $d_1 < d_2 < \cdots < d_h$ be the used target dates the algorithm* PTD *produces. It follows that $d_i - d_{i-1} \geq \delta$ for $i = 2, 3, \ldots, h$.*

Proof. Given an ONLINETDAP with restricted request set $\mathcal{R}_\delta$ where $\delta \in \mathbb{N}$ and consider a sequence $\sigma = r_1, r_2, \ldots, r_n$ of $n \in \mathbb{N}$ requests from $\mathcal{R}_\delta$. From the way the algorithm PTD operates and the fact that a restricted request set $\mathcal{R}_\delta$ is given it follows that for each request of σ at most one used target date exists which is also a feasible target date for this request. If for a request r

of σ a feasible and used target date exists, then the algorithm PTD uses the pack together tactic and assigns this request to this target date and does not generate a new used target date. Otherwise, all δ feasible target dates of request r are not used and the algorithm uses the delay strategy and assigns this request to its deadline $T(r)$. This produces a new used target date which is at least δ time units away from the last one since after the assignment the first $\delta - 1$ feasible target dates of request r are still not used. Hence, a target date which is not further away than δ time units from a used target date cannot turn into a used target date. Therefore, the used target dates the algorithm PTD produces for σ are at least δ time units apart. □

Note that the above lemma holds for ONLINETDAPs with a restricted request set and an arbitrary set $\mathcal{C}$ of cost functions.

Lemma 3.15. *Let $(Q, \mathcal{R}_\delta)$ be a* MIN-TOTAL ONLINETDAP *with $\delta \in \mathbb{N}$, $\sigma = r_1, r_2, \ldots, r_n$ a sequence of $n \in \mathbb{N}$ requests from $\mathcal{R}_\delta$, and $D \subset \mathbb{N}$ the set of used target dates the algorithm* PTD *produces for σ, then the following equation holds:*

$$\mathsf{PTD}(\sigma) = \sum_{d \in D} \mathsf{PTD}(\sigma_d).$$

Proof. Consider a MIN-TOTAL ONLINETDAP with restricted request set $\mathcal{R}_\delta$ where $\delta \in \mathbb{N}$. For a given request sequence $\sigma = r_1, r_2, \ldots, r_n$ of $n \in \mathbb{N}$ requests from $\mathcal{R}_\delta$, let $\{d_1, d_2, \ldots, d_h\} \subset \mathbb{N}$ denote the set of target dates the algorithm PTD uses to serve σ. The first requests of the subsequences $\sigma_{d_1}, \sigma_{d_2}, \ldots, \sigma_{d_h}$ of σ are assigned by PTD using the delay tactic. Hence, the assignments for these subsequences are not influencing each other, that is,

$$\mathsf{PTD}[\sigma] = (\mathsf{PTD}[\sigma_{d_1}], \mathsf{PTD}[\sigma_{d_2}], \ldots, \mathsf{PTD}[\sigma_{d_h}]).$$

Therefore,

$$\begin{aligned}
\mathsf{PTD}(\sigma) &= \mathrm{cost}(\sigma, \mathsf{PTD}[\sigma]) \\
&= \mathrm{cost}(\sigma, \mathsf{PTD}[\sigma_{d_1}], \mathsf{PTD}[\sigma_{d_2}], \ldots, \mathsf{PTD}[\sigma_{d_h}]) \\
&= \sum_{i=1}^{h} \mathrm{downcost}(\sigma_{d_i}) \\
&= \sum_{i=1}^{h} \mathrm{cost}(\sigma_{d_i}, \mathsf{PTD}[\sigma_{d_i}]) \\
&= \sum_{i=1}^{h} \mathsf{PTD}(\sigma_{d_i}).
\end{aligned}$$

□

By the results of Lemma 3.14 and Lemma 3.15 it is easy to prove that PTD is 2-competitive for a MIN-TOTAL ONLINETDAP $(Q, \mathcal{R}_\delta)$ with $\delta \in \mathbb{N}$ if the downstream problem Q satisfies some properties.

Theorem 3.16 ([HKM+05]). *Let $(Q, \mathcal{R}_\delta)$ be a* MIN-TOTAL ONLINETDAP *with $\delta \in \mathbb{N}$ and a downstream problem $Q = (\mathcal{I}, F, K, \mathcal{M})$ which has unlimited resources, that is, $F(I) \neq \emptyset$ for all $I \in \mathcal{I} \setminus \{\emptyset\}$ (each input instance is feasible). For the given problem $(Q, \mathcal{R}_\delta)$ the algorithm* PTD *is 2-competitive, if the following two properties hold*

(i) The downcost function from Q is a monotonically increasing function, that is,

$$\text{downcost}(\bar{I}) \leq \text{downcost}(I) \quad \forall\, I \in \mathcal{I},\ \bar{I} \subset I;$$

(ii) For each disjoint partition $I^{(1)}, I^{(2)}, \ldots, I^{(k)}$ with $k \in \mathbb{N}$ of any given input instance $I \in \mathcal{I}$ of Q it holds

$$\text{downcost}(I) \leq \sum_{i=1}^{k} \text{downcost}(I^{(i)}).$$

Proof. Let $(Q, \mathcal{R}_\delta)$ be a MIN-TOTAL ONLINETDAP with $\delta \in \mathbb{N}$ meeting the conditions of the theorem. Since the downstream problem Q has unlimited resources available, an online algorithm which assigns each request to a feasible target date is already a feasible online algorithm. Therefore, the algorithm PTD is a feasible online algorithm for the given problem. For a given sequence $\sigma = r_1, r_2, \ldots, r_n$ of $n \in \mathbb{N}$ requests from $\mathcal{R}_\delta$, let $d_1 < d_2 < \cdots < d_h$ be all target dates used by PTD. The used target dates are divided into two disjoint sets

$$D_{\text{odd}} := \{d_i \mid 1 \leq i \leq h \wedge i \text{ odd}\} \quad \text{and} \quad D_{\text{even}} := \{d_i \mid 1 \leq i \leq h \wedge i \text{ even}\}.$$

These sets induce

$$\sigma_{\text{odd}} := \bigcup_{d \in D_{\text{odd}}} \sigma_d \quad \text{and} \quad \sigma_{\text{even}} := \bigcup_{d \in D_{\text{even}}} \sigma_d$$

as the subsequences of requests assigned to "odd" respectively "even" target dates. With Lemma 3.15 follows

$$\text{PTD}(\sigma) = \text{PTD}(\sigma_{d_1}) + \text{PTD}(\sigma_{d_2}) + \cdots + \text{PTD}(\sigma_{d_h}) = \text{PTD}(\sigma_{\text{odd}}) + \text{PTD}(\sigma_{\text{even}}). \tag{3.1}$$

From Lemma 3.14 it is known that the used target dates are at least δ time units apart. Hence,

$$\forall d, \bar{d} \in D_a : d \neq \bar{d} \Rightarrow |d - \bar{d}| \geq 2\delta \qquad a \in \{\text{odd}, \text{even}\}.$$

This implies that two requests of σ_{odd} (σ_{even}) that have not been assigned to the same target date have no overlap in their feasible target dates. Therefore, PTD assigns the requests from the subsequence σ_{odd} (σ_{even}) which have an overlap in their feasible target dates to the same target date. Furthermore, Property (ii) implies for all $i \in \{1, 2, \ldots, h\}$ that

$$\text{OPT}(\sigma_{d_i}) \geq \text{downcost}(\sigma_{d_i}).$$

Putting the last two arguments together it follows

$$\mathsf{OPT}(\sigma_{\text{odd}}) = \sum_{\substack{i=1: \\ i \text{ odd}}}^{h} \mathsf{OPT}(\sigma_{d_i}) \geq \sum_{\substack{i=1: \\ i \text{ odd}}}^{h} \text{downcost}(\sigma_{d_i}) = \mathsf{PTD}(\sigma_{\text{odd}}).$$

In the same manner it is shown that $\mathsf{OPT}(\sigma_{\text{even}}) \geq \mathsf{PTD}(\sigma_{\text{even}})$. Hence,

$$\mathsf{PTD}(\sigma_{\text{odd}}) = \mathsf{OPT}(\sigma_{\text{odd}}) \quad \text{and} \quad \mathsf{PTD}(\sigma_{\text{even}}) = \mathsf{OPT}(\sigma_{\text{even}}). \tag{3.2}$$

Moreover, an optimal assignment for σ gives a feasible assignment for σ_{odd} (σ_{even}) by removing the irrelevant requests. Since the downcost function is monotonically increasing (Property (i)), it follows

$$\mathsf{OPT}(\sigma_{\text{odd}}) \leq \mathsf{OPT}(\sigma) \quad \text{and} \quad \mathsf{OPT}(\sigma_{\text{even}}) \leq \mathsf{OPT}(\sigma).$$

Putting this together with Equation (3.1) and Equations (3.2), we have

$$\begin{aligned} \mathsf{PTD}(\sigma) &= \mathsf{PTD}(\sigma_{\text{odd}}) + \mathsf{PTD}(\sigma_{\text{even}}) \\ &= \mathsf{OPT}(\sigma_{\text{odd}}) + \mathsf{OPT}(\sigma_{\text{even}}) \\ &\leq 2 \cdot \mathsf{OPT}(\sigma). \end{aligned}$$

Hence, PTD is a feasible online algorithm and 2-competitive for the given MIN-TOTAL ONLINETDAP. □

3.3.2 Downstream Problem Bin-Packing

We now analyze MIN-TOTAL ONLINETDAP w. r. t. bin-packing as the downstream problem and a restricted request set $\mathcal{R}_\delta$ with $\delta \in \{2, 3, \dots\} \cup \{\infty\}$.

An instance of the bin-packing problem consists of $b \in \mathbb{N} \cup \{\infty\}$ unit-capacity bins and accepts as input a sequence $L = s_1, s_2, \dots, s_n$ of $n \in \mathbb{N}$ items with size $s_i \in (0, 1]$ for $i = 1, 2, \dots, n$. The downcost function determines the minimum number b^* of bins to serve an input sequence if $b^* \leq b$. Otherwise, the sequence has no feasible solution. A request r has a size $s(r) \in (0, 1]$ which represents the specific downstream problem information and a release date $t(r) \in \mathbb{N}_0$. Hence, a request r is a pair $(t(r), s(r))$. A more specific description of bin-packing as downstream problem is given in Section 3.2.3.

There are three cases of interest. The first one assumes that all requests have no deadline ($\delta = \infty$). The other two cases consider that all requests have a deadline and differ in their assumption on the number of available bins. One case assumes that a bounded number of bins ($b \in \mathbb{N}$) is available on each target date. The other case considers that as many bins ($b = \infty$) as necessary are available on each target date. Table 3.1 summarizes the results presented in this section.

Parameter		arbitrary item size		equal item size $s \in (0, 1]$	
δ	b	lower bound	upper bound	lower bound	upper bound
∞	$\mathbb{N} \cup \{\infty\}$	no competitive online algorithm		$\lfloor 1/s \rfloor$	$\lfloor 1/s \rfloor$
$\mathbb{N}_{\geq 2}$	$\mathbb{N}$	no feasible online algorithm		$\min\{\lfloor 1/s \rfloor, \delta\}$	$\lfloor 1/s \rfloor$
$\mathbb{N}_{\geq 2}$	∞	$3/2$	2	1	1

Table 3.1: Lower bounds on the competitive ratio of deterministic online algorithms as well as upper bounds for the best known deterministic online algorithms for MIN-TOTAL ONLINETDAP w. r. t. bin-backing and a restricted request set $\mathcal{R}_\delta$. The number of available unit-capacity bins is denoted by b.

Unlimited number of feasible target dates ($\delta = \infty$)

If each request has no deadline ($\delta = \infty$), there exists no competitive deterministic online algorithm independently of the number of bins available on each target date. Only if all request have the same size, there exist competitive deterministic online algorithms.

Theorem 3.17. *Consider the* MIN-TOTAL ONLINETDAP *with downstream problem bin-packing where each request has no deadline ($\delta = \infty$). For this problem setting there exists no competitive deterministic online algorithm.*

Proof. Given a MIN-TOTAL ONLINETDAP $(Q, \mathcal{R}_\infty)$ where Q is a bin-packing problem with $b \in \mathbb{N} \cup \{\infty\}$ unit-capacity bins, let ALG be an arbitrary deterministic online algorithm for the problem $(Q, \mathcal{R}_\infty)$ which assigns each request to a feasible target date. Since ALG is a deterministic online algorithm, it follows that the answer sequence for any request sequence is known in advance. With that knowledge the claim of the theorem is proved by showing that for all $n \in \mathbb{N}$ there exists a sequence σ of requests from $\mathcal{R}_\infty$ with

$$\mathsf{ALG}(\sigma) = n \cdot \mathsf{OPT}(\sigma).$$

This implies that ALG is not competitive. Let $n \in \mathbb{N}$ and $\sigma = r_1, r_2, \ldots, r_n$ be a sequence of n requests from $\mathcal{R}_\infty$ where each request r_i has size $s(r_i) = 1/n$ for $i = 1, 2, \ldots, n$. Moreover, the release dates are defined as:

$$t(r_i) = \begin{cases} 0, & \text{if } i = 0 \\ \mathsf{ALG}[r_{i-1}], & \text{otherwise.} \end{cases}$$

ALG is not able to assign any two of the requests from σ to the same target date since a new request is released on the date where the previous request is assigned to (This is possible since ALG is a deterministic algorithm). Therefore, the algorithm needs n bins to serve the request sequence σ. Assigning all requests to a joint target date after the release date of request r_n leads to a feasible solution, since all requests have no deadline. Moreover, such an assignment needs 1 bin which is the optimal cost for the given sequence σ. Hence, the algorithm ALG is not competitive. □

If it is additionally assumed in this setting that all requests have the same size $s \in (0, 1]$, any feasible deterministic online algorithm has a competitive ratio of $\lfloor 1/s \rfloor$. For example, the algorithm which assigns each request r to the earliest feasible target date $d = t(r) + 1$ is $\lfloor 1/s \rfloor$-competitive.

Theorem 3.18. *If additionally to the setting of the previous theorem is assumed that all requests have the same size $s \in (0, 1]$, then any feasible deterministic online algorithm has a competitive ratio of $\lfloor 1/s \rfloor$.*

Proof. Consider a MIN-TOTAL ONLINETDAP $(Q, \mathcal{R}_\infty)$ where Q is a bin-packing problem with $b \in \mathbb{N} \cup \{\infty\}$ unit-capacity bins and all requests from $\mathcal{R}_\infty$ have the same size $s \in (0, 1]$. Let $k = \lfloor 1/s \rfloor$ define the maximum number of request an unit-capacity bin can hold. Moreover, let ALG be an arbitrary feasible deterministic online algorithm for this problem. To prove this theorem the following two properties are shown:

1. ALG is k-competitive for the given MIN-TOTAL ONLINETDAP;
2. and k is a lower bound on the competitive ratio of ALG for the problem $(Q, \mathcal{R}_\infty)$.

Proof of Property 1. Let $\sigma = r_1, r_2, \ldots, r_n$ be a sequence of $n \in \mathbb{N}$ requests from $\mathcal{R}_\infty$. Since one bin can not hold more than k requests, it follows:

$$\mathsf{OPT}(\sigma) \geq \left\lceil \frac{n}{k} \right\rceil \geq \frac{n}{k} \quad \Leftrightarrow \quad n \leq k \cdot \mathsf{OPT}(\sigma).$$

Moreover, $\mathsf{ALG}(\sigma) \leq n$. Therefore,

$$\mathsf{ALG}(\sigma) \leq n \leq k \cdot \mathsf{OPT}(\sigma).$$

Hence, ALG is k-competitive for the given problem $(Q, \mathcal{R}_\infty)$.

Proof of Property 2. Consider the following sequence $\sigma = r_1, r_2, \ldots, r_k$ of k requests from $\mathcal{R}_\delta$ with release dates

$$t(r_i) = \begin{cases} 0, & \text{if } i = 0 \\ \mathsf{ALG}[r_{i-1}], & \text{otherwise.} \end{cases}$$

This is the same construction as in the proof for Theorem 3.17. The algorithm ALG is not able to assign any two of the requests to the same target date. Hence, this algorithm needs k bins to serve this request sequence. The optimal cost is 1 which is achieved by assigning all k requests to a joint target date after the release date of request r_k. Therefore, k is a lower bound on the competitive ratio of ALG.

Hence, the algorithm ALG has a competitive ratio of k for the given MIN-TOTAL ONLINE-TDAP. □

The following two cases assume that all requests have a deadline ($\delta \in \{2, 3, \ldots\}$).

Input : A request r, the current assignments $\sigma_{t(r)+1}, \sigma_{t(r)+2}, \ldots, \sigma_{T(r)}$ to the feasible target dates of request r, and the feasibility function F of a downstream problem Q.
Output : An assignment $d \in \mathbb{N}$ for request r or the message infeasible.

for $d \leftarrow t(r) + 1$ **to** $T(r)$ **do**
  if $F(\sigma_d \cup r) \neq \emptyset$ **then**
    // request r fits to target date d;
    return d;
// there exists no feasible target date where request r fits to;
return infeasible;

Algorithm 2: FIRSTFIT

Bounded number of available bins ($b \in \mathbb{N}$)

For the case where each target date has a bounded number of bins available and all requests have a deadline ($\delta \in \{2, 3, \ldots\}$) we consider the algorithm FIRSTFIT. This algorithm assigns each request to the earliest feasible target date d such that the downstream problem concerning target date d is still feasible. Algorithm 2 specifies the workflow of FIRSTFIT. Note that it is possible that the algorithm is not able to return a feasible assignment for a given request.

Analyzing the bounded case ($b \in \mathbb{N}$) shows that the algorithm FIRSTFIT is the only online algorithm which has a chance to be feasible for the considered problem even if all requests have the same size. Furthermore, this result also holds for ONLINETDAPs with the currently considered downstream problem bin-packing and an arbitrary set $\mathcal{C}$ of cost functions.

Lemma 3.19. *Consider an* ONLINETDAP *with downstream problem bin-packing which has $b \in \mathbb{N}$ bins available. Moreover, suppose that all requests have $\delta \in \{2, 3, \ldots\}$ feasible target dates and the same size $s \in (0, 1]$. The algorithm* FIRSTFIT *is the only deterministic online algorithm which can be feasible for the given problem.*

Proof. Let $(Q, \mathcal{R}_\delta, \mathcal{C})$ be an ONLINETDAP where Q is a bin-packing problem with $b \in \mathbb{N}$ bins available, $\delta \in \{2, 3, \ldots\}$, and all requests from $\mathcal{R}_\delta$ have the same size $s \in (0, 1]$. Let $k = \lfloor 1/s \rfloor$ which defines the maximum number of requests a unit-capacity bin can hold. Moreover, let ALG be an arbitrary deterministic online algorithm for the given problem which does not behave like FIRSTFIT, that is, there exists a request sequence σ where FIRSTFIT and ALG produce a feasible solution and FIRSTFIT$[\sigma] \neq$ ALG$[\sigma]$. Consider a minimal (in terms of number of requests) sequence $\sigma = r_1, r_2, \ldots, r_m$ of $m \in \mathbb{N}$ requests from $\mathcal{R}_\delta$ which satisfies the following conditions:

1. FIRSTFIT and ALG generate a feasible assignment for σ;
2. ALG assigns the first $m - 1$ requests of σ to the same target dates as FIRSTFIT, that is, FIRSTFIT$[r_1, r_2, \ldots, r_{m-1}] =$ ALG$[r_1, r_2, \ldots, r_{m-1}]$;

3. Request r_m is assigned by the algorithm ALG to a target date d_{ALG} which differs from the target date d_{FIRSTFIT} which denotes the target date chosen by FIRSTFIT for request r_m. Therefore, $\mathsf{FIRSTFIT}[r_1, r_2, \ldots, r_m] \neq \mathsf{ALG}[r_1, r_2, \ldots, r_m]$.

Note that such a request sequence exists since $\delta > 1$. From the description of the algorithm FIRSTFIT it follows that $d_{\mathsf{FIRSTFIT}} < d_{\mathsf{ALG}}$. Otherwise, the assignment of ALG for the request sequence σ is infeasible. Moreover, Condition 1 implies that $d_{\mathsf{ALG}} \leq T(r_m)$ since the assignment of ALG is feasible.

Consider now a sequence $\bar{\sigma} = r_1, r_2, \ldots, r_n$ of $n = m + k \cdot b \cdot \delta$ requests from $\mathcal{R}_\delta$ where the first m requests are equal to the requests from σ and the other have a release date of $d = d_{\mathsf{ALG}} - 1$. Since ALG is a deterministic online algorithm, the first m request from $\bar{\sigma}$ are assigned to the same target dates as the requests from σ. Therefore, request r_m of $\bar{\sigma}$ is assigned to date d_{ALG}. Note that through the assignment of ALG the time period $T = \{d_{\mathsf{ALG}}, d_{\mathsf{ALG}} + 1, \ldots, d_{\mathsf{ALG}} + \delta - 1\}$ only can handle $k \cdot b \cdot \delta - 1$ more requests of equal size s. Since this time period represents the feasible target dates for the last $k \cdot b \cdot \delta$ requests from $\bar{\sigma}$, the algorithm ALG is not able to generate a feasible assignment for $\bar{\sigma}$. FIRSTFIT has a capacity of $k \cdot b \cdot \delta$ requests left in the time period T and is able to produce a feasible solution for $\bar{\sigma}$.

Therefore, FIRSTFIT is the only deterministic online algorithm which has a chance to be feasible for the given problem. □

Note that the above lemma does not state that FIRSTFIT is feasible. It only shows that this algorithm is the only one which has a chance to be feasible in the considered setting. Unfortunately, if the number of bins per target date is restricted and the requests do not have the same size, the online algorithm FIRSTFIT is infeasible. Again this result also holds for the case of an ONLINETDAP with the currently considered downstream problem bin-packing and an arbitrary set $\mathcal{C}$ of cost functions.

Lemma 3.20. *Consider an* ONLINETDAP *with downstream problem bin-packing which has $b \in \mathbb{N}$ bins available and each request has $\delta \in \{2, 3, \ldots\}$ feasible target dates. For this problem the online algorithm* FIRSTFIT *is infeasible.*

Proof. Let $(Q, \mathcal{R}_\delta, \mathcal{C})$ be an ONLINETDAP where Q is a bin-packing problem with $b \in \mathbb{N}$ bins available and $\delta \in \{2, 3, \ldots\}$. Moreover, let $\sigma = r_1, r_2, \ldots, r_{2b\delta}$ be a sequence of requests from $\mathcal{R}_\delta$ where each request is released on date 0 and the first $b \cdot \delta$ requests of σ have a size of $2/5$ and the other a size of $3/5$.

A feasible assignment for the request sequence σ is given by $(a_1, a_2, \ldots, a_{2b\delta}) \in \mathbb{N}^{2b\delta}$ with

$$a_i = \begin{cases} \lceil \frac{i}{b} \rceil, & \text{if } 1 \leq i \leq b\,\delta \\ \lceil \frac{i}{b} \rceil - \delta, & \text{if } b\delta + 1 \leq i \leq 2\,b\,\delta. \end{cases}$$

Figure 3.2 illustrates such a feasible assignment for $b = 2$ and $\delta = 5$. FIRSTFIT is not able to

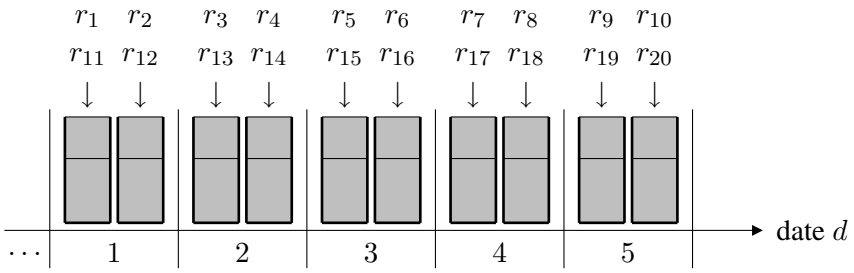

Figure 3.2: A feasible assignment for σ with $b = 2$ and $\delta = 5$.

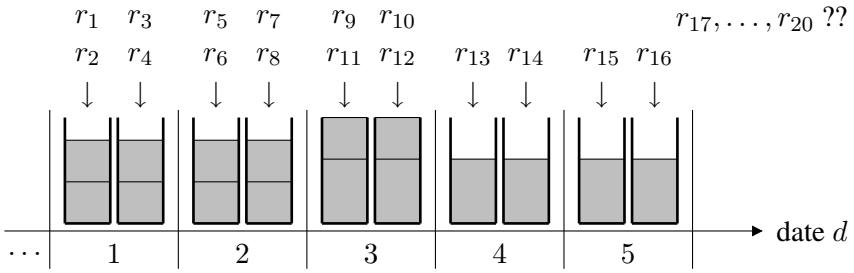

Figure 3.3: FIRSTFIT produces an infeasible assignment for σ with $b = 2$ and $\delta = 5$.

generate a feasible solution for the request sequence σ. This is illustrated in Figure 3.3 for $b = 2$ and $\delta = 5$. Hence, FIRSTFIT is not a feasible online algorithm for the given problem $(Q, \mathcal{R}_\delta, \mathcal{C})$. □

Using the last two lemmas we have the following result for ONLINETDAP w. r. t. bin-packing as downstream problem.

Theorem 3.21. *Consider an* ONLINETDAP *with downstream problem bin-packing which has $b \in \mathbb{N}$ bins available and all requests have $\delta \in \{2, 3, \dots\}$ feasible target dates. For this problem there exists no feasible deterministic online algorithm.*

Proof. The theorem follows directly from Lemma 3.19 and Lemma 3.20. □

Note that the above theorem holds in particular for the MIN-TOTAL ONLINETDAP setting considered in this section.

In the case that all requests have the same size the situation changes a little bit. Lemma 3.19 states that FIRSTFIT is the only online algorithm which has a chance to be feasible. Fortunately, in this setting FIRSTFIT is a feasible online algorithm. This result also holds for ONLINETDAP with the currently considered downstream problem bin-packing and an arbitrary set of cost functions.

Lemma 3.22. *If additionally to the setting of the previous theorem is assumed that all requests have the same size $s \in (0, 1]$, then the algorithm* FIRSTFIT *is the only feasible online algorithm.*

Proof. Consider an ONLINETDAP $(Q, \mathcal{R}_\delta, \mathcal{C})$ where Q is a bin-packing problem with $b \in \mathbb{N}$ bins available, $\delta \in \{2, 3, \dots\}$, and all requests from $\mathcal{R}_\delta$ have the same size $s \in (0, 1]$. Lemma 3.19 gives that FIRSTFIT is the only deterministic online algorithm which has a chance to be feasible for the problem $(Q, \mathcal{R}_\delta, \mathcal{C})$. Suppose that FIRSTFIT is not feasible. Then, there exists a minimal (in terms of number of requests) sequence $\sigma = r_1, r_2, \dots, r_n$ of $n \in \mathbb{N}$ requests from $\mathcal{R}_\delta$ which has a feasible assignment and FIRSTFIT is not able to generate a feasible solution. Since σ is a minimal counter example for the feasibility of FIRSTFIT, it follows that FIRSTFIT assigns the requests $r_1, r_2, \dots, r_{n-1}$ to the target dates $t(r_n) + 1, t(r_n) + 2, \dots, T(r_n)$. Moreover, these requests fill every bin in this time period with the maximum number of requests a bin can hold (if not, σ is not a minimal counter example). Hence, the first $(n-1)$ requests are released at the same release date as request r_n. The time period of feasible target dates of request r_n can only serve $n - 1 = k \cdot b \cdot \delta$ requests where k denotes the maximum number of requests a unit-capacity bin can hold. Therefore, there exists no feasible assignment for σ which is a contradiction to the assumption that σ has a feasible assignment. This shows that FIRSTFIT is a feasible online algorithm for the given problem $(Q, \mathcal{R}_\delta, \mathcal{C})$. □

We aimed to prove that FIRSTFIT has a competitive ratio of $\min\{\lfloor 1/s \rfloor, \delta\}$ in this setting where s denotes the equal item size of all requests. Until now, it was not possible for us to prove this conjecture. However, the following theorem states that FIRSTFIT is $\lfloor 1/s \rfloor$-competitive and shows that $\min\{\lfloor 1/s \rfloor, \delta\}$ is a lower bound on the competitive ratio of this algorithm.

Theorem 3.23. *Consider a* MIN-TOTAL ONLINETDAP *with downstream problem bin-packing which has $b \in \mathbb{N}$ bins available. Moreover, suppose that all requests have $\delta \in \{2, 3, \dots\}$ feasible target dates and the same size $s \in (0, 1]$. For this problem setting the algorithm* FIRSTFIT *is $\lfloor 1/s \rfloor$-competitive and $\min\{\lfloor 1/s \rfloor, \delta\}$ is a lower bound on the competitive ratio of this algorithm.*

Proof. Let $(Q, \mathcal{R}_\delta)$ be a MIN-TOTAL ONLINETDAP where Q is a bin-packing problem with $b \in \mathbb{N}$ bins available, $\delta \in \{2, 3, \dots\}$, and all requests from $\mathcal{R}_\delta$ have the same size $s \in (0, 1]$. Moreover, let $k = \lfloor 1/s \rfloor$ be the maximum number of requests an unit-capacity bin can hold. Lemma 3.22 shows in particular that the algorithm FIRSTFIT is feasible for this problem. Therefore, it is left to prove that FIRSTFIT is k-competitive and $\min\{k, \delta\}$ is a lower bound on the competitive ratio.

Let $\sigma = r_1, r_2, \dots, r_n$ be a sequence of $n \in \mathbb{N}$ requests from $\mathcal{R}_\delta$. Since one bin can not hold more than k requests, we have:

$$\mathsf{OPT}(\sigma) \geq \left\lceil \frac{n}{k} \right\rceil \geq \frac{n}{k} \quad \Leftrightarrow \quad n \leq k \cdot \mathsf{OPT}(\sigma).$$

Moreover, $\mathsf{FIRSTFIT}(\sigma) \leq n$. Therefore,

$$\mathsf{FIRSTFIT}(\sigma) \leq n \leq k \cdot \mathsf{OPT}(\sigma).$$

Hence, FIRSTFIT is k-competitive for the given MIN-TOTAL ONLINETDAP.

It follows the proof that $\min\{k, \delta\}$ is a lower bound on the competitive ratio of FIRSTFIT. Let $m = \min\{k, \delta\}$ and consider the sequence $\sigma = r_1, r_2, \ldots, r_m$ of m requests from $\mathcal{R}_\delta$ where each request has a release date $t(r_i) = i - 1$ for $i = 1, 2, \ldots, m$. An optimal assignment is

$$\mathsf{OPT}[\sigma] = (\delta, \delta, \ldots, \delta) \in \mathbb{N}^m \quad \text{with} \quad \mathsf{OPT}(\sigma) = 1.$$

FIRSTFIT produces the following assignment

$$\mathsf{FIRSTFIT}[\sigma] = (1, 2, \ldots, m) \in \mathbb{N}^m \quad \text{with} \quad \mathsf{FIRSTFIT}(\sigma) = m.$$

Hence,

$$\mathsf{FIRSTFIT}(\sigma) = m \cdot \mathsf{OPT}(\sigma) = \min\{k, \delta\} \cdot \mathsf{OPT}(\sigma).$$

Therefore, $\min\{k, \delta\}$ is a lower bound on the competitive ratio of FIRSTFIT for the given problem $(Q, \mathcal{R}_\delta)$.

This shows the claims of the theorem. □

Remark. Lemma 3.19, Lemma 3.20, Theorem 3.21 and Lemma 3.22 consider a ONLINETDAP with downstream problem bin-packing and an arbitrary set of cost functions. Therefore, these results hold also for MIN-MAX ONLINETDAP w. r. t. bin-packing (see Section 3.4.2).

Infinite number of available bins ($b = \infty$)

In this part the classical bin-packing problem is considered as downstream problem. This means, an infinite number of bins ($b = \infty$) is available on each single target date. Therefore, each input instance for the bin-packing problem is feasible. Hence, any online algorithm which assign each request to a feasible target date is already feasible. However, any deterministic online algorithm has a competitive ratio of at least $3/2$.

Theorem 3.24 ([HKM+05]). *Consider a* MIN-TOTAL ONLINETDAP *with downstream problem bin-packing which has an infinite number of bins ($b = \infty$) available and each request has $\delta \in \{2, 3, \ldots\}$ feasible target dates. For this problem no deterministic online algorithm has a competitive ratio less than* $3/2$.

Proof. Given a MIN-TOTAL ONLINETDAP $(Q, \mathcal{R}_\delta)$ where Q is a bin-packing problem with an infinite number of bins ($b = \infty$) available and $\delta \in \{2, 3, \ldots\}$. Let ALG be an arbitrary feasible deterministic online algorithm for the given problem $(Q, \mathcal{R}_\delta)$ and $0 < \varepsilon < 1/2$. Moreover, consider the following request sequences:

$$\sigma^{(1)} = r_1, r_3 \quad \text{and} \quad \sigma^{(2)} = r_1, r_2 \quad \text{and} \quad \sigma^{(3)} = r_1, r_2, r_3, r_4$$

with

$$r_1 = (0, \tfrac{1}{2} - \varepsilon),\ r_2 = (1, \tfrac{1}{2} - \varepsilon),\ r_3 = (\delta - 1, \tfrac{1}{2} + \varepsilon),\ \text{and}\ r_4 = (\delta, \tfrac{1}{2} + \varepsilon).$$

To prove the claim of the theorem it is shown that there exists an $i \in \{1, 2, 3\}$ such that

$$\mathsf{ALG}(\sigma^{(i)}) \geq \frac{3}{2} \cdot \mathsf{OPT}(\sigma^{(i)}).$$

This means, one of the three request sequences forces the algorithm ALG to a cost of at least $3/2$ time the optimal cost which proves the theorem. The following table presents for each sequence an optimal solution and the optimal cost.

sequence	an optimal assignment	the optimal cost
$\sigma^{(1)}$	(δ, δ)	1
$\sigma^{(2)}$	(δ, δ)	1
$\sigma^{(3)}$	$(\delta, \delta + 1, \delta, \delta + 1)$	2

If the algorithm ALG does not assign request r_1 to its deadline $T(r_1) = \delta$, the request sequence $\sigma^{(1)}$ forces ALG to use one bin for each request. In that case the algorithm ALG needs 2 bins to serve this sequence while an optimal solution only needs 1 bin. Therefore, $\mathsf{ALG}(\sigma^{(1)}) \geq 2 \cdot \mathsf{OPT}(\sigma^{(1)})$. Otherwise, assume that ALG assigns request r_1 to its deadline $T(r) = \delta$. In this case, the request sequences $\sigma^{(2)}$ and $\sigma^{(3)}$ are of interest. If ALG does not assign request r_2 to the same target date δ as request r_1, the request sequence $\sigma^{(2)}$ forces ALG to use 2 bins while an optimal assignment for $\sigma^{(2)}$ needs only 1 bin. Therefore, $\mathsf{ALG}(\sigma^{(2)}) \geq 2 \cdot \mathsf{OPT}(\sigma^{(2)})$. Hence, request sequence $\sigma^{(3)}$ is of interest if ALG assigns the requests r_1 and r_2 to target date δ. In that case the algorithm needs 3 bins to serve the sequence $\sigma^{(3)}$. An optimal solution for $\sigma^{(3)}$ needs 2 bins. Therefore, $\mathsf{ALG}(\sigma^{(3)}) \geq 3/2 \cdot \mathsf{OPT}(\sigma^{(3)})$.

The above case distinction proves that the competitive ratio of ALG is at least $3/2$. □

On the other hand the algorithm PTD has a competitive ratio of 2. This is proved using the general result for this algorithm (Theorem 3.16).

Theorem 3.25 ([HKM$^+$05]). *Consider the same* MIN-TOTAL ONLINETDAP *as in the previous theorem. For this problem the online algorithm* PTD *has a competitive ratio of* 2.

Proof. Let $(Q, \mathcal{R}_\delta)$ be a MIN-TOTAL ONLINETDAP where Q is a bin-packing problem with an infinite number of bins ($b = \infty$) available and $\delta \in \{2, 3, \dots\}$. To prove the claim of the theorem it is to show that:

1. PTD is 2-competitive for the given MIN-TOTAL ONLINETDAP;
2. and 2 is a lower bound on the competitive ratio of PTD for the problem $(Q, \mathcal{R}_\delta)$.

Proof of Claim 1. It is shown that $(Q, \mathcal{R}_\delta)$ meets the conditions of Theorem 3.16 which proves that PTD is 2-competitive for the given problem $(Q, \mathcal{R}_\delta)$.

Since an infinite number of bins is available, it follows that any input instance for the bin-packing problem is feasible. Therefore, the downstream problem Q has unlimited resources.

The monotonicity (Property (i) of Theorem 3.16) of the downcost function from Q holds since any feasible solution of an input instance I gives a feasible solution for any sub-instance $I' \subset I$ by removing the irrelevant items. Hence, $\mathrm{downcost}(I') \leq \mathrm{downcost}(I)$.

In order to prove that the downstream problem Q satisfies the second property of Theorem 3.16, let I be an input instance from Q, $k \in \mathbb{N}$, and $I^{(1)}, I^{(2)}, \ldots, I^{(k)}$ be a disjoint partition of I. Moreover, denote by b_j^* the minimum number of bins needed for the sub-instance $I^{(j)}$ for $j = 1, 2, \ldots, k$. If all items from the sub-instances $I^{(1)}, I^{(2)}, \ldots, I^{(k)}$ are presented at once to the downstream problem Q, no more than $b_1^* + b_2^* + \cdots + b_k^*$ bins are needed to pack these items. Hence,

$$\mathrm{downcost}(I) \leq \sum_{j=1}^{k} \mathrm{downcost}(I^{(j)}).$$

Therefore, it follows from Theorem 3.16 that PTD is a 2-competitive online algorithm for the given problem $(Q, \mathcal{R}_\delta)$.

Proof of Claim 2. It is shown that for all $\varepsilon > 0$ there exists a sequence σ of requests from $\mathcal{R}_\delta$ with

$$\mathsf{PTD}(\sigma) > (2 - \varepsilon)\, \mathsf{OPT}(\sigma).$$

This means, it is possible to construct for any $\varepsilon > 0$ a request sequences σ which forces the online algorithm PTD to a cost which is greater than $(2 - \varepsilon)$ times the optimal cost. In the following we construct request sequences which force the algorithm PTD to use almost on each used target date 2 bins whereas an optimal solution needs the same amount of used target dates but only uses one bin on these dates.

Let $k \in \mathbb{N}$, $k \geq 3$ and $0 < \lambda < 1/(2k-4)$. Moreover, let $\sigma^{(0)}$ be the request sequence consisting of one request $r_1 = (0, 1)$. For $i = 1, 2, \ldots, k$ the request sequence $\sigma^{(i)}$ is defined recursively as $\sigma^{(i)} = \sigma^{(i-1)} \cup (r_{2i}, r_{2i+1})$ with

$$r_{2i} = (i\,\delta - 1, 1/2 + (i-2)\,\lambda) \quad \text{and} \quad r_{2i+1} = (i\,\delta, 1/2 - (i-2)\,\lambda).$$

The assignment of PTD for the request sequence $\sigma^{(k)}$ is:

$$\mathsf{PTD}[\sigma^{(k)}] = (a_1, a_2, \ldots, a_{2k+1}) \in \mathbb{N}^{2k+1}$$

with $a_{2i-1} = a_{2i} = i\,\delta$ for $i = 1, 2, \ldots, k$ and $a_{2k+1} = (k+1)\,\delta$. Therefore, the requests with an odd index are assigned by PTD using the delay tactic and the other requests with the pack together strategy. The assignment of PTD is illustrated in Figure 3.4. The request sequence is constructed such that

$$s(r_{2i-1}) + s(r_{2i}) = \frac{1}{2} - (i - 1 - 2)\,\lambda + \frac{1}{2} + (i-2)\,\lambda = 1 + \lambda > 1$$

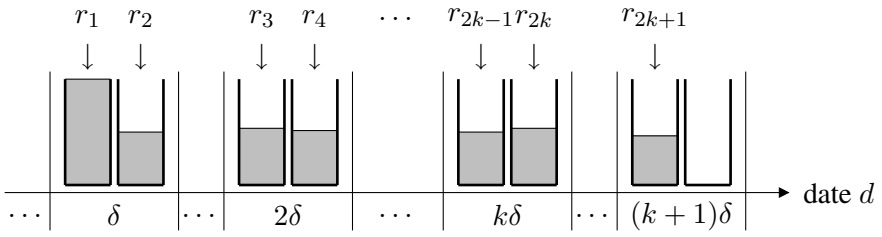

Figure 3.4: Assignment of PTD for the request sequence $\sigma^{(k)}$.

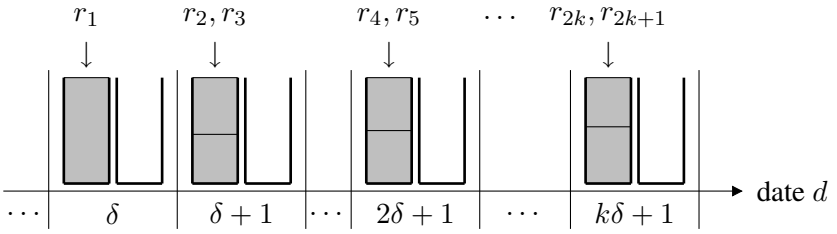

Figure 3.5: An optimal assignment for the request sequence $\sigma^{(k)}$ which is also the assignment of the algorithm PFD.

for $i = 2, 3, \ldots, k$. Hence, PTD needs $2k + 1$ bins to serve the request sequence $\sigma^{(k)}$. Since $\delta \geq 2$, the requests r_{2i} and r_{2i+1} can be assigned to the same target date $d = (i\,\delta + 1)$ and have together a size of 1 for $i = 1, 2, \ldots, k$. Figure 3.5 shows such an assignment which is an optimal solution for $\sigma^{(k)}$. Hence, the optimal cost for serving $\sigma^{(k)}$ is $k + 1$. Since

$$\lim_{k\to\infty} \frac{2k+1}{k+1} = 2,$$

there exists for any $\varepsilon > 0$ a $k \in \mathbb{N}$ with $k \geq 3$ such that

$$\mathsf{PTD}(\sigma^{(k)}) > (2 - \varepsilon)\,\mathsf{OPT}(\sigma^{(k)}).$$

Hence, 2 is a lower bound on the competitive ratio of PTD for the given problem $(Q, \mathcal{R}_\delta)$. □

Another online algorithm named PACKFIRSTORDELAY, or briefly PFD, which is described as Algorithm 3 seems to be even more promising in this setting. PFD assigns a request r to the earliest, used, and feasible target date d such that the assignment of r to this target date does not increase the downstream cost. If such a target date does not exist PFD uses a delay strategy and assigns a request r to its deadline $T(r)$. Note that this algorithm only works in general if all requests have a deadline. Otherwise, the delay strategy is not well defined. Moreover, this algorithm does not always compute a feasible assignment since the delay tactic can produce input instances for the downstream problem which are infeasible. In the case where this algorithm is considered in this work it is ensured that any input instance of the downstream problem is feasible and all requests have a deadline.

Input : A request r, the current assignments $\sigma_{t(r)+1}, \sigma_{t(r)+2}, \ldots, \sigma_{T(r)}$ to the feasible target dates of request r, and the downcost function $\mathrm{downcost}$ of a downstream problem Q.
Output : An assignment $d \in \mathbb{N}$ for request r.

for $d \leftarrow t(r)+1$ **to** $T(r)$ **do**
  if $\mathrm{downcost}(\sigma_d) = \mathrm{downcost}(\sigma_d \cup r)$ **then**
    // pack together tactic since no cost increase;
    return d;
// delay tactic;
return $d \leftarrow T(r)$;

Algorithm 3: PACKFIRSTORDELAY (PFD)

PFD achieves a better solution on the lower bound instance for PTD in the previous proof, in fact PFD generates an optimal solution for this instance, which is illustrated in Figure 3.5. However, it is not clear if the worst case performance of PFD is better than that of PTD since request sequences exist where PTD yields a better solution than PFD. An example for such a sequence is given next.

Example 3.26 (PTD vs. PFD). Given a MIN-TOTAL ONLINETDAP $(Q, \mathcal{R}_\delta)$ where Q is a bin-packing problem with an infinite number of bins ($b = \infty$) available and $\delta \in \{2, 3, \ldots\}$. Moreover, let $\sigma = r_1, r_2, \ldots, r_6$ be a request sequence with:

$$r_1 = (0, 2/5),\ r_2 = (0, 1/5),\ r_3 = (0, 1/5),$$
$$r_4 = (\delta - 1, 2/5),\ r_5 = (\delta - 1, 2/5), \text{and } r_6 = (\delta - 1, 2/5).$$

The algorithm PTD assigns all request to the joint target date δ. Therefore,

$$\mathsf{PTD}[\sigma] = (\delta, \delta, \delta, \delta, \delta, \delta) \quad \text{and} \quad \mathsf{PTD}(\sigma) = 2.$$

On the other hand PFD only assigns the requests r_1, r_2, and r_3 to the target date δ. The other requests are assigned by PFD to the target date $(2\,\delta - 1)$. Hence,

$$\mathsf{PFD}[\sigma] = (\delta, \delta, \delta, 2\,\delta - 1, 2\,\delta - 1, 2\,\delta - 1) \quad \text{and} \quad \mathsf{PFD}(\sigma) = 3.$$

Therefore, the assignment of PFD needs one bin more as the solution of PTD. Both assignments are visualized in Figure 3.6. △

The algorithm PFD has to solve several downstream problems to decide which target date to choose for a request. Since a downstream problem and therefore, an offline optimization problem sometimes is not solvable in a real-time setting, this algorithm is not useful in these cases. In the case of bin-packing as downstream problem where it is assumed that all items have the same size, it is no problem to determine the minimum number of used bins for an input instance. Moreover, if all items have identical size the situation changes in such a way that PFD is a 1-competitive online algorithm for this problem setting.

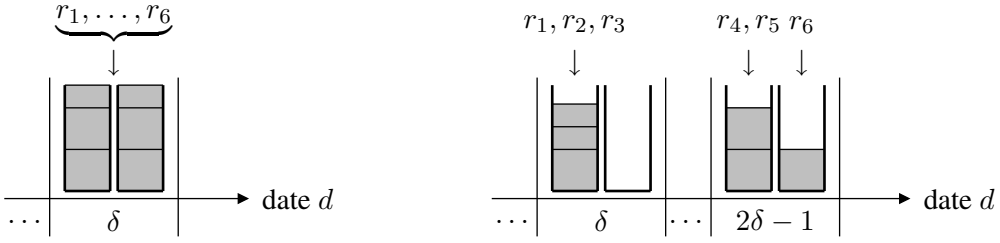

Figure 3.6: Assignment for the request sequence σ of PTD (PFD) is depicted on the left (right) side.

Theorem 3.27 ([HKM+05]). *Consider a* MIN-TOTAL ONLINETDAP *with downstream problem bin-packing which has an infinite number of bins ($b = \infty$) available. Moreover, suppose that all requests have $\delta \in \{2, 3, \dots\}$ feasible target dates and the same size $s \in (0, 1]$. For this problem the algorithm* PFD *is a* 1*-competitive online algorithm.*

Proof. Let $(Q, \mathcal{R}_\delta)$ be a MIN-TOTAL ONLINETDAP where Q is a bin-packing problem with an infinite number of bins ($b = \infty$) available, $\delta \in \{2, 3, \dots\}$, and all requests from $\mathcal{R}_\delta$ have the same size $s \in (0, 1]$. Given a sequence $\sigma = r_1, r_2, \dots, r_n$ of $n \in \mathbb{N}$ requests from $\mathcal{R}_\delta$. For each input instance for the downstream problem bin-packing there exists an optimal packing in such a way that at most one bin is partially filled. Assume that the bin-packing instances are solved in this way. Therefore, let $d_1 < d_2 < \ldots < d_p$ be the target dates on which PFD produced partially filled bins for the given request sequence σ. Moreover, let σ' be the subsequence of σ containing all requests that are packed in a fully filled bin and for each partially filled bin the request that opens this bin. Note that $\mathsf{PFD}(\sigma') = \mathsf{PFD}(\sigma)$.

Now σ' is partitioned into subsequences $\sigma^{(k)}$ with $k = 1, 2, \dots, p+1$ where

- $\sigma^{(1)}$ consists of all requests r of σ' with $t(r) < d_1$;
- $\sigma^{(k)}$ contains all requests r of σ' with $d_{k-1} \leq t(r) < d_k$ for $k = 2, 3, \dots, p$;
- $\sigma^{(p+1)}$ consists of all requests r of σ' with $t(r) \geq d_p$.

Note that the last request in $\sigma^{(k)}$ for $k = 1, 2, \dots, p$ is the one which opens the partially filled bin. Therefore, this request is assigned using the delay tactic of PFD. Moreover, the first request of each sequence $\sigma^{(k)}$ for $k = 1, 2, \dots, p+1$ is also assigned by the delay tactic of PFD. Since it is considered that each request has δ feasible target dates, it follows that there is no overlap in the feasible target dates of requests of different subsequences. Hence,

$$\mathsf{PFD}(\sigma') = \sum_{k=1}^{p+1} \mathsf{PFD}(\sigma^{(k)}) \quad \text{and} \quad \mathsf{OPT}(\sigma') = \sum_{k=1}^{p+1} \mathsf{OPT}(\sigma^{(k)}).$$

Furthermore, at most one bin of a packing of PFD for a subsequence $\sigma^{(k)}$ is partially filled, and thus we have

$$\mathsf{PFD}(\sigma^{(k)}) = \mathsf{OPT}(\sigma^{(k)})$$

for $k = 1, 2, \ldots, p+1$. Combining these equalities, leads to

$$\mathsf{PFD}(\sigma) = \mathsf{PFD}(\sigma') = \sum_{k=1}^{p+1} \mathsf{PFD}(\sigma^{(k)}) = \sum_{k=1}^{p+1} \mathsf{OPT}(\sigma^{(k)}) = \mathsf{OPT}(\sigma') \leq \mathsf{OPT}(\sigma).$$

Therefore, PFD is a 1-competitive online algorithm for the given problem $(Q, \mathcal{R}_\delta)$. □

3.3.3 Downstream Problem Parallel-Machine Scheduling

In this section we analyze MIN-TOTAL ONLINETDAP w. r. t. parallel-machine scheduling as downstream problem and a restricted request set $\mathcal{R}_\delta$ with $\delta \in \{2, 3, \ldots\} \cup \{\infty\}$.

The parallel-machine scheduling problem accepts as input a sequence $L = p_1, p_2, \ldots, p_n$ of $n \in \mathbb{N}$ jobs with processing time $p_i > 0$ for $i = 1, 2, \ldots, n$. The downcost function determines for an input sequence the minimum makespan, that is, the latest completion time of all jobs, to serve the given input sequence on $m \in \mathbb{N} \cup \{\infty\}$ parallel and identical machines in a non-preemptive way. For a more specific description read Section 3.2.3. In this setting, a request r has a processing time $p(r) > 0$ additionally to its release date $t(r) \in \mathbb{N}_0$ and is given by the pair $(t(r), p(r))$. Note that any input instance for the downstream problem parallel-machine scheduling is feasible independently of the number of machines available. Hence, an online algorithm which assigns each request to a feasible target date is already feasible.

Studying MIN-TOTAL ONLINETDAP w. r. t. parallel-machine scheduling and a restricted request set, two cases of interest occur. One case considers a real deadline for each request, that is, each request has only a bounded number $\delta \in \mathbb{N}$ of feasible target dates. The other case assumes that each request has no deadline ($\delta = \infty$), that is, all target dates after the release date of a request are feasible. Table 3.2 summarizes the results proved for MIN-TOTAL ONLINETDAP w. r. t. parallel-machine scheduling in this section.

Before the results for the two cases are presented assume that the downstream problem parallel-machine scheduling only has one machine ($m = 1$) available. In this case each feasible assignment of a request sequence σ is optimal since any feasible assignment yields a cost of $\sum_{r \in \sigma} p(r)$. In particular, FIRSTFIT, PFD, and PTD are feasible online algorithms for this problem setting with competitive ratio of 1.

Theorem 3.28. *Consider a* MIN-TOTAL ONLINETDAP $(Q, \mathcal{R})$ *where Q is a machine scheduling problem on one machine ($m = 1$). For this problem every feasible online algorithm is* 1*-competitive.*

The following cases assume that the downstream problem parallel-machine scheduling provides more than one machine ($m > 1$).

Parameter		arbitrary processing time		equal processing time	
δ	m	lower bound	upper bound	lower bound	upper bound
$\mathbb{N} \cup \{\infty\}$	1	1	1	1	1
∞	∞	no competitive online algorithm		no competitive online algorithm	
∞	$\mathbb{N}_{\geq 2}$	m	m	m	m
$\mathbb{N}_{\geq 2}$	$\mathbb{N}_{\geq 2} \cup \{\infty\}$	$\sqrt{2}$	2	1	1

Table 3.2: Lower bounds on the competitive ratio of deterministic online algorithms as well as upper bounds for the best known deterministic online algorithms for MIN-TOTAL ONLINETDAP w. r. t. parallel-machine scheduling on m parallel and identical machines and a restricted request set $\mathcal{R}_\delta$.

Unbounded number of feasible target dates ($\delta = \infty$)

In the setting where each request has no deadline ($\delta = \infty$) the number of available machines m is a lower and an upper bound on the competitive ratio of any feasible deterministic online algorithm. Hence, in the case where $m = \infty$ there exists no deterministic algorithm which is competitive. Even if all requests have the same processing time, the bounds can not be improved.

Lemma 3.29. *Consider a* MIN-TOTAL ONLINETDAP *with downstream problem parallel-machine scheduling which has $m \in \{2, 3, \dots\}$ parallel and identical machines available and each request has no deadline ($\delta = \infty$). Even if all requests have the same processing time $p > 0$, there exists no deterministic online algorithm which has a competitive ratio less than m.*

Proof. Given a MIN-TOTAL ONLINETDAP $(Q, \mathcal{R}_\infty)$ where Q is a parallel-machine scheduling problem with $m \in \{2, 3, \dots\}$ parallel and identical machines available and all requests from $\mathcal{R}_\infty$ have the same processing time $p > 0$. Let ALG be an arbitrary feasible deterministic online algorithm for this problem $(Q, \mathcal{R}_\infty)$. Since ALG is a deterministic online algorithm, it follows that the answer sequence for any request sequence is known in advance. With that knowledge consider a sequence $\sigma = r_1, r_2, \dots, r_m$ of m requests from $\mathcal{R}_\infty$ with release dates:

$$t(r_i) = \begin{cases} 0, & \text{if } i = 0 \\ \mathsf{ALG}[r_{i-1}], & \text{otherwise.} \end{cases}$$

ALG is not able to assign any two of the requests from σ to the same target date since a new request is released on the date where the previous request is assigned to (This is possible since ALG is a deterministic algorithm). Therefore, $\mathsf{ALG}(\sigma) = m \cdot p$. Assigning all requests of σ to a joint target date after the release date of request r_m leads to a feasible solution since all requests have no deadline. Moreover, such an assignment has a cost of p since m machines are available on each target date. This assignment is also optimal. Hence, $\mathsf{ALG}(\sigma) = m \cdot \mathsf{OPT}(\sigma)$. Therefore, the algorithm ALG has a competitive ratio which is greater than or equal to m for given MIN-TOTAL ONLINETDAP. □

From the above lemma follows in particular that there exists no competitive deterministic online algorithm if the downstream problem parallel-machine scheduling has an infinite number of machines, even if all requests have the same processing time.

Corollary 3.30. *Consider a* MIN-TOTAL ONLINETDAP *with downstream problem parallel-machine scheduling which has an infinite number of parallel and identical machines ($m = \infty$) available and each request has no deadline ($\delta = \infty$). Even if all requests have the same processing time $p > 0$, there exists no competitive deterministic online algorithm for this problem.*

Proof. Let $(Q, \mathcal{R}_\infty)$ be a MIN-TOTAL ONLINETDAP where Q is a parallel-machine scheduling problem with an infinite number of parallel and identical machines ($m = \infty$) available and all requests from $\mathcal{R}_\infty$ have the same processing time $p > 0$. Moreover, let ALG be an arbitrary feasible deterministic online algorithm for the given problem. To prove the claim we use the same construction of a request sequence as in the proof of the previous lemma and show that for all $n \in \mathbb{N}$ there exists a sequence σ of requests from $\mathcal{R}_\infty$ with

$$\mathsf{ALG}(\sigma) = n \cdot \mathsf{OPT}(\sigma).$$

This implies that ALG is not competitive. Let $n \in \mathbb{N}$ and $\sigma = r_1, r_2, \ldots, r_n$ be a sequence of n requests from $\mathcal{R}_\infty$ with release dates:

$$t(r_i) = \begin{cases} 0, & \text{if } i = 0 \\ \mathsf{ALG}[r_{i-1}], & \text{otherwise.} \end{cases}$$

With the same arguments as in the proof of the last lemma it follows that $\mathsf{ALG}(\sigma) = n \cdot p$ and $\mathsf{OPT}(\sigma) = p$. Therefore, the algorithm ALG is not competitive. □

Now an upper bound on the competitive ratio of an arbitrary feasible deterministic online algorithm is shown in the case that the downstream problem parallel-machine scheduling has a bounded number of machines.

Lemma 3.31. *Consider a* MIN-TOTAL ONLINETDAP *with downstream problem parallel-machine scheduling which has $m \in \{2, 3, \ldots\}$ parallel and identical machines available and each request has no deadline ($\delta = \infty$). For this problem any feasible deterministic online algorithm is m-competitive.*

Proof. Let $(Q, \mathcal{R}_\infty)$ be a MIN-TOTAL ONLINETDAP where Q is a parallel-machine scheduling problem with $m \in \{2, 3, \ldots\}$ parallel and identical machines available. Moreover, let ALG be an arbitrary feasible deterministic online algorithm for the given problem and $\sigma = r_1, r_2, \ldots, r_n$ be a sequence of $n \in \mathbb{N}$ requests from $\mathcal{R}_\infty$. The total cost for serving the request sequence σ is bounded from above by the sum of all processing times. Therefore,

$$\mathsf{ALG}(\sigma) \leq \sum_{r \in \sigma} p(r).$$

Assigning all requests of σ to a joint target date after the release date of request r_n yields a feasible solution since all requests have no deadline. Moreover, such an assignment is optimal. Therefore,

$$\mathsf{OPT}(\sigma) \geq \frac{1}{m} \sum_{r \in \sigma} p(r) \Leftrightarrow m \cdot \mathsf{OPT}(\sigma) \geq \sum_{r \in \sigma} p(r).$$

Hence,

$$\mathsf{ALG}(\sigma) \leq \sum_{r \in \sigma} p(r) \leq m \cdot \mathsf{OPT}(\sigma)$$

which proves that the algorithm ALG is m-competitive for given MIN-TOTAL ONLINETDAP. □

With the last two lemmas the main result for this problem setting is already proved.

Theorem 3.32. *Consider the same* MIN-TOTAL ONLINETDAP *as in the previous lemma. For this problem setting any feasible deterministic online algorithm has a competitive ratio of m, even if all requests have the same processing time $p > 0$.*

Proof. The theorem follows from Lemma 3.29 and Lemma 3.31. □

Bounded number of feasible target dates ($\delta \in \mathbb{N}_{\geq 2}$)

It is now assumed that all requests have $\delta \in \{2, 3, \dots\}$ feasible target dates and therefore, a deadline. In this setting it is shown that the online algorithm PTD has a competitive ratio of 2 using the general result for this algorithm (Theorem 3.16). Moreover, any deterministic online algorithm can be forced to a cost of $\sqrt{2}$ times the optimal cost. If all requests have the same processing time, the algorithm PFD is 1-competitive.

Theorem 3.33 ([HKM$^+$05]). *Consider a* MIN-TOTAL ONLINETDAP *with downstream problem parallel-machine scheduling which has more than one parallel and identical machines ($m > 1$) available and each request has $\delta \in \{2, 3, \dots\}$ feasible target dates. For this problem the online algorithm* PTD *has a competitive ratio of* 2.

Proof. Consider a MIN-TOTAL ONLINETDAP $(Q, \mathcal{R}_\delta)$ where Q is a parallel-machine scheduling problem with $m \in \{2, 3, \dots\} \cup \{\infty\}$ parallel and identical machines available and $\delta \in \{2, 3, \dots\}$. The following claims are shown to prove the theorem:

1. PTD is 2-competitive for the given MIN-TOTAL ONLINETDAP;
2. and 2 is a lower bound on the competitive ratio of PTD for the problem $(Q, \mathcal{R}_\delta)$.

Proof of Claim 1. It is shown that $(Q, \mathcal{R}_\delta)$ meets the conditions of Theorem 3.16 which proves that PTD is 2-competitive for the given MIN-TOTAL ONLINETDAP.

The downstream problem parallel-machine scheduling provides unlimited capacity independently of the number of available machines m. This means, for every feasible input instance for the parallel-machine scheduling problem there exists a feasible solution. Therefore, the downstream problem Q has unlimited resources.

The monotonicity (Property (i) of Theorem 3.16) of the downcost function from Q is given since any feasible solution of a feasible input instance I gives a feasible solution for any sub-instance $I' \subset I$ by removing the irrelevant jobs. Hence, $\mathrm{downcost}(I') \leq \mathrm{downcost}(I)$.

In order to prove that the downstream problem Q satisfies the second property of Theorem 3.16, let I be a feasible input instance from Q, $k \in \mathbb{N}$ and $I^{(1)}, I^{(2)}, \ldots, I^{(k)}$ be a disjoint partition of I. Moreover, we denote by m_j^* the minimum makespan for serving the sub-instance $I^{(j)}$ for $j = 1, 2, \ldots, k$. If all jobs from the sub-instances $I^{(1)}, I^{(2)}, \ldots, I^{(k)}$ are presented at once to the downstream problem Q, the minimum makespan is not greater than $m_1^* + m_2^* + \cdots + m_k^*$. This follows since the combination in series of the schedules for the sub-instances $I^{(1)}, I^{(2)}, \ldots, I^{(k)}$ lead to a feasible solution of I. Hence,

$$\mathrm{downcost}(I) \leq \sum_{j=1}^{k} \mathrm{downcost}(I^{(j)}).$$

Therefore, Theorem 3.16 states that PTD is a 2-competitive online algorithm for the given problem.

Proof of Claim 2. Let $p \in (0, 1)$ and $\sigma = r_1, r_2, r_3$ be a sequence of 3 requests from $\mathcal{R}_\delta$ with:

$$r_1 = (0, p),\ r_2 = (1, 1), \text{and } r_3 = (\delta, 1).$$

The algorithm PTD assigns the first two requests of σ to the same target date δ and request r_3 to target date $2\,\delta$. Therefore,

$$\mathsf{PTD}[\sigma] = (\delta, \delta, 2\,\delta) \quad \text{and} \quad \mathsf{PTD}(\sigma) = 2.$$

The optimal cost for the given sequences is not smaller than $1 + p$ since request r_1 and request r_3 have no overlapping in their feasible target dates. Therefore, the following assignment is optimal

$$\mathsf{OPT}[\sigma] = (\delta, \delta + 1, \delta + 1) \quad \text{with} \quad \mathsf{OPT}(\sigma) = 1 + p.$$

Figure 3.7 illustrates the assignment of PTD for σ and shows an optimal solution. Since

$$\lim_{p \to 0} \frac{2}{1 + p} = 2,$$

it follows that for all $\varepsilon > 0$ there exists a sequence σ of requests from $\mathcal{R}_\delta$ with

$$\mathsf{PTD}(\sigma) > (2 - \varepsilon)\,\mathsf{OPT}(\sigma).$$

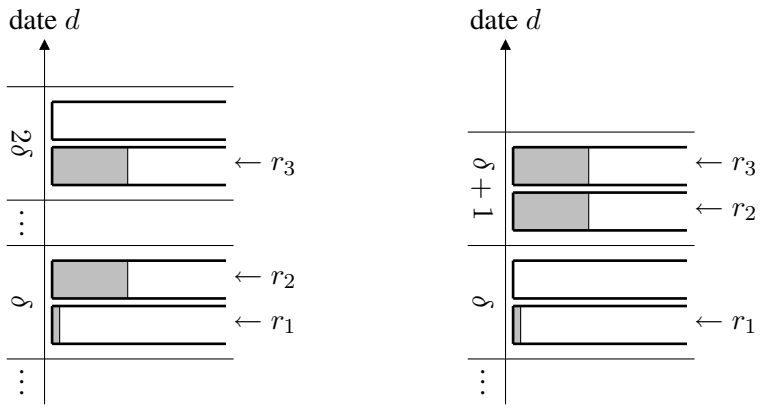

Figure 3.7: Assignment of PTD for the request sequence σ on the left side and an optimal solution for σ on the right side.

Therefore, 2 is a lower bound on the competitive ratio of PTD for the problem $(Q, \mathcal{R}_\delta)$.

Hence, PTD has a competitive ratio of 2 for the given MIN-TOTAL ONLINETDAP. □

Next, we show a lower bound on the competitive ratio of any deterministic online algorithm for the current problem setting.

Theorem 3.34 ([HKM$^+$05]). *Consider the same* MIN-TOTAL ONLINETDAP *as in the previous theorem. For this problem there exists no deterministic online algorithm with competitive ratio less than* $\sqrt{2}$.

Proof. Given a MIN-TOTAL ONLINETDAP $(Q, \mathcal{R}_\delta)$ where Q is a parallel-machine scheduling problem with $m \in \{2, 3, \dots\} \cup \{\infty\}$ parallel and identical machines available and $\delta \in \{2, 3, \dots\}$. Let ALG be an arbitrary feasible deterministic online algorithm for the problem $(Q, \mathcal{R}_\delta)$. Moreover, consider the following two request sequences:

$$\sigma^{(1)} = r_1, r_2 \quad \text{and} \quad \sigma^{(2)} = r_1, r_2, r_3$$

with

$$r_1 = (0, 1),\ r_2 = (\delta - 1, 1 + \sqrt{2}), \text{and } r_3 = (\delta, 1 + \sqrt{2}).$$

Optimal assignments and the optimal cost for these request sequences are presented in the table below.

sequence	an optimal assignment	the optimal cost
$\sigma^{(1)}$	(δ, δ)	$1 + \sqrt{2}$
$\sigma^{(2)}$	$(\delta, \delta + 1, \delta + 1)$	$2 + \sqrt{2}$

To prove the claim of the theorem it is shown:

1. $\mathsf{ALG}(\sigma^{(1)}) < \sqrt{2} \cdot \mathsf{OPT}(\sigma^{(1)}) \Rightarrow \mathsf{ALG}(\sigma^{(2)}) = \sqrt{2} \cdot \mathsf{OPT}(\sigma^{(2)})$;
2. $\mathsf{ALG}(\sigma^{(2)}) < \sqrt{2} \cdot \mathsf{OPT}(\sigma^{(2)}) \Rightarrow \mathsf{ALG}(\sigma^{(1)}) = \sqrt{2} \cdot \mathsf{OPT}(\sigma^{(1)})$.

Case 1: Assume that $\mathsf{ALG}(\sigma^{(1)}) < \sqrt{2} \cdot \mathsf{OPT}(\sigma^{(1)}) = \sqrt{2}+2$. Since the sum of processing times of the two requests of $\sigma^{(1)}$ is $\sqrt{2}+2$, it follows that both requests of $\sigma^{(1)}$ are assigned by ALG to the same target date d. Moreover,

$$d \leq \min\{T(r_1), T(r_2)\} = \delta.$$

The first two requests of sequence $\sigma^{(2)}$ are assigned to the same target date d since ALG is a deterministic algorithm. Request r_3 is released on target date δ. Therefore, it is not possible to assign this request to the same target date as the first two requests of $\sigma^{(2)}$. Hence,

$$\mathsf{ALG}(\sigma^{(2)}) = (1+\sqrt{2}) + (1+\sqrt{2}) = 2 + 2\sqrt{2} = \sqrt{2}\,(2+\sqrt{2}) = \sqrt{2} \cdot \mathsf{OPT}(\sigma^{(2)})$$

which proves claim 1.

Case 2: Consider $\mathsf{ALG}(\sigma^{(2)}) < \sqrt{2} \cdot \mathsf{OPT}(\sigma^{(2)}) = 2 + 2\sqrt{2}$. Therefore, the algorithm ALG assigns the two request r_2 and r_3 of $\sigma^{(2)}$ to the same target date d since these requests together have a processing time of $2 + 2\sqrt{2}$. Moreover,

$$d > \max\{t(r_2), t(r_3)\} = \delta.$$

ALG is a deterministic algorithm. Therefore, this algorithm assigns request r_2 of request sequence $\sigma^{(1)}$ to the same target date d. Since $d > \delta$, $t(r_1) = 0$, and ALG is a feasible algorithm, it follows that request r_1 of $\sigma^{(1)}$ is not assigned to date d. Hence,

$$\mathsf{ALG}(\sigma^{(1)}) = 1 + (1+\sqrt{2}) = 2 + \sqrt{2} = \sqrt{2}\,(1+\sqrt{2}) = \sqrt{2} \cdot \mathsf{OPT}(\sigma^{(1)})$$

which proves claim 2.

Hence, $\sqrt{2}$ is a lower bound on the competitive ratio of the algorithm ALG for given MIN-TOTAL ONLINETDAP. □

The construction of the request sequences in the last proof heavily depends on different processing times. In the case that all requests have the same processing time, the problem collapses into a much easier problem. In this situation the online algorithm PFD is 1-competitive as it is in the corresponding bin-packing case.

Theorem 3.35 ([HKM+05]). *If additionally to the problem setting of the previous theorem is assumed that all requests have the same processing time, then the algorithm* PFD *is* 1*-competitive.*

Proof. Let $(Q, \mathcal{R}_\delta)$ be a MIN-TOTAL ONLINETDAP where Q is a parallel-machine scheduling problem with $m \in \mathbb{N} \cup \{\infty\}$ parallel and identical machines available, $\delta \in \{2, 3, \dots\}$, and all requests from $\mathcal{R}_\delta$ have the same processing time $p > 0$. To prove the theorem a case distinction is made. One case assumes that $m \in \mathbb{N}$, the other one considers $m = \infty$.

Case $m \in \mathbb{N}$: It is shown that the given MIN-TOTAL ONLINETDAP $(Q, \mathcal{R}_\delta)$ is equivalent to a MIN-TOTAL ONLINETDAP $(Q', \mathcal{R}'_\delta)$ where Q' is a bin-packing problem with an infinite number of unit-capacity bins ($b = \infty$) available and all requests from $\mathcal{R}'_\delta$ have the same size $s = 1/m \in (0, 1]$. Therefore, Theorem 3.27 implies that PFD is 1-competitive for this problem $(Q, \mathcal{R}_\delta)$ if $m \in \mathbb{N}$. Let $\sigma = r_1, r_2, \ldots, r_n$ be a sequence of $n \in \mathbb{N}$ requests from $\mathcal{R}_\delta$. This sequence is transformed into a request sequence $\sigma' = r'_1, r'_2, \ldots, r'_n$ of requests from $\mathcal{R}'_\delta$. Request r'_i has the same release date as request r_i for $i = 1, 2, \ldots, n$, that is, $t(r'_i) = t(r_i)$ for each i. Since the parallel-machine scheduling problem Q has m machines available on each target date and all requests from $\mathcal{R}_\delta$ have the same processing time p, the downcost function of Q is equal to

$$\text{downcost}_Q(I) = \left\lceil \frac{|I|}{m} \right\rceil \cdot p \quad \forall\, I \in \mathcal{I}$$

where $|I|$ defines the number of jobs represented by I and $\mathcal{I}$ is the input set of the downstream problem Q. In the case of the bin-packing problem Q' where all items have the same size $s = 1/m$ the downcost function is equal to

$$\text{downcost}_{Q'}(I) = \left\lceil \frac{|I|}{m} \right\rceil \quad \forall\, I \in \mathcal{I}'.$$

Again $|I|$ defines the number of items represented by I and $\mathcal{I}'$ is the input set of the downstream problem Q'. Therefore, in both cases the number of requests assigned to a joint target date define the downstream cost for this target date. Furthermore, both downcost functions are step functions where

$$J = \{n \in \mathbb{N}_0 \mid (n \bmod m) = 1\}$$

is the set of all jumps of both functions. Therefore, the assignment $(d_1, d_2, \ldots, d_n) \in \mathbb{N}^n$ of PFD for σ and the assignment $(d'_1, d'_2, \ldots, d'_n) \in \mathbb{N}^n$ of PFD for σ' are equal since request r_i has the same release date as request r'_i for $i = 1, 2, \ldots, n$. Hence,

$$\text{PFD}_Q(\sigma) = p \cdot \text{PFD}_{Q'}(\sigma').$$

This shows that the given MIN-TOTAL ONLINETDAP $(Q, \mathcal{R}_\delta)$ is equivalent to the MIN-TOTAL ONLINETDAP $(Q', \mathcal{R}'_\delta)$. Since the assignment of PFD for σ' is optimal (Theorem 3.27) it follows that the assignment of PFD for σ is also optimal. Therefore, the algorithm PFD is 1-competitive for the given problem $(Q, \mathcal{R}_\delta)$ if $m \in \mathbb{N}$.

Case $m = \infty$: If an infinite number of machines is available on each target date, the downcost function of the given downstream problem Q can only take two values, that are,

$$\text{downcost}(I) \in \{0, p\} \quad \forall\, I \in \mathcal{I}.$$

This means, if a target date is used than the downstream cost is p independently of the number of requests assigned to this target date. Otherwise, the downstream cost is 0 if the target date is not used. Therefore, it follows that an assignment for a request sequence is optimal if this assignment uses as less as possible target dates. To prove the claim of the theorem for $m = \infty$ it is shown:

1. PFD is equivalent to PTD for the given problem $(Q, \mathcal{R}_\delta)$ if $m = \infty$, that is, $\mathsf{PFD}[\sigma] = \mathsf{PTD}[\sigma]$ for all possible request sequences σ;
2. PTD is 1-competitive for the given MIN-TOTAL ONLINETDAP if $m = \infty$.

Proof of Claim 1. If a target date is used, the downstream cost is p independently of the number of requests assigned to this target date (since unlimited machines are available). Therefore, the delay tactic of the algorithm PFD is only used if for a request no used and feasible target date exists. Hence, for the given problem $(Q, \mathcal{R}_\delta)$ the algorithm PFD produces for any sequence of requests from $\mathcal{R}_\delta$ the same assignment as the algorithm PTD if it is assumed that $m = \infty$. This proves Claim 1.

Proof of Claim 2. Therefore, let $\sigma = r_1, r_2, \ldots, r_n$ be a sequence of $n \in \mathbb{N}$ requests from $\mathcal{R}_\delta$ and $d_1 < d_2 < \cdots < d_h$ be all target dates used by the algorithm PTD for the request sequence σ. Lemma 3.14 states that $d_i - d_{i-1} \geq \delta$ for $i = 2, 3, \ldots, h$. Consider the subsequences $\sigma_{d_1}, \sigma_{d_2}, \ldots, \sigma_{d_h}$ of σ. From the description of the algorithm PTD it follows that the first request of each subsequence is assigned by the algorithm PTD using the delay tactic. Therefore, for these requests there exists no used and feasible target date. Moreover, any two of these requests have no overlapping in the set of feasible target dates. Hence, the assignment of the algorithm PTD uses for the request sequence σ the minimum number of target dates. This shows that the algorithm PTD is 1-competitive for the given problem (Claim 2).

Both cases prove that PFD is a 1-competitive online algorithm for the given MIN-TOTAL ONLINETDAP. □

3.4 Competitive Analysis for MIN-MAX ONLINETDAPs

Section 3.2.2 introduced the class of MIN-MAX ONLINETDAPs. This section provides results for elementary examples of this class obtained by competitive analysis. In particular, results are shown for MIN-MAX ONLINETDAP with downstream problem bin-packing or parallel-machine scheduling where additionally is assumed that a restricted request set is given. That means, all requests have $\delta \in \mathbb{N} \cup \{\infty\}$ feasible target dates. Again some of the results were obtained in joint work with Sven O. Krumke, Nicole Megow, Jörg Rambau, Andreas Tuchscherer and Tjark Vredeveld [HKM+05].

Since we consider only restricted request sets in this section, a request r is given by the pair $(t(r), I(r))$ as it was in the previous section. Moreover, we suppose that each request has more than one feasible target date. Otherwise, the problem setting is trivial (see Section 3.3).

3.4.1 The General Online Algorithm BALANCE

We start by introducing the general online algorithm BALANCE. Since the overall objective is to minimize the maximum downstream cost over all target dates, an effective strategy for an online

Input : A request r, the current assignments $\sigma_{t(r)+1}, \sigma_{t(r)+2}, \ldots, \sigma_{T(r)}$ to the feasible target dates of request r, and an approximation algorithm APPROX for the downstream problem Q.

Output : An assignment $d \in \mathbb{N}$ for request r.

```
d ← t(r) + 1;
min ← APPROX(σ_d ∪ r) − APPROX(σ_d);
for i ← t(r) + 2 to T(r) do
    newmin ← APPROX(σ_i ∪ r) − APPROX(σ_i);
    if newmin < min then
        d ← i;
        min ← newmin;
return d;
```

Algorithm 4: BALANCE (BAL)

algorithm is to balance the load of all requests of a given request sequence over all target dates. Therefore, an algorithm which assigns a given request to the earliest feasible target date such that the increase in the objective value is minimal seems to be promising. Such an algorithm has to solve several downstream problems to decide which target date to choose for a request. Since the considered downstream problem may not be solvable under real-time aspects, this algorithm is not useful in these cases. Therefore, the idea to balance the load of all requests has to go together with an approximation algorithm for the downstream problem. Algorithm 4 specifies the algorithm BALANCE (BAL) which implements the balancing idea with help of an approximation algorithm APPROX for the considered downstream problem.

In [HKM$^+$05] we only considered that the downstream problems are solvable under real time aspects. We show in this section that the results for the algorithm BAL in [HKM$^+$05] also hold if a specific approximation algorithm is used to determine a target date for a given request.

For each downstream problem an approximation algorithm is specified in the corresponding section.

3.4.2 Downstream Problem Bin-Packing

We now analyze MIN-MAX ONLINETDAP w. r. t. bin-packing as downstream problem and a restricted request set $\mathcal{R}_\delta$ with $\delta \in \{2, 3, \ldots\} \cup \{\infty\}$.

The notation and downstream problem definition are the same as in the corresponding MIN-TOTAL ONLINETDAP section (see Section 3.3.2). The downstream problem bin-packing consists of $b \in \mathbb{N} \cup \{\infty\}$ unit-capacity bins and the downcost function determines the minimum number b^* of bins to serve an input instance for the downstream problem if $b^* \leq b$. Otherwise, the downstream

Input : A bin-packing problem and a list $L = s_1, s_2, \ldots, s_n$ of $n \in \mathbb{N}$ items with size $s_i \in (0, 1]$ for $i = 1, 2, \ldots, n$.

Output : The number of used unit-capacity bins for the list L.

1 Reorganize the list L in the way that $s_1 \geq s_2 \geq \cdots \geq s_n$;
2 Go through the reorganized list L and pack the current item s_i in the first bin where s_i fits;
3 Return the number of used bins;

Algorithm 5: FIRSTFITDECREASING (FFD)

problem is infeasible for the given input instance. Moreover, a request r has a size $s(r) \in (0, 1]$ and a release date $t(r) \in \mathbb{N}_0$. Therefore, each request r is a pair $(t(r), s(r))$ where the deadline date is given indirectly by $T(r) = t(r) + \delta$. A more detailed description of bin-packing as offline optimization problem is found in Section 3.2.3.

Before results for MIN-MAX ONLINETDAP w. r. t bin-packing are presented, the approximation algorithm FIRSTFITDECREASING, or briefly FFD, for the downstream problem bin-packing is introduced. FFD packs each item in order of non-increasing size into the first bin in which it fits. Algorithm 5 specifies the workflow of FFD.

In [Bak85] it is proved that FFD always produces a solution which does not need more bins than 4 plus $11/9$ times the number of bins an optimal solution needs. This means, for a given sequence $L = s_1, s_2, \ldots, s_n$ of $n \in \mathbb{N}$ items with size $s_i \in (0, 1]$ for $i = 1, 2, \ldots, n$ it follows

$$\mathsf{FFD}(L) \leq \frac{11}{9}\,\mathsf{OPT}(L) + 4,$$

where $\mathsf{FFD}(L)$ and $\mathsf{OPT}(L)$ denote the used bins for the sequence L by FFD and an optimal solution, respectively. Known proofs for this approximation guarantee require several pages and are not readily intuitive. Much easier to see is that

$$\mathsf{FFD}(L) \leq 2 \cdot \mathsf{OPT}(L).$$

This result is obvious since all bins are half full on average if more than one bin is used.

Analyzing MIN-MAX ONLINETDAP w. r. t. bin-packing two cases of interest occur. One case assumes that a bounded number of bins ($b \in \mathbb{N}$) are available on each target date. The other case supposes that on each target date as many bins ($b = \infty$) as necessary are available. Table 3.3 summarizes the results presented in this section for MIN-MAX ONLINETDAP w. r. t. bin-packing and a restricted request set.

In the case that each request has no deadline date ($\delta = \infty$) the problem turns out to be trivial. An online algorithm which assigns each request to a feasible target date in such a way that in the end a target date has at most one request to serve produces always a feasible solution. This algorithm does not need more than one bin per date. One bin is a lower bound on the optimal cost if the request sequence is not empty. Hence, this algorithm is 1-competitive.

Parameter		arbitrary item size		equal item size	
δ	b	lower bound	upper bound	lower bound	upper bound
∞	$\mathbb{N} \cup \{\infty\}$	1	1	1	1
$\mathbb{N}_{\geq 2}$	$\mathbb{N}$	no feasible online algorithm		$\min\{b, \delta\}$	$\min\{b, \delta\}$
$\mathbb{N}_{\geq 2}$	∞	2	$\min\{4, \delta\}$	$3/2$	2

Table 3.3: Lower Bound on the competitive ratio of deterministic online algorithms as well as upper bounds for the best known deterministic online algorithms for MIN-MAX ONLINETDAP w. r. t. bin-backing and a restricted request set $\mathcal{R}_\delta$. The number of available unit-capacity bins is denoted with b.

Theorem 3.36. *For a* MIN-MAX ONLINETDAP *with downstream problem bin-packing where each request has no deadline ($\delta = \infty$) the algorithm which assigns at most one request to each target date by respecting the feasible target dates of each request is a* 1*-competitive online algorithm.*

Suppose from now on that each request has a deadline ($\delta \in \{2, 3, \dots\}$).

Bounded number of available bins ($b \in \mathbb{N}$)

In the case where each target date has a bounded number of bins available and all requests have $\delta \in \{2, 3, \dots\}$ feasible target dates, Theorem 3.21 already states that there exists no competitive deterministic online algorithm. A competitive algorithm only exists if additionally is assumed that all requests have the same size. In this case FIRSTFIT is the only feasible online algorithm and has a competitive ratio of $\min\{b, \delta\}$.

Theorem 3.37. *Consider a* MIN-MAX ONLINETDAP *with downstream problem bin-packing where all requests have $\delta \in \{2, 3, \dots\}$ feasible target dates and the same size $s \in (0, 1]$. For this setting the algorithm* FIRSTFIT *has a competitive ratio of $\min\{b, \delta\}$ and is the only feasible online algorithm.*

Proof. Given a MIN-MAX ONLINETDAP $(Q, \mathcal{R}_\delta)$ where Q is a bin-packing problem with $b \in \mathbb{N} \cup \{\infty\}$ bins available, $\delta \in \{2, 3, \dots\}$, and all requests from $\mathcal{R}_\delta$ have the same size $s \in (0, 1]$. Lemma 3.22 states that FIRSTFIT is the only feasible online algorithm for this problem setting. Therefore, it is left to prove that

1. FIRSTFIT is $\min\{b, \delta\}$-competitive for the given MIN-MAX ONLINETDAP;
2. and $\min\{b, \delta\}$ is a lower bound on the competitive ratio of FIRSTFIT for the problem $(Q, \mathcal{R}_\delta)$.

Proof of Claim 1. On the one hand the number of available bins b is obviously an upper bound on the competitive ratio of any feasible online algorithm since the objective is to minimize the maximum

downstream cost over all target dates. Therefore, FIRSTFIT is b-competitive. On the other hand δ is also an upper bound on the competitive ratio of FIRSTFIT. In order to see that consider a sequence $\sigma = r_1, r_2, \ldots, r_n$ of $n \in \mathbb{N}$ requests from $\mathcal{R}_\delta$ and the assignments $\sigma_1, \sigma_2, \ldots, \sigma_{T(r_n)}$ of FIRSTFIT for this sequence. Moreover, we denote by d the first target date where the assignment of FIRSTFIT attained the maximum number of used bins, that is,

$$\begin{aligned} &\textsf{FIRSTFIT}(\sigma) = \text{downcost}(\sigma_d) \\ &\text{downcost}(\sigma_i) < \text{downcost}(\sigma_d), \quad \text{if } 1 \leq i < d \\ &\text{downcost}(\sigma_i) \leq \text{downcost}(\sigma_d), \quad \text{if } d < i \leq T(r_n). \end{aligned}$$

Therefore, all requests of σ_d are released on date $(d-1)$ and have a joint set of feasible target dates, that is, $\{d, d+1, \ldots, d+\delta-1\}$. Otherwise, d is not the first target date with the claimed properties. Hence, an optimal solution can divide σ_d on not more than δ target dates. This leads to

$$\textsf{OPT}(\sigma_d) \geq \frac{\text{downcost}(\sigma_d)}{\delta} \Leftrightarrow \text{downcost}(\sigma_d) \leq \delta \cdot \textsf{OPT}(\sigma_d).$$

Since $\textsf{FIRSTFIT}(\sigma) = \text{downcost}(\sigma_d)$ and $\textsf{OPT}(\sigma_d) \leq \textsf{OPT}(\sigma)$, it follows

$$\textsf{FIRSTFIT}(\sigma) \leq \delta \cdot \textsf{OPT}(\sigma).$$

Therefore, δ is an upper bound on the competitive ratio of FIRSTFIT in this setting. Hence, FIRSTFIT is $\min\{b, \delta\}$-competitive for the given MIN-MAX ONLINETDAP.

Proof of Claim 2. Let $m = \min\{b, \delta\}$ and $k = \lfloor 1/s \rfloor$ which determines the maximum number of items a unit-capacity bin can hold. Consider a sequence $\sigma = r_1, r_2, \ldots, r_{k \cdot m}$ of $k \cdot m$ requests from $\mathcal{R}_\delta$ where each request is released on date $d \in \mathbb{N}$. The algorithm FIRSTFIT assigns all requests of σ to the joint target date $(d+1)$. Therefore,

$$\textsf{FIRSTFIT}(\sigma) = m = \min\{b, \delta\}.$$

The optimal cost is $\textsf{OPT}(\sigma) = 1$ which is achieved by assigning no more than k requests to the same target date. The following assignment represents such an optimal solution for σ:

$$\sigma_{d+i} = r_{(i-1)k+1}, r_{(i-1)k+2}, \ldots, r_{(i-1)k+k} \quad i = 1, 2, \ldots, m.$$

Hence, the competitive ratio of FIRSTFIT for the given problem is not smaller than $\min\{b, \delta\}$. □

Unbounded number of bins available ($b = \infty$)

In this part it is assumed that each target date provides as many bins as necessary. Therefore, each input instance for the downstream problem bin-packing is feasible. Theorem 3.37 declares for the current problem setting that FIRSTFIT is a feasible online algorithm with a competitive ratio of δ. Furthermore, the general algorithm BAL is 4-competitive if the algorithm FFD is used to approximate the downstream problem. Moreover, 2 is a lower bound on the competitive ratio of any deterministic online algorithm.

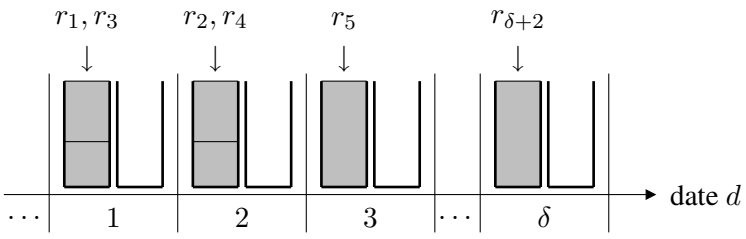

Figure 3.8: An optimal assignment for the request sequence σ.

Theorem 3.38 ([HKM$^+$05]). *Consider the* MIN-MAX ONLINETDAP *with downstream problem bin-packing which has an infinite number of bins ($b = \infty$) available and each request has $\delta \in \{2, 3, \dots\}$ feasible target dates. For this problem there exists no deterministic online algorithm which has competitive ratio less than* 2*.*

Proof. Let $(Q, \mathcal{R}_\delta)$ be a MIN-MAX ONLINETDAP where Q is a bin-packing problem with an infinite number of bins ($b = \infty$) available and $\delta \in \{2, 3, \dots\}$. Moreover, let ALG be an arbitrary feasible deterministic online algorithm for the given problem and $0 < \varepsilon < 1/2$. In order to obtain the lower bound of 2 on the competitive ratio of ALG consider a request sequence σ with the following first two requests: $r_1 = (0, \varepsilon)$ and $r_2 = (0, \varepsilon)$.

If the algorithm ALG assigns the same target date to both requests, then the sequence σ is completed by the requests:

$$r_3 = (0, 1-\varepsilon), \quad r_4 = (0, 1-\varepsilon), \quad r_i = (0, 1) \quad 5 \leq i \leq \delta + 2.$$

An optimal assignment for the resultant sequence is shown in Figure 3.8. Therefore, we have $\mathsf{OPT}(\sigma) = 1$. Note that every used bin of an optimal assignment is completely filled. This implies that $\mathsf{ALG}(\sigma) \geq 2$ since the same target date is assigned to both requests r_1 and r_2, $s(r_1) + s(r_2) < 1$, and for all $i \in \{3, 4, \dots, \delta + 2\}$ it follows that $s(r_1) + s(r_2) + s(r_i) > 1$. Therefore, in this setting the competitive ratio of ALG is not smaller than 2.

Suppose now that the algorithm ALG assigns different target dates to the requests r_1 and r_2, then the following additional requests are given:

$$r_3 = (0, 1-2\,\varepsilon), \quad r_i = (0, 1) \quad 4 \leq i \leq \delta + 2.$$

Figure 3.9 depicts an optimal assignment for this sequence σ. Again $\mathsf{OPT}(\sigma) = 1$ and every used bin of an optimal assignment is completely filled. This indicates that $\mathsf{ALG}(\sigma) \geq 2$ since there exists no request r of σ such that $s(r_1) + s(r) = 1$. Therefore, if the algorithm behaves like assumed, the competitive ratio of ALG for the given problem has a lower bound of 2.

Hence, no deterministic online algorithm for the given MIN-MAX ONLINETDAP has a competitive ratio less than 2. □

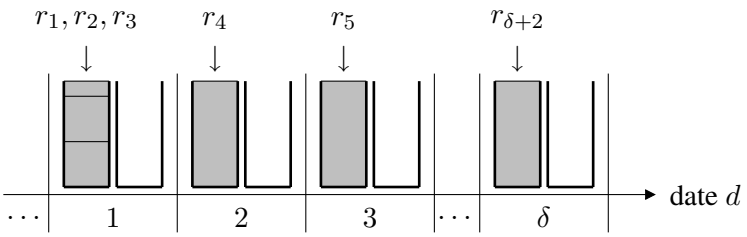

Figure 3.9: An optimal assignment for the request sequence σ.

Next, the algorithm BAL is analyzed for the MIN-MAX ONLINETDAP with downstream problem bin-packing.

Theorem 3.39. *Consider the same* MIN-MAX ONLINETDAP *as in the previous theorem. For this problem setting the algorithm* BAL *is* 4*-competitive if* FFD *is used to approximate the downstream cost.*

Proof. Given a MIN-MAX ONLINETDAP $(Q, \mathcal{R}_\delta)$ where Q is a bin-packing problem with an infinite number of bins ($b = \infty$) available and $\delta \in \{2, 3, \dots\}$. The crucial observation is the following: Given a sequence $\sigma = r_1, r_2, \dots, r_n$ of $n \in \mathbb{N}$ requests from $\mathcal{R}_\delta$ the total size of all requests assigned by BAL within the time interval of feasible target dates of request r_n is bounded from below by half the number of bins required in this interval, whenever more than one bin is used in this period of dates.

This claim can be shown by induction on the number of requests from the given sequence. Obviously, the claim holds when the given sequence contains only one request. Assume that the claim is true for sequences with $(n-1)$ requests and consider a sequence $\sigma = r_1, r_2, \dots, r_n$ of n requests from $\mathcal{R}_\delta$. If after the first $(n-1)$ requests of σ no bin is used in the time window $[t(r_n)+1, T(r_n)]$ the claim obviously also holds after assigning r_n. Suppose that at least one bin is used in the considered time period of dates. If $s(r_n) \geq 1/2$ the total size in the considered time interval increases by at least $1/2$ and the number of used bins by at most one. Hence, in this case the claim also holds after assigning r_n. Assume that $s(r_n) < 1/2$. If BAL can assign r_n to some date such that FFD on this date does not increase the number of used bins, we are also done. However, BAL assigns a date to request r_n such that FFD needs to use a new bin at this date, we know that previously the load of each used bin at the dates $t(r_n)+1, t(r_n)+2, \dots, T(r_n)$ was at least $1 - s(r_n) > 1/2$ which proves the claim.

Now we can prove that BAL is 4-competitive. Consider a sequence $\sigma = r_1, r_2, \dots, r_n$ of $n \in \mathbb{N}$ requests from $\mathcal{R}_\delta$ and the assignments $\sigma_1, \sigma_2, \dots, \sigma_{T(r_n)}$ computed by the algorithm BAL for σ. We denote by d the earliest date where FFD uses the most bins, that is,

$$\begin{aligned} \mathsf{FFD}(\sigma_i) &< \mathsf{FFD}(\sigma_d), \quad \text{if } 1 \leq i < d \\ \mathsf{FFD}(\sigma_i) &\leq \mathsf{FFD}(\sigma_d), \quad \text{if } d < i \leq T(r_n). \end{aligned}$$

Let r_k be the first request in the sequence σ which forces FFD on date d to the maximum number of used bins. From the way the algorithm BAL works together with FFD it follows that $\mathsf{BAL}[r_k] = t(r_k) + 1$. Let $\bar{\sigma}$ be the subsequence of all requests from σ up to r_k that have been assigned by BAL to a target date $d' > t(r_k)$. Note that all requests from the subsequence $\bar{\sigma}$ have a release date which is not smaller than $(t(r_k) - \delta + 1)$. Therefore, an optimal assignment only can balance the load of these requests over the time period $t(r_k) - \delta + 2$ to $T(r_k) = t(r_k) + \delta$. Hence,

$$\mathsf{OPT}(\sigma) \geq \mathsf{OPT}(\bar{\sigma}) \geq \frac{1}{2\,\delta - 1} \sum_{r \in \bar{\sigma}} s(r) > \frac{1}{2\,\delta} \sum_{r \in \bar{\sigma}} s(r) \Leftrightarrow 2\,\delta \cdot \mathsf{OPT}(\sigma) > \sum_{r \in \bar{\sigma}} s(r).$$

We assume that $\mathsf{FFD}(\sigma_d) > 1$. Otherwise, there is nothing to show. Each target date in the time period $t(r_k)+2$ to $T(r_k)$ uses $\mathsf{FFD}(\sigma_d) - 1$ bins if FFD packs the items. These bins in the considered time window are on average half full (follows from the crucial observation). Hence,

$$\delta \cdot (\mathsf{FFD}(\sigma_d) - 1) \leq 2 \sum_{r \in \bar{\sigma}} s(r).$$

Putting this together with the previous argument, it follows

$$\mathsf{BAL}(\sigma) \leq \mathsf{FFD}(\sigma_d) \leq \frac{2}{\delta} \sum_{r \in \bar{\sigma}} s(r) + 1 < 4 \cdot \mathsf{OPT}(\sigma) + 1.$$

The downcost function of the downstream problem bin-packing is an integral function. Hence, $\mathsf{BAL}(\sigma) \leq 4 \cdot \mathsf{OPT}(\sigma)$. □

As in the previous sections the situation improves if we consider that all requests have the same size. In this setting the algorithm BAL with FFD as approximation algorithm for the downstream cost of bin-packing is 2-competitive. Note that in the case where all items of the downstream problem bin-packing have the same size it is clear that the algorithm FFD always produces an optimal packing.

Theorem 3.40. *Assume that a* MIN-MAX ONLINETDAP *with downstream problem bin-packing is given which has an infinite number of bins ($b = \infty$) available. Moreover, suppose that all requests have $\delta \in \{2, 3, \dots\}$ feasible target dates and the same size $s \in (0, 1]$. For this problem the algorithm* BAL *is a 2-competitive online algorithm if* FFD *is used to compute the downstream cost.*

Proof. Let $(Q, \mathcal{R}_\delta)$ be a MIN-MAX ONLINETDAP where Q is a bin-packing problem with an infinite number of bins ($b = \infty$) available, $\delta \in \{2, 3, \dots\}$, and all requests from $\mathcal{R}_\delta$ have the same size $s \in (0, 1]$. Consider a sequence $\sigma = r_1, r_2, \dots, r_n$ of $n \in \mathbb{N}$ requests from $\mathcal{R}_\delta$. Let r_k be the first request of the given sequence σ such that the maximum downstream cost is attained, that is, $\mathsf{BAL}(r_1, r_2, \dots, r_k) = \mathsf{BAL}(\sigma)$. Note that the optimal packings computed by FFD have at most one partially filled bin. With respect to such optimal packings each target date of the time period $t(r_k) + 1$ to $T(r_k)$ has $\mathsf{BAL}(\sigma) - 1$ fully filled bins. Since an optimal solution requires the same number of bins ($\delta\,(\mathsf{BAL}(\sigma) - 1)$) to serve the requests assigned from BAL to this time window and these requests have a release date not earlier than $t(r_k) - \delta + 1$, we have

$$\mathsf{OPT}(\sigma) \geq \frac{1}{2\,\delta - 1}(\delta\,(\mathsf{BAL}(\sigma) - 1)) > \frac{1}{2}(\mathsf{BAL}(\sigma) - 1).$$

Hence, $\mathsf{BAL}(\sigma) < 2 \cdot \mathsf{OPT}(\sigma) + 1$. This gives that BAL is 2-competitive since the cost of bin-packing is integral. □

It follows a lower bound on the competitive ratio of any deterministic online algorithm for the considered MIN-MAX ONLINETDAP with downstream problem bin-packing.

Theorem 3.41 ([HKM+05]). *Consider the same* MIN-MAX ONLINETDAP *as in the previous theorem. For the given problem there exists no deterministic online algorithm which has a competitive ratio less than* $3/2$.

Proof. Given a MIN-MAX ONLINETDAP $(Q, \mathcal{R}_\delta)$ where Q is a bin-packing problem with an infinite number of bins ($b = \infty$) available, $\delta \in \{2, 3, \dots\}$, and all requests from $\mathcal{R}_\delta$ have the same size $s \in (0, 1]$. Let ALG be an arbitrary deterministic online algorithm for this problem and $k = \lfloor 1/s \rfloor$ which denotes the maximum number of items a unit-capacity bin can hold. Consider a sequence σ of requests from $\mathcal{R}_\delta$ starting with $k \cdot \delta$ requests released on date 0. In order for ALG to achieve a competitive ratio better than 2, this algorithm has to assign these requests in such a way that no more than one bin is used on each target date. At that point the target dates 1 to δ have one fully filled bin. Additionally $k\,(\delta + 2)$ requests are released on date 1. These additional requests need at least $\delta + 2$ bins in the time period $[2, \delta + 1]$. In this period are already $\delta - 1$ bins fully filled from the previous requests. Hence,

$$\mathsf{ALG}(\sigma) \geq \left\lceil \frac{2\,\delta + 1}{\delta} \right\rceil = 3.$$

Obviously, the optimal cost for the given sequence is 2. Therefore, no deterministic online algorithm has a competitive ratio less than $3/2$. □

3.4.3 Downstream Problem Parallel-Machine Scheduling

In this section we analyze the MIN-MAX ONLINETDAP w. r. t. parallel-machine scheduling as downstream problem and a restricted request set $\mathcal{R}_\delta$ with $\delta \in \{2, 3, \dots\} \cup \{\infty\}$.

The notation and the downstream problem definition are the same as in the corresponding MIN-TOTAL ONLINETDAP section (see Section 3.3.3). The downstream problem parallel-machine scheduling consists of $m \in \mathbb{N} \cup \{\infty\}$ parallel and identical machines. The downcost function determines the minimum makespan to serve a given input instance. A request r is given by the pair $(t(r), p(r))$ where $p(r) > 0$ represents the processing time and $t(r) \in \mathbb{N}_0$ the release date. The deadline date is given indirectly by $T(r) = t(r) + \delta$. A more detailed description of the downstream problem parallel-machine scheduling is given in Section 3.2.3. Note that any online algorithm which assigns each request of a given sequence to a feasible target date is already a feasible online algorithm since the downstream problem and therefore the target dates have unlimited capacity. Hence, the algorithm BAL is a feasible online algorithm for this problem setting.

Input : m parallel and identical machines and a list $L = p_1, p_2, \ldots, p_n$ of $n \in \mathbb{N}$ jobs with processing time $p_i > 0$ for $i = 1, 2, \ldots, n$.

Output : A makespan to serve the list L of jobs on m parallel and identical machines where each job is processed in a non-preemptive way.

1 Reorganize the list L in the way that $p_1 \geq p_2 \geq \ldots \geq p_n$;
2 Go through the reorganized list L and assign the current jobs p_i to a least loaded machine;
3 Return the makespan of the computed schedule;

Algorithm 6: LISTDECREASING (LD)

Parameter		arbitrary processing time		equal processing time	
δ	m	lower bound	upper bound	lower bound	upper bound
$\mathbb{N} \cup \{\infty\}$	∞	1	1	1	1
∞	$\mathbb{N} \cup \{\infty\}$	1	1	1	1
$\mathbb{N}_{\geq 2}$	$\mathbb{N}$	$3/2$	$3 - 1/\delta$	$3/2$	2

Table 3.4: Lower bounds on the competitive ratio of deterministic online algorithms as well as upper bounds for the best known deterministic online algorithms for MIN-MAX ONLINETDAP w. r. t. parallel-machine scheduling on m parallel and identical machine and a restricted request set $\mathcal{R}_\delta$.

Before the results for MIN-MAX ONLINETDAP w. r. t. parallel-machine scheduling are given, the approximation algorithm LISTDECREASING, or briefly LD, for the downstream problem parallel-machine scheduling is presented. The approximation algorithm LD assigns the jobs of an input sequence in order of non-increasing size to a least loaded machine. Algorithm 6 formalizes the workflow of LD. The following approximation guarantee is shown in [Gra69]. For all input sequences $L = p_1, p_2, \ldots, p_n$ of $n \in \mathbb{N}$ jobs follows:

$$\mathsf{LD}(L) \leq \frac{1}{3}\left(4 - \frac{1}{m}\right)\mathsf{OPT}(L),$$

where m denotes the number of parallel and identical machines and $\mathsf{LD}(L)$ and $\mathsf{OPT}(L)$ represents the makespan of the schedule obtained for the sequence L by LD and an optimal solution, respectively.

Only one case of interest occurs when analyzing MIN-MAX ONLINETDAP w. r. t. parallel-machine scheduling. This case considers that each request has $\delta \in \{2, 3, \ldots\}$ feasible target dates and the downstream problem parallel-machine scheduling has a finite number of machines. All other cases turned out to be trivial. Table 3.4 summarizes the results shown for MIN-MAX ONLINETDAP w. r. t. parallel-machine scheduling as downstream problem.

We start to present two trivial cases. Suppose that the downstream problem parallel-machine scheduling has an infinite number of machines ($m = \infty$) available. In this situation any feasible

solution of a given request sequence σ yields a cost of $\max_{r \in \sigma} p(r)$. Therefore, any feasible online algorithm in this problem setting is 1-competitive.

Theorem 3.42. *For the* MIN-MAX ONLINETDAP *with downstream problem parallel-machine scheduling which has an infinite number of parallel and identical machines ($m = \infty$) available any feasible online algorithm is* 1*-competitive.*

The problem changes also into a trivial problem if all requests have no deadline date ($\delta = \infty$). In that case an algorithm which assigns to each target date at most one request by respecting the release dates of each request computes a feasible assignment. This assignment yields a cost of $\max_{r \in \sigma} p(r)$ which is also a lower bound on the optimal cost. Therefore, this algorithm is feasible and 1-competitive.

Theorem 3.43. *Consider a* MIN-MAX ONLINETDAP *with downstream problem parallel-machine scheduling where each request has no deadline ($\delta = \infty$). The algorithm which assigns at most one request to each target date by respecting the feasible target dates of each request is* 1*-competitive for the given problem.*

The following case assumes that the downstream problem provides a finite number of parallel and identical machines and all requests have $\delta \in \{2, 3, \dots\}$ feasible target dates. In this problem setting no deterministic online algorithm has a competitive ratio less than $3/2$. This lower bound also holds if all request have the same processing time.

Theorem 3.44 ([HKM$^+$05]). *Suppose that a* MIN-MAX ONLINETDAP *with downstream problem parallel-machine scheduling is given which has $m \in \mathbb{N}$ parallel and identical machines available. Moreover, assume that all requests have $\delta \in \{2, 3, \dots\}$ feasible target dates and the same processing time $p > 0$. For this setting there exists no deterministic online algorithm with a competitive ratio less than $3/2$.*

Proof. Given a MIN-MAX ONLINETDAP $(Q, \mathcal{R}_\delta)$ where Q is a parallel-machine scheduling problem with $m \in \mathbb{N}$ parallel and identical machines available, $\delta \in \{2, 3, \dots\}$, and all requests from $\mathcal{R}_\delta$ have the same processing time $p > 0$. Let ALG be an arbitrary deterministic online algorithm for the given problem. Consider a sequence σ of requests from $\mathcal{R}_\delta$ which starts with $m \cdot \delta$ requests released on date 0. In order for ALG to achieve a competitive ratio better than 2, this algorithm has to serve the given requests in such a way that no more than m requests are assigned to the same target date. Otherwise, the competitive ratio of ALG is not smaller than 2. At that point the target dates 1 to δ have m requests to serve. Additionally, $m\,(\delta + 2)$ requests are released on date 1. These additional requests require a total processing time of $m \cdot p\,(\delta + 2)$ in time period $[2, \delta + 1]$. In this period previous requests already require a total processing time of $m \cdot p\,(\delta - 1)$. Hence,

$$\mathsf{ALG}(\sigma) \geq \frac{m \cdot p\,(2\,\delta + 1)}{m\,\delta} > 2\,p.$$

It follows that $\mathsf{ALG}(\sigma) \geq 3p$ since the downstream cost only can take values which are multiples of p. The optimal cost for the given request sequence is $2p$ which is achieved by assigning no more

than $2 \cdot m$ requests to a joint target date. Therefore, the competitive ratio of ALG is not smaller than $3/2$ for the given MIN-MAX ONLINETDAP. □

Remark. The above theorem also could be proved using Theorem 3.41 by transforming the downstream problem parallel-machine scheduling problem into a corresponding bin-packing problem as it is done in proof of Theorem 3.35 on page 48.

The online algorithm BAL is $(3 - 1/\delta)$-competitive for the current problem setting if LD is used as approximation algorithm for the parallel-machine scheduling problem.

Theorem 3.45. *Consider a* MIN-MAX ONLINETDAP *with downstream problem parallel-machine scheduling which has $m \in \mathbb{N}$ parallel and identical machines available and each request has $\delta \in \{2, 3, \dots\}$ feasible target dates. For this problem setting the online algorithm* BAL *is $(3 - 1/\delta)$-competitive if the approximation algorithm* LD *is used to approximate the downstream cost.*

Proof. Let $(Q, \mathcal{R}_\delta)$ be a MIN-MAX ONLINETDAP where Q is a parallel-machine scheduling problem with $m \in \mathbb{N}$ parallel and identical machines available and $\delta \in \{2, 3, \dots\}$. Consider a request sequence $\sigma = r_1, r_2, \dots, r_n$ of $n \in \mathbb{N}$ requests from $\mathcal{R}_\delta$ and the assignments $\sigma_1, \sigma_2, \dots, \sigma_{T(r_n)}$ computed by the algorithm BAL for σ. We denote by d the earliest date where LD has the greatest makespan, that is,

$$\begin{aligned} \mathsf{LD}(\sigma_i) &< \mathsf{LD}(\sigma_d), \quad \text{if } 1 \leq i < d \\ \mathsf{LD}(\sigma_i) &\leq \mathsf{LD}(\sigma_d), \quad \text{if } d < i \leq T(r_n). \end{aligned}$$

Let r be the first request in the given sequence which forces LD on target date d to the maximum makespan. Consider the schedule obtained by BAL before r is released. Moreover, let w denote the load of a least loaded machine over all feasible target dates of r. Then, the makespan of LD on date d for the given sequence σ is at most $w + p(r)$. Since all feasible target dates for r have a load of at least $w \cdot m$, the total load in this time period is at least $w \cdot m \cdot \delta + p(r)$. Any of the corresponding requests in this time window could not be issued earlier than δ dates before the release date of request r. Hence, even an optimal offline algorithm OPT obeying feasibility conditions has at least the following cost for the sequence σ:

$$\mathsf{OPT}(\sigma) \geq \frac{w\,m\,\delta + p(r)}{(2\,\delta - 1)\,m} > \frac{w\,\delta}{2\,\delta - 1}.$$

Hence, we have:

$$w < \left(2 - \frac{1}{\delta}\right) \mathsf{OPT}(\sigma).$$

Since $\mathsf{OPT}(\sigma)$ is bounded from below by $p(r)$ we conclude

$$\mathsf{BAL}(\sigma) \leq \mathsf{LD}(\sigma_d) \leq w + p(r) < \left(2 - \frac{1}{\delta}\right) \mathsf{OPT}(\sigma) + \mathsf{OPT}(\sigma) = \left(3 - \frac{1}{\delta}\right) \mathsf{OPT}(\sigma).$$

Therefore, BAL is $(3 - 1/\delta)$-competitive for the given MIN-MAX ONLINETDAP if LD is used to approximate the downstream problem Q. □

Next, we consider the case where all requests have the same processing time. Note that in this setting it is clear that the algorithm LD produces an optimal schedule for any input instance. Moreover, the upper bound on the competitive ratio of BAL can be reduced to 2.

Theorem 3.46. *Assume additionally to the previous theorem that all requests have the same processing time $p > 0$. In this case the algorithm* BAL *is 2-competitive if the algorithm* LD *is used to compute the downstream cost.*

Proof. Given a MIN-MAX ONLINETDAP $(Q, \mathcal{R}_\delta)$ where Q is a parallel-machine scheduling problem with $m \in \mathbb{N}$ parallel and identical machines available, $\delta \in \{2, 3, \dots\}$, and all requests from $\mathcal{R}_\delta$ have the same processing time $p > 0$. Consider a sequence $\sigma = r_1, r_2, \dots, r_n$ of $n \in \mathbb{N}$ requests from $\mathcal{R}_\delta$. Let r_k be the first request of the given sequence which causes the maximum makespan, that is, $\mathsf{BAL}(r_1, r_2, \dots, r_k) = \mathsf{BAL}(\sigma)$. Consider the schedule obtained by BAL before r_k is released with respect to the offline optimum and let w denote the load of a least loaded machine over all feasible target dates of r. Since all request have the same processing time p it follows that all machines in the time period $t(r_k) + 1$ to $T(r_k)$ have the same load w before request r_k is released. Hence, $\mathsf{BAL}(\sigma) = w + p$. Since all feasible target dates for r_k have load of at least $w \cdot m$, the total load in this time window is $w \cdot m \cdot \delta + p(r)$. Any of the corresponding requests could not be issued earlier than δ dates before the release date of request r_k. Hence, even an optimal offline algorithm OPT obeying feasibility conditions has at least the following cost on sequence σ:

$$\mathsf{OPT}(\sigma) \geq \frac{w\, m\, \delta + p}{(2\,\delta - 1)\, m} > \frac{w}{2}.$$

Hence, we have

$$\mathsf{BAL}(\sigma) = w + p < 2 \cdot \mathsf{OPT}(\sigma) + p.$$

Since the downcost function of the parallel-machine scheduling only can take values in multiples of p, it follows $\mathsf{BAL}(\sigma) \leq 2 \cdot \mathsf{OPT}(\sigma)$. □

Remark. Again, the last result also could be proved with help of Theorem 3.40 by transforming the parallel-machine scheduling problem into a bin-packing problem as we showed in the proof of Theorem 3.35.

4

Approximate the Optimal Value Function of a Discounted Markov Decision Problem Locally

This chapter presents results from a joint work with Volker Kaibel, Matthias Peinhardt, Jörg Rambau, and Andreas Tuchscherer [HKP+05]. In Section 2.2 we introduced standard techniques to compute for a given discounted Markov Decision Problem (MDP) the optimal value function J^* and an optimal policy μ^*. These approaches are often infeasible if the state space is "large". This chapter introduces a method to approximate locally for a given state i_0 the optimal value $J^*(i_0)$ and to find a (near) optimal control for this state.

The outline of this chapter is as follows. In Section 4.1 we present the idea which is used to approximate the optimal value function of a discounted Markov decision problem locally. After that, in Section 4.2 we give the approximation result. Section 4.3 is devoted to further results which follow from the result presented in the previous section. We derive two algorithms which estimate the optimal value $J^*(i_0)$ for any given state i_0 of an MDP in Section 4.4.

4.1 The Idea

Markov decision processes arising in real-life applications usually have a state space whose size is exponential in the common input parameters of the original problem. The complexity of the standard techniques introduced in Section 2.2 depends at least linearly on the state space. A reason for this complexity is that these approaches aim to find the optimal value function and an optimal policy globally for each state. Therefore, it is usually computationally impossible for these methods to compute the optimal value function or an optimal policy.

In real-life applications it is often satisfying to find (near) optimal controls for certain states locally. This observation motivates the approach to approximate for a given state i_0 of an MDP

the optimal value $J^*(i_0)$ and find a (near) optimal control for the considered state. In order to approximate for a given state i_0 the optimal value $J^*(i_0)$, we must provide an upper bound $\overline{J}(i_0)$ and a lower bound $\underline{J}(i_0)$ on $J^*(i_0)$, that is,

$$\underline{J}(i_0) \leq J^*(i_0) \leq \overline{J}(i_0).$$

On the other hand, often policies are known that behave well in simulation experiments, and one would like to have a method to estimate the distance of the expected total discounted cost $J_\mu(i_0)$ of such a policy μ to the optimal cost $J^*(i_0)$ for a given state i_0. Because of the complexity for computing $J^*(i_0)$ and $J_\mu(i_0)$ exactly for a given state i_0, we do not only have to approximate the optimal value. We also have to estimate $J_\mu(i_0)$ to evaluate the distance between these two values. Therefore, we have to provide a lower bound $\underline{J}^*(i_0)$ for $J^*(i_0)$ and an upper bound $\overline{J}_\mu(i_0)$ for $J_\mu(i_0)$ since

$$J_\mu(i_0) - J^*(i_0) \leq \overline{J}_\mu(i_0) - \underline{J}^*(i_0).$$

Furthermore, the relative distance of these values is also reasonable to estimate. This is also possible with the above introduced bounds since

$$\frac{J_\mu(i_0) - J^*(i_0)}{J^*(i_0)} \leq \frac{\overline{J}_\mu(i_0) - \underline{J}^*(i_0)}{\underline{J}^*(i_0)}.$$

In recent years there has been a significant progress in handling large-scale linear programs. We aim to develop an approach based on the standard linear programming technique to compute the optimal value function J^* and an optimal policy μ^* (see Section 2.2). The main idea is to modify the linear program (2.6) (on page 15) in such a way that the resulting linear program is solvable and its optimal solution approximates the optimal value $J^*(i_0)$ for a given state i_0 from above or below, respectively.

4.2 The Approximation Result

This section presents the approximation result of [HKP$^+$05]. Before this result is introduced we analyze the standard linear programming approach and give a necessary definition concerning a Markov decision process.

Given an MDP with Markov decision process (S, C, P, cost) and discount factor $\alpha \in (0, 1)$, we consider for any subset $\hat{S} \subseteq S$ the following linear program:

$$\begin{aligned} &\max \sum_{i \in \hat{S}} J(i) && (4.1)\\ &\text{subject to } J(i) \leq \overline{\text{cost}}(i, u) + \alpha \sum_{j \in \hat{S}} p_{ij}(u) J(j) && \forall i \in \hat{S}, \forall u \in C(i). \end{aligned}$$

Note that in case of $\hat{S} = S$ the above linear program is equal to LP (2.6). Let $J^*(i)$ for all $i \in \hat{S}$ be an optimal solution of the considered LP (4.1). Then, for each state $i \in \hat{S}$ there exists a control $u \in C(i)$ with

$$J^*(i) = \overline{\text{cost}}(i,u) + \alpha \sum_{j \in S} p_{ij}(u) J^*(j).$$

That is, for each state $i \in \hat{S}$ there exists a control $u \in C(i)$ such that the constraint corresponding to state i and control u is tight. Otherwise, there exists a state $i \in \hat{S}$ such that for all $u \in C(i)$ the corresponding constraints are loose, that is,

$$J^*(i) < \text{cost}(i,u) + \alpha \sum_{j \in S} p_{ij}(u) J^*(j),$$

but then the solution J^* cannot be optimal since the variable $J^*(i)$ can be increased. A control u is called *critical* for state i w. r. t. the optimal solution J^* if the corresponding constraint of LP (4.1) is tight. Note that in the case of $\hat{S} = S$ a critical control is optimal.

Definition 4.1 (r-neighborhood)
Given a Markov decision process (S, C, P, cost) and a state $i_0 \in S$. The *r-neighborhood* $S(i_0, r)$ of state i_0 with $r \in \mathbb{N}_0$ is a subset of states that can be reached from i_0 with positive probability within at most r transitions. That is $S(i_0, 0) := \{i_0\}$ and for $r > 0$ we define

$$S(i_0, r) := S(i_0, r-1) \cup \{j \in S \mid \exists\, i \in S(i_0, r-1)\, \exists\, u \in C(i) : p_{ij}(u) > 0\}.$$

△

Next, the approximation result is presented.

Theorem 4.2 ([HKP$^+$05]). *Consider an* MDP *with discount factor $\alpha \in (0,1)$ and Markov decision process (S, C, P, cost). Assume that, the* MDP *exhibits for every state at most a constant number $c \in \mathbb{N}$ of controls, that is, $|C(i)| \leq c$ for all $i \in S$. Furthermore, suppose that for each feasible control there is a positive transition probability for at most a constant number $d \in \mathbb{N}$ of destination states, that is,*

$$\forall\, i \in S\, \forall\, u \in C(i) : |\{j \in S \mid p_{ij}(u) > 0\}| \leq d.$$

Let $M \in \mathbb{R}_{\geq 0}$ be an upper bound on the expected stage cost $\overline{\text{cost}}(i,u)$ for all $i \in S$ and $u \in C(i)$. Then, for all $i_0 \in S$ and $\varepsilon > 0$, there exists a set of states $\hat{S} \subseteq S$ with the following properties:

(i) $|\hat{S}| \leq (cd)^{\left\lceil \log\left(\frac{\varepsilon(1-\alpha)}{M}\right)/\log(\alpha)\right\rceil - 1}$*, that is, the number of states in $\hat{S}$ does not depend on $|S|$;*

(ii) the function $\underline{J} : \hat{S} \to \mathbb{R}_{\geq 0}$ defined as an optimal solution of the linear program

$$\begin{aligned} &\max \sum_{i \in \hat{S}} J(i) && (4.2)\\ &\text{subject to } J(i) \leq \overline{\text{cost}}(i,u) + \alpha \sum_{j \in \hat{S}} p_{ij}(u) J(j) && \forall i \in \hat{S}, \forall u \in C(i) \end{aligned}$$

is an ε-close lower bound to J^* in the given state i_0, that is, $0 \leq J^*(i_0) - \underline{J}(i_0) \leq \varepsilon$;

(iii) the function $\overline{J} : \hat{S} \to \mathbb{R}_{\geq 0}$ defined as an optimal solution of the linear program

$$\max \sum_{i \in \hat{S}} J(i) \tag{4.3}$$
$$\text{subject to } J(i) \leq \overline{\text{cost}}(i,u) + \alpha \sum_{j \in \hat{S}} p_{ij}(u) J(j) + \alpha \sum_{j \in S \setminus \hat{S}} p_{ij}(u) \frac{M}{1-\alpha} \quad \forall i \in \hat{S}, \forall u \in C(i)$$

is an ε-close upper bound to J^* in the given state i_0, that is, $0 \leq \overline{J}(i_0) - J^*(i_0) \leq \varepsilon$.

Proof. Consider an MDP with Markov decision process (S, C, P, cost) and discount factor $\alpha \in (0,1)$ meeting the conditions of the theorem. Moreover let $\varepsilon > 0$, $i_0 \in S$, and

$$r = \left\lceil \log\left(\frac{\varepsilon\,(1-\alpha)}{M}\right) / \log(\alpha) \right\rceil - 1.$$

The claim is that $\hat{S} = S(i_0, r) \subseteq S$ satisfies the properties of the theorem.

Proof of Property (i). Since the number of applicable controls for any state $i \in S$ and the number of succeeding states from state i using any control $u \in C(i)$ are bounded by c and d, respectively, we have

$$|\hat{S}| \leq (cd)^r.$$

Therefore, $\hat{S}$ satisfies Property (i) of the theorem.

Proof of Property (ii). Let J^* be the optimal value function of the given MDP. Therefore, J^* is an optimal solution of LP (2.6). Now consider the following linear program:

$$\max \sum_{i \in S} J(i) \tag{4.4}$$
$$\text{subject to } J(i) \leq \overline{\text{cost}}(i,u) + \alpha \sum_{j \in S} p_{ij}(u) J(j) \qquad \forall i \in S, \forall u \in C(i)$$
$$J(i) = 0 \qquad \forall i \in S \setminus \hat{S}.$$

Let $\underline{J}$ be an optimal solution of LP (4.4). Note that $\underline{J}|_{\hat{S}}$ is an optimal solution of LP (4.2) and $\underline{J}(i) = 0$ for all $i \in S \setminus \hat{S}$. Therefore, LP (4.2) is equivalent to LP (4.4). Moreover, LP (4.4) is derived from LP (2.6) by adding the restriction $J(i) = 0$ for all $i \in S \setminus \hat{S}$. Therefore, $\underline{J}$ is a feasible solution for LP (2.6) and is a lower bound on the optimal objective value of LP (2.6), that is,

$$\sum_{i \in S} \underline{J}(i) = \sum_{i \in \hat{S}} \underline{J}(i) \leq \sum_{i \in S} J^*(i).$$

Moreover, $\underline{J}(i) \leq J^*(i)$ for all $i \in S$ since $\overline{\text{cost}}(i,u) \geq 0$ for all $i \in S$ and $u \in C(i)$. In particular, it follows that $0 \leq J^*(i_0) - \underline{J}(i_0)$ which proves the first inequality of Property (ii).

Since $\underline{J}|_{\hat{S}}$ is an optimal solution of LP (4.2), we know that each state $i \in \hat{S}$ has a critical control. Therefore, let $\underline{u} : \hat{S} \to C$ be a function which maps any state $i \in \hat{S}$ to a critical control $u \in C(i)$ according to LP (4.2) with optimal solution $\underline{J}|_{\hat{S}}$. Hence,

$$\begin{aligned} J^*(i) &\leq \overline{\text{cost}}(i, \underline{u}(i)) + \alpha \sum_{j \in S} p_{ij}(\underline{u}(i)) J^*(j) && \forall i \in \hat{S} \\ \underline{J}(i) &= \overline{\text{cost}}(i, \underline{u}(i)) + \alpha \sum_{j \in S} p_{ij}(\underline{u}(i)) \underline{J}(j) && \forall i \in \hat{S}, \end{aligned}$$

which gives

$$J^*(i) - \underline{J}(i) \leq \alpha \sum_{j \in S} p_{ij}(\underline{u}(i))(J^*(j) - \underline{J}(j)) \qquad \forall i \in \hat{S}. \tag{4.5}$$

By successively applying (4.5) onto the difference $J^*(j) - \underline{J}(j)$ we can show the claimed approximation. Note that this is only possible for $j \in \hat{S} = S(i_0, r)$. This leads to the following approximation:

$$\begin{aligned} J^*(i_0) - \underline{J}(i_0) &\leq \alpha \sum_{i_1 \in S} p_{i_0 i_1}(\underline{u}(i_0))(J^*(i_1) - \underline{J}(i_1)) \\ &= \alpha \sum_{i_1 \in \hat{S}} p_{i_0 i_1}(\underline{u}(i_0))(J^*(i_1) - \underline{J}(i_1)) \\ &\leq \alpha \sum_{i_1 \in \hat{S}} p_{i_0 i_1}(\underline{u}(i_0)) \left(\alpha \sum_{i_2 \in S} p_{i_1 i_2}(\underline{u}(i_1))(J^*(i_2) - \underline{J}(i_2)) \right) \\ &= \alpha^2 \sum_{i_1 \in \hat{S}} p_{i_0 i_1}(\underline{u}(i_0)) \sum_{i_2 \in \hat{S}} p_{i_1 i_2}(\underline{u}(i_1))(J^*(i_2) - \underline{J}(i_2)) \\ &\leq \alpha^{r+1} \sum_{i_1 \in \hat{S}} p_{i_0 i_1}(\underline{u}(i_0)) \cdots \sum_{i_{r+1} \in \hat{S}} p_{i_r i_{r+1}}(\underline{u}(i_r))(J^*(i_{r+1}) - \underline{J}(i_{r+1})). \end{aligned}$$

From Inequality (2.1) on page 12 follows in particular that $J^*(i_{r+1}) \leq M/(1-\alpha)$. Furthermore, $\underline{J}(i_{r+1}) = 0$ since $i_{r+1} \in S \setminus \hat{S}$. Therefore,

$$\begin{aligned} J^*(i_0) - \underline{J}(i_0) &\leq \alpha^{r+1} \underbrace{\sum_{i_1 \in \hat{S}} p_{i_0 i_1}(\underline{u}(i_0))}_{=1} \cdots \underbrace{\sum_{i_{r+1} \in \hat{S}} p_{i_r i_{r+1}}(\underline{u}(i_r))}_{=1} \cdot \frac{M}{(1-\alpha)} \\ &= \alpha^{r+1} \cdot \frac{M}{(1-\alpha)} \\ &= \alpha^{\left\lceil \log\left(\frac{\varepsilon(1-\alpha)}{M}\right) / \log(\alpha) \right\rceil} \cdot \frac{M}{(1-\alpha)} \\ &\leq \alpha^{\log\left(\frac{\varepsilon(1-\alpha)}{M}\right) / \log(\alpha)} \cdot \frac{M}{(1-\alpha)} \\ &= \varepsilon. \end{aligned}$$

Hence, $\hat{S}$ satisfies Property (ii) since $0 \leq J^*(i_0) - \underline{J}(i_0) \leq \varepsilon$.

Property (iii) is proved in the same way as Property (ii). Consider the following linear program:

$$\begin{aligned} \max \sum_{i \in S} J(i) & & (4.6) \\ \text{subject to } J(i) \leq \overline{\text{cost}}(i,u) + \alpha \sum_{j \in S} p_{ij}(u) J(j) & \qquad \forall i \in \hat{S}, \forall u \in C(i) \\ J(i) \leq \frac{M}{1-\alpha} & \qquad \forall i \in S \setminus \hat{S}. \end{aligned}$$

Let $\overline{J}$ be an optimal solution of LP (4.6). Note that $\overline{J}(i) = M/(1-\alpha)$ for all $i \in S \setminus \hat{S}$. Therefore, $\overline{J}|_{\hat{S}}$ is an optimal solution of LP (4.3). Hence, LP (4.6) is equivalent to LP (4.3). Obviously, LP (4.6) is a relaxation of LP (2.6) since LP (2.6) is derived from LP (4.6) with additional restrictions. Therefore, the optimal objective value of LP (4.6) is an upper bound on the optimal objective value of LP (2.6), that is,

$$\sum_{i \in S} \overline{J}(i) \geq \sum_{i \in S} J^*(i).$$

Moreover, $\overline{J}(i) \geq J^*(i)$ for all $i \in S$ since $\overline{\text{cost}}(i,u) \geq 0$ for all $i \in S$ and $u \in C(i)$. In particular, $0 \leq \overline{J}(i_0) - J^*(i_0)$ which proves the first inequality of Property (iii). Let μ^* be an optimal policy for the given MDP. Therefore,

$$\begin{aligned} J^*(i) &= \overline{\text{cost}}(i, \mu^*(i)) + \alpha \sum_{j \in S} p_{ij}(\mu^*(i)) J^*(j) && \forall i \in \hat{S} \\ \overline{J}(i) &\leq \overline{\text{cost}}(i, \mu^*(i)) + \alpha \sum_{j \in S} p_{ij}(\mu^*(i)) \overline{J}(j) && \forall i \in \hat{S}, \end{aligned}$$

which gives

$$\overline{J}(i) - J^*(i) \leq \alpha \sum_{j \in S} p_{ij}(\mu^*(i))(\overline{J}(j) - J^*(j)) \qquad \forall i \in \hat{S}.$$

Following the same arguments as above for proving the second inequality of Property (ii) gives:

$$\overline{J}(i_0) - J^*(i_0) \leq \varepsilon.$$

Hence, $\hat{S}$ satisfies Property (iii) since $0 \leq \overline{J}(i_0) - J^*(i_0) \leq \varepsilon$.

This showed that $\hat{S} = S(i_0, r)$ satisfies the properties of the theorem. □

Remark 4.3. LP (4.2) and LP (4.3) for computing a lower bound and an upper bound, respectively, have unique optimal solutions. To see that, let $J_1^*(i)$ for all $i \in \hat{S}$ and $J_2^*(i)$ for all $i \in \hat{S}$ be two optimal solutions of LP (4.2). Moreover, let

$$J^*(i) = \max\{J_1^*(i), J_2^*(i)\}$$

for all $i \in \hat{S}$. The claim is that J^* is a feasible solution of LP (4.2). This follows since the discount factor α is greater than zero, $p_{ij} \geq 0$ for all $i, j \in \hat{S}$, the cost $\overline{\text{cost}}(i,u) \geq 0$ for all $i \in \hat{S}$ and $u \in C(i)$, and

$$\begin{aligned} J^*(i) &= \max\{J_1^*(i), J_2^*(i)\} \\ &\leq \max\{\overline{\text{cost}}(i,u) + \alpha \sum_{j \in \hat{S}} p_{ij}(u) J_1^*(j), \overline{\text{cost}}(i,u) + \alpha \sum_{j \in \hat{S}} p_{ij}(u) J_2^*(j)\} \\ &\leq \overline{\text{cost}}(i,u) + \alpha \sum_{j \in \hat{S}} p_{ij}(u) \cdot \max\{J_1^*(j), J_2^*(j)\} \\ &= \overline{\text{cost}}(i,u) + \alpha \sum_{j \in \hat{S}} p_{ij}(u) J^*(j) \end{aligned}$$

for all $i \in \hat{S}$ and $u \in C(i)$. Moreover, J^* is obviously also an optimal solution of LP (4.2), that is,

$$\sum_{i \in \hat{S}} J_1^*(i) = \sum_{i \in \hat{S}} J^*(i) = \sum_{i \in \hat{S}} J_2^*(i).$$

Hence, $J_1^* \equiv J^* \equiv J_2^*$ which shows that LP (4.2) has a unique optimal solution. In the same manner it follows that LP (4.3) has a unique optimal solution.

4.3 Further Results

The above theorem gives an approximation result for estimating the optimal value $J^*(i_0)$ for a given state i_0 of an MDP. In Section 4.1 we mentioned that it is also of interest to evaluate known policies. In doing so, we have to approximate both, the optimal value $J^*(i_0)$ and the expected total discounted cost $J_\mu(i_0)$ for a given policy μ. Furthermore, it turned out that it is also desirable to evaluate a single control (not the entire policy) in given state i_0 to estimate the cost increase for using in this state a control $u \in C(i_0)$ instead of an optimal control. This section shows that the approximation result of Theorem 4.2 carries over for these problems.

4.3.1 Evaluating Known Policies

Consider an MDP with Markov decision process (S, C, P, cost) and discount factor $\alpha \in (0,1)$. For a given policy μ, the expected total discounted cost function $J_\mu : S \to \mathbb{R}_{\geq 0}$ can be computed as the optimal value function of a modified MDP. The only modification we have to make from the original MDP is that the feasible controls of each state are restricted to the control chosen by the policy μ, that is, $C'(i) = \{\mu(i)\}$ for all $i \in S$. With that modification it is possible to compute the expected total discounted cost function J_μ as the optimal value function of the modified MDP. Hence, the approximation result of Theorem 4.2 carries over for approximating the expected total discounted cost $J_\mu(i_0)$ for a given policy μ and state $i_0 \in S$. Note that in this case the number of applicable

controls in any state is one. Therefore, the set of necessary states to estimate the expected total discounted cost $J_\mu(i_0)$ is "smaller" than in the case of approximation the optimal value. Moreover, for each state there exists only one side constraint in the corresponding linear programs.

From the above observation follows, that it is possible to estimate the relative and absolute cost increase in state $i_0 \in S$ when using the policy μ instead of an optimal policy, that are,

$$\frac{J_\mu(i_0) - J^*(i_0)}{J^*(i_0)} \quad \text{and} \quad J_\mu(i_0) - J^*(i_0).$$

4.3.2 Evaluating Single Controls

Given an MDP with Markov decision process (S, C, P, cost) and discount factor $\alpha \in (0.1)$. Let $i_0 \in S$ be a state of the MDP for which we want to evaluate a control $u \in C(i_0)$. This means, we want to know the cost increase for using in state i_0 control u instead of an optimal control. Therefore, the remaining decisions are made w. r. t. an optimal policy. This cost can be determine as the optimal value of state i_0 of a modified MDP. We only have to restrict the set of feasible controls for state i_0 to the control u, that means, $C'(i_0) = \{u\}$. The optimal value $J^*(i_0)$ of the modified MDP gives the expected cost of interest. We denote this cost by $J_u(i_0)$.

It is obviously that the approximation result of Theorem 4.2 also holds for this expected cost. Therefore, the relative and absolute cost increase in state i_0 when using the control u instead of an optimal control can be estimated, that are,

$$\frac{J_u(i_0) - J^*(i_0)}{J^*(i_0)} \quad \text{and} \quad J_u(i_0) - J^*(i_0).$$

Note that if a control $u \in C(i_0)$ is optimal for state i_0, then it follows that $J_u(i_0) = J^*(i_0)$.

4.4 Two Algorithms

Section 4.2 states an approximation result for the optimal value $J^*(i_0)$ for a state i_0 of an MDP which satisfies some properties. This section describes two algorithms for approximating $J^*(i_0)$. We consider in this section only MDPs which satisfy the assumption of Theorem 4.2.

4.4.1 The Algorithm ApproxByStaticNeighborhood (ASN)

Consider an MDP with Markov decision process (S, C, P, cost) and discount factor $\alpha \in (0, 1)$. This section introduces the algorithm ApproxByStaticNeighborhood (ASN) which yields for a given $\varepsilon > 0$ an ε-close approximation for a state $i_0 \in S$ on optimal value $J^*(i_0)$. This means, the algorithm returns a lower bound and an upper bound which differ by at most ε.

Input : An MDP with Markov decision process (S, C, P, cost) and discount factor $\alpha \in (0,1)$, a state $i_0 \in S$, and $\varepsilon > 0$.

Output : A lower bound $\underline{J}(i_0)$ and an upper bound $\overline{J}(i_0)$ on the optimal value $J^*(i_0)$ which differ by at most ε, that is, $\overline{J}(i_0) - \underline{J}(i_0) \leq \varepsilon$.

$r \leftarrow \left\lceil \log\left(\frac{\varepsilon(1-\alpha)}{2\,M}\right) / \log(\alpha) \right\rceil - 1$;
initialize lower bound LP (4.2) with $\hat{S} = S(i_0, r)$ and solve it;
$\underline{J} \leftarrow$ optimal solution of lower bound LP (4.2);
initialize upper bound LP (4.3) with $\hat{S} = S(i_0, r)$ and solve it;
$\overline{J} \leftarrow$ optimal solution of upper bound LP (4.3);
return $\underline{J}(i_0), \overline{J}(i_0)$;

Algorithm 7: APPROXBYSTATICNEIGHBORHOOD (ASN)

The proof of Theorem 4.2 showed that for a certain $r \in \mathbb{N}$ the r-neighborhood of i_0 defines a subset of S such that an optimal solution of LP 4.2 (LP 4.3) is an ε-close lower bound (upper bound) on the optimal value function J^* in state i_0. The algorithm ASN is based on this fixed neighborhood.

The algorithm ASN needs as input an MDP, the state i_0 of interest, and an $\varepsilon > 0$. Since ASN wants to return a lower bound and an upper bound which differ by at most ε, the necessary set $S(i_0, r) \subseteq S$ has to be chosen to ensure this approximation guarantee. Therefore, $r \in \mathbb{N}$ has to be adjusted to

$$r = \left\lceil \log\left(\frac{\varepsilon(1-\alpha)}{2\,M}\right) / \log(\alpha) \right\rceil - 1,$$

which differs slightly to the used r in the proof of Theorem 4.2. With the computed subset $S(i_0, r)$ the algorithm initializes LP (4.2) and LP (4.3) and solves them. Let $\underline{J}$ and $\overline{J}$ be an optimal solution of LP (4.2) and LP (4.3), respectively. ASN returns $\underline{J}(i_0)$ as lower bound and $\overline{J}(i_0)$ as upper bound on the optimal value $J^*(i_0)$. Algorithm 7 summarizes this method for approximating the optimal value $J^*(i_0)$.

For the linear programs used by ASN follows from Theorem 4.2 that $J^*(i_0) - \underline{J}(i_0) \leq \varepsilon/2$ and $\overline{J}(i_0) - J^*(i_0) \leq \varepsilon/2$ where $\underline{J}$ is an optimal solution of LP (4.2), $\overline{J}$ is an optimal solution of LP (4.3), and J^* is the optimal value function of the considered MDP. Therefore, $\overline{J}(i_0) - \underline{J}(i_0) \leq \varepsilon$ which proves the correctness of the algorithm ASN.

Experimental results showed that for particular applications the set $S(i_0, r)$ is already pretty large for reasonable ε and α and that the computed bounds $\underline{J}(i_0)$ and $\overline{J}(i_0)$ are already much closer than ε (see Appendix C.1).

4.4.2 The Algorithm APPROXBYDYNAMICNEIGHBORHOOD (ADN)

This section introduces the algorithm APPROXBYDYNAMICNEIGHBORHOOD (ADN) which uses the technique of column generation to approximate the optimal value $J^*(i_0)$ for a given state i_0 of an MDP. For more details about column generation see [LD05, Wil01].

Consider an MDP with Markov decision process (S, C, P, cost) and discount factor $\alpha \in (0, 1)$. Let $i_0 \in S$ be the state whose optimal value $J^*(i_0)$ we want to estimate. The idea of this algorithm is to start with a "small" set $\hat{S} \subseteq S$ which is used to initialize the linear programs to compute a lower bound and an upper bound on $J^*(i_0)$. After that, states are added successively to the set $\hat{S}$ until the rearranged linear programs yield a lower bound and an upper bound which achieve the desired approximation. Therefore, the neighborhood of state i_0 grows dynamically. Since we are only interested in $J^*(i_0)$ and not in $J^*(i)$ for any other state $i \in S$ with $i \neq i_0$, we want to extend the set $\hat{S}$ with states that force the approximation values for $J^*(i_0)$ to converge fast. To reach this goal we use to following linear program to compute a lower bound on the optimal value and to obtain new states:

$$\begin{aligned} &\max J(i_0) && \\ &\text{subject to } J(i) \leq \overline{\text{cost}}(i,u) + \alpha \sum_{j \in \hat{S}} p_{ij}(u) J(j) && \forall i \in \hat{S}, \forall u \in C(i) \\ &J(i) \geq 0 && \forall i \in \hat{S}. \end{aligned} \tag{4.7}$$

The objective value of this linear program is equal to $\underline{J}(i_0)$ where $\underline{J}$ is the optimal solution of the corresponding LP (4.2) used in Theorem 4.2 to determine a lower bound on the optimal value $J^*(i_0)$. This can be proved in the same way as Remark 4.3.

In each step of the column generation algorithm we have to solve a pricing problem which selects a state $i \in S \setminus \hat{S}$ which is added to the set $\hat{S}$. To formulate the pricing problem we first state the dual linear program to LP (4.7). For all $i \in \hat{S}$ and $u \in C(i)$ we define for the corresponding constraint of LP (4.7) the dual variable $\pi(i,u)$. This gives the dual linear program to LP (4.7):

$$\begin{aligned} &\min \sum_{i \in \hat{S}} \sum_{u \in C(i)} \overline{\text{cost}}(i,u)\, \pi(i,u) && \\ &\text{subject to } \sum_{u \in C(i_0)} \pi(i_0,u) \geq \alpha \sum_{i \in \hat{S}} \sum_{u \in C(i)} p_{ii_0}(u)\, \pi(i,u) + 1 && \\ &\sum_{u \in C(j)} \pi(j,u) \geq \alpha \sum_{i \in \hat{S}} \sum_{u \in C(i)} p_{ij}(u)\, \pi(i,u) && \forall j \in \hat{S} \setminus \{i_0\} \\ &\pi(i,u) \geq 0 && \forall i \in \hat{S}, \forall u \in C(i). \end{aligned} \tag{4.8}$$

Input : An MDP with Markov decision process (S, C, P, cost) and discount factor $\alpha \in (0,1)$, a state $i_0 \in S$, and $\varepsilon > 0$.

Output : A lower bound $\underline{J}(i_0)$ and an upper bound $\overline{J}(i_0)$ on the optimal value $J^*(i_0)$ which differ by at most ε, that is, $\overline{J}(i_0) - \underline{J}(i_0) \leq \varepsilon$.

1 $\hat{S} \leftarrow \{i_0\}$;
2 $S_C \leftarrow S(i_0, 1) \setminus \{i_0\}$;
repeat
3 initialize lower bound LP (4.7) with $\hat{S}$ and solve it;
4 $\underline{J} \leftarrow$ optimal solution of lower bound LP (4.7);
5 initialize upper bound LP (4.3) with $\hat{S}$ and solve it;
6 $\overline{J} \leftarrow$ optimal solution of upper bound LP (4.3);
if the reduced costs for all states in S_C are not positive **then**
// in this case $\underline{J}(i_0) = J^*(i_0)$;
7 **return** $\underline{J}(i_0)$;
8 $j \leftarrow$ a state of S_C with maximum reduced cost w. r. t. LP (4.7) and $\underline{J}$;
9 $\hat{S} \leftarrow \hat{S} \cup \{j\}$;
10 $S_C \leftarrow S_C \cup S(j, 1) \setminus \hat{S}$;
until $\overline{J}(i_0) - \underline{J}(i_0) \leq \varepsilon$;
11 **return** $\underline{J}(i_0), \overline{J}(i_0)$;

Algorithm 8: APPROXBYDYNAMICNEIGHBORHOOD (ADN)

From the theory of linear programming follows that if for all $i \in S \setminus \hat{S}$ the reduced costs are not positive, then the objective value of the current LP (4.7) is equal to the optimal value $J^*(i_0)$. Moreover, a states $j \in S \setminus \hat{S}$ is only of interest to be added to the set $\hat{S}$ if this state has positive reduced cost. From the dual LP (4.8) we see that the reduced cost of a state $j \in S \setminus \hat{S}$ is

$$\alpha \sum_{i \in \hat{S}} \sum_{u \in C(i)} p_{ij}(u)\, \pi^*(i,u),$$

where π^* is an optimal solution of the dual linear program. Note that these costs are zero for all states in $S \setminus \hat{S}$ which are not reachable within one transition from a state of $\hat{S}$ since the corresponding transition probabilities are zero. Therefore, we only have to consider states as candidates to be added into the set $\hat{S}$ which are reachable with one transition with positive probability. Therefore, the candidates are

$$S_C = \{j \in S \setminus \hat{S} \mid \exists\, i \in \hat{S}\, \exists\, u \in C(i) : p_{ij}(u) > 0\}.$$

The algorithm ADN has to check if there exists a state in S_C which has positive reduced cost. If this is not the case, then the algorithm can stop and has found the optimal value $J^*(i_0)$. Otherwise, a state j from S_C with maximum reduced cost is added to the set $\hat{S}$. Algorithm 8 specifies this method to approximate the optimal value $J^*(i_0)$. Obviously, this algorithm works correctly.

5

Computational Results for PACKTOGETHERORDELAY and PACKFIRSTORDELAY

In Section 3.3 and Section 3.4 we used competitive analysis to analyze instantiations of ONLINE-TDAPs. In the case of MIN-TOTAL ONLINETDAPs we considered the general online algorithm PACKTOGETHERORDELAY (PTD) which turned out to be competitive for specific problem settings. Furthermore, we proposed the online algorithm PACKFIRSTORDELAY (PFD) which seems to be even more promising than PTD. However, it was not possible to prove any satisfying result for the competitive ratio of PFD. In this chapter we analyze these two algorithms concerning their average case behavior for associated Markov decision process formulations. Therefore, we use the approach introduced in the previous chapter to approximate the expected costs for both algorithms. The computational results were obtained using a self-developed approximation tool.

This chapter is organized as follows. In Section 5.1 we briefly introduce the used approximation tool. In Section 5.2 we describe the problem setting of MIN-TOTAL ONLINETDAPs used to compare the algorithms PTD and PFD. Moreover, we specify the stochastic assumptions on which the average case analysis for these algorithms is based. For the considered problems in Section 5.3 we give a model as Markov decision process since the used approach is based on an MDP. Finally, in Section 5.4 we state and analyze computational results for the two algorithms PTD and PFD applied to the associated Markov decision processes of the considered problem settings.

5.1 Features of the Used Approximation Tool

This section briefly describes the features of the self-developed approximation tool. This tool was developed together with Andreas Tuchscherer and is based on the algorithm ADN (Algo-

rithm 8), introduced in the previous chapter. It accepts as input an MDP with Markov decision process (S, C, P, cost) and discount factor α and has the following three main features:

1. It can be used to approximate for a given state $i_0 \in S$ the optimal value $J^*(i_0)$ for a desired approximation guarantee of $\varepsilon > 0$. Therefore, this tool returns a lower and an upper bound on $J^*(i_0)$ which differ by at most ε;

2. If additionally to a state $i_0 \in S$ a policy μ is given, then this tool can estimate the expected total discounted cost $J_\mu(i_0)$ until a certain accuracy $\varepsilon > 0$ is reached. Therefore, a lower and an upper bound on $J_\mu(i_0)$ is returned which differ by at most ε;

3. It is also possible to estimate for a state $i_0 \in S$ the expected total discounted cost $J_u(i_0)$ for a certain control $u \in C(i_0)$ (for a definition of $J_u(i_0)$ see Section 4.3.2). The approximation tool returns in this case a lower and an upper bound on $J_u(i_0)$ which differ by at most $\varepsilon > 0$.

The approximation tool uses linear programs to compute the lower and upper bounds on the expected costs of interest. In particular, this tool uses the technique of column generation to obtain reasonable states (for more details see Section 4.4.2). After for a state i_0 any of the three expected costs is estimated, the generated state by the technique of column generation can be useful to build the starting linear programs for approximating the expected cost for any successor state of i_0. This is the case since the obtained states in the previous approximation phase are often also of specific interest for approximating the expected cost for any successor state of i_0. The approximation tool exploits this fact in the case where one of the expected costs has to be estimated for a sample path of states. This gives often a significant speed up compared to approximate the expected cost for any state of the sample path from scratch.

All these features are used to analyze the algorithms PTD and PFD applied to the associated Markov decision processes of the MIN-TOTAL ONLINETDAPs introduced in the next section.

5.2 Considered MIN-TOTAL ONLINETDAPs

In this chapter we report computational results for two MIN-TOTAL ONLINETDAPs. The first one is a MIN-TOTAL ONLINETDAP with downstream bin-packing. The second one is a MIN-TOTAL ONLINETDAP w. r. t. parallel machine scheduling as downstream problem. For both problems we only consider a restricted request set, that is, all request have $\delta \in \mathbb{N}$ feasible target dates. This section introduces these two problems and the considered stochastic assumptions which determine the probabilities for future requests. This allows to perform an average case analysis for the algorithms PTD and PFD w. r. t. these stochastic assumptions.

MIN-TOTAL ONLINETDAP w. r. t. bin-packing

Section 3.3.2 showed that the problem setting MIN-TOTAL ONLINETDAP w. r. t. bin-packing and a restricted request set only has one setting which is reasonable for comparing both algorithms. This is the setting where the downstream problem bin-packing provides an infinite number of bins ($b = \infty$) on each target date and all requests have $\delta \in \{2, 3, \dots\}$ feasible target dates (see Table 3.1 on page 30). With that knowledge we consider a MIN-TOTAL ONLINETDAP with downstream bin-packing which has an infinite number of bins ($b = \infty$) with bin-capacity 5 available and a restricted request set

$$\mathcal{R}_\delta = \{(t(r), T(r), s(r)) \mid t(r) \in \mathbb{N}_0 \wedge T(r) = t(r) + \delta \wedge s(r) \in \{1, 2\}\},$$

where $\delta \in \{2, 3, \dots\}$. Note that for a request $r \in \mathcal{R}_\delta$ the release date of r is given by $t(r)$, the deadline date is $T(r)$, and $s(r)$ is the size of this request.

MIN-TOTAL ONLINETDAP w. r. t. parallel-machine scheduling

In the case of MIN-TOTAL ONLINETDAP w. r. t. parallel-machine scheduling and a restricted request set, Table 3.2 (on page 43) states that there is only one problem setting which is suitable for analyzing the average case behavior of the two algorithms. This setting assumes that all requests have $\delta \in \{2, 3, \dots\}$ feasible target dates and the downstream problem parallel-machine scheduling has more than one machine available. Therefore, we consider a MIN-TOTAL ONLINETDAP with downstream parallel-machine scheduling which has 2 parallel and identical machines and a restricted request set

$$\mathcal{R}_\delta = \{(t(r), T(r), p(r)) \mid t(r) \in \mathbb{N}_0 \wedge T(r) = t(r) + \delta \wedge p(r) \in \{1, 2\}\},$$

where $\delta \in \{2, 3, \dots\}$. Note that this request set is equal to the one in the last paragraph. However, a request $r \in \mathcal{R}_\delta$ has a processing time $p(r)$ instead of a size.

Stochastic Assumptions

Since we want to analyze the average case behavior of the algorithms PTD and PFD, we have to make stochastic assumptions. These assumptions determine the possibilities of future requests. Thereby, we have to ensure that the next request has a release date which is not smaller than that of the previously processed one. Otherwise, the generated sequence of requests is not reasonable for these problems. To ensure this we define a probability for the release date of the next request and a probability for the size or processing time of the next request. Since both problems have the same request set $\mathcal{R}_\delta$, we are using the same stochastic assumptions for them.

We assume that on each date at least one request is released but no more than 6. Therefore, we give a probability that a *time shift* arises. This means, if this is the case, the next request has a release

date one greater than the just assigned one. Otherwise, the new request has the same release date as the previously processed one. The time shift probability depends on the number of requests released after the last shift. This probability is given by the function $p_S : \{1, 2, \ldots, 6\} \to [0, 1]$ with

$$p_S(k) = \begin{cases} 0.2, & \text{if } k = 1 \\ \frac{2k-1}{10}, & \text{if } 2 \leq k \leq 5 \\ 1, & \text{if } k = 6. \end{cases}$$

The size or processing time of the next request is generated according to the uniform distribution which is given by the following function

$$p_D : \{1, 2\} \to [0, 1], \quad i \mapsto 0.5.$$

These two probabilities define the stochastic assumptions we consider. For example, suppose that $r \in \mathcal{R}_\delta$ is the k-th ($k \in \{1, 2, \ldots, 6\}$) request after the last time shift, then there are at most four possible requests of $\mathcal{R}_\delta$ which can appear with a positive probability. These requests are

$$r_1 = (t(r), T(r), 1), \quad r_2 = (t(r), T(r), 2),$$

$$r_3 = (t(r) + 1, T(r) + 1, 1), \text{ and } \quad r_4 = (t(r) + 1, T(r) + 1, 2).$$

The probabilities that these requests arise are given by

$$p(r_1) = p(r_2) = 0.5 \cdot (1 - p_S(k)), \text{ and } \quad p(r_3) = p(r_4) = 0.5 \cdot p_S(k).$$

Note that in the case of $k = 6$ request r_1 and request r_2 have probability zero to occur as next request since the probability that a time shift arises in this case is one. Moreover, for all $k \in \{1, 2, \ldots, 6\}$ follows

$$p(r_1) + p(r_2) + p(r_3) + p(r_4) = 1.$$

These stochastic assumptions ensure that reasonable request sequences are generated.

5.3 Markov Decision Process Model

This section describes briefly how the considered MIN-TOTAL ONLINETDAPs with the given stochastic assumptions are modeled as a Markov decision process. For both downstream problems the resulting Markov decision processes are similar. They only differ in the cost functions.

The parameters of interest for the following formulation are the given restricted request set $\mathcal{R}_\delta$ and the stochastic assumptions. The main part of the formulation as a Markov decision process (S, C, P, cost) is to define a suitable state space S. By modeling this space we have to be aware of the information a state has to provide for specifying the transition probability matrix P and the cost function cost. A state of the state space S includes the following information:

1. the number of released requests $k \in \{1, 2, \ldots, 6\}$ after the last time shift;
2. a demand $d \in \{1, 2\}$ which is either a size or a processing time depending on the downstream problem;
3. and δ slot assignments each of which specifies the assignment of some target date and is given by a pair of integers (d_1, d_2) where d_1 and d_2 specify the number of assigned requests with demand 1 and 2, respectively.

We now define the other objects of the resulting Markov decision process which shows why a state has to contain these information. The control set is $C = \{1, 2, \ldots, \delta\}$ where a control $u \in C$ determines the slot to use for the current request demand. The transition probability matrix P has to reflect the considered stochastic assumptions. These assumptions depend on the number of requests released on the same date which explains why a state has to include the number of released requests k after the last time shift. The cost structure of the resulting Markov decision process has to reflect the overall objective of a MIN-TOTAL ONLINETDAP. If the cost function cost returns the cost increase w. r. t. the downstream problem after a certain request demand is assigned, then the sum of all cost increases equals the overall objective of a MIN-TOTAL ONLINETDAP. Since each request has δ feasible target dates and therefore, δ feasible controls, δ slot assignments are necessary to compute the cost increase after an arbitrary control $u \in C$ is used. This observation explains why a state has to contain δ slot assignments besides the request demand d.

Note that after a time shift occurs the first slot assignment is not relevant anymore for future assignments. Furthermore, a new target date has to be taken into account where no request has yet been assigned to. This date is represented by a new empty slot assignment. Moreover, the release date and the deadline date of a request does not appear as a state information. These dates are indirectly given through the δ slot assignments. It follows a small example, which illustrates the described construction of the Markov decision process.

Example 5.1. Consider a MIN-TOTAL ONLINETDAP with downstream parallel-machine scheduling which has 2 parallel and identical machines available and a restricted request set

$$\mathcal{R}_2 = \{(t(r), T(r), p(r)) \mid t(r) \in \mathbb{N}_0 \wedge T(r) = T(r) + 2 \wedge p(r) = 1\}.$$

Furthermore, we assume that at least one and at most two requests are released at the same target date. Hence, we have $k \in \{1, 2\}$, $d = 1$, and $\delta = 2$. Table 5.1 presents all states of the resulting state space S and also for each state $i \in S$ the possible successor states.

Let $\sigma = r_1, r_2, \ldots, r_6$ be a randomly generated (w. r. t. the considered stochastic assumptions) sequence of requests from $\mathcal{R}_2$ with

$$r_1 = (1, 3, 1) = r_2 = (1, 3, 1), \quad r_3 = (2, 4, 1), \quad r_4 = r_5 = (3, 5, 1), \text{ and } \quad r_6 = (4, 6, 1).$$

The assignment and incurred cost of the algorithm PTD for σ are

$$\mathsf{PTD}[\sigma] = (3, 3, 3, 5, 5, 5) \quad \text{and} \quad \mathsf{PTD}(\sigma) = 4.$$

State	Request r		Assignment		successor states for	
Index	k	$p(r)$	Slot 1	Slot 2	Control 1	Control 2
1	1	1	0	0	$1, 5$	$2, 4$
2	1	1	1	0	$1, 7$	$2, 6$
3	1	1	2	0	$1, 9$	$2, 8$
4	2	1	0	1	2	3
5	2	1	1	0	1	2
6	2	1	1	1	2	3
7	2	1	2	0	1	2
8	2	1	2	1	2	3
9	2	1	3	0	1	2

Table 5.1: All states of S if $\delta = 2$, each request has a processing time of 1, and on each target date at least one and at most two requests are released. Note that k denotes the number of requests already released on the current date and a slot assignment is represented only by one integer since there is only one possible processing time.

The associated Markov decision process starts in the state with state index 1 since after the release of request r_1 the feasible target dates for this request are empty and this request is the first released on this date. The algorithm PTD selects for request r_1 the deadline date $T(r_1) = 3$ which is control 2 in the corresponding Markov decision processes. The next request is also released at date 1. Therefore, the system evolves after applying control 2 in the starting state to the state with state index 4. Furthermore, this leads to a cost increase of one. PTD selects in this situation again control 2. The process ends after applying this control in the state with state index 3 since a time shift occurs. For this control the cost function equals zero. Processing the whole sequences σ leads to following sample path of visited states:

$$1 \xrightarrow{2;1} 4 \xrightarrow{2;0} 3 \xrightarrow{1;1} 1 \xrightarrow{2;1} 4 \xrightarrow{2;0} 3 \xrightarrow{1;1} i.$$

The first number over each arrow defines the chosen control and the second number determines the amount of cost increase. Note that after processing the last request it is undetermined in which state i the associated Markov decision process ends since the outcome of the stochastic process is unknown. Furthermore, the sum of all cost increases is 4 which matches the cost of the algorithm PTD in the corresponding MIN-TOTAL ONLINETDAP. △

Before we present the computational results we illustrate how many states a Markov decision process has which results from the above formulation. The number of states depends on the number of feasible target dates δ, the maximum number of request which can be released on the same target date k (in our case 6), and the number of different demands. Table 5.2 shows the number of states of a Markov decision process resulting from the above formulation for $\delta \in \{1, 2, 3, 4, 5, 6\}$,

$k \setminus \delta$	1	2	3	4	5	6
6	42	$4 \cdot 10^3$	$230 \cdot 10^3$	$16 \cdot 10^6$	$1 \cdot 10^9$	$100 \cdot 10^9$
7	56	$8 \cdot 10^3$	$590 \cdot 10^3$	$55 \cdot 10^6$	$5 \cdot 10^9$	$577 \cdot 10^9$
8	72	$14 \cdot 10^3$	$1,353 \cdot 10^3$	$160 \cdot 10^6$	$20 \cdot 10^9$	$2,700 \cdot 10^9$
9	90	$25 \cdot 10^3$	$2,841 \cdot 10^3$	$417 \cdot 10^6$	$65 \cdot 10^9$	$10,712 \cdot 10^9$
10	110	$39 \cdot 10^3$	$5,552 \cdot 10^3$	$988 \cdot 10^6$	$187 \cdot 10^9$	$37,207 \cdot 10^9$

Table 5.2: Number of states of a Markov decision process which results from the formulation described in Section 5.3. This number depends on the number of feasible target dates δ, the maximal number of request released on the same date k, and number of different demands which equals 2.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$t(r_i)$	1	1	2	2	2	3	3	3	3	4	4	5	5	5	6	6	6	6	7	7
$d(r_i)$	1	1	1	1	2	1	1	2	1	1	1	1	2	1	2	2	1	1	1	2

Table 5.3: The considered request sequence $\sigma = r_1, r_2, \ldots, r_{20}$ for the computational results. The deadline date for each request r is $T(r) = t(r) + 3$.

$k \in \{6, 7, 8, 9, 10\}$, and 2 different request demands which are either a size or a processing time of requests depending on the downstream problem.

This shows that already small examples lead to Markov decision processes which have a huge number of states.

5.4 Results

Finally, we present the computational results for the algorithms PTD and PFD applied to the associated Markov decision processes of the problems described in Section 5.2 where $\delta = 3$ is chosen. From now on we use the term policy instead of the term algorithm.

We made for both policies the same computations to obtain results which are comparable. Therefore, we generated w. r. t. the considered stochastic assumptions (see Section 5.2) a sequence $\sigma = r_1, r_2, \ldots r_{20}$ of requests from $\mathcal{R}_\delta$. This sequence is stated in Table 5.3 and is considered for all computations.

The request sequence σ, the considered policy $\mu \in \{\mathsf{PTD}, \mathsf{PFD}\}$, and the considered downstream problem define a sample path of 20 states. For each state i_0 of this sample path we are interested in the relative and absolute errors

$$\frac{J_\mu(i_0) - J^*(i_0)}{J^*(i_0)} \quad \text{and} \quad J_\mu(i_0) - J^*(i_0)$$

and

$$\frac{J_u(i_0) - J^*(i_0)}{J^*(i_0)} \quad \text{and} \quad J_u(i_0) - J^*(i_0)$$

where $u = \mu(i_0)$ (for the definition of $J_u(i_0)$ see Section 4.3.2). Since in general it is computational impossible to compute the expected total discounted costs $J^*(i_0)$, $J_\mu(i_0)$, and $J_u(i_0)$, we use the approach introduced in Chapter 4 to estimate these costs choosing a discount factor $\alpha \in \{0.5, 0.7\}$. We compute a lower and an upper bound on each of these values. A lower bound is denoted with a line under the corresponding symbol and an upper bound with a line over the corresponding symbol. With these bounds the relative and absolute errors of interest can be approximated. Furthermore, sometimes it is possible to tell if a policy μ is not optimal. This is the case if a state i_0 exists with

$$\underline{J}_\mu(i_0) > \overline{J}^*(i_0). \tag{5.1}$$

Note that this does not mean that the control choice of μ in state i_0 is not optimal. To tell this it has to be shown that

$$\underline{J}_u(i_0) > \overline{J}^*(i_0) \tag{5.2}$$

where $u = \mu(i_0)$.

We are further interested to know if the control choice of a policy is optimal for the explored states. Since in general it is computational impossible to compute the optimal value function J^*, Bellman's equation (2.3) does not help to decide whether a certain control is optimal or not. However, if for a state i_0 and a control $u \in C(i_0)$ follows that

$$\overline{J}_u(i_0) \leq \underline{J}_{u'}(i_0) \tag{5.3}$$

for all $u' \in C(i_0) \setminus \{u\}$, then the control u is optimal for state i_0. Therefore, we also approximate for any state i_0 of the considered sample path the expected total discounted cost $J_u(i_0)$ for all $u \in C(i_0)$.

We estimate all values of interest until the lower bound and the upper bound differ by at most 1%, this is, until the following inequality is satisfied:

$$\text{Lower Bound} + 0.01 \cdot \text{Lower Bound} \geq \text{Upper Bound}.$$

Before the computational results are finally presented for each downstream problem separately, we want to give a general explanation for the following figures. Each figure includes two pictures: one showing the results for the discount factor $\alpha = 0.5$, and the other for the discount factor $\alpha = 0.7$. Each picture presents for each state i_0 of the current sample path, policy μ, and α the estimated bounds on the expected costs $J^*(i_0)$ (black), $J_\mu(i_0)$ (red), and $J_u(i_0)$ (blue) with $u = \mu(i_0)$. This means, for each pair of bounds we draw a filled rectangle (in the corresponding color) where the "upper" side states the computed upper bound and the "lower" side the computed lower bound. Tables of the actual values of the computed bounds are presented in Appendix C.2. Since we also estimated for each state i_0 the expected cost $J_u(i_0)$ for all $u \in C(i_0)$, it is often possible to answer

the question: Is the control chosen by μ optimal? This answer is also shown in each picture. The computed bounds which are necessary to answer the optimality question are given in tables shown in Appendix C.3. Moreover, the answer "Yes" is given if the control chosen by the policy satisfies Inequality (5.3). This question is answered with "No" if the control selected by the policy meets Inequality (5.2). In the case that the selected control by the policy neither satisfies Inequality (5.2) nor Inequality (5.3), then the answer "Maybe" is stated.

5.4.1 Downstream Problem Bin-Packing

In this section we consider the MIN-TOTAL ONLINETDAP with downstream bin-packing as it is described in Section 5.2 where $\delta = 3$ is chosen. For this problem setting we present the computational results for the policies PTD and PFD applied to the associated Markov decision processes.

As mentioned before the sample path of states depends among other things on the considered policy μ. Table 5.4 states the sample path for the policy PTD and Table 5.5 for the policy PFD resulting in the current problem setting. For both policies the computational results are depicted in Figure 5.1 and Figure 5.2.

First of all, both figures illustrate the influence of the discount factor α. For $\alpha = 0.5$ the computed bounds on the expected costs are smaller than the corresponding bounds for $\alpha = 0.7$. This is the case since for a greater discount factor, the future costs matter for the current cost more as for a smaller one. This also explains that if for a state i_0 and policy $\mu \in \{\mathsf{PTD}, \mathsf{PFD}\}$ follows that

$$\underline{J}_\mu(i_0) > \overline{J}^*(i_0) \quad \text{or} \quad \underline{J}_u(i_0) > \overline{J}^*(i_0),$$

then the gaps between these values are in the case of $\alpha = 0.7$ greater than the corresponding gaps in the case of $\alpha = 0.5$. For instance, this can be observed in Figure 5.1 for the state with state index 9 and 17.

This figure also shows that PTD is not an optimal policy. This is the case since in almost all investigated states for the policy PTD the Inequality (5.1) is satisfied (since there is a gap between the red and black rectangle for these states). Furthermore, PTD selects independently of the discount factor α for certain states obviously a control which is not optimal at all, e. g. , for the states with state index 9, 14, and 17. This can be observed since there exists a positive gap between the black and blue rectangles for these states and therefore, the computed bound for these states meet Inequality (5.2). It is also obvious that in these states the gaps are often quite "large". This indicates that the policy PTD is not even a near-optimal policy for the considered MDPs. This conjecture is strengthened by the observation that even in the cases where PTD selects an optimal control, Inequality (5.1) is satisfied often and the resulting gaps are relatively larger.

By Figure 5.2 the policy PFD is also not optimal for the considered MDPs since for each discount factor α the policy PFD selects in the state with state index 20 a control which is not optimal. However, Figure 5.2, which shows the computed bounds for the policy PFD, looks slightly different to Figure 5.1, which states the computed bounds for the policy PTD. This is the case since for each

State Index	State							
	Request r		Assignment					
	k	$d(r)$	Slot 1		Slot 2		Slot 3	
			#size 1	#size 2	#size 1	#size 2	#size 1	#size 2
1	1	1	0	0	0	0	0	0
2	2	1	0	0	0	0	1	0
3	1	1	0	0	2	0	0	0
4	2	1	0	0	3	0	0	0
5	3	2	0	0	4	0	0	0
6	1	1	4	1	0	0	0	0
7	2	1	5	1	0	0	0	0
8	3	2	6	1	0	0	0	0
9	4	1	6	2	0	0	0	0
10	1	1	0	0	0	0	0	0
11	2	1	0	0	0	0	1	0
12	1	1	0	0	2	0	0	0
13	2	2	0	0	3	0	0	0
14	3	1	0	0	3	1	0	0
15	1	2	4	1	0	0	0	0
16	2	2	4	2	0	0	0	0
17	3	1	4	3	0	0	0	0
18	4	1	5	3	0	0	0	0
19	1	1	0	0	0	0	0	0
20	2	2	0	0	0	0	1	0

Table 5.4: The sample path of states for associated Markov decision processes for the considered MIN-TOTAL ONLINETDAPs where $\delta = 3$ is chosen and the policy (algorithm) PTD is used. Note that k denotes the number of requests after the last time shift.

State Index	State							
	Request r		Assignment					
	k	$d(r)$	Slot 1		Slot 2		Slot 3	
			#size 1	#size 2	#size 1	#size 2	#size 1	#size 2
1	1	1	0	0	0	0	0	0
2	2	1	0	0	0	0	1	0
3	1	1	0	0	2	0	0	0
4	2	1	0	0	3	0	0	0
5	3	2	0	0	4	0	0	0
6	1	1	4	0	0	1	0	0
7	2	1	5	0	0	1	0	0
8	3	2	5	0	1	1	0	0
9	4	1	5	0	1	2	0	0
10	1	1	1	2	1	0	0	0
11	2	1	1	2	2	0	0	0
12	1	1	3	0	0	0	0	0
13	2	2	4	0	0	0	0	0
14	3	1	4	0	0	0	0	1
15	1	2	0	0	0	1	0	0
16	2	2	0	0	0	2	0	0
17	3	1	0	0	0	2	0	1
18	4	1	0	0	1	2	0	1
19	1	1	1	2	1	1	0	0
20	2	2	1	2	2	1	0	0

Table 5.5: This table shows the obtained sample path of states for associated Markov decision processes for the considered MIN-TOTAL ONLINETDAPs w. r. t. bin-packing where $\delta = 3$ is chosen and the policy (algorithm) PFD is used. Note that k denotes the number of requests after the last time shift.

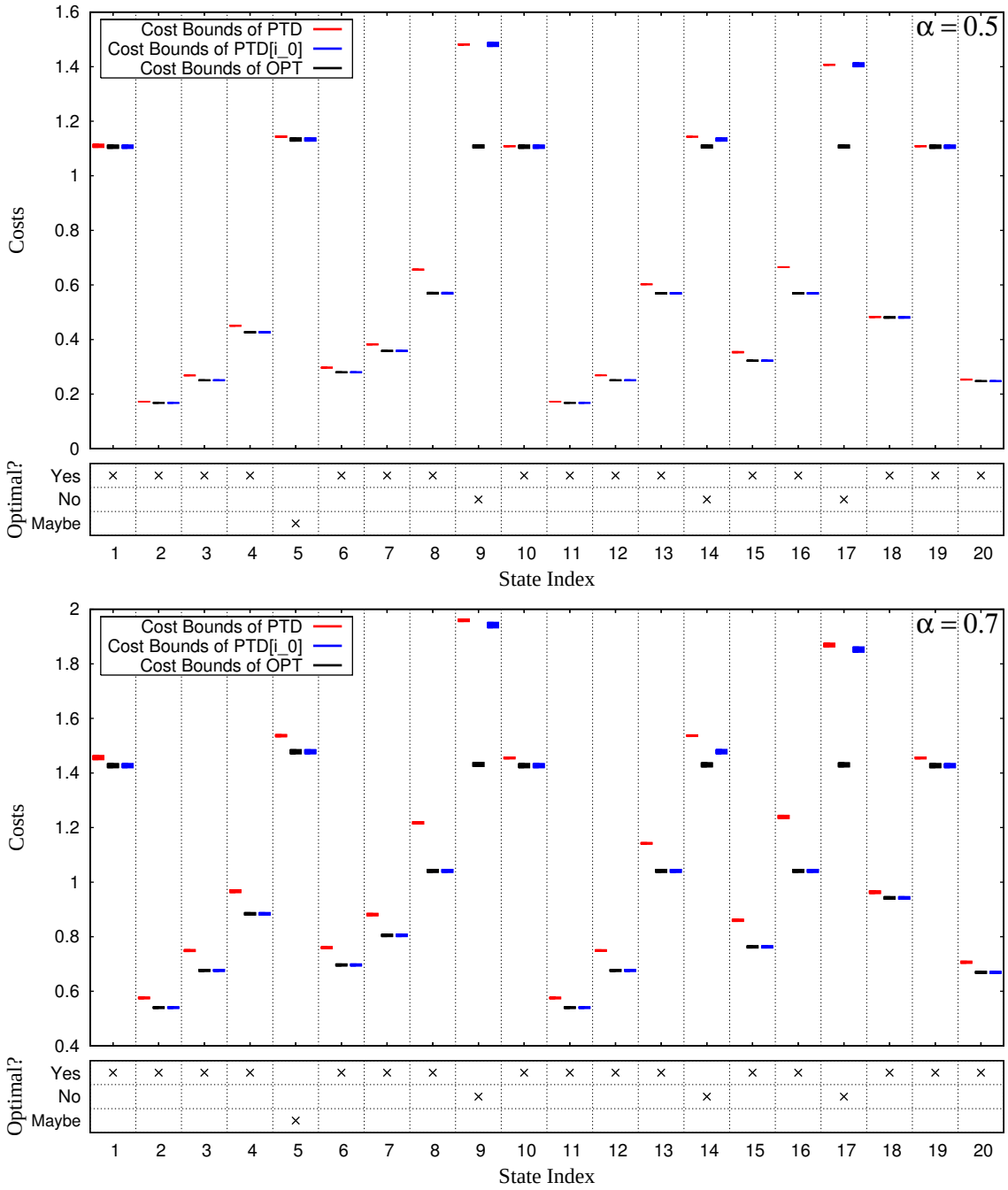

Figure 5.1: The computational results for the policy (algorithm) PTD applied to the MIN-TOTAL ONLINETDAP w. r. t. bin-packing as described in Section 5.2 with $\delta = 3$ and $\alpha \in \{0.5, 0.7\}$. The actual values for the bounds on the expected costs are given in Table C.2 (see Appendix C.2) and the optimality decisions are based on Inequality (5.3) with the bounds stated in Table C.6 and Table C.7 (see Appendix C.3).

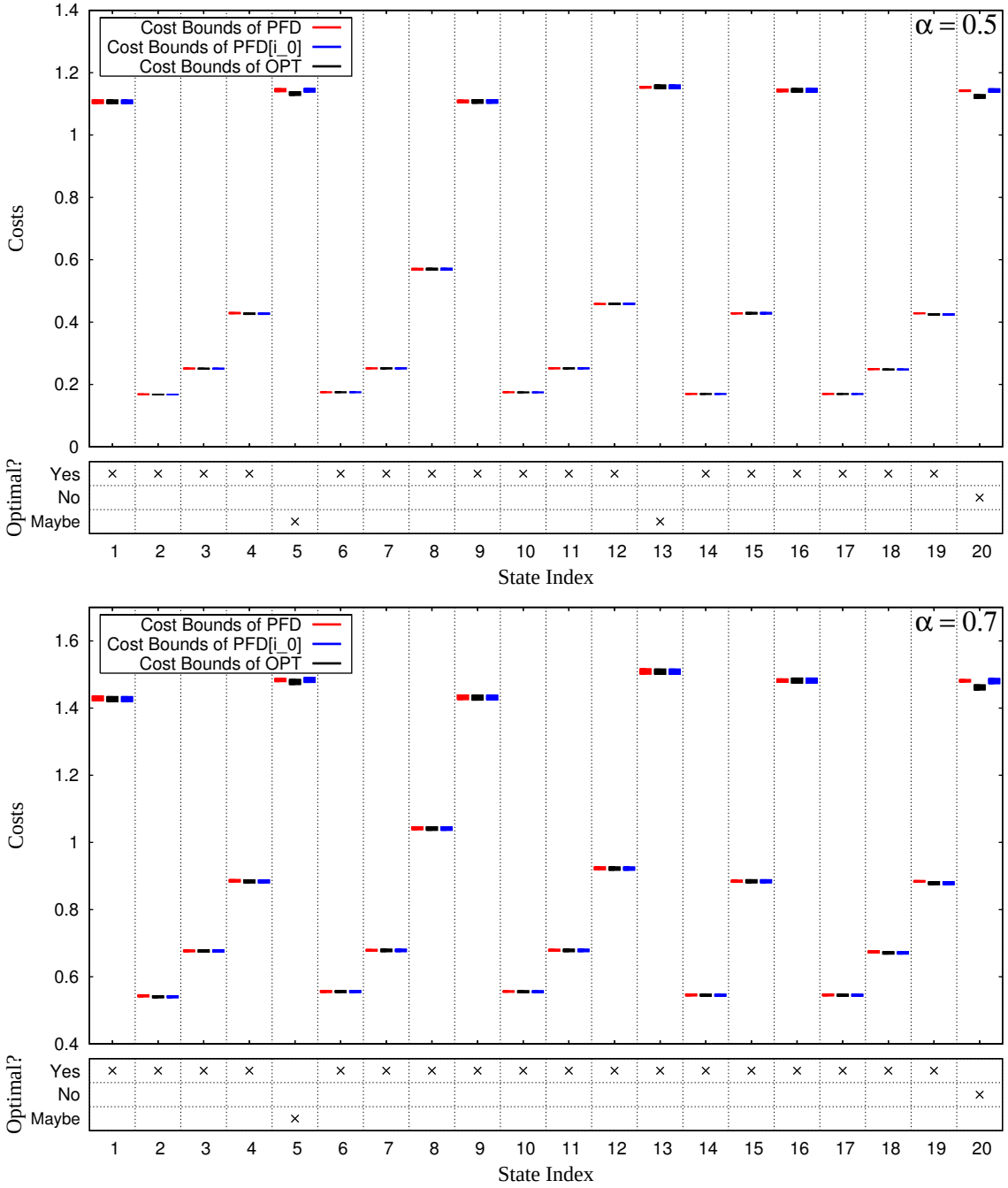

Figure 5.2: The computational results for the policy (algorithm) PFD applied to the MIN-TOTAL ONLINETDAP w. r. t. bin-packing as described in Section 5.2 with $\delta = 3$ and $\alpha \in \{0.5, 0.7\}$. The actual values for the bounds on the expected costs are given in Table C.3 (see Appendix C.2) and the optimality decisions are based on Inequality (5.3) with the bounds stated in Table C.6 and Table C.7 (see Appendix C.3).

investigated state for PFD the drawn rectangles in Figure 5.2, for the computed bounds on the three expected costs, are often on the same level unlike the drawn rectangles in Figure 5.1. In particular, except for two states, these are the states with state index 5 and 20, the computed bounds on the expected costs for PFD are not satisfying Inequality (5.1) and Inequality (5.2). This emphasizes the conjectures that PFD seems to be more promising than PTD and that PFD is a near-optimal policy for the considered MDPs.

Before we go on to the case of downstream parallel-machine scheduling, we want to point out that the optimality patterns given in each picture of the two figures are the same except for the state with state index 13 in Figure 5.2. Looking at the Table C.2, which includes the computed bounds on which these decisions are based in this state, make us suppose that if we were approximate the expected costs a little bit further, then the decision in the case of $\alpha = 0.5$ would flip into a "Yes" as it is in the case of $\alpha = 0.7$.

5.4.2 Downstream Problem Parallel-Machine Scheduling

We consider in this section the MIN-TOTAL ONLINETDAP with parallel-machine scheduling as downstream problem as it is described in Section 5.2 where $\delta = 3$ is chosen. For this problem setting the computational results for the policies PTD and PFD applied to the associated Markov decision processes are presented in this section.

The policy PTD works in this problem setting equivalent to the case of downstream bin-packing, this is, each request is assigned to the same target date. Moreover, the associated Markov decision process for the current problem setting differs only in the cost function compared to the Markov decision process considered for the case of bin-packing as downstream problem. In particular, the state space of these Markov decision processes are equal. Therefore, the policy PTD occupies for both downstream problems the same states. Table 5.4 shows these states. Since the decision made by the policy PFD is influenced by the downstream problem, the visited states for the case of parallel-machine scheduling as downstream problem differ to the case of downstream bin-packing. The sample path of states occupied by the policy PFD for the current problem setting is stated in Table 5.6. For both policies the computational results are depicted in Figures 5.3 and Figure 5.4. Again, both figures illustrate the influence of the discount factor α as described in the case of downstream bin-packing. That is, a greater discount factor leads to greater expected costs compared to a smaller discount factor since future costs matter for a greater discount factor more as for a smaller one.

Figure 5.3 shows that PTD is not an optimal policy for the considered MDPs. For both discount factors there exist many states where the computed bounds for the expected costs meet Inequality 5.1. This can be observed in the figure since there exists often between the red and black rectangles a positive gap. For instance, in the case of $\alpha = 0.5$ Inequality 5.1 is satisfied by the states with state index 5 and 8. In the case of $\alpha = 0.7$ this inequality is satisfied by the computed bounds of all considered states. Similar to the previous section there are states, independently of the chosen discount factor, for which the policy PTD selects a control which is not optimal. This is stated in the figure by a positive gap between the black and blue rectangles, for instance, for the states with

State Index	State							
	Request r		Assignment					
	k	$d(r)$	Slot 1		Slot 2		Slot 3	
			#size 1	#size 2	#size 1	#size 2	#size 1	#size 2
1	1	1	0	0	0	0	0	0
2	2	1	0	0	0	0	1	0
3	1	1	0	0	2	0	0	0
4	2	1	0	0	2	0	1	0
5	3	2	0	0	2	0	2	0
6	1	1	2	0	2	1	0	0
7	2	1	2	0	2	1	1	0
8	3	2	2	0	2	1	2	0
9	4	1	2	0	2	1	2	1
10	1	1	2	1	3	1	0	0
11	2	1	2	1	4	1	0	0
12	1	1	4	1	1	0	0	0
13	2	2	4	1	2	0	0	0
14	3	1	4	1	2	0	0	1
15	1	2	2	0	1	1	0	0
16	2	2	2	0	1	1	0	1
17	3	1	2	0	1	1	0	2
18	4	1	2	0	2	1	0	2
19	1	1	2	1	1	2	0	0
20	2	2	2	1	2	2	0	0

Table 5.6: This table shows the obtained sample path of states for the considered MIN-TOTAL ONLINETDAPs w. r. t. parallel-machine scheduling where $\delta = 3$ is chosen and the policy (algorithm) `PFD` is used. Note that k denotes the number of requests after the last time shift.

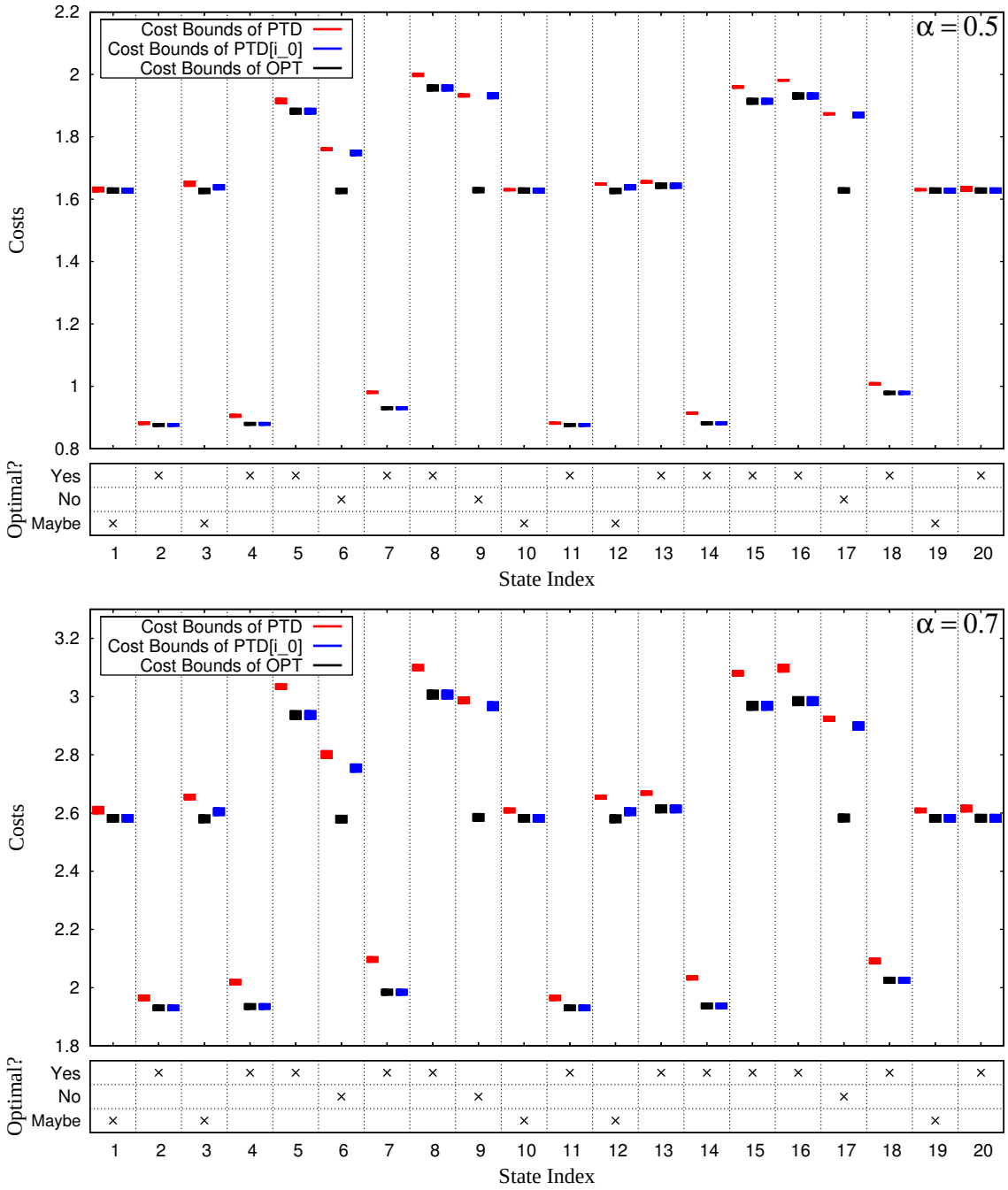

Figure 5.3: The computational results for the algorithm PTD applied to the MIN-TOTAL ONLINETDAP w. r. t. parallel-machine scheduling as described in Section 5.2 with $\delta = 3$ and $\alpha \in \{0.5, 0.7\}$. The actual values for the bounds on the expected costs are given in Table C.4 (see Appendix C.2) and the optimality decisions are based on Inequality (5.3) with the bounds stated in Table C.8 and Table C.9 (see Appendix C.3).

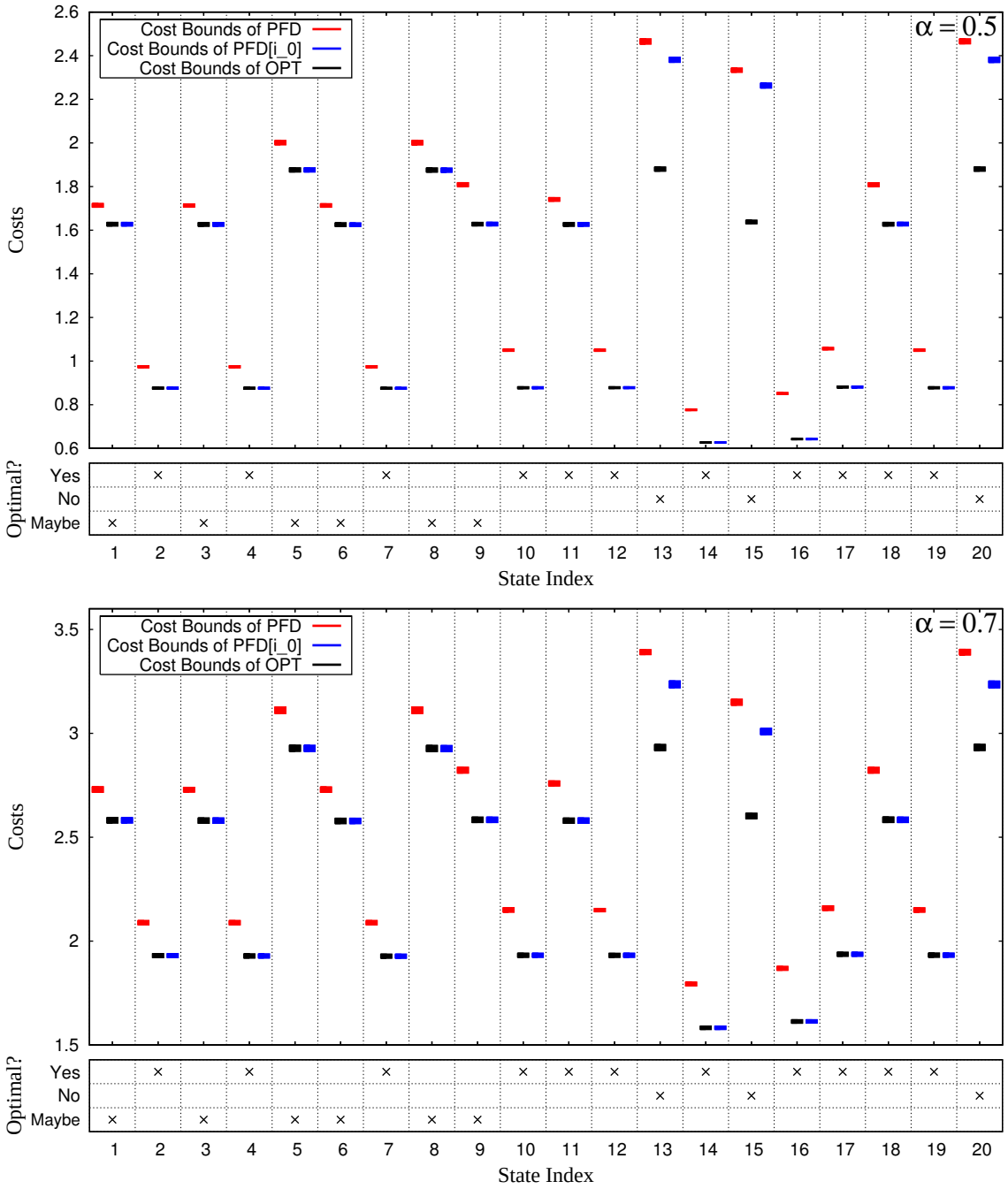

Figure 5.4: The computational results for the algorithm PFD applied to the MIN-TOTAL ONLINETDAP w. r. t. parallel-machine scheduling as described in Section 5.2 with $\delta = 3$ and $\alpha \in \{0.5, 0.7\}$. The actual values for the bounds on the expected costs are given in Table C.5 (see Appendix C.2) and the optimality decisions are based on Inequality (5.3) with the bounds stated in Table C.8 and Table C.9 (see Appendix C.3).

state index 13, 15, and 20. This means, the computed bounds for these states meet Inequality (5.2). Furthermore, these gaps a relatively large. Finally, all these observations show that the policy PTD is not even near-optimal for the MDPs considered in this section.

The policy PFD is not optimal for the current problem setting. This can be observed in Figure 5.4 since there exists a positive gap between the red and black rectangle for all visited states, independently of the chosen discount factor. Hence, for all these states Inequality 5.1 is satisfied by the computed bounds for both discount factors. Moreover, the policy PFD selects, independently of the chosen discount factor, for the states with state index 13, 15, and 20 a control which is not optimal. This can be seen in the figure since for these states there exists a positive gap between the black and blue rectangles. Therefore, the computed bounds for these state satisfy Inequality (5.2). Furthermore, these gaps are relatively large. Hence, the policy PFD is not even a near-optimal policy for the considered MDPs. Furthermore, all these observations show that for the considered MDPs the average performance of PFD is not better than the average performance of PTD.

Finally, in this problem setting the answer "Maybe" is given quite often for the question: If the control chosen by $\mu \in \{\mathsf{PTD}, \mathsf{PFD}\}$ optimal? Looking for these states at the actual values for the computed bounds, which are posted in the tables in Appendix C.3, shows that for these states often two controls exist which have similar bounds. This could indicate that in these states more than one control is optimal. However, to prove this conjecture we have to know the exact expected costs. However, these are in general computational impossible to compute.

6

Summary, Conclusion, and Outlook

In this diploma thesis we introduced the novel concept of an Online Target Date Assignment Problem (ONLINETDAP) as a general framework for online problems featuring a special two-stage decision process. In a first stage target dates have to be assigned immediately and irrevocably to arising requests, while in the second stage certain "sub-instances", these are instances formed by requests assigned to a joint target date, can be solved to optimality later. The cost at a target date is given by the downstream cost which is the optimal cost of processing all requests assigned to this date w. r. t. a given downstream problem. There are two intuitive objectives: Minimizing the sum over all downstream costs (MIN-TOTAL ONLINETDAP) and minimizing the maximum downstream cost over all target dates (MIN-MAX ONLINETDAP). A real-life application was given in Section 1.1: The customer service of Hermes Technischer Kundendienst.

We investigated first basic examples of ONLINETDAPs. In particular, we provided general online algorithms for the MIN-TOTAL ONLINETDAP and MIN-MAX ONLINETDAP independently of the downstream problem. As the first basic examples, we analyzed these algorithms for the particular academic downstream problems of bin-packing and non-preemptive scheduling on identical and parallel machines. Therefore, we used competitive analysis to analyze the competitive ratios of these algorithms. In some cases, where the results obtained by competitive analysis were not satisfying, we preformed an average case analysis for these algorithms to evaluate the expected performance of them. Therefore, we applied these algorithms to associated Markov decision processes of instances of MIN-TOTAL ONLINETDAPs.

Competitive Analysis

First of all, we observed for both objective functions that special settings lead to trivial problems or prevent any deterministic online algorithm from achieving a constant competitive ratio. In particular, this is the case if each request has no deadline (independently of the considered downstream problem and overall objective). Furthermore, in the case of bin-packing as downstream problem there exists

no competitive deterministic online algorithm if this downstream problem only provides a bounded number of bins. However, there are also interesting settings of MIN-TOTAL ONLINETDAP and MIN-MAX ONLINETDAP which do not yield trivial results.

In the case of MIN-TOTAL ONLINETDAP we introduced the general online algorithm PACKTOGETHERORDELAY (PTD) which is 2-competitive if the downstream problem satisfies some properties and each request has $\delta \in \mathbb{N}$ feasible target dates. Parallel-machine scheduling as downstream problem always meets these properties. The downstream problem bin-packing only conforms these properties if it provides an infinite number of bins. Furthermore, we proved for these profiles of the downstream problems and the assumption that all requests have $\delta \in \{2, 3, \dots\}$ feasible target dates lower bounds on the competitive ratio of any deterministic online algorithm. In particular, we showed in the case of downstream bin-packing that no deterministic online algorithm has a competitive ratio better than $3/2$. For parallel-machine scheduling as downstream problem we constructed request sequences which force any deterministic online algorithm to a cost of $\sqrt{2}$ times the optimal cost if additionally is assumed that more than one machine is available.

In the case of MIN-MAX ONLINETDAP we introduced the online algorithm BALANCE (BAL). If each request has $\delta \in \{2, 3, \dots\}$ feasible target dates, this algorithm is competitive for specific profiles of the considered downstream problems. In particular, if the downstream problem bin-packing provides an infinite number of bins, then the algorithm BAL is 4-competitive. Moreover, for this problem setting we proved that no deterministic online algorithm has competitive ratio less than 2. For parallel-machine scheduling as downstream problem the algorithm BAL is $(3 - 1/\delta)$-competitive. Furthermore, if this downstream problem has more than one machine, we showed that $3/2$ is a lower bound on the competitive ratio of any deterministic online algorithm.

The presented results for MIN-TOTAL ONLINETDAPs and MIN-MAX ONLINETDAPs using competitive analysis were satisfying in some cases. On the one hand, it was surprising that a great portion of the investigated problem setting turned out to be in some way trivial. On the other hand, it was dissatisfying that there are still gaps between the proved lower and upper bound on the competitive ratio of the considered online algorithms for reasonable problem setting, in particular since the investigated downstream problems are well known. Furthermore, in the case of MIN-TOTAL ONLINETDAP we introduced the algorithm PACKFIRSTORDELAY (PFD) which seemed to be more promising than PTD. However, using competitive analysis this conjecture could not be proved. Therefore, we carried out an average case analysis for these two algorithms to receive a satisfying result.

Expected Performance

We analyzed the expected performance of the two algorithms PTD and PFD applied to associated Markov decision processes for instances of MIN-TOTAL ONLINETDAPs. We showed that the associated Markov decision processes have a "very large" state space. For these Markov decision

processes the standard approaches of the field stochastic dynamic optimization are in general infeasible to compute the optimal value function. This is the case since they aim to find the optimal value function globally for each state of the system. We presented a method for approximating the optimal value function of discounted Markov decision problem (MDP) locally. This approach is based on the classical linear programming formulation. Its running time complexity does not depend on the number of states of the considered MDP. We derived from this approach an algorithm which uses the technique of column generation to approximate the optimal value function locally. A version of this algorithm was used to analyze the expected performance of the two algorithms PTD and PFD.

In particular, we considered two instances of MIN-TOTAL ONLINETDAPs. One instance had bin-packing as downstream problem and the other parallel-machine scheduling. Furthermore, we assumed some probabilities on which the average case analysis is based. The presented computational results were in some way surprising. In the case of downstream bin-backing these results emphasize the conjecture that PFD has a better performance than PTD. Furthermore, it seemed that PFD is even a near-optimal policy for this problem setting (w. r. t. the considered stochastic assumptions). For parallel-machine scheduling as downstream problem the computational results showed that in general the performance of PFD is not better than the of PTD. Moreover, it turned out that both algorithms are not near-optimal.

Open Problems

For the introduced ONLINETDAP framework, there are several open problems which are directly and indirectly related to this thesis. The proved bounds, received by competitive analysis in this work, still offer room for improvements since they are not tight. In this thesis we assumed that each request has to be assigned to a future target date. Therefore, the downstream problem of an ONLINETDAP can be solved offline. If we were consider that the release date of a request itself is a feasible target date, the situation would chances. With this assumption the downstream problem is not solvable offline anymore. The question which occurs is: How the proved bounds, stated in this thesis, change if this is assumed? It is also conceivable to consider various other downstream problem. In particular, downstream problems which are abundant in reality such as the vehicle dispatching problem arising at Hermes Technischer Kundendienst. Furthermore, we assumed that each request has the same number of feasible target date. Therefore, the question, which arises, is: What happens if each request has an arbitrary number of feasible target dates? There are several other versions which are conceivable to be verify.

We conjecture that the theoretical result stated in Theorem 4.2 is not tight. That means, we believe that the used state space (a certain r-neighborhood) in the proof of this theorem, to prove the approximation result, is too large. This conjecture is strengthened by the computational results given in Appendix C.3 even though these results are for a fixed application. In particular, these results show that for reasonable approximation guarantees and discount factors the needed state space is pretty large to achieve a provable approximation result. Furthermore, the computational results show that

the computed bounds are much closer than the desired approximation guarantee. Therefore, it would be desirable to improve the theoretical approximation results of Theorem 4.2.

By generating the computational results, we figured out that for MDPs, which result from particular applications, it is often possible to tell for certain states that a certain control is not optimal. If this observation were formalized into rules for particular MDPs, it would be possible to speed up the running time for approximating the optimal value function. This is conjecture since for example the corresponding side constraints for these controls in the linear programs used to compute a lower and an upper bound could be removed without shrinking the feasibility region of these programs. It would be interesting to analyze this observation for particular MDPs.

List of Algorithms

Bibliography

[AFG01] Norbert Ascheuer, Matteo Fischetti, and Martin Grötschel, *Solving the Asymmetric Travelling Salesman Problem with time windows by branch-and-cut*, Mathematical Programming, Series A **90** (2001), no. 3, 475–506 (English).

[Alb03] Susanna Albers, *Online Algorithms: A Survey*, Mathematical Programming **97** (2003), no. 1–2, 3–26.

[Bak85] Brenda S. Baker, *A new proof for the First-Fit Decreasing bin-packing algorithm*, Journal of Algorithms **6** (1985), no. 1, 47–70.

[BDBK+94] Shai Ben-David, Allan Borodin, Richard M. Karp, Gábor Tardos, and Avi Wigderson, *On the Power of Randomization in On-Line Algorithms.*, Algorithmica **11** (1994), no. 1, 2–14.

[Ber01] Dimitri P. Bertsekas, *Dynamic programming and optimal control*, 2 ed., vol. 1 and 2, Athena Scientific, 2001.

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

[BL97] John R. Birge and François Louveaux, *Introduction to stochastic programming*, Springer-Verlag, New York, 1997.

[BLMS+03] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, Guido Schaefer, and T. Vredeveld, *Average case and smoothed competitive analysis of the multi-level feedback algorithm*, in Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (2003), 462–471.

[CGJ96] Edward G. Coffman, Jr., Michael R. Garey, and David S. Johnson, *Approximation Algorithms for $\mathcal{NP}$-Hard Problems*, ch. Approximation algorithms for bin packing: A survey, pp. 46–93, PWS Publishing, Boston, 1996.

[Chv83] Vašek Chvátal, *Linear Programming*, A Series of Books in the Mathematical Sciences, Freeman, 1983.

[dFR] Daniela Pucci de Farias and Benjamin Van Roy, *A Linear Program for Bellman Error Minimization with Performance Guarantees*, submitted to Mathematics of Operations Research.

[dFR03] ———, *The Linear Programming Approach to Approximate Dynamic Programming*, Operations Research **51** (2003), no. 6, 850–865.

[DH03] Herold Dehling and Beate Haupt, *Einführung in die Wahrscheinlichkeitstheorie und Statistik*, Springer, 2003.

[FW98] Amos Fiat and Gerhard J. Woeginger (eds.), *Online Algorithms: The State of the Art*, Lecture Notes in Computer Science, vol. 1442, Springer, 1998.

[GJ79] Michael R. Garey and David S. Johnson, *Computers and Intractability (A guide to the theory of $\mathcal{NP}$-completeness)*, W.H. Freeman and Company, New York, 1979.

[Gra69] Ronald L. Graham, *Bounds on multiprocessing timing anomalies*, SIAM Journal on Applied Mathematics **17** (1969), no. 2, 416–429.

[Grö03a] Martin Grötschel, *Graphen- und Netzwerkalgorithmen (ADM I)*, Lecture Notes, Technical University of Berlin, 2003.

[Grö03b] ———, *Lineare Optimierung (ADM II)*, Lecture Notes, Technical University of Berlin, 2003.

[HKM+05] Stefan Heinz, Sven O. Krumke, Nicole Megow, Jörg Rambau, Andreas Tuchscherer, and Tjark Vredeveld, *The Online Target Date Assignment Problem*, Approximation and Online Algorithms, Lecture Notes in Computer Science, 2005, to appear.

[HKP+05] Stefan Heinz, Volker Kaibel, Matthias Peinhardt, Jörg Rambau, and Andreas Tuchscherer, *Relative Policy Evaluation in Constant-Degree Markov Decision Problems*, Technical Report, ZIB, 2005.

[Joh74] David S. Johnson, *Fast algorithms for bin packing*, Journal of Computer and System Sciences **8** (1974), 272–314.

[KMN02] Michael J. Kearns, Yishay Mansour, and Andrew Y. Ng, *A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes*, Machine Learning **49** (2002), no. 2-3, 193–208.

[KMRS88] A. Karlin, M. Manasse, L. Rudolph, and D. Sleator, *Competitive snoopy paging*, Algorithmica **3** (1988), 70–119.

[KR02] Sven O. Krumke and Jörg Rambau, *Online Optimierung*, Lecture Notes, Technical University of Berlin, 2002.

[Kru01] Sven.O. Krumke, *Online Optimization - Competitive Analysis and Beyond.*, Habilitation Thesis, Technical University of Berlin, 2001.

[LD05] Marco E. Lübbecke and Jacques Desrosiers, *Selected Topics in Column Generation*, Operations Research **53** (2005), no. 6, In press.

[Leu04] Joseph Y.-T. Leung (ed.), *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, CRC Press, 2004.

[MO70] Hisashi Mine and Shunji Osaki, *Markovian Decision Processes*, Modern Analytic and Computational Methods in Science and Mathematics, no. 25, American Elsevier, New York, 1970.

[MR95] Rajeev Motwani and Prabhakar Raghaven, *Randomized Algorithms*, Cambridge University Press, Cambridge UK, 1995.

[Put94] Martin L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, John Wiley & Sons, Inc., 1994.

[Sga98] Jiří Sgall, *On-Line Scheduling – A Survey*, in Online Algorithms: The State of the Art (Amos Fiat and Gerhard J. Woeginger, eds.), Lecture Notes in Computer Science, vol. 1442, Springer, 1998.

[SP04] Michael Z. Spivey and Warren B. Powell, *The dynamic assignment problem*, Transportation Science **38** (2004), 399–419.

[ST85] Daniel Dominic Sleator and Robert Endre Tarjan, *Amortized Efficiency of List Update and Paging Rules.*, Communications of the ACM **28** (1985), no. 2, 202–208.

[TV02] Paolo Toth and Daniele Vigo, *The Vehicle Routing Problem*, SIAM Monographs on Discrete Mathematics and Applications, 2002.

[Wil01] Wilbert E. Wilhelm, *A Technical Review of Column Generation in Integer Programming*, Optimization and Engineering **2** (2001), no. 2, 159–200.

Appendices

A

Mathematical Symbols and Notations

The usage of mathematical symbols and notations differ from one to the other. In this chapter we state the mathematical symbols and notations with the corresponding definition we use in this thesis.

Symbol	**Definition**
$\mathbb{N}$	the natural numbers $\mathbb{N} = \{1, 2, 3, 4, \dots\}$;
$\mathbb{N}_0$	the natural numbers including zero $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$;
$\mathbb{R}$	the real numbers;
$\mathbb{R}_+$	the positive real numbers $\mathbb{R}_+ = \{x \in \mathbb{R} \mid x > 0\}$;
$\mathbb{R}_{\geq 0}$	the positive real numbers including zero $\mathbb{R}_{\geq 0} = \{x \in \mathbb{R} \mid x \geq 0\}$;
S^n	$S^n = \{(s_1, s_2, \dots, s_n) \mid s_1, s_2, \dots, s_n \in S\}$ denotes the Cartesian product;
$\lvert S \rvert$	number of elements the set S contains;
$\lceil x \rceil$	the ceiling function which gives the smallest integer greater than or equal to $x \in \mathbb{R}$;
$\lfloor x \rfloor$	the floor function which gives the largest integer less than or equal to $x \in \mathbb{R}$;
X	a discrete random variable;
$\mathbb{E}[X]$	the expectation of the random variable X;
$\mathbb{E}[X_1 \mid X_2]$	the conditional expectation;

B

Zusammenfassung

Viele Planungsaufgaben, die in der Praxis vorkommen, unterliegen einem zweistufigen Entscheidungsprozess. In der ersten Stufe werden Ressourcen aufkommenden Anfragen zugewiesen. Dies muss sofort nach Auftreten dieser Anfrage und ohne Wissen über zukünftige Ereignisse geschehen. In einer zweiten Stufe werden alle Anfragen, die einer bestimmten Ressource zugewiesen wurden, bezüglich eines gegeben Kostenkriteriums, optimal abgearbeitet. Ein Beispiel für ein derartiges Problem ist die Servicetechniker-Einsatzplanung der Firma Hermes Technischer Kundendienst. In der ersten Stufe wird während des Telefonates mit einem Kunden ein Zeitrahmen (zum Beispiel ein Wochentag) innerhalb eines festen Zeithorizonts (zum Beispiel die folgenden zwei Wochen) vereinbart, um einen Serviceauftrag beim Kunden auszuführen. In der zweiten Stufe wird vor jedem Außendiensttag ein bezüglich eines Kostenkriteriums optimaler Techniker-Einsatzplan berechnet. Dieser umfasst alle vereinbarten Kundentermine des betrachteten Tages.

In dieser Diplomarbeit entwickelten wir ein mathematisches Modell für die Klasse von Planungsaufgaben, bei der die zuzuweisende Ressource Zeitrahmen sind. Wir nennen dieses Modell das Online Target Date Assignment Problem, oder kurz ONLINETDAP. Ein Auftrag eines ONLINETDAP muss sofort einem Zeitrahmen innerhalb eines gegebenen Zeitfensters unwiderruflich zugewiesen werden, d. h. ohne Wissen über zukünftige Aufträge und somit *online*. Die Kosten für einen Zeitrahmen ergegeben sich aus den *nachgelagerten Kosten*. Das sind die Kosten für das Bearbeiten aller Aufträge, die diesem Zeitrahmen zugewiesen wurden. Dabei nehmen wir an, dass diese hinsichtlich eines gegebenen *nachgelagerten Optimierungsproblems* optimal sind. Dieses nachgelagerte Optimierungsproblem kann *offline* gelöst werden, da zu diesem Zeitpunkt alle Aufträge des nächsten Zeitrahmens bekannt sind. Die Zuweisungsentscheidungen definieren also den Inputs mehrerer nachgelagerter Optimierungsprobleme und beeinflussen damit die Summe aller nachgelagerter Kosten und die maximalen nachgelagerter Kosten über alle Zeitrahmen. Dies führt für diese Problemformulierung zu den beiden Zielsetzungen: Minimierung der Summe aller nachgelagerten Kosten und Minimierung der maximalen nachgelagerten Kosten über alle Zeitrahmen.

Wir präsentierten für diese Problemklasse Online-Algorithmen in Abhängigkeit von der Zielsetzung, jedoch unabhängig vom nachgelagerten Problem. Diese Algorithmen wurden für elementare

nachgelagerte Probleme, wie Bin-Packing und ein Maschinen-Scheduling Problem, analysiert. Im Mittelpunkt stand die Untersuchung des worst-case Verhaltens dieser Algorithmen unter Zuhilfenahme der *kompetitiven Analyse*. In den Fällen, wo die kompetitive Analyse keine zufriedenstellenden Ergebnisse lieferte, haben wir die Algorithmen bezüglich ihres durchschnittlichen Verhaltens analysiert. Dafür entwickelten wir eine neue Methode, welche die optimale Wertfunktion eines diskontierten Markov-Entscheidungsproblems lokal approximiert.

Kompetitive Analyse

Unter der Zielsetzung, die maximalen nachgelagerten Kosten über alle Zeitrahmen zu minimieren, ist es eine vielversprechende Strategie, die ankommenden Aufträge über alle Zeitrahmen auszubalancieren. Diese Idee liefert den Online-Algorithmus BALANCE. In dieser Arbeit konnten wir zeigen, dass dieser Algorithmus für das nachgelagerte Problem des Bin-Packings 4-kompetitiv ist. Für diese Problemstellung haben wir außerdem bewiesen, dass 2 eine untere Schranke für den kompetitiven Faktor für jeden deterministischen Online-Algorithmus ist. Im Falle von Maschinen-Scheduling haben wir bewiesen, dass BALANCE 3-kompetitiv ist und $3/2$ eine untere Schranke an den kompetitiven Faktor für jeden beliebigen deterministischen Online-Algorithmus ist.

Für die Problemstellung, die Summe aller nachgelagerten Kosten zu minimieren, haben wir den Algorithmus PACKTOGETHERORDELAY (PTD) vorgestellt. Die Grundidee dieses Algorithmus' ist, auftretende Aufträge zu bündeln. Wir haben gezeigt, dass dieser Algorithmus 2-kompetitiv ist, wenn das nachgelagerte Problem bestimmte Eigenschaften erfüllt. Diese Eigenschaften werden vom betrachteten Maschinen-Scheduling Problem eingehalten. Im Falle des Bin-Packing Problem sind diese Eigenschaften nur erfüllt, wenn beliebig viele Bins zur Verfügung stehen. Weiterhin haben wir bewiesen, dass für das nachgelagerte Problem des Maschinen-Scheduling kein deterministischer Online-Algorithmus einen kompetitiven Faktor kleiner als $\sqrt{2}$ haben kann. Wenn Bin-Packing das nachgelagerte Problem ist, folgt eine untere Schranke von $3/2$ für den kompetitiven Faktor eines jeden deterministischen Online-Algorithmus für dieses Problem. Nachfolgend haben wir den Online-Algorithmus PACKFIRSTORDELAY (PFD) vorgestellt. Dieser ist eine modifizierte Version von PTD und wir hatten die Vermutung, dass dieser Algorithmus eine bessere Gütegarantie als PTD hat. Jedoch war es nicht möglich, ein zufriedenstellendes Resultat mit Hilfe der kompetitiven Analyse zu beweisen. Daher haben wir diese beiden Algorithmen bezüglich ihres durchschnittlichen Verhaltens analysiert.

Durchschnittsanalyse

Wir haben die beiden Algorithmen PTD und PFD auf Markov-Entscheidungsprozesse angewandt. Diese Prozesse gehören zu Instanzen von ONLINETDAPs mit der Zielsetzung, die Summe aller

nachgelagerten Kosten zu minimieren. Die klassischen Methoden aus der stochastischen dynamischen Optimierung zur Berechnung der diskontierten Kosten sind im Allgemeinen nicht anwendbar, wenn der Zustandsraum zu "groß" ist. Ein Grund dafür ist, dass diese Verfahren für jeden Zustand eines Markov-Entscheidungsprozesses die diskontierten Kosten global berechnen. In dieser Arbeit haben wir ein neues Verfahren vorgestellt, welches die diskontierten Kosten eines Zustandes lokal approximiert. Mit Hilfe dieses Verfahrens haben wir beide Algorithmen bezüglich ihres durchschnittlichen Verhaltens untersucht. Dazu betrachteten wir zwei Probleminstanzen des ONLINE-TDAPs mit dem Ziel, die Summe der nachgelagerten Kosten zu minimieren. Eine Instanz hat Bin-Packing als nachgelagertes Problem, die andere Maschinen-Scheduling. Weiterhin haben wir Übergangswahrscheinlichkeiten angenommen, auf welchen die Durchschnittsanalyse beruht. Die präsentierten Ergebnisse haben im Falle von Bin-Packing als nachgelagertes Problem die Vermutung verstärkt, dass PFD eine besseres "Verhalten" als PTD hat. Im Fall von Maschinen-Scheduling wurde diese Vermutung im Allgemeinen jedoch widerlegt.

C

Additional Computational Results

This chapter states additional computational results. Mainly, we present the computational result tables for the figures shown in Section 5.4.

C.1 ApproxByStaticNeighborhood vs. ApproxByDynamicNeighborhood

This section presents some experimental results which compare the number of necessary states the two algorithms ApproxByStaticNeighborhood (ASN) and ApproxByDynamicNeighborhood (ADN) need to generate, in order to achieve some approximation guarantee.

We considered as input instances for both algorithms:

- the associated Markov decision process which results from the Min-Total OnlineTDAP with downstream problem bin-packing as described in Section 5.2 where $\delta = 3$ is chosen;
- $\alpha \in \{0.5, 0.7\}$;
- the state with state index 1 shown in Table 5.4;
- $\varepsilon \in \{0.5, 0.3, 0.1, 0.05, 0.01\}$.

Table C.1 shows the computed bounds as well as the number of necessary states to achieve a provable approximation result for both algorithms.

This table shows that the required state space of the algorithm ASN is pretty large for reasonable α and ε and the computed bounds are much closer than ε. For the algorithm ADN the required number of states to achieve a certain approximation guarantee is much smaller than the corresponding number of ASN . Therefore, these experimental results emphasize the conjecture that the result of Theorem 4.2 is not tight in the sense that the necessary state space to reach a given approximation guarantee is probably much smaller.

Parameters		ASN				ADN		
α	ε	r	$\lvert S(i_0, r)\rvert$	lower bound	upper bound	$\lvert S\rvert$	lower bound	upper bound
0.5	0.5	1	12	1	1.49735	8	1	1.5
0.5	0.3	2	88	1.001	1.24475	17	1	1.29
0.5	0.1	4	1210	1.09168	1.1406	85	1.07728	1.17608
0.5	0.05	5	3242	1.09592	1.12011	180	1.08988	1.13957
0.5	0.01	7	12412	1.10324	1.1066	625	1.1014	1.11135
0.7	0.5	5	3242	1.31973	1.54286	162	1.27753	1.77745
0.7	0.3	6	6756	1.36461	1.48506	368	1.32418	1.624
0.7	0.1	9	32358	1.41933	1.43338	1072	1.39302	1.49286
0.7	0.05	11	67134	1.42389	1.42513	1750	1.408	1.45788
0.7	0.01	13	209524	1.42426	1.42426	3629	1.42104	1.43104

Table C.1: Number of necessary states for the algorithms ASN and ADN to achieve a desired approximation guarantee.

C.2 Computational Result Tables

This section contains the tables for computational results depicted in the figures shown in Section 5.4. For each figure we present one table including all bounds drawn in the corresponding figure.

The following table shows which table belongs to which figure.

Figure	Computational Result Table
Figure 5.1	Table C.2
Figure 5.2	Table C.3
Figure 5.3	Table C.4
Figure 5.4	Table C.5

State Index	$\alpha = 0.5$						$\alpha = 0.7$					
	$J^*(i_0)$		$J_u(i_0)$		$J_{\mathsf{PTD}}(i_0)$		$J^*(i_0)$		$J_u(i_0)$		$J_{\mathsf{PTD}}(i_0)$	
	lower bound	upper bound	lower bound	upper bound	lower bound	upper bound	lower bound	upper bound	lower bound	upper bound	lower bound	upper bound
1	1.10116	1.11206	1.10115	1.11213	1.10436	1.11529	1.41974	1.43392	1.41977	1.43387	1.44926	1.46334
2	0.166842	0.168484	0.166846	0.168492	0.171884	0.173504	0.537628	0.542987	0.537628	0.542987	0.572999	0.57851
3	0.249448	0.251936	0.249448	0.251936	0.267678	0.269588	0.67318	0.679853	0.67318	0.679853	0.745986	0.752453
4	0.424781	0.428953	0.424781	0.428953	0.449383	0.452262	0.879524	0.888224	0.879524	0.888224	0.962046	0.971548
5	1.1274	1.13841	1.12726	1.13852	1.14123	1.14583	1.47058	1.48514	1.47055	1.48518	1.53311	1.54039
6	0.27891	0.281652	0.27891	0.281652	0.295902	0.298575	0.693231	0.700151	0.693231	0.700151	0.756663	0.763351
7	0.35653	0.360058	0.35653	0.360058	0.381088	0.383548	0.801049	0.80898	0.801049	0.80898	0.877091	0.885353
8	0.566757	0.57234	0.566757	0.57234	0.654149	0.657846	1.03625	1.04658	1.03625	1.04658	1.21318	1.22159
9	1.10214	1.11297	1.47409	1.4885	1.47947	1.48274	1.42413	1.43816	1.93226	1.95152	1.95489	1.96421
10	1.10116	1.11206	1.10115	1.11213	1.10699	1.1101	1.41974	1.43392	1.41977	1.43387	1.45194	1.45826
11	0.166842	0.168484	0.166846	0.168492	0.172013	0.173288	0.537628	0.542987	0.537628	0.542987	0.573105	0.578335
12	0.249448	0.251936	0.249448	0.251936	0.26802	0.269062	0.67318	0.679853	0.67318	0.679853	0.746207	0.752078
13	0.566741	0.572231	0.566734	0.57224	0.601469	0.603076	1.03615	1.04648	1.03616	1.04646	1.1395	1.14458
14	1.10185	1.11258	1.12731	1.13851	1.14195	1.14443	1.42264	1.43681	1.47065	1.48528	1.53373	1.53935
15	0.320966	0.324155	0.320966	0.324155	0.352041	0.355276	0.759112	0.766678	0.759112	0.766678	0.856396	0.863964
16	0.566786	0.572322	0.566786	0.572322	0.664336	0.666365	1.03616	1.04646	1.03616	1.04646	1.23328	1.24468
17	1.10169	1.11265	1.39983	1.41382	1.40548	1.4079	1.42269	1.43676	1.84307	1.86131	1.86246	1.87515
18	0.478587	0.483311	0.478587	0.483311	0.481491	0.484355	0.93734	0.94666	0.93734	0.94666	0.958568	0.967883
19	1.10116	1.11206	1.10115	1.11213	1.10704	1.11	1.41974	1.43392	1.41977	1.43387	1.45194	1.45826
20	0.246602	0.249064	0.246602	0.249064	0.252104	0.254492	0.666093	0.672723	0.666093	0.672728	0.703143	0.709886

Table C.2: Computed bounds for PTD applied to the MIN-TOTAL ONLINETDAP w. r. t. bin-packing as described in Section 5.2 where $\delta = 3$ and $\alpha \in \{0.5, 0.7\}$ is chosen and $u = \mathsf{PTD}[i_0]$. Cf. Figure 5.1

State Index	$\alpha = 0.5$						$\alpha = 0.7$					
	$J^*(i_0)$		$J_u(i_0)$		$J_{\mathsf{PFD}}(i_0)$		$J^*(i_0)$		$J_u(i_0)$		$J_{\mathsf{PFD}}(i_0)$	
	lower bound	upper bound	lower bound	upper bound	lower bound	upper bound	lower bound	upper bound	lower bound	upper bound	lower bound	upper bound
1	1.10116	1.11206	1.10115	1.11213	1.10163	1.1125	1.41974	1.43392	1.41977	1.43387	1.42164	1.43585
2	0.166842	0.168484	0.166846	0.168492	0.167933	0.169582	0.537628	0.542987	0.537628	0.542987	0.540674	0.545999
3	0.249448	0.251936	0.249448	0.251936	0.25005	0.252224	0.67318	0.679853	0.67318	0.679853	0.674866	0.679257
4	0.424781	0.428953	0.424781	0.428953	0.427007	0.430747	0.879524	0.888224	0.879524	0.888224	0.882346	0.889091
5	1.1274	1.13841	1.13812	1.14916	1.13933	1.1483	1.47058	1.48514	1.47637	1.49102	1.47915	1.48823
6	0.174314	0.176049	0.174321	0.176031	0.174458	0.175956	0.553084	0.55861	0.553084	0.558609	0.553686	0.558746
7	0.250603	0.253092	0.250603	0.253092	0.250926	0.252888	0.67512	0.681839	0.67512	0.681839	0.676381	0.680984
8	0.566706	0.572301	0.566706	0.572301	0.567117	0.572252	1.03629	1.04663	1.03629	1.04663	1.03794	1.04613
9	1.1022	1.11322	1.1022	1.11322	1.10365	1.11163	1.424	1.43823	1.42406	1.43811	1.42575	1.43884
10	0.173961	0.17568	0.173961	0.17568	0.174556	0.175867	0.552694	0.558155	0.552694	0.558155	0.554086	0.557751
11	0.250658	0.253128	0.250658	0.253128	0.250892	0.253218	0.67514	0.681887	0.67514	0.681887	0.676139	0.681362
12	0.456463	0.46098	0.456463	0.46098	0.457233	0.459322	0.917422	0.92657	0.917422	0.92657	0.918096	0.926921
13	1.14861	1.16008	1.14877	1.15997	1.15062	1.15606	1.50086	1.51581	1.50084	1.51584	1.50155	1.51655
14	0.168624	0.17028	0.168622	0.170292	0.168852	0.170484	0.542393	0.547769	0.542382	0.547791	0.543478	0.548551
15	0.426443	0.430621	0.426443	0.430621	0.426946	0.429716	0.879946	0.888735	0.879946	0.888735	0.881507	0.887256
16	1.13777	1.14863	1.13774	1.14897	1.13886	1.14697	1.47459	1.4893	1.4746	1.4893	1.47675	1.48687
17	0.168639	0.170284	0.168632	0.170288	0.168903	0.170564	0.542389	0.547766	0.542378	0.547755	0.543447	0.548491
18	0.247023	0.249471	0.247023	0.249471	0.248347	0.250393	0.66764	0.674261	0.667632	0.674293	0.67096	0.677196
19	0.422521	0.426711	0.422521	0.426711	0.427156	0.429008	0.874165	0.882778	0.874165	0.882778	0.881976	0.886192
20	1.11815	1.12916	1.13706	1.14798	1.13955	1.14417	1.45423	1.46868	1.4732	1.48793	1.47782	1.48452

Table C.3: Computed bounds for PFD applied to the MIN-TOTAL ONLINETDAP w. r. t. bin-packing as described in Section 5.2 where $\delta = 3$ and $\alpha \in \{0.5, 0.7\}$ is chosen and $u = \mathsf{PFD}[i_0]$. Cf. Figure 5.2

State Index	$\alpha = 0.5$						$\alpha = 0.7$					
	$J^*(i_0)$		$J_u(i_0)$		$J_{\mathsf{PTD}}(i_0)$		$J^*(i_0)$		$J_u(i_0)$		$J_{\mathsf{PTD}}(i_0)$	
	lower bound	upper bound	lower bound	upper bound	lower bound	upper bound	lower bound	upper bound	lower bound	upper bound	lower bound	upper bound
1	1.61957	1.63575	1.61954	1.63558	1.62356	1.63866	2.56901	2.59452	2.56877	2.59445	2.59742	2.62207
2	0.872075	0.880691	0.872075	0.880691	0.878025	0.88626	1.92073	1.93987	1.92074	1.93992	1.95547	1.97464
3	1.6185	1.63462	1.63024	1.64629	1.64154	1.65769	2.56716	2.59281	2.59145	2.61727	2.64522	2.66441
4	0.87538	0.884105	0.87538	0.884105	0.901122	0.910109	1.92566	1.94489	1.92564	1.94488	2.00996	2.02827
5	1.87324	1.8911	1.87324	1.8911	1.9064	1.92358	2.92195	2.95112	2.92228	2.95144	3.02531	3.04339
6	1.61858	1.63461	1.74001	1.75734	1.7563	1.76464	2.56631	2.59194	2.74062	2.76799	2.78755	2.81405
7	0.925583	0.934807	0.925583	0.934807	0.977619	0.984434	1.9743	1.99404	1.97427	1.99399	2.0876	2.10594
8	1.94718	1.96631	1.94718	1.96631	1.9934	2.00324	2.99184	3.02163	2.99179	3.02164	3.08834	3.10973
9	1.6209	1.63685	1.92238	1.94115	1.92808	1.93754	2.57198	2.5977	2.9521	2.98156	2.9752	2.99888
10	1.61957	1.63575	1.61954	1.63558	1.62719	1.63359	2.56901	2.59452	2.56877	2.59445	2.60086	2.61689
11	0.872075	0.880691	0.872075	0.880691	0.878665	0.884981	1.92073	1.93987	1.92074	1.93992	1.95611	1.97368
12	1.6185	1.63462	1.63024	1.64629	1.64489	1.65164	2.56716	2.59281	2.59145	2.61727	2.64771	2.66083
13	1.63513	1.65119	1.63513	1.65119	1.65181	1.65916	2.60107	2.62702	2.60116	2.62692	2.66144	2.67583
14	0.877203	0.885793	0.877203	0.885793	0.910986	0.917648	1.92756	1.94674	1.92757	1.94672	2.02669	2.04117
15	1.90476	1.92298	1.90476	1.92298	1.95578	1.96366	2.95354	2.98306	2.95355	2.98306	3.07074	3.08919
16	1.9216	1.94053	1.9216	1.94053	1.97848	1.98379	2.96938	2.99906	2.96939	2.99905	3.08422	3.11093
17	1.62013	1.63632	1.86051	1.87865	1.8703	1.87674	2.57001	2.59569	2.88431	2.91304	2.91478	2.93233
18	0.974162	0.9839	0.974162	0.9839	1.00422	1.01186	2.01539	2.03543	2.01536	2.03545	2.08179	2.10122
19	1.61957	1.63575	1.61954	1.63558	1.62731	1.63347	2.56901	2.59452	2.56877	2.59445	2.60107	2.61654
20	1.61994	1.63585	1.61994	1.63585	1.62535	1.64065	2.56922	2.59489	2.56919	2.59487	2.60404	2.62664

Table C.4: Computed bounds for PTD applied to the MIN-TOTAL ONLINETDAP w. r. t. parallel-machine scheduling as described in Section 5.2 where $\delta = 3$ and $\alpha \in \{0.5, 0.7\}$ is chosen and $u = \mathsf{PTD}[i_0]$. Cf. Figure 5.3

State Index	$\alpha = 0.5$						$\alpha = 0.7$					
	$J^*(i_0)$		$J_u(i_0)$		$J_{\mathsf{PFD}}(i_0)$		$J^*(i_0)$		$J_u(i_0)$		$J_{\mathsf{PFD}}(i_0)$	
	lower bound	upper bound	lower bound	upper bound	lower bound	upper bound	lower bound	upper bound	lower bound	upper bound	lower bound	upper bound
1	1.61957	1.63575	1.61954	1.63558	1.70516	1.72205	2.56901	2.59452	2.56877	2.59445	2.71605	2.74315
2	0.872075	0.880691	0.872075	0.880691	0.969307	0.978901	1.92073	1.93987	1.92074	1.93992	2.07813	2.09889
3	1.6185	1.63462	1.61861	1.63472	1.70578	1.72034	2.56716	2.59281	2.56713	2.59269	2.71761	2.74025
4	0.871508	0.880183	0.871508	0.880183	0.969252	0.978882	1.91904	1.93823	1.91908	1.93827	2.07815	2.09884
5	1.86709	1.88553	1.867	1.88559	1.9917	2.01098	2.91372	2.94282	2.91313	2.94218	3.09619	3.12679
6	1.61755	1.63353	1.61706	1.63308	1.70553	1.72103	2.56568	2.59128	2.56607	2.59165	2.71642	2.74209
7	0.871038	0.879625	0.871011	0.879642	0.969326	0.978702	1.9181	1.9372	1.91793	1.93706	2.07818	2.09874
8	1.86623	1.88489	1.86592	1.88454	1.99119	2.01089	2.91266	2.94159	2.91229	2.94132	3.09599	3.12685
9	1.61978	1.63569	1.61997	1.6361	1.79994	1.81772	2.57099	2.5966	2.57102	2.59667	2.80868	2.83667
10	0.873648	0.882255	0.873648	0.882255	1.04464	1.05509	1.92216	1.94138	1.92217	1.94132	2.13944	2.16042
11	1.61846	1.63452	1.61857	1.63449	1.73307	1.74942	2.56707	2.59219	2.56717	2.59274	2.74675	2.77084
12	0.873885	0.88197	0.873885	0.88197	1.04473	1.05465	1.92222	1.94117	1.9222	1.94129	2.14152	2.15726
13	1.87084	1.88904	2.36973	2.39253	2.45349	2.4768	2.91714	2.9463	3.21993	3.25209	3.37929	3.40264
14	0.62399	0.63022	0.62399	0.63022	0.772791	0.780425	1.57491	1.59058	1.57497	1.59067	1.78448	1.80229
15	1.62963	1.64559	2.25174	2.27403	2.32426	2.34257	2.58941	2.61518	2.9939	3.0238	3.13575	3.16421
16	0.639287	0.645559	0.639287	0.645559	0.847164	0.855597	1.60522	1.62125	1.60532	1.62133	1.85969	1.87796
17	0.876737	0.885361	0.876737	0.885361	1.05168	1.0621	1.92717	1.94629	1.92725	1.9463	2.14832	2.16968
18	1.61969	1.63567	1.61981	1.63596	1.79998	1.81741	2.57118	2.59685	2.57086	2.59654	2.80896	2.83592
19	0.873567	0.882256	0.873567	0.882256	1.04482	1.05507	1.92265	1.94188	1.92268	1.94187	2.13948	2.16023
20	1.87057	1.88924	2.36944	2.39287	2.45477	2.47688	2.91778	2.94669	3.21921	3.25125	3.37774	3.40441

Table C.5: Computed bounds for PFD applied to the MIN-TOTAL ONLINETDAP w. r. t. parallel-machine scheduling as described in Section 5.2 where $\delta = 3$ and $\alpha \in \{0.5, 0.7\}$ is chosen and $u = \mathsf{PFD}[i_0]$. Cf. Figure 5.4

C.3 Evaluating all Controls of Certain States

Consider an MDP with Markov decision process (S, C, P, cost) and discount factor $\alpha \in (0, 1)$. Theorem 2.12 states known results for MDPs. One of them is that a policy μ is optimal if for every state $i \in S$, the control $\mu(i)$ attains the minimum in Bellman's equation (2.3). To check this criterion the optimal value function J^* is necessary. Since it is usually computational impossible the compute J^*, this criterion does not help. However, there is a possibility to determine for some states an optimal control without knowing J^* exactly.

The idea is to estimate for a state i_0 the expected total discounted cost $J_u(i_0)$ for all controls $u \in C(i_0)$. If we have for a control $u \in C(i_0)$ that the computed upper bound on $J_u(i_0)$ is smaller than or equal to the computed lower bound on $J_{u'}(i_0)$ for all $u' \in C(i_0) \setminus \{u\}$, then the control u is an optimal control for state i_0. On the other hand, a control $u \in C(i_0)$ is not optimal if a control $u' \in C(i_0)$ exists such that the upper bound on $J_{u'}(i_0)$ is smaller than the lower bound on $J_u(i_0)$.

Section 5.4 presented computational results for the algorithms `PTD` and `PFD`. For certain states of given MDPs we illustrated approximation results. The following tables present for each state i_0 considered in Section 5.4 a lower and upper bounds on the expected total discounted cost $J_u(i_0)$ for any possible control $u \in C(i_0)$. For the states where it is possible to tell which control is optimal we print the lower and upper bound with a bold face.

The tables are organized as follows. Table C.6 and Table C.7 present the bounds for the states shown in Table 5.4 and Table 5.5 for associated Markov decision processes for specific MIN-TOTAL ONLINETDAP w. r. t. bin-packing and discount factor $\alpha \in \{0.5, 0.7\}$. Table C.8 and Table C.9 state the bounds for the states shown in Table 5.4 and Table 5.6 for associated Markov decision processes of certain MIN-TOTAL ONLINETDAP w. r. t. parallel-machine scheduling and discount factor $\alpha \in \{0.5, 0.7\}$. Note that each state is coded by the values of the last 8 columns of the sample path tables (see Table 5.4,Table 5.5, and Table 5.6). For instance, the state with state index 1 in Table 5.4 is coded by 11000000.

State	State Index		Control 1		Control 2		Control 3	
	PTD	PFD	lower bound	upper bound	lower bound	upper bound	lower bound	upper bound
11000000	1, 10, 19	1	1.26466	1.27705	1.11264	1.12351	**1.10115**	**1.11213**
11002000	3, 12	3	1.08351	1.09396	**0.249448**	**0.251936**	1.02928	1.03944
11121000	–	10	1.0627	1.07247	**0.173961**	**0.17568**	1.02039	1.03052
11121100	–	19	1.11052	1.12114	**0.422521**	**0.426711**	1.04317	1.05352
11300000	–	12	**0.456463**	**0.46098**	1.07343	1.08354	1.05744	1.06765
11400100	–	6	**0.174321**	**0.176031**	0.195519	0.197461	1.02508	1.03475
11410000	6	–	**0.27891**	**0.281652**	1.06269	1.07245	1.04379	1.05413
12000100	–	15	1.09256	1.10338	**0.426443**	**0.430621**	1.05243	1.06266
12410000	15	–	**0.320966**	**0.324155**	1.08286	1.09323	1.07278	1.0828
21000010	2, 11	2	1.05601	1.06626	1.02257	1.03279	**0.166846**	**0.168492**
21003000	4	4	1.13994	1.1508	**0.424781**	**0.428953**	1.04414	1.05455
21122000	–	11	1.10546	1.11622	**0.250658**	**0.253128**	1.03128	1.04146
21500100	–	7	1.10539	1.11612	**0.250603**	**0.253092**	1.03109	1.04118
21510000	7	–	**0.35653**	**0.360058**	1.08278	1.0932	1.05926	1.06982
22000010	20	–	1.05931	1.06973	1.03094	1.04115	**0.246602**	**0.249064**
22000200	–	16	1.25322	1.26552	1.15162	1.16305	**1.13774**	**1.14897**
22003000	13	–	1.14681	1.158	**0.566734**	**0.57224**	1.0695	1.07978
22122100	–	20	1.2527	1.26441	**1.1181**	**1.12923**	1.13706	1.14798
22400000	–	13	1.31186	1.32474	1.15918	1.17003	1.14877	1.15997
22420000	16	–	**0.566786**	**0.572322**	1.11784	1.12827	1.10512	1.11586
31000201	–	17	1.09885	1.10957	**0.168632**	**0.170288**	0.172527	0.174233
31003100	14	–	1.39995	1.4138	1.12731	1.13851	**1.10182**	**1.11273**
31400001	–	14	**0.168622**	**0.170292**	1.03683	1.04658	0.220311	0.2225
31430000	17	–	1.39983	1.41382	1.12739	1.13823	**1.10164**	**1.11263**
32004000	5	5	1.30861	1.32148	1.12726	1.13852	1.13812	1.14916
32501100	–	8	1.18245	1.1939	**0.566706**	**0.572301**	1.07137	1.08202
32610000	8	–	**0.566757**	**0.57234**	1.1406	1.15133	1.12315	1.13438
41001201	–	18	1.13981	1.15076	1.04315	1.05311	**0.247023**	**0.249471**
41501200	–	9	1.47416	1.4883	1.13592	1.14673	**1.1022**	**1.11322**
41530000	18	–	**0.478587**	**0.483311**	1.12394	1.13475	1.08616	1.09676
41620000	9	–	1.47409	1.4885	1.13581	1.14676	**1.10211**	**1.11301**

Table C.6: Lower and upper bounds on the expected cost $J_u(i_0)$ for each state i_0 shown in Table 5.4 and Table 5.5 and each feasible control $u \in C(i_0)$ for the associated Markov decision process of the MIN-TOTAL ONLINETDAP with downstream problem bin-packing, $\delta = 3$, and $\alpha = 0.5$. If an optimal control is detectable, then the corresponding bounds are printed with a bold face.

State	State Index		Control 1		Control 2		Control 3	
	PTD	PFD	lower bound	upper bound	lower bound	upper bound	lower bound	upper bound
11000000	1, 10, 19	1	1.69332	1.71025	1.44477	1.45892	**1.41977**	**1.43387**
11002000	3, 12	3	1.36243	1.37585	**0.67318**	**0.679853**	1.22535	1.23757
11121000	–	10	1.3081	1.32108	**0.552694**	**0.558155**	1.18889	1.20069
11121100	–	19	1.42198	1.43607	**0.874165**	**0.882778**	1.27527	1.28802
11300000	–	12	**0.917422**	**0.92657**	1.3469	1.36028	1.31176	1.32475
11400100	–	6	**0.553084**	**0.558609**	0.58769	0.593541	1.20559	1.21754
11410000	6	–	**0.693231**	**0.700151**	1.30782	1.32082	1.26381	1.27634
12000100	–	15	1.39053	1.40429	**0.879946**	**0.888735**	1.29469	1.30761
12410000	15	–	**0.759112**	**0.766678**	1.36027	1.37378	1.33643	1.34975
21000010	2, 11	2	1.30137	1.31428	1.19774	1.20966	**0.537628**	**0.542987**
21003000	4	4	1.4824	1.49714	**0.879524**	**0.888224**	1.27656	1.2891
21122000	–	11	1.41287	1.42693	**0.67514**	**0.681887**	1.23188	1.24407
21500100	–	7	1.41255	1.42657	**0.67512**	**0.681839**	1.23156	1.2438
21510000	7	–	**0.801049**	**0.80898**	1.36446	1.37805	1.31258	1.32557
22000010	20	–	1.31285	1.32591	1.23183	1.24413	**0.666093**	**0.672728**
22000200	–	16	1.65669	1.67303	1.50486	1.51987	**1.4746**	**1.4893**
22003000	13	–	1.49926	1.51414	**1.03616**	**1.04646**	1.34627	1.35957
22122100	–	20	1.65506	1.67142	**1.45418**	**1.46863**	1.4732	1.48793
22400000	–	13	1.73108	1.74825	1.52101	1.53595	**1.50084**	**1.51584**
22420000	16	–	**1.03616**	**1.04646**	1.44543	1.45982	1.41947	1.43366
31000201	–	17	1.39281	1.40672	**0.542378**	**0.547755**	0.549814	0.555234
31003100	14	–	1.84403	1.86241	1.47065	1.48528	**1.4227**	**1.43682**
31400001	–	14	**0.542382**	**0.547791**	1.24731	1.2597	0.625066	0.63126
31430000	17	–	1.84307	1.86131	1.47067	1.48536	**1.4227**	**1.43683**
32004000	5	5	1.73171	1.74899	1.47055	1.48518	1.47637	1.49102
32501100	–	8	1.56237	1.57788	**1.03629**	**1.04663**	1.35159	1.36506
32610000	8	–	**1.03625**	**1.04658**	1.4896	1.50441	1.45613	1.47058
41001201	–	18	1.48799	1.50284	1.26714	1.27976	**0.667632**	**0.674293**
41501200	–	9	1.93226	1.9514	1.48552	1.50029	**1.42406**	**1.43811**
41530000	18	–	**0.93734**	**0.94666**	1.45626	1.47081	1.38447	1.39827
41620000	9	–	1.93226	1.95152	1.4856	1.50033	**1.42404**	**1.43825**

Table C.7: Lower and upper bounds on the expected cost $J_u(i_0)$ for each state i_0 shown in Table 5.4 and Table 5.5 and each feasible control $u \in C(i_0)$ for the associated Markov decision process of the MIN-TOTAL ONLINETDAP with downstream problem bin-packing, $\delta = 3$, and $\alpha = 0.7$. If an optimal control is detectable, then the corresponding bounds are printed with a bold face.

State	State Index		Control 1		Control 2		Control 3	
	PTD	PFD	lower bound	upper bound	lower bound	upper bound	lower bound	upper bound
11000000	1, 10, 19	1	1.75282	1.7701	1.63054	1.64657	1.61954	1.63558
11002000	3, 12	3	1.69074	1.70753	1.63024	1.64629	1.61861	1.63472
11202100	–	6	1.69053	1.70707	1.62947	1.64574	1.61706	1.63308
11211200	–	19	1.57347	1.5892	**0.873567**	**0.882256**	1.53741	1.55268
11213100	–	10	1.57386	1.58945	**0.873648**	**0.882255**	1.53777	1.55293
11410000	6	–	1.74001	1.75734	1.63001	1.64562	1.61865	1.63465
11411000	–	12	1.57337	1.58907	**0.873885**	**0.88197**	1.53725	1.5522
12201100	–	15	1.62926	1.64524	1.62955	1.64567	2.25174	2.27403
12410000	15	–	**1.90476**	**1.92298**	2.39201	2.4152	2.38345	2.40648
21000010	2, 11	2	1.56989	1.58545	1.53981	1.55467	**0.872075**	**0.880691**
21002010	–	4	1.56845	1.58397	1.54006	1.55489	**0.871508**	**0.880183**
21003000	4	–	1.58933	1.60475	**0.87538**	**0.884105**	1.53996	1.55535
21202110	–	7	1.56916	1.58435	1.53975	1.55485	**0.871011**	**0.879642**
21214100	–	11	1.72321	1.74015	1.63509	1.65112	**1.61857**	**1.63449**
21510000	7	–	**0.925583**	**0.934807**	1.58911	1.60492	1.56982	1.58517
22000010	20	–	2.39242	2.41535	2.25284	2.27524	**1.61994**	**1.63585**
22003000	13	–	2.41158	2.43566	**1.63513**	**1.65119**	2.25717	2.27974
22201101	–	16	1.26093	1.27348	1.26097	1.27349	**0.639287**	**0.645559**
22212200	–	20	1.87071	1.88932	1.87092	1.88929	2.36944	2.39287
22412000	–	13	1.87072	1.88903	1.87054	1.8888	2.36973	2.39253
22420000	16	–	**1.9216**	**1.94053**	2.40248	2.4264	2.39061	2.41382
31003100	14	–	1.61227	1.62767	**0.877203**	**0.885793**	1.54266	1.55795
31201102	–	17	1.61223	1.62827	**0.876737**	**0.885361**	1.54256	1.55764
31412001	–	14	1.33889	1.35203	1.268	1.28068	**0.62399**	**0.63022**
31430000	17	–	1.86051	1.87865	1.64257	1.65854	**1.62021**	**1.63615**
32002020	–	5	2.63795	2.66321	1.86708	1.88558	1.867	1.88559
32004000	5	–	2.64437	2.67073	**1.87324**	**1.8911**	2.37252	2.39577
32202120	–	8	1.8664	1.88468	1.86614	1.88467	1.86592	1.88454
32610000	8	–	**1.94718**	**1.96631**	2.4138	2.43724	2.39668	2.42051
41202102	–	18	1.81726	1.83508	1.64895	1.66513	**1.61981**	**1.63596**
41202121	–	9	1.80902	1.82674	1.63162	1.64744	1.61997	1.6361
41530000	18	–	**0.974162**	**0.9839**	1.63229	1.64825	1.60165	1.61746
41620000	9	–	1.92238	1.94115	1.64894	1.66513	**1.62091**	**1.63647**

Table C.8: Lower and upper bounds on the expected cost $J_u(i_0)$ for each state i_0 shown in Table 5.4 and Table 5.6 and each feasible control $u \in C(i_0)$ for the associated Markov decision process of the MIN-TOTAL ONLINETDAP with downstream problem parallel-machine scheduling, $\delta = 3$, and $\alpha = 0.5$. If an optimal control is detectable, then the corresponding bounds are printed with a bold face.

State	State Index		Control 1		Control 2		Control 3	
	PTD	PFD	lower bound	upper bound	lower bound	upper bound	lower bound	upper bound
11000000	1, 10, 19	1	2.77916	2.80693	2.59166	2.61746	2.56877	2.59445
11002000	3, 12	3	2.68478	2.71156	2.59145	2.61727	2.56713	2.59269
11202100	–	6	2.68338	2.71002	2.5901	2.61538	2.56607	2.59165
11211200	–	19	2.47574	2.50041	**1.92268**	**1.94187**	2.39025	2.41401
11213100	–	10	2.47659	2.50121	**1.92217**	**1.94132**	2.39049	2.41434
11410000	6	–	2.74062	2.76799	2.59055	2.61642	2.56641	2.59189
11411000	–	12	2.47647	2.50118	**1.9222**	**1.94129**	2.39019	2.41401
12201100	–	15	2.58961	2.6155	2.58991	2.61536	2.9939	3.0238
12410000	15	–	**2.95355**	**2.98306**	3.25761	3.29009	3.23761	3.26992
21000010	2, 11	2	2.46822	2.49263	2.3975	2.42145	**1.92074**	**1.93992**
21002010	–	4	2.46672	2.49124	2.39693	2.42089	**1.91908**	**1.93827**
21003000	4	–	2.51147	2.53641	**1.92564**	**1.94488**	2.39693	2.42082
21202110	–	7	2.46646	2.4911	2.39627	2.42003	**1.91793**	**1.93706**
21214100	–	11	2.73094	2.75822	2.60057	2.62655	**2.56717**	**2.59274**
21510000	7	–	**1.97427**	**1.99399**	2.5115	2.53659	2.467	2.4913
22000010	20	–	3.22047	3.25259	2.99819	3.02814	**2.56919**	**2.59487**
22003000	13	–	3.26353	3.29612	**2.60116**	**2.62692**	3.00561	3.03564
22201101	–	16	2.00983	2.02989	2.00948	2.02957	**1.60532**	**1.62133**
22212200	–	20	2.91804	2.94709	2.91875	2.9479	3.21921	3.25125
22412000	–	13	2.91801	2.94717	2.91808	2.9472	3.21993	3.25209
22420000	16	–	**2.96939**	**2.99905**	3.27398	3.30662	3.24711	3.27932
31003100	14	–	2.5562	2.58172	**1.92757**	**1.94672**	2.40479	2.42884
31201102	–	17	2.55601	2.58146	**1.92725**	**1.9463**	2.40452	2.4285
31412001	–	14	2.15858	2.1801	2.02389	2.04407	**1.57497**	**1.59067**
31430000	17	–	2.88431	2.91304	2.61406	2.63987	**2.5701**	**2.5957**
32002020	–	5	3.6075	3.64346	2.91387	2.94295	2.91313	2.94218
32004000	5	–	3.61847	3.65456	**2.92228**	**2.95144**	3.22516	3.2572
32202120	–	8	2.91222	2.94121	2.91229	2.94132	2.91229	2.94132
32610000	8	–	**2.99179**	**3.02164**	3.29076	3.32346	3.2553	3.28782
41202102	–	18	2.85435	2.88261	2.62546	2.65162	**2.57086**	**2.59654**
41202121	–	9	2.84158	2.86999	2.59666	2.6226	2.57102	2.59667
41530000	18	–	**2.01536**	**2.03545**	2.59386	2.61979	2.53488	2.56019
41620000	9	–	2.9521	2.98156	2.62575	2.652	**2.57205**	**2.59775**

Table C.9: Lower and upper bounds on the expected cost $J_u(i_0)$ for each state i_0 shown in Table 5.4 and Table 5.6 and each feasible control $u \in C(i_0)$ for the associated Markov decision process of the MIN-TOTAL ONLINETDAP with downstream problem parallel-machine scheduling, $\delta = 3$, and $\alpha = 0.7$. If an optimal control is detectable, then the corresponding bounds are printed with a bold face.