Uniwersytet Wrocławski
Wydział Matematyki i Informatyki

Cena opóźnień w algorytmach online

Autor:
Artur Kraska

Rozprawa sporządzona pod opieką
dr. hab. Marcina Bieńkowskiego

2021

University of Wrocław
Faculty of Mathematics and Computer Science

The Price of Delays in Online Algorithms

Author:
Artur Kraska

Doctoral dissertation supervised by
dr hab. Marcin Bieńkowski

2021

Streszczenie

W niniejszej rozprawie rozważamy kilka klasycznych i dobrze znanych problemów online w nowym kontekście, w którym dopuszczamy opóźnioną obsługę żądań. W tym wariancie algorytm online nie musi podejmować decyzji od razu po pojawieniu się nowych żądań, lecz może odłożyć ich realizację, płacąc za to pewien zależny od opóźnienia koszt.

Pierwszym rozważanym w rozprawie problemem jest problem podróżującego mechanika (*traveling repairperson problem*). Skonstruowaliśmy algorytm, którego współczynnik konkurencyjności wynosi 4 w przypadku deterministycznym, a w przypadku randomizowanym $1 + 2/\ln 3 \approx 2.81$. Algorytm można łatwo rozszerzyć (zachowując jego konkurencyjność) do wariantu wyboru przejazdów (*dial-a-ride problem*) lub wariantu z wieloma mechanikami.

Powyższy algorytm udało się nam zaadaptować do — wydawałoby się zupełnie odmiennego — problemu szeregowania zadań na niepowiązanych maszynach, gdzie celem jest minimalizacja średniego czasu zakończenia zadania. W ten sposób otrzymaliśmy ulepszone współczynniki konkurencyjności dla tego problemu: 3 w przypadku deterministycznym i $1 + 1/\ln 2 \approx 2.443$ w przypadku randomizowanym.

Rozważyliśmy również problem obsługi z opóźnieniami (*online service with delay*). Przedstawiliśmy deterministyczny $O(\log n)$-konkurencyjny algorytm dla przypadku, gdy żądania mogą pojawiać się w n równoodległych punktach na linii.

Ostatnim rozważanym przez nas zagadnieniem jest problem skojarzeń z opóźnieniami (*online matching with delays*). Pokazaliśmy deterministyczny $O(m)$-konkurencyjny algorytm, gdzie $2m$ jest liczbą punktów w sekwencji wejściowej, które trzeba połączyć w pary. Nasz algorytm działa również w wariancie „dwudzielnym”, w którym każdy punkt jest albo dodatni albo ujemny i dopuszczalne jest łączenie tylko punktów o różnych znakach.

Abstract

We study well-known online problems through a modern lens of delayed decision-making. In this setting, an online algorithm may not react immediately once a request arrives, but postpone its service and instead pay an extra cost depending on the length of such delay.

First, we focus on the *traveling repairperson problem*. We create an algorithm whose competitive ratio is equal to 4 in the deterministic case and $1 + 2/\ln 3 \approx 2.81$ in the randomized one. Our algorithm can be easily extended (while maintaining its competitive ratios) to other variants of this problem, such as the dial-a-ride problem or a case with multiple repairpersons.

It turns out that the algorithm above can be tuned to solve a seemingly unrelated problem of *online machines scheduling* (with the goal of minimizing average completion time). This yields improved guarantees for this scheduling variant: 3 in the deterministic scenario and $1 + 1/\ln 2 \approx 2.443$ in the randomized one.

Next, we study the problem of *online service with delay*. We present a deterministic $O(\log n)$-competitive algorithm for the setting where requests may appear at n equidistant points of the real line.

Finally, we construct a deterministic algorithm for the problem of *online matching with delays*. The algorithm is $O(m)$-competitive, where $2m$ is the number of points that appear in the input and need to be connected into pairs. Our algorithm works also in the "bipartite" case, where each request has polarity and only points with different polarities may be connected together.

Contents

1 Introduction 11

1.1 Optimization Problems . . . 12

1.2 Online Algorithms . . . 12

1.3 Delays in Online Algorithms . . . 13

1.4 Problems Considered in This Thesis . . . 14

1.5 Linear Programming Techniques . . . 18

1.6 Biographical Notes . . . 21

2 Traveling Repairperson Problem and Unrelated Machines Scheduling 23

2.1 Introduction . . . 23

2.2 Resettable Scheduling . . . 26

2.3 Our Contribution . . . 28

2.4 Warmup: Algorithm Basic . . . 29

2.5 Deterministic and Randomized Algorithms: Routine Mimic . . . 34

2.6 Tightness of the Analysis . . . 48

2.7 Flaw in the Existing Randomized Lower Bound . . . 49

3 Online Service with Delay 51

3.1 Introduction . . . 51

3.2 Algorithm Bucket . . . 53

3.3 Competitiveness . . . 56

4 Online Matching with Delays 63

4.1 Introduction . . . 63

4.2 Primal-Dual Formulation . . . 67

4.3 Algorithm Greedy Dual 69
4.4 Correctness 72
4.5 Cost Analysis 74
4.6 Tightness of the Analysis 77
4.7 Alternative Deterministic Approaches 78

5 Afterword **81**

Chapter 1

Introduction

Optimization problems arise in natural manner in many areas of modern world, with transportation and logistics being prime examples. Big delivery companies need to deliver millions of parcels every day, dispatching thousands of trucks to travel an enormous number of kilometers. Another showcases are people transportation companies (e.g., Uber) that need to transport multiple clients to their destinations, using a group of available drivers. In both of these examples, companies can save a great amount of money, using algorithms that optimize assignments of tasks or people to drivers. Problems that need to be solved are *inherently online*: new clients or deliveries may appear while the company already dispatched some drivers. While routing decisions are usually irrevocable (e.g., it is unacceptable for a taxi to drop customers in the middle of the route), new requests can easily cause the currently executed solution to become very unoptimal.

From the practical point of view, some routing decisions do not have to be made immediately, and one can gain just by delaying them a bit and observing what happens in the nearest future. However, such delay may incur additional costs. This leads to a central assumption that we make in this thesis:

"Decisions are irrevocable, but can be delayed at a certain cost."

For example, in the Uber case, one can wait with dispatching the drivers to customers, gain knowledge about subsequent delivery requests, and find a better route. Such modification, however, changes the goal from simply minimizing the distance traveled by the drivers, to include also the minimization of customer waiting times. The waiting may make them less satisfied from the service, and in effect incur additional monetary costs.

This thesis focuses on the achievable trade-offs between the cost of service and the cost of waiting for many natural and previously studied online combinatorial problems, such as Traveling Repairperson Problem, Online Service with Delay, and Online Matching with Delays.

Before we describe particular problems we tackle in this thesis, we present our setup in a broader context.

1.1 Optimization Problems

Many optimization problems admit algorithms that compute an optimal solution (usually defined as a minimum-cost solution) and perform *polynomial number of operations*. That is, their running time is bounded by a polynomial function of the problem instance size. For example, if our goal is to find a perfect matching of a minimum cost in graph, there exist algorithms whose running times are polynomial in the number of graph vertices.

That said, for many problems, e.g., for Steiner Tree [WH16] and Set Cover [You16] (under the well-established hypothesis that computational classes $\mathbb{P}$ and $\mathbb{NP}$ are not equivalent), no polynomial algorithm can exist. In such cases, a natural approach is to seek a polynomial-time *approximation* algorithm, which for a given input computes a solution with a provably low cost. The ratio between the cost incurred by our algorithm and that of the optimal solution is called *approximation ratio* and is subject to minimization.

1.2 Online Algorithms

For many natural scenarios, the data given to an algorithm may not be readily available at the beginning, and new information may arrive during execution, *in online manner*. (The precise definition of how the data arrives is problem-dependent.) In the online setting, the approximation ratio is called *competitive ratio*. More precisely, we say that an online algorithm is γ-competitive if, for any possible input σ, its cost is at most γ times the cost of the best possible (offline) solution for σ, i.e., it holds that

$$\forall \sigma \quad \textsc{Alg}(\sigma) \leq \gamma \cdot \textsc{Opt}(\sigma)\,.$$

For a randomized algorithm $\textsc{Alg}$, we replace its cost by its expected value. The competitive ratio of $\textsc{Alg}$ is then the infimum over all values γ such that $\textsc{Alg}$ is γ-competitive [BE98].

While we use the notion above for all problems studied in this thesis, we note that the definitions in the literature (see, e.g., [BE98]) sometimes relax it by admitting an additional constant term on the right hand side.

Most classical optimization problems have their natural online counterparts. For example, in the online Steiner Tree problem [IW91], requests (terminals that need to be connected) arrive over time, and our goal is to buy appropriate edges, always keeping all terminals connected to the root. The online variant usually involves some irrevocability constraints: in the Steiner Tree case, edges can only be bought and cannot be sold back. Other well-studied examples include multiple adaptations of matching problems where edges or vertices of the graph appear online [Meh13].

If we restrict our attention to polynomial-time solutions, then (for appropriate definitions of online variants) the achievable competitive ratios can only be higher than the achievable approximation ratios. This is a trivial observation as an online algorithm generates a feasible solution to an offline problem as well. For instance, the currently best approximation ratio for the Steiner tree is below 1.39 [BGRS10], while, in the online variant, no polynomial-time algorithm can achieve a competitive ratio better than $\Omega(\log n)$ (where n is the number of terminal nodes) [IW91]. Interestingly however, the overwhelming majority of the existing lower bounds for online algorithms (including the above-mentioned one) hold even if we lift the polynomial-time requirement and assume online algorithms that have unbounded computational power. That is, for many online problems, the only source of hardness is the lack of knowledge about the future. Therefore, most of the research in this area focuses on the question what results can be achieved in the information-theoretic sense, and the computational complexity of online algorithms is only of secondary importance.

1.3 Delays in Online Algorithms

The usual setting, where the online algorithm has to make an immediate, irrevocable decision right after a request is presented has lead to many great algorithmic advancements. However, in many cases such setting does not reflect practical circumstances and leads to over-pessimistic results. Thus, over the course of time, many relaxations were introduced.

One of the simplest relaxations is to give a look-ahead to an online algorithm, allowing it to see not only the current request, but also a fixed number of subsequent ones. This is equivalent to having a buffer of fixed capacity that keeps most recent requests, and being forced to service only the oldest of them. Another buffer-related option is the buffer reordering model [RSW02], where an algorithm may choose which buffered request to service once the buffer is full.

The *delays* extension we consider in this thesis is based on a different concept. There, the requests arrive in time and an algorithm may service them at their arrival *or anytime* *afterwards*. Postponing the decision may allow an algorithm to learn subsequent requests, which in turn allows it to construct a better solution. Of course, such delays are not for free: each request arrives equipped with a non-decreasing *waiting function* that describes how much additional cost is incurred if it is pending for a given time.

We can distinguish two types of online problems that involve waiting of requests. We illustrate this distinction on an example where an algorithm has a server in some metric space (e.g., on a plane). In both cases, incoming requests are represented by points and in order to service them, the server needs to be moved to the requested point.

Time-based problems. In the first type of problems, *the actions of algorithm take a certain amount of time*. In our server example, the server moves with a fixed speed, so traveling to a new position takes an appropriate amount of time. In such problems, the amount that needs to be optimized is simply a function (e.g., a sum) of waiting times of requests. The described server problem is called the (online) Traveling Repairperson Problem [FS01].

Delay-augmented problems. In the second type of problems, *actions of an algorithm are immediate, but there is a certain cost associated with them*. In the server example, the server moves with an infinite speed, but it needs to pay for the traveled distance. Waiting of requests incur an additional cost, and the goal is to optimize the sum of service and waiting costs. This problem is called the Online Service with Delay [AGGP17].

We note that both types of server problems essentially boil down to deciding which requests are close enough and should be serviced right away and which are far and can be serviced later in a single batch. However, the employed techniques and difficulties that need to be overcome are surprisingly different among these two scenarios.

The time-based problems include, in particular, different kinds of machine scheduling problems, where jobs can be executed on a machine. Jobs can be executed at any time after their arrival and such execution takes some specified amount of time. On the other hand, the delay-augmented problems include various types of online request-answer games, such as online network design [AT20], online matching variants [EKW16] or online set cover [ACKT20].

1.4 Problems Considered in This Thesis

In this thesis, we study both variants of delayed online problems. We focus on time-based variants in Chapter 2, where we investigate two server problems: the Traveling

Repairperson Problem and the Dial-A-Ride Problem. Later, in Chapter 2, we show how similar approach can be applied to Unrelated Machines Scheduling. Then, we switch our attention to delay-augmented scenarios and study Online Service with Delay in Chapter 3 and Matching with Delays in Chapter 4.

Below we give a short overview of the studied problems, including the current state of art, and outlining the contribution of this thesis.

1.4.1 Traveling Repairperson Problem and Dial-A-Ride Problem

In the *Traveling Repairperson Problem* (TRP), there is a metric space with a distinguished point called origin. An algorithm has a server, initially located at the origin, that is able to move in the metric space with a constant speed. Requests arrive in online manner; each of them occupies a specified point of the metric and has a fixed weight. In order to service such request, an algorithm needs to move the server to the request position. The cost associated with a given request is the time that elapsed from the very beginning of the algorithm execution, multiplied by the weight of the request. The goal of the algorithm is to minimize the total cost of servicing all requests.

The *Dial-A-Ride Problem* (DARP) is an extension of the TRP. Here, every request can be thought of as delivery demand and is defined by its weight and a pair of metric points: a source and a destination. To service a request, an algorithm needs to move the server to its source point first, "pick up" a request, then move to its destination, at which time it is considered serviced. Similarly to the TRP, algorithm needs to minimize the total weighted service cost. There are multiple variations of this problem. For example, the server may have capacity, which allows it to pick up many requests before delivering them. If the capacity is finite, then one may study also a preemptive model, where the requests can be dropped and picked-up later, or non-preemptive one, where such dropping is not allowed. In both TRP and DARP problems, there are variants with precedence constraints, where some requests need to be serviced before others.

Till recently, the best online deterministic algorithm for the TRP and DARP variants, achieving the ratio of 5.14 was given by Hwang and Jaillet [HJ18]. They also showed how to randomize their solution to obtain a 3.641-competitive algorithm. In Chapter 2, we present algorithms that achieve a deterministic competitive ratio of 4 and a randomized competitive ratio of $1 + 2/\ln 3 \approx 2.81$, working for multiple variants of both problems.

A general approach used in our algorithm is to perform a delayed simulation of optimal solutions. That is, at time t, an online algorithm has a full knowledge about all requests that appeared till time t. Hence, it is able to compute a solution (a path in the metric space) for these requests that optimizes a certain function (closely resembling the actual optimization goal). Our algorithm then follows this pre-computed path and afterwards, at time t', it starts the next simulation.

In the analysis, we upper bound the competitive ratio of our algorithm by the objective value of a carefully crafted linear program (LP). This technique, called factor-revealing LP is described in more detail in Subsection 1.5.3.

1.4.2 Unrelated Machines Scheduling

A problem seemingly different from the TRP and the DARP is scheduling on m unrelated machines (denoted as $R|r_j|\sum w_j C_j$ in the Graham et al. notation [GLLK79]). There, weighted jobs arrive in time, each with a vector of size m describing execution times of the job when assigned to a given machine. A single machine can execute at most one job at a time. The goal is to assign each job (at or after its arrival) to one of the machines to minimize the weighted sum of completion times. This problem comes in two flavors: in the preemptive one, job execution may be interrupted and picked up later, while in the non-preemptive one, such interruption is not possible. As an extension, each job may have precedence constraints, i.e., can be executed only once some other jobs are completed.

So far, the best online algorithm for this problem was given by Hall et al. [HSSW97]. They gave an 8-competitive polynomial-time algorithm, which would be 4-competitive if the polynomial-time requirement was lifted. Chakrabarti et al. showed how to randomize this algorithm, achieving the ratio of $2/\ln 2 \approx 2.886$ [CPS+96]. They also observe that both algorithms can handle precedence constraints. The currently best deterministic lower bound of 1.309 is due to Vestjens [Ves97], and the best randomized one of 1.157 is due to Seiden [Sei00].

It turns out that it is possible to use our algorithmic framework for the TRP problem to also give improved bounds for the Unrelated Machines Scheduling problem. In fact, in Chapter 2, we present an algorithm for a slightly generalized problem that we call θ-resettable scheduling which contains variants of the TRP, the DARP, and the machine scheduling as specific instances. This yields a new bound of 3 on the deterministic competitive ratio of the Unrelated Machines Scheduling problem and $1 + 1/\ln 2 \approx 2.443$ for the randomized one.

1.4.3 Online Service with Delay

Similarly to the TRP problem, in the *Online Service with Delay* (OSD) problem, there is a server that starts at a point of a metric space called origin and during runtime, some points are requested and they have to be serviced eventually. As in the TRP problem, to service a request, an algorithm has to move its server to the requested point. In contrast to the TRP, such movement is immediate (takes no time). However, each request incurs a *waiting cost*, which is in this case a non-decreasing function of the delay between the time a request is issued and the time it is served by an algorithm. Another incurred cost is a *distance cost*, equal to the total distance traveled by the server. (Note that this type of cost was not present for the TRP.) The goal is to minimize the total cost, defined as the sum of distance cost and the waiting costs of all issued requests.

The results for this problem usually assumed that the metric space consists of finitely many points. The currently best solution, given by Azar et al. [AT19], is an $O(\log^2 n)$-competitive randomized algorithm for any metric space of n points. They use the fact that any n-point metric space on n points can be randomly approximated by a hierarchically separated tree (HST) of depth $O(\log n)$ with the expected distance distortion of $O(\log n)$ [BBMN15, FRT04] and provide a competitive algorithm for such trees.

In Chapter 3, we study the OSD problem on a line consisting of n equidistant points. We present a *deterministic* $O(\log n)$-competitive algorithm for this problem. Our algorithm combines ideas from the Moving Partition [GS09] and the batched service approach used for the OSD and MLA problems [AGGP17, BBB+16]. Namely, once our algorithm identifies an interval I of a line, such that the total waiting cost of requests pending in I is comparable to the cost of traveling to I, it decides to serve requests from I. However, a crucial insight for ensuring low competitive ratio is that our algorithm serves not only requests from I, but also proactively all requests from its surrounding. As we prove, this extra work significantly reduces the algorithm cost in the future.

1.4.4 Matching with Delays

In the *Min-cost Perfect Matching with Delays (MPMD)* problem [EKW16], $2m$ requests arrive in time at points in an n-point metric space. At any moment in time, an algorithm is able to connect a pair of requests that have already appeared. The cost of such connection is the metric distance between them plus the waiting costs of both of them. Waiting cost of a request is defined as a time that elapsed from its arrival to the time it became served (connected). The goal is to eventually connect all the requests into the pairs, minimizing the total cost.

The currently best solution for the MPMD problem is a *randomized* $O(\log n)$-competitive algorithm due to Azar et al. [ACK17], and the randomized lower bound of $\Omega(\log n / \log \log n)$ was given by Ashlagi et al. [AAC$^+$17].

The MPMD problem was also considered in a *bipartite* variant, called *Min-cost Bipartite Perfect Matching with Delays (MBPMD)* [AAC$^+$17]. There requests have polarities: one half of them is positive, and the other half is negative. An algorithm may match only requests of different signs. There, the achievable competitive ratios are between $\Omega\left(\sqrt{\log n / \log \log n}\right)$ [AAC$^+$17] and $O(\log n)$ [ACK17].

In Chapter 4, we study *deterministic* solutions both for the MPMD and MBPMD problems. We present a simple $O(m)$-competitive algorithm based on a primal-dual scheme that works in both settings; we highlight this scheme in Subsection 1.5.4. More concretely, we base our approach on the moat-growing framework, developed originally for (offline) constrained connectivity problems (e.g., for Steiner problems) by Goemans and Williamson [GW95].

Concurrently and independently of our result, Azar and Jacob-Fanani [AJF18] improved the deterministic ratio to roughly $O(m^{0.58})$ using a simple unbalanced greedy approach.

1.5 Linear Programming Techniques

The linear programming techniques proved useful in many algorithmic areas, both as a guide for algorithmic decisions and also as an analysis tool, for checking the algorithm effectiveness. These tools were commonly used in approximation algorithms, and in the last two decades these techniques paved their way also into online algorithms (see, e.g., the survey by Buchbinder and Naor [BN09]).

In this section, we outline some basic facts about linear programming, used later in Chapter 2 and Chapter 4. In particular, in Subsection 1.5.3, we sketch how to analyze an algorithm using *factor revealing* linear program. This technique is used in Chapter 2 to bound the competitive ratios of our algorithms. Next, in Subsection 1.5.4, we give a brief idea how linear programs can help in making decisions during an algorithm execution, presenting the *primal-dual* approach, used in the algorithm in Chapter 4.

1.5.1 Linear Program

An instance of a linear program (LP) consists of a linear objective function to be maximized (or minimized), subject to a set of linear inequalities (constraints) that must be fulfilled. More formally, the goal of a linear program is to *maximize* the expression

$$\sum_{i=1}^{n} c_i \cdot x_i \,,$$

subject to constraints

$$\begin{aligned} &\sum_{i=1}^{n} a_{ij} \cdot x_i \le b_j && \text{for all } j = 1, \ldots, m \,, \\ &x_i \ge 0 && \text{for all } i = 1, \ldots, n \,. \end{aligned}$$

There, x_i are real-valued variables that need to be set, and a_{ij}, b_j and c_i are constants. A set of values for variables x_i that satisfies the constraints is called a feasible solution.

While we do not use this property, it is worth mentioning that there exist algorithms that find an optimal solution to any LP instance in polynomial time (e.g., the ellipsoid method [GLS88]).

1.5.2 Dual Program

On the basis of any LP instance, one can construct (in an automated way) another LP instance, called a *dual program*. Its goal is to *minimize* the expression

$$\sum_{j=1}^{m} b_j \cdot y_j \,,$$

subject to constraints

$$\begin{aligned} &\sum_{j=1}^{m} a_{ij} \cdot y_j \ge c_i && \text{for all } i = 1, \ldots, n \,, \\ &y_j \ge 0 && \text{for all } j = 1, \ldots, m \,. \end{aligned}$$

where y_j are variables that need to be set. We call the former linear program the *primal* and the latter one the *dual*.

Note that the primal LP consists of n variables and m constraints, while the dual LP consists of m variables and n constraints. Furthermore, these LPs are linked by a simple but powerful relationship.

Theorem 1.1 (weak duality). *For any feasible solution $x_1, \ldots, x_n$ for the primal maximization LP and any feasible solution $y_1, \ldots, y_m$ for dual minimization LP, it holds that*

$$\sum_{i=1}^{n} c_i \cdot x_i \le \sum_{j=1}^{m} b_j \cdot y_j \,.$$

To show weak duality, one can use that all constraints in the primal and dual LPs have to hold, immediately obtaining that $\sum_{i=1}^{n} c_i \cdot x_i \leq \sum_{i=1}^{n} \sum_{j=1}^{m} a_{ij} \cdot y_j \cdot x_i = \sum_{j=1}^{m} \sum_{i=1}^{n} a_{ij} \cdot x_i \cdot y_j \leq \sum_{j=1}^{m} b_j \cdot y_j$. The weak duality relation allows to give an upper bound for all feasible solutions to the primal LP by finding *any* feasible solution to the dual LP. We use this in the analysis of our algorithm in Chapter 2.

1.5.3 Factor Revealing LPs

The factor revealing LP is a tool used for the algorithm analysis. It has been commonly used in approximation algorithms, for example for the facility location problem (e.g. in [JMM+03]). Examples where this technique was adapted to online algorithms are rather limited (see, e.g., [MY11] or [BBM17]).

The general idea behind this technique is quite simple. Once an algorithm is fixed, we create a maximization linear program (LP), whose variables describe all possible input instances and whose constraints correspond to guarantees provided by an algorithm. Furthermore, the objective function of the linear program is chosen to be an upper bound on the algorithm cost. Now, the worst-case cost of an algorithm is upper-bounded by the optimal solution to such maximization LP. (It is convenient to think that producing the worst possible instance corresponds to setting LP variables in a way that maximizes the objective value.)

It is rarely possible to describe the algorithm exactly by such LP: some algorithm properties may not be expressible by simple linear constraints or the LP objective may be only a rough estimate of the algorithm cost. Of course, the more precisely the LP describes the algorithm behavior, the better bound it gives. This issue is particularly problematic when the LP has to describe an online algorithm as in such case the input instance can be of arbitrary length.

Once a suitable LP is defined, its objective value can be found simply by running any LP solver. However, sometimes the algorithm performance can be only bounded by a whole family of linear programs. For example, in Chapter 2, we define an algorithm that executes Q phases, where Q is input-dependent. For any fixed Q we may write a factor-revealing LP, however running an LP solver for infinitely many possible values of Q is clearly impossible. Thus, we need to resort to a more analytical approach. That is, we obtain an upper bound for such maximization LPs by using weak duality (see Subsection 1.5.2) and finding *feasible* solutions to their dual programs.

The technique above can be used not only for upper-bounding algorithms cost, but also their competitive ratios. In Chapter 2, we use the factor revealing technique to give upper bounds for a whole group of online problems, including the TRP, the DARP, and unrelated machines scheduling, both for the deterministic and for the randomized setting.

1.5.4 Primal-Dual Technique

Another branch of LP applications includes cases where an LP is used in the algorithm definition. More precisely, the variables of the LP describe a current configuration of an algorithm and updating them is equivalent to actual choices of an algorithm. Besides maintaining an (initially infeasible and finally feasible) solution to the (primal) LP, an algorithm maintains also a feasible solution to the corresponding dual LP, hence the *primal-dual* name.

Such an algorithm usually updates simultaneously its solutions to both primal and dual LPs. The dual program plays the role of an oracle that gives the information how primal variables should be updated, i.e., what action an algorithm should perform. Usually, some carefully chosen set of dual variables is increased in a certain fashion as long as all the dual constraints are feasible. When it is no longer possible (some dual constraints becomes tight, i.e., hold with equality), an algorithm increases some primal variables and chooses another set of dual variables to increase.

Because of weak duality, this technique is a popular tool for designing efficient approximation algorithms: If only we ensure that the growth of the primal objective value is at most α times the growth of the dual objective, we are guaranteed that the created solution is an α-approximation.

This technique was successfully applied to (non-delayed) online computations. There, the setting is more complex as both primal and dual programs evolve (new variables and constraints appear) when new requests are presented to an algorithm. A particular framework, in which primal variables are increased at exponential pace in comparison to dual variables, gave rise to many online algorithms with polylogarithmic competitive ratios [BN09].

In Chapter 4, we show how to adapt the primal-dual framework to the problem of Min-cost Perfect Matching with Delays. We show that for the delayed problems, we may increase dual variables in a natural manner, together with the passing time.

1.6 Biographical Notes

All the results of the thesis were presented in a preliminary form at various conferences and published in the corresponding proceedings. Results for the Traveling

Repairperson Problem, the Dial-A-Ride Problem and the Unrelated Machines Scheduling problem presented in Chapter 2 appeared previously in the *48th International Colloquium on Automata, Languages, and Programming* (ICALP 2021) [BKL21]. This thesis extends results from the paper by the algorithm Basic that is a simplified version of the main algorithm for the TRP; we hope it will simplify the understanding of the whole result. The result for the Online Service with Delay on a line metric, given in Chapter 3, was first presented in the *25th International Colloquium on Structural Information and Communication Complexity* (SIROCCO 2018) [BKS18]. Finally, the results for Online Matching with Delays presented in Chapter 4, appeared first in the *16th International Workshop on Approximation and Online Algorithms* (WAOA 2018) [BKLS18].

Chapter 2

Traveling Repairperson Problem and Unrelated Machines Scheduling

2.1 Introduction

In this part, we present a unified framework for handling online scheduling problems, such as the Traveling Repairperson Problem (TRP), the Dial-a-Ride Problem (DARP) and the Unrelated Machines Scheduling, described in Subsection 1.4.1 and Subsection 1.4.2. We present an algorithm Mimic that yields substantially improved competitive ratios for all previously mentioned variants of these problems.

2.1.1 Previous Work

The currently best algorithms for the TRP, the DARP, and machine scheduling on unrelated machines share a common framework. Namely, each of these algorithms works in phases of geometrically increasing lengths. In each phase, it computes and executes an auxiliary schedule for the requests presented so far. (In the case of the TRP and the DARP, the server additionally returns to the origin afterward.) The auxiliary schedule optimizes a certain function, such as maximizing the weight of served requests [KdPPS03, KdPPS06, JW06, BL19, HSSW97, CPS+96] or minimizing the sum of completion times with an additional penalty for non-served requests [HJ18].[1] Moreover, known randomized algorithms are also based on a common idea: they delay the execution of the deterministic algorithm by a random offset [KdPPS03, KdPPS06,

[1]Computing such auxiliary schedule usually involves optimally solving an NP-hard task. This is quite common for online algorithms, where the focus is on information-theoretic aspects and not on computational complexity.

	deterministic		randomized	
	lower	upper	lower	upper
TRP	2.414 [FS01]	5.14 [HJ18]	2.333 [KdPPS03]	3.641 [HJ18]
DARP	3 [FS01]	5.14* [HJ18]	2.410 [KdPPS03]	3.641* [HJ18]
k-TRP	2 [BS09]	5.14* [HJ18]	2 [BS09]	3.641* [HJ18]
k-DARP	2 [BS09]	5.14* [HJ18]	2 [BS09]	3.641* [HJ18]
k-TRP, k-DARP		**4**		**2.821**
unrel. machines	1.309 [Ves97]	4 [HSSW97] **3**	1.157 [Sei00]	2.886 [CPS+96] **2.443**

Table 2.1: Previous and current bounds on the competitive ratios for the TRP and the DARP problems. Asterisked results were not given in the referenced papers, but they are immediate consequences of the arguments therein. All upper bounds for the TRP/DARP variants hold for any number k of servers, any server capacities, both in the preemptive and the non-preemptive case. Upper bounds for unrelated machines scheduling hold also in the presence of precedence constraints. Bounds proven in this thesis are given in boldface.

CPS+96, HJ18]. We call these approaches *phase based*. The currently best results are gathered in Table 2.1; we discuss their history below.

Traveling Repairperson and Dial-a-Ride Problems. The online variant of the TRP has been first investigated by Feuerstein and Stougie [FS01]. By adapting an algorithm for the so-called cow-path problem [BYCR93], they gave a 9-competitive solution for line metrics. The result has been improved by Krumke et al. [KdPPS03], who gave a phase-based deterministic algorithm Interval attaining competitive ratio of $3 + 2\sqrt{2} < 5.829$ for an arbitrary metric space. A slightly different algorithm with the same competitive ratio was given by Jaillet and Wagner [JW06]. Bienkowski and Liu [BL19] applied postprocessing to auxiliary schedules, serving heavier requests earlier, and improved the ratio to 5.429 on line metrics. Finally, Hwang and Jaillet proposed a phase-based algorithm Plan-And-Commit [HJ18]. They give a computer-based upper bound of 5.14 for the competitive ratio and an analytical upper bound of 5.572.

Randomized counterparts of algorithms Interval and Plan-And-Commit achieve ratios of 3.874 [KdPPS03, KdPPS06] and 3.641 [HJ18], respectively. Interestingly, the latter bound is not a direct randomization of the deterministic algorithm, but uses a different parameterization, putting more emphasis on penalizing requests not served by auxiliary schedules.

The phase-based algorithm Interval extends in a straightforward fashion to the DARP problem with an arbitrary assumption on the server capacity, both for the preemptive and non-preemptive variants: all the details of the solved problem are encapsulated in the computations of auxiliary schedules [KdPPS03]. In the same manner, Interval can be enhanced to handle k-TRP and k-DARP variants, where an algorithm has k servers at its disposal (also for any k, any server capacities, and any preemptiveness assumptions) [BS09]. Although this was not explicitly stated in [HJ18], the algorithm Plan-And-Commit can be extended in the same way.

From the impossibility side, Feuerstein and Stougie [FS01] gave a lower bound for the TRP (that also holds already for a line) of $1 + \sqrt{2} > 2.414$, while the bound of $7/3$ for randomized algorithms was presented by Krumke et al. [KdPPS03]. For the variant of the TRP with multiple servers, the deterministic lower bound is only 2 [BS09] (it holds for any number of servers). Clearly, all these lower bounds hold also for any variant of the DARP problem. For the DARP with a single server of capacity 1, the deterministic lower bound can be improved to 3 [FS01] and the randomized one to 2.410 [KdPPS03].

The authors of [FKW09] claimed a lower bound of 3 for randomized k-DARP (for any k). This contradicts the upper bound of 2.821 we present in this chapter. In Section 2.7, we pinpoint a flaw in their argument.

Unrelated Machines Scheduling Problem. As already mentioned in Subsection 1.4.2, the currently best online algorithms for this problem have been 4-competitive for the deterministic case [HSSW97] and $2/\ln 2 \approx 2.886$-competitive for the randomized one [CPS+96]. Both of them work for multiple problem variants (e.g., with or without preemption, with precedence constraints). The best deterministic lower bound is 1.309 [Ves97], and the best randomized one is 1.157 [Sei00].

2.1.2 Related Results

To put the work in a broader context, we briefly discuss also related results on the TRP, the DARP, and machine scheduling.

Traveling Repairperson and Dial-a-Ride Problems. Both the online TRP and DARP problems were considered under different objectives, such as minimizing the total makespan (when the TRP becomes the online Traveling Salesperson Problem (TSP)) [AKR00, AFL+94, AFL+95, AFL+01, BD20, BDS19, BDH+17, BKdPS01, CDLW19, JMS19, JW08, LLdP+04] or maximum flow time [HKR00, KdPP+05, KLL+02].

The offline variants of the TRP and the DARP have been extensively studied both from the computational hardness (see, e.g., [SG76, dPLS+04]) and approximation algorithms perspectives. In particular, the TRP, also known as the *minimum latency problem* problem, is NP-hard already on weighted trees [Sit02] (where the closely related TSP problem [Blä16] becomes trivial) and the best known approximation factor in general graphs is 3.59 [CGRT03]. For some metrics (Euclidean plane, planar graphs or weighted trees), the TRP admits a PTAS [AK03, Sit14].

Machine Scheduling. While for unrelated machines, the results described above have not been beaten for the last 25 years, the competitive ratios for simpler models were improved substantially. For example, for parallel identical machines, a sequence of papers lowered the ratio to 1.791 [CW09, SS02, MS04, Sit10].

The Unrelated Machines Scheduling problem has also been studied intensively in the offline regime. Both weighted preemptive and non-preemptive variants were shown to be APX-hard [HSW01, Sit17]. On the positive side, a 1.698-approximation for the preemptive case was given by Sitters [Sit17], and a 1.5-approximation for the non-preemptive case by Skutella [Sku98]. A PTAS for a constant number of machines is due to Afrati et al. [ABC+99].

2.2 Resettable Scheduling

The phase-based algorithms for DARP variants (including the TRP problem) and machine scheduling on unrelated machines both execute auxiliary schedules, but the ones for the DARP variants need to bring the server back to the origin between schedules. We call the latter action *resetting*. To provide a single algorithm for all these problems, we define a class of *resettable scheduling* problems.

In these problems, we assume that jobs are handled by an *executor*, which has a set of possible states. At time 0, it is in a distinguished *initial state*. An input to the problem consists of a sequence of jobs $\mathcal{I}$ released over time. Each job r is characterized by its arrival time $a(r)$, its weight $w(r)$, and possibly other parameters that determine its execution time. We extend the definition of weight w to any request set R in a natural way, i.e., $w(R) = \sum_{r \in R} w(r)$. The executor cannot start executing job r before its arrival time $a(r)$. We will slightly abuse the notation and use $\mathcal{I}$ to also denote the *set* of all jobs from the input sequence. There is a problem-specific way of executing jobs, and we use $s_{\text{ALG}}(r)$ to denote the *completion time* of a job by an algorithm ALG. The cost of ALG is defined as the weighted sum of job completion times, $\text{COST}_{\text{ALG}}(\mathcal{I}) = \sum_{r \in \mathcal{I}} w(r) \cdot s_{\text{ALG}}(r)$.

For any time τ, let $\mathcal{I}_\tau$ be the set of jobs that appear till τ. An *auxiliary τ-schedule* is a problem-specific way of feasibly executing a subset of jobs from $\mathcal{I}_\tau$. Such schedule starts at time 0, terminates at time τ, and leaves no job partially executed. We require that the following properties hold for any resettable scheduling problem; later we argue that they are satisfied for the DARP variants and the Unrelated Machines Scheduling problem.

Delayed execution. At any time t, if the executor is in the initial state, it can execute an arbitrary auxiliary τ-schedule (for $\tau \leq t$). Such action takes place in time interval $[t, t+\tau)$. Any job r that would be completed at time $z \in [0, \tau)$ by the τ-schedule started at time 0 is now completed at time $t + z$ (unless it has been already completed before).

Resetting executor. Assume that at time t, the executor was in the initial state, and then executed a τ-schedule, ending at time $t + \tau$. Then, it is possible to *reset* the executor using extra $\gamma \cdot \tau$ time, where γ is a parameter characteristic to the problem. That is, at time $t + (1+\gamma) \cdot \tau$, the executor is again in its initial state.

Learning minimum. We define $\min(\mathcal{I})$ to be the earliest time at which OPT may complete some job. We require that the value of $\min(\mathcal{I})$ is learned by an online algorithm at or before time $\min(\mathcal{I})$, and that $\min(\mathcal{I}) > 0$.

We call scheduling problems that obey these restrictions γ-resettable.

Example 1: Unrelated Machines Scheduling is 0-Resettable. For the machine scheduling problem, the executor is always in the initial state, and no resetting is necessary. As we may assume that processing of any job takes positive time, $\min(\mathcal{I}) > 0$ holds for any input $\mathcal{I}$.

Example 2 : DARP Problems are 1-Resettable. For the DARP variants, the executor state is the position of the algorithm server, with the origin used as the initial state.[2] Jobs are requests for transporting objects and an auxiliary τ-schedule is a fixed path of length τ starting at the origin, augmented with actions of picking up and dropping particular objects. (In the preemptive variants, preemption is allowed *inside* an auxiliary schedule, provided that after a τ-schedule terminates, each job is either completed or untouched.) It is feasible to execute a τ-schedule starting at any time t when the server is at the origin. In such case, jobs are completed with an extra delay of t. Furthermore, right after serving the τ-schedule, the distance between the server

[2]In the variants with k servers, the executor state is a k-tuple describing the positions of all servers.

and the origin is at most τ. Thus, it is possible to reset the executor to the initial state within extra time $1 \cdot \tau$.

Finally, as we may assume that there are no requests that arrive at time 0 with both start and destination at the origin, $\min(\mathcal{I}) > 0$ for any input $\mathcal{I}$.

2.3 Our Contribution

As a warm-up, we present a deterministic algorithm Basic, that solves 1-resettable scheduling problems and achieves a competitive ratio 6. Then, we provide a more general deterministic routine Mimic and its randomized version that solves any γ-resettable scheduling problem. It achieves a deterministic ratio of $3 + \gamma$ and a randomized one of $1 + (1 + \gamma)/\ln(2 + \gamma)$.

That is, for 1-resettable scheduling problems (the DARP variants with arbitrary server capacity, an arbitrary number of servers, and both in the preemptive and non-preemptive setting, or the TRP problem with an arbitrary number of servers), this gives solutions whose ratios are at most 4 and $1 + 2/\ln 3 < 2.821$, respectively. For 0-resettable scheduling problems (that include scheduling on unrelated machines with or without precedence constraints), the ratios of our solutions are 3 and $1 + 1/\ln 2 < 2.443$.

In both cases, our results constitute a substantial improvement over currently best ratios as illustrated in Table 2.1. Our result for the scheduling on unrelated machines is the first improvement in the last 25 years for this problem.

Challenges and Techniques. Basic and Mimic work in phases of geometrically increasing lengths. At the beginning of each phase, at time τ, they compute an auxiliary τ-schedule that optimizes the total completion time of jobs seen so far with an additional penalty for non-completed jobs: they are penalized as if they were completed by schedules at time τ. Then, within the phase they execute these schedules and afterward they reset the executor. We obtain a randomized variant by delaying the start of Mimic by an offset randomly chosen from a *continuous* distribution.

Admittedly, this idea is not new, and in fact, when we apply Mimic to the TRP problem, it becomes a slightly modified variant of Plan-And-Commit [HJ18]. Hence, the main technical contribution of this chapter is a careful and exact analysis of such an approach. The crux here is to observe several structural properties and relations among schedules produced by Mimic in consecutive phases, carefully tracking the overlaps of the job sets completed by them. On this basis, and for a fixed number Q of phases, we construct a maximization linear program (LP), whose optimal value upper-bounds the competitive ratio of Mimic. Roughly speaking, the LP encodes, in a sparse manner, an adversarially created input. To upper bound its value, we explicitly construct a solution to its dual (minimization) program and show that its value is at most 4 for any number of phases Q.

Bounding the competitive ratio for the randomized version of Mimic is substantially more complicated as we need to combine the discrete world of an LP with uncountably many random choices of the algorithm. To tackle this issue, we consider an intermediate solution Disc which approximates the random choice of Mimic to a given precision, choosing an offset randomly from a discrete set of M values. This way, we upper-bound the ratio of Mimic by $1 + (1/M) \cdot \sum_{j=1}^{M} (2+\gamma)^{j/M}$. This bound holds for an arbitrary value of M, and thus by taking the limit, we obtain the desired bound on the competitive ratio. Interestingly, we use the same LP for analyzing both the deterministic and the randomized solution.

2.4 Warmup: Algorithm Basic

As an illustration of our techniques, we start with an algorithm Basic that solves 1-resettable scheduling problems. We present a simple analysis proving that it is 6-competitive. In the subsequent sections, we will extend this idea to achieve a 4-competitive deterministic algorithm Mimic and its 2.82-competitive randomization. To make the remaining part of this chapter self-contained, some of definitions presented here will be defined once again later.

Algorithm Basic follows the general phase-based framework described in Subsection 2.1.1. In this section, without loss of generality, we assume that $\min(\mathcal{I}) = 1$. This property can be easily achieved by an appropriate scaling. (Note that the algorithm learns the value of $\min(\mathcal{I})$ before it needs to use it for the first time.)

Let $\alpha = 3$. For any integer $k \geq 1$, at time α^k, Basic computes and executes an auxiliary α^k-schedule A_k (i.e., a schedule of duration α^k) that we will define below. Clearly it ends executing this schedule at time $2 \cdot \alpha^k$, and afterwards it resets the executor to the initial state; this operation terminates latest at time $3 \cdot \alpha^k = \alpha^{k+1}$.

We call the time interval $[\alpha^{k-1}, \alpha^k]$ *the k-th phase*. To complete the definition of Basic, for any α^k-schedule A, we define its value as

$$\textsc{val}_k^{\textsc{Basic}}(A) = \sum_{j=1}^{k} \alpha^{j-1} \cdot w\left(R_j(A)\right) + \alpha^k \cdot w\left(\mathcal{I}_{\alpha^k} \setminus R(A)\right),$$

where $R(A)$ represents the set of all requests served by A and $R_j(A)$ represents the subset of requests served by A during the j-th phase. $\textsc{val}_k^{\textsc{Basic}}(A)$ is closely related to an actual cost of an algorithm that would follow A in time interval $[0, \alpha^k]$. To see that, first note that no request can be served by A before the first phase, i.e., before $\alpha^0 = 1 = \min(\mathcal{I})$. Second, note that in the definition of $\textsc{val}_k^{\textsc{Basic}}(A)$, we charge A for the requests it serves in phase j as if they were served at the beginning of the phase j, and for the unserved requests as if they were served at the end of schedule A. Therefore, if all requests are served by A, then $\textsc{val}_k^{\textsc{Basic}}(A)$ is a lower bound on the actual cost of A executed in the interval $[0, \alpha^k]$.

The auxiliary α^k-schedule A_k used in algorithm Basic is the schedule that minimizes $\textsc{val}_k^{\textsc{Basic}}$ (with ties broken arbitrarily). An illustration of an example execution is given in Figure 2.1 on page 36.

2.4.1 Relating Basic to Opt

We fix any input $\mathcal{I}$. Let T be the set of all integers k, such that $R(A_k) = \mathcal{I}$, i.e., the auxiliary α^k-schedule used by Basic serves all requests. The following lemma shows that Basic eventually terminates.

Lemma 2.1. *For any input $\mathcal{I}$, the corresponding set T is non-empty.*

Proof. By scaling, we may assume that the weight of each request is at least 1. For a fixed $\mathcal{I}$, there exists an integer ℓ and an auxiliary α^ℓ-schedule B that serves all requests from $\mathcal{I}$. (Note that it does not imply that the schedule that minimizes $\textsc{val}_\ell^{\textsc{Basic}}$ serves all requests.)

Choose $k \geq \ell$ sufficiently large, so that $\alpha^k > \textsc{val}_\ell^{\textsc{Basic}}(B)$. As $k \geq \ell$, B is also an auxiliary α^k-schedule and $\textsc{val}_k^{\textsc{Basic}}(B) = \textsc{val}_\ell^{\textsc{Basic}}(B)$. For any auxiliary α^k-schedule C not serving all requests, it holds that

$$\textsc{val}_k^{\textsc{Basic}}(C) \geq \alpha^k > \textsc{val}_\ell^{\textsc{Basic}}(B) = \textsc{val}_k^{\textsc{Basic}}(B),$$

where the first relation follows by the second summand of $\textsc{val}_k^{\textsc{Basic}}$. Thus, the schedule minimizing $\textsc{val}_k^{\textsc{Basic}}$ has to serve all requests, and therefore T is non-empty. □

In these terms, Basic terminates in phase $\min(T)$. Note that Opt may terminate in a different phase, also in a phase not from T.

We proceed with definitions and structural properties that will be used in Basic. (We will reuse these definitions for Mimic, reminding them in the next section.) For any k, we partition $R(A_k)$ into two disjoint parts:

- the set of *fresh requests* $R^{\mathrm{F}}(A_k) = R(A_k) \setminus \bigcup_{\ell=1}^{k-1} R(A_\ell)$,

- the set of *stale requests* $R^{\mathrm{S}}(A_k) = R(A_k) \cap \bigcup_{\ell=1}^{k-1} R(A_\ell) = R(A_k) \setminus R^{\mathrm{F}}(A_k)$.

In other words, $R(A_k)$ is the set of requests that would be served by A_k if it was started at time 0. However, in the actual runtime of Basic, schedule A_k is executed starting at time α^k, when some of the requests from $R(A_k)$ may have already been served by Basic. That is, fresh requests from $R^{\mathrm{F}}(A_k)$ are served by Basic in the $(k+1)$-th phase, while stale requests from $R^{\mathrm{S}}(A_k)$ have been already served in some previous phases. Note that $R(A_1)$ has no stale requests.

Lemma 2.2. *Fix any $t \in T$. For any $k \in \{1, \ldots, t\}$, it holds that $R^{\mathrm{S}}(A_k) \subseteq \bigcup_{\ell=1}^{k-1} R^{\mathrm{F}}(A_\ell)$. For $k = t$, the inclusion can be replaced by equality.*

Proof. By a simple induction, it can be shown that for any $k \geq 1$, it holds that $\bigcup_{\ell=1}^{k-1} R^{\mathrm{F}}(A_\ell) = \bigcup_{\ell=1}^{k-1} R(A_\ell)$. Since $R^{\mathrm{S}}(A_k) \subseteq \bigcup_{\ell=1}^{k-1} R(A_\ell)$, the first part of the lemma follows.

As A_t serves all the requests, $\bigcup_{\ell=1}^{t-1} R(A_\ell) \subseteq R(A_t) = R^{\mathrm{F}}(A_t) \uplus R^{\mathrm{S}}(A_t)$. By the definition of fresh requests, $R^{\mathrm{F}}(A_t)$ does not contain any request from $\bigcup_{\ell=1}^{t-1} R(A_\ell)$. Thus, $\bigcup_{\ell=1}^{t-1} R(A_\ell) \subseteq R^{\mathrm{S}}(A_t)$, and hence $\bigcup_{\ell=1}^{t-1} R^{\mathrm{F}}(A_\ell) = \bigcup_{\ell=1}^{t-1} R(A_\ell) \subseteq R^{\mathrm{S}}(A_t)$. □

For any A_k, sets $R^{\mathrm{F}}(A_k)$ and $R^{\mathrm{S}}(A_k)$ are subsets of $R(A_k)$. Therefore, similarly to partitioning of the set $R(A_k)$ into sets $R_j(A_k)$, both fresh and stale requests sets can be partitioned into requests served in particular phases: $R^{\mathrm{F}}(A_k) = \biguplus_{j=1}^{k} R_j^{\mathrm{F}}(A_k)$ and $R^{\mathrm{S}}(A_k) = \biguplus_{j=1}^{k} R_j^{\mathrm{S}}(A_k)$. For succinctness, we introduce the following shorthand notations: $w_{kj} = w(R_j(A_k))$, $w_{kj}^{\mathrm{F}} = w(R_j^{\mathrm{F}}(A_k))$ and $w_{kj}^{\mathrm{S}} = w(R_j^{\mathrm{S}}(A_k))$. We may now use these values to estimate both costs of Opt and Basic.

Lemma 2.3. *Fix any input $\mathcal{I}$, the corresponding auxiliary schedules $\{A_k\}_{k \geq 1}$ and set T. Let $K = \arg\min_{k \in T}\{\mathrm{VAL}_k^{\mathrm{BASIC}}(A_k)\}$. Then, the following inequalities hold.*

$$\textsc{Opt}(\mathcal{I}) \geq \sum_{j=1}^{K} \alpha^{j-1} \cdot w_{Kj} \tag{2.1}$$

$$\textsc{Basic}(\mathcal{I}) \leq \sum_{k=1}^{K} \sum_{j=1}^{k} (\alpha^k + \alpha^j) \cdot w_{kj}^{\mathrm{F}} \tag{2.2}$$

$$\sum_{j=1}^{\ell} w_{\ell j}^{\mathrm{S}} \leq \sum_{k=1}^{\ell-1} \sum_{j=1}^{k} w_{kj}^{\mathrm{F}} \qquad \textit{for all } 1 \leq \ell \leq K \tag{2.3}$$

$$\sum_{k=1}^{K-1} \sum_{j=1}^{k} w_{kj}^{\mathrm{F}} = \sum_{j=1}^{K} w_{Kj}^{\mathrm{S}} \tag{2.4}$$

$$\sum_{j=1}^{k} (\alpha^{j-1} - \alpha^k) \cdot w_{kj} \leq \sum_{j=1}^{k} (\alpha^{j-1} - \alpha^k) \cdot w_{\ell j} \qquad \textit{for all } 1 \leq k < \ell \leq K \tag{2.5}$$

Proof. Let B be a schedule used by OPT and let $\ell(B)$ be the phase, in which B terminates. Then, $\text{OPT}(\mathcal{I}) \geq \text{VAL}^{\text{BASIC}}_{\ell(B)}(B) \geq \text{VAL}^{\text{BASIC}}_{\ell(B)}(A_{\ell(B)}) \geq \text{VAL}^{\text{BASIC}}_{K}(A_K)$. As $K \in T$, $R(A_K) = \mathcal{I}$, and thus $\text{VAL}^{\text{BASIC}}_{K}(A_K) = \sum_{j=1}^{K} \alpha^{j-1} \cdot w_{Kj}$, which implies (2.1).

On the other hand, BASIC terminates latest in phase $K+1$. In phase $k+1$ (for $k \in \{1,\ldots,K\}$), it pays only for serving fresh requests from A_k (as stale requests from A_k were served in some previous phase of BASIC). Moreover, requests from $R^{\text{F}}_j(A_k)$ are served by BASIC in the time interval $[\alpha^k + \alpha^{j-1}, \alpha^k + \alpha^j]$, i.e., latest at time $\alpha^k + \alpha^j$. This implies (2.2).

Relations (2.3) and (2.4) follow immediately by noting that $K \in T$ and applying weights to the relations guaranteed by Lemma 2.2.

Finally, to show (2.5), we fix any k and ℓ such that $1 \leq k < \ell \leq K$. Recall that schedule A_k minimizes the value of function $\text{VAL}^{\text{BASIC}}_{k}$ among all auxiliary α^k-schedules. Let $\tilde{A}_\ell$ be the auxiliary schedule consisting of the first k phases of schedule A_ℓ. Then,

$$\text{VAL}^{\text{BASIC}}_{k}(A_k) = \sum_{j=1}^{k} \alpha^{j-1} \cdot w_{kj} + \alpha^k \cdot \left(w(\mathcal{I}_{\alpha^k}) - \sum_{j=1}^{k} w_{kj} \right) \quad \text{and}$$

$$\text{VAL}^{\text{BASIC}}_{k}(\tilde{A}_\ell) = \sum_{j=1}^{k} \alpha^{j-1} \cdot w_{\ell j} + \alpha^k \cdot \left(w(\mathcal{I}_{\alpha^k}) - \sum_{j=1}^{k} w_{\ell j} \right) .$$

As A_k is a minimizer of $\text{VAL}^{\text{BASIC}}_{k}$, we obtain $\text{VAL}^{\text{BASIC}}_{k}(A_k) \leq \text{VAL}^{\text{BASIC}}_{k}(\tilde{A}_\ell)$, and thus $\sum_{j=1}^{k} (\alpha^{j-1} - \alpha^k) \cdot w_{kj} \leq \sum_{j=1}^{k} (\alpha^{j-1} - \alpha^k) \cdot w_{\ell j}$ for all $1 \leq k < \ell \leq K$. □

2.4.2 Bounding the Competitive Ratio

Now we take the relations (2.1)–(2.5) and perform the following operations.

- Take (2.1) and multiply it by 2α.
- Take (2.2) as it is.
- Take (2.3) for a fixed ℓ and multiply both sides by $2\alpha^{\ell+1} - 2\alpha^{\ell}$. Add all these inequalities for $\ell \in \{1,\ldots,K-1\}$.
- Take (2.4) as an inequality and multiply both sides by $2\alpha^K$.
- Take (2.5) for a fixed $k \in \{1,\ldots,K-1\}$ and $\ell = k+1$, then sum all these inequalities with both sides multiplied by 2α. Futhermore, we replace w_{kj} by $w^{\text{F}}_{kj} + w^{\text{S}}_{kj}$.

The resulting inequalities are as follows.

$$\begin{aligned}
\sum_{j=1}^{K} 2\alpha^j \cdot w_{Kj} &\leq 2\alpha \cdot \textsc{Opt}(\mathcal{I}) \\
\textsc{Basic}(\mathcal{I}) &\leq \sum_{k=1}^{K}\sum_{j=1}^{k}(\alpha^k + \alpha^j) \cdot w^{\mathrm{F}}_{kj} \\
\sum_{\ell=1}^{K-1}\sum_{j=1}^{\ell}(2\alpha^{\ell+1} - 2\alpha^{\ell}) \cdot w^{\mathrm{S}}_{\ell j} &\leq \sum_{\ell=1}^{K-1}\sum_{k=1}^{\ell-1}\sum_{j=1}^{k}(2\alpha^{\ell+1} - 2\alpha^{\ell}) \cdot w^{\mathrm{F}}_{kj} \\
&= \sum_{k=1}^{K-1}\sum_{j=1}^{k}(2\alpha^{K} - 2\alpha^{k+1}) \cdot w^{\mathrm{F}}_{kj} \\
\sum_{k=1}^{K-1}\sum_{j=1}^{k} 2\alpha^K \cdot w^{\mathrm{F}}_{kj} &\leq \sum_{j=1}^{K} 2\alpha^K \cdot w^{\mathrm{S}}_{Kj} \\
\sum_{k=1}^{K-1}\sum_{j=1}^{k}(2\alpha^{k+1} - 2\alpha^j) \cdot (w^{\mathrm{F}}_{k+1,j} + w^{\mathrm{S}}_{k+1,j}) &\leq \sum_{k=1}^{K-1}\sum_{j=1}^{k}(2\alpha^{k+1} - 2\alpha^j) \cdot (w^{\mathrm{F}}_{kj} + w^{\mathrm{S}}_{kj})
\end{aligned}$$

Adding these inequalities together and moving all the summands except $2\alpha \cdot \textsc{Opt}(\mathcal{I})$ to the left side, we can observe that all the coefficients at variables w^{F}_{kj} and w^{S}_{kj} are non-negative. (To this end, we use that in all these coefficients, it holds that $k \geq j$, and thus also $\alpha^k \geq \alpha^j$.) This yields the relation $\textsc{Basic}(\mathcal{I}) \leq 2\alpha \cdot \textsc{Opt}(\mathcal{I})$, so the competitive ratio of Basic is at most $2 \cdot \alpha = 6$.

2.4.3 Relation to Factor-Revealing LP

In the previous section, we show that if the relations (2.1)–(2.5) are satisfied for arbitrary valuation of weights (variables w^{F}_{kj} and w^{S}_{kj}), then together they imply that $\textsc{Basic}(\mathcal{I}) \leq 2\alpha \cdot \textsc{Opt}(\mathcal{I})$. Equivalently, we could add an extra constraint stating that $\textsc{Opt}(\mathcal{I}) \leq 1$ and show that $\textsc{Basic}(\mathcal{I}) \leq 2\alpha$. As weights can be scaled without changing the validity of remaining constraints, this would lead to $\textsc{Basic}(\mathcal{I}) \leq 2\alpha \cdot \textsc{Opt}(\mathcal{I})$.

In other words, our goal was to show that for any values of variables w^{F}_{kj} and w^{S}_{kj}, as long as they satisfy constraints (2.1)–(2.5) and the constraint $\textsc{Opt}(\mathcal{I}) \leq 1$, the value of $\textsc{Basic}(\mathcal{I})$ can be upper-bounded by 2α. In turn, this is equivalent to analyzing an LP, whose goal is to maximize the value of $\textsc{Basic}(\mathcal{I})$, subject to constraints (2.1)–(2.5) and $\textsc{Opt}(\mathcal{I}) \leq 1$: if we could show that the optimal value of such LP is at most 2α, then we would obtain the desired upper bound on $\textsc{Basic}(\mathcal{I})$.

Reducing the proof to bounding the value of such maximization LP allows us to utilize the power of linear programming and approach the problem in a more systematic way. First, the coefficients by which we multiplied the inequalities in the previous section are in fact dual variables in some feasible solution to the dual program: there is no need of guessing them. Second, by weak duality (cf. Theorem 1.1 in Subsection 1.5.2), the value of such maximization LP can be upper-bounded by the value of any feasible solution to the dual program. This property will be used in upper-bounding the competitive ratio of Mimic in subsequent sections.

2.5 Deterministic and Randomized Algorithms: Routine Mimic

To describe our approach for γ-resettable scheduling, we start with defining auxiliary schedules used by our routine Mimic. The parameter γ will be used to define partitioning of time into phases. Both our deterministic and randomized solutions will run Mimic, however, the randomized one will execute it for a random choice of parameters.

Auxiliary Schedules. As introduced already in Section 2.2, an (auxiliary) τ-schedule A describes a sequence of job executions, has the total duration τ, and may be executed whenever the executor is in the initial state. For the preemptive variants, we require that once such a schedule terminates, each job is processed either completely or not at all.

For a fixed input $\mathcal{I}$, and a τ-schedule A, we use $R(A)$ to denote the set of jobs that would be served by A if it was executed from time 0, i.e., in the interval $[0, \tau)$. For any set of jobs $R \subseteq R(A)$, let

$$\textsc{cost}_A(R) = \textstyle\sum_{r \in R} w(r) \cdot s_A(r). \tag{2.6}$$

Note that if a schedule A serves all jobs from the input ($R(A) = \mathcal{I}$), then $\textsc{cost}_A(R(A))$ coincides with the cost of an algorithm that executes schedule A at time 0.

Recall that $\mathcal{I}_\tau \subseteq \mathcal{I}$ denotes the set of jobs that arrive till time τ. For any τ-schedule A, we define its value as

$$\textsc{val}_\tau(A) = \textsc{cost}_A(R(A)) + \tau \cdot w\left(\mathcal{I}_\tau \setminus R(A)\right). \tag{2.7}$$

The value corresponds to the actual cost of completing jobs from $\mathcal{I}_\tau$ by schedule A in interval $[0, \tau)$, but we charge A for unprocessed jobs as if they were completed at time τ.

Definition 2.1. *For any $\tau \geq 0$, let S_τ be the τ-schedule minimizing function $\textsc{val}_\tau$. Ties are broken arbitrarily, but in a deterministic fashion.*

Routine MIMIC. For solving the γ-resettable scheduling problem, we define routine Mimic(γ, ω), where $\omega \in (-1, 0]$ is an additional parameter that controls the initial delay.

- Our deterministic algorithm is simply Mimic$(\gamma, 0)$.
- Our randomized algorithm first chooses a value ω uniformly at random from the range $(-1, 0]$. Then, it executes Mimic(γ, ω).

Internally, Mimic(γ, ω) uses a parameter $\alpha = 2 + \gamma$. It splits time into phases in the following way. For any k, let $\tau_k = \tau(k) = \min(\mathcal{I}) \cdot \alpha^{k+\omega}$. The k-th phase (for $k \geq 1$) starts at time $\tau_{k-1} = \min(\mathcal{I}) \cdot \alpha^{k-1+\omega}$ and ends at time $\tau_k = \min(\mathcal{I}) \cdot \alpha^{k+\omega}$. The time interval $[0, \tau_0) = [0, \alpha^{\omega} \cdot \min(\mathcal{I}))$ does not belong to any phase. As $\alpha^{\omega} \cdot \min(\mathcal{I}) \leq \min(\mathcal{I})$, no jobs can be completed within this interval, by the definition of $\min(\mathcal{I})$ (see Section 2.2).

Mimic does nothing till the end of phase 1 (till time $\tau_1 = \alpha^{1+\omega} \cdot \min(\mathcal{I})$). Since $\omega \geq -1$, we have $\tau_1 \geq \min(\mathcal{I})$. As Mimic learns the value of $\min(\mathcal{I})$ latest at time $\min(\mathcal{I})$, it can thus correctly identify the value of τ_1 before or at time τ_1.

For a phase $k+1$, where $k \geq 1$, similarly to Basic, Mimic behaves in the following way. We ensure that at time τ_k, at the beginning of phase $k+1$, Mimic is in its initial state. At this time, Mimic computes the τ_k-schedule $S_{\tau(k)}$ (see Definition 2.1), executes it within time interval $[\tau_k, 2 \cdot \tau_k)$ and afterwards, it resets its state to the initial one. The execution of $S_{\tau(k)}$ will not be interrupted or modified when new jobs arrive within phase $k+1$. Furthermore, Mimic serves only those requests from $S_{\tau(k)}$ it has not yet served earlier. The resetting part takes time $\gamma \cdot \tau_k$, and is thus finished at time $(2 + \gamma) \cdot \tau_k = \alpha \cdot \tau_k = \tau_{k+1}$ when the next phase starts.

It is worth observing that for the deterministic case of 1-resettable scheduling (e.g., for the TRP problem), assuming that $\min(\mathcal{I}) = 1$, phases of Mimic$(1, 0)$ coincide with phases of Basic. (The function they minimize different, though.) An illustration is given in Figure 2.1.

2.5.1 Intermediate Algorithm Disc

As mentioned in Section 2.3, we introduce an additional intermediate algorithm Disc, whose analysis will allow us to bound the competitive ratios of both our deterministic and randomized solution. For an integer ℓ, we use $[\ell]$ to denote the set $\{0, \ldots, \ell - 1\}$.

Disc(γ, M, β) solves the γ-resettable scheduling problem, and is additionally parameterized by a positive integer M, and a real number $\beta \in (0, 1/M]$. Disc(γ, M, β) first

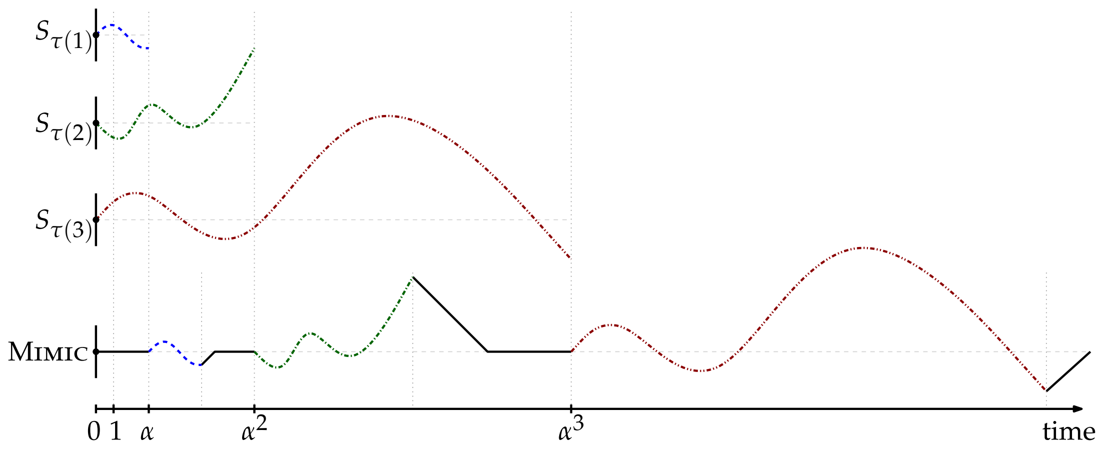


Figure 2.1: An example execution of algorithms BASIC and MIMIC$(1, 0)$ applied for the TRP problem. We use $\alpha = 3$ and we assume that $\min(\mathcal{I}) = 1$. Within time interval $[\tau(k) = \alpha^k, 2 \cdot \alpha^k)$ of phase $k+1$, BASIC executes an α^k-schedule that optimizes function $\text{VAL}_k^{\text{BASIC}}$ and MIMIC executes a $\tau(k)$-schedule $S_{\tau(k)}$ that optimizes function $\text{VAL}_{\tau(k)}$. Afterwards within time interval $[2 \cdot \alpha^k, \tau(k+1) = 3 \cdot \alpha^k)$, both of them reset their states to the initial ones (the server of TRP returns to the origin).

chooses a random integer $m \in [M]$. Then, it executes MIMIC$(\gamma, \omega = -1 + m/M + \beta)$. The main result of this chapter is the following bound, whose proof will be given in the remaining part of this section.

Theorem 2.1. *For any γ, any positive integer M, and any $\beta \in (0, 1/M]$, the competitive ratio of* DISC(γ, M, β) *for the γ-resettable scheduling is at most $1 + (1/M) \cdot \sum_{j=1}^{M} (2+\gamma)^{j/M}$.*

Corollary 2.1. *For any γ, the competitive ratio of our* MIMIC*-based deterministic solution is at most $3 + \gamma$ and the ratio of randomized one at most $1 + (1+\gamma)/\ln(2+\gamma)$.*

Proof. Let $\xi_M = 1 + (1/M) \cdot \sum_{j=1}^{M} \alpha^{j/M}$. First, we note that DISC$(\gamma, M = 1, \beta = 1)$ chooses deterministically $m = 0$ and executes MIMIC$(\gamma, \omega = -1 + 0 + 1 = 0)$, i.e., is equivalent to our deterministic algorithm. Hence, by Theorem 2.1, the corresponding competitive ratio is at most $\xi_1 = 3 + \gamma$.

For analyzing our randomized algorithm, we observe that instead of choosing a random $\omega \in (-1, 0]$, we may choose a random integer $m \in [M]$ and a random real $\beta \in (0, 1/M]$ and set $\omega = -1 + m/M + \beta$. Thus, for any fixed integer M, our randomized algorithm is equivalent to choosing random $\beta \in (0, 1/M]$ and running DISC(γ, M, β).

Fix any input $\mathcal{I}$. By Theorem 2.1, $\mathbf{E}_m[\text{COST}_{\text{DISC}(\gamma,M,\beta)}(\mathcal{I})] \leq \xi_M \cdot \text{COST}_{\text{OPT}}(\mathcal{I})$ holds for any $\beta \in (0, 1/M]$, where the expected value is taken over random choice of m. Clearly, this relation holds also when β is chosen randomly, i.e., $\mathbf{E}_\omega[\text{COST}_{\text{MIMIC}(\gamma,\omega)}] = \mathbf{E}_\gamma \mathbf{E}_m[\text{COST}_{\text{DISC}(\gamma,M,\beta)}(\mathcal{I})] \leq \xi_M \cdot \text{COST}_{\text{OPT}}(\mathcal{I})$. As the bound is valid for any M, and the competitive ratio of our randomized algorithm is at most $\inf_{M \in \mathbb{N}} \{\xi_M\} = \lim_{M \to \infty} \xi_M = 1 + (1+\gamma)/\ln(2+\gamma)$. □

2.5.2 Structural Properties of DISC

In this section, we build relations useful for analyzing the performance of $\text{Disc}(\gamma, M, \beta)$ on any instance $\mathcal{I}$ of the γ-resettable scheduling problem.

We start by presenting structural properties of schedules S_τ. We note that even if there exists a τ-schedule A that completes all jobs from $\mathcal{I}$, S_τ may leave some jobs untouched. However, a sufficiently long schedule S_τ completes all jobs.

Lemma 2.4. *Fix any input $\mathcal{I}$. There exists a value $T_\mathcal{I}$, such that for any $\tau \geq T_\mathcal{I}$, S_τ completes all jobs of $\mathcal{I}$ and is an optimal (cost-minimal) solution for $\mathcal{I}$.*

Proof. Let Opt be a cost-optimal schedule for $\mathcal{I}$ and let t be its length. Let w be the weight of the lightest job from $\mathcal{I}$. We fix $T_\mathcal{I} = \max\{t, (\text{VAL}_t(\text{Opt}) + 1)/w\}$. Now, we pick any $\tau \geq T_\mathcal{I}$, and investigate properties of S_τ.

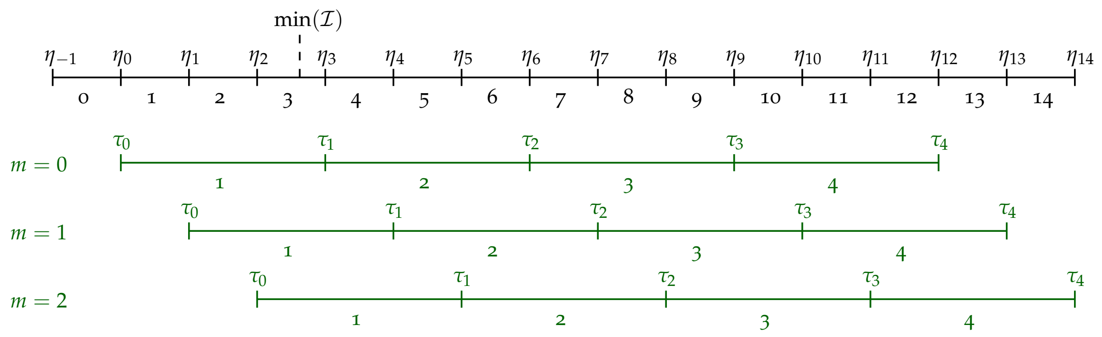
As $\tau \geq T_\mathcal{I} \geq t$, the schedule of Opt can be trivially extended to a τ-schedule A that does nothing in its suffix of length $\tau - t$. Both A and Opt complete all jobs, and thus $\text{VAL}_\tau(A) = \text{VAL}_t(\text{Opt})$. Moreover, as S_τ minimizes function VAL_τ, $\text{VAL}_\tau(S_\tau) \leq \text{VAL}_\tau(A) = \text{VAL}_t(\text{Opt}) < T_\mathcal{I} \cdot w \leq \tau \cdot w$, and thus S_τ completes all jobs (as otherwise VAL_τ would include a penalty of at least $\tau \cdot w$). As S_τ and Opt complete all jobs, $\text{COST}_{S_\tau}(\mathcal{I}) = \text{VAL}_\tau(S_\tau) \leq \text{VAL}_t(\text{Opt}) = \text{COST}_{\text{Opt}}(\mathcal{I})$, i.e., S_τ is an optimal solution for $\mathcal{I}$. □

Sub-phases. Recall that the algorithm $\text{Disc}(\gamma, M, \beta)$ chooses a random integer $m \in [M]$, and executes $\text{Mimic}(\gamma, \omega = -1 + m/M + \beta)$. To compare Disc executions for different random choices, we introduce sub-phases. Recall that $\alpha = 2 + \gamma$; let $\delta = \alpha^{1/M}$.

Recall that the k-th phase of Mimic starts at time τ_{k-1} and ends at time τ_k, where $\tau_k = \min(\mathcal{I}) \cdot \alpha^{k-1+m/M+\beta} = \min(\mathcal{I}) \cdot \alpha^{\beta-1} \cdot \delta^{m+k \cdot M}$. For any q, we define

$$\eta_q = \eta(q) = \min(\mathcal{I}) \cdot \alpha^{\beta-1} \cdot \delta^q. \tag{2.8}$$

In these terms, $\tau_k = \eta_{m+k\cdot M}$. We define the q-th sub-phase (for $q \geq 0$) as the time interval starting at time η_{q-1} and ending at time η_q. Then, phase k of $\text{Disc}(\gamma, M, \beta)$ consists of exactly M sub-phases, numbered from $(k-1) \cdot M + m + 1$ to $k \cdot M + m$. An example of phases and sub-phases is given in Figure 2.2. We emphasize that the start and the end of a sub-phase is a deterministic function of the parameters of Disc, while the start and end of a phase depend additionally on the value $m \in [M]$ that Disc chooses randomly.



$\min(\mathcal{I})$
η_{-1} η_0 η_1 η_2 η_3 η_4 η_5 η_6 η_7 η_8 η_9 η_{10} η_{11} η_{12} η_{13} η_{14}
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
$m = 0$
τ_0 τ_1 τ_2 τ_3 τ_4
1 2 3 4
$m = 1$
τ_0 τ_1 τ_2 τ_3 τ_4
1 2 3 4
$m = 2$
τ_0 τ_1 τ_2 τ_3 τ_4
1 2 3 4

Figure 2.2: Example of phases (green) and sub-phases (black) of algorithm Disc($\gamma, M = 3, \beta$) for all possible choices of m. The time interval lengths are in logarithmic scale. The starts and ends of sub-phases are deterministic functions of γ, M, and β, but the start of a phase depends additionally on the integer $m \in [M]$ chosen randomly by Disc. Sub-phase 0 is not contained in any phase, but will be used in our analysis.

Recall that our deterministic algorithm is equivalent to Mimic$(\gamma, 0) \equiv$ Disc$(\gamma, 1, 1)$. In this case $m = 0$, and thus $\eta_q = \tau_q$ for any q, i.e., each phase consists of one sub-phase, and their indexes coincide.

Sub-phases vs Auxiliary Schedules. We now identify the times when auxiliary schedules are computed by Disc(γ, M, β). Recall that at the beginning of any phase $k + 1$ (where $k \geq 1$), i.e., at time $\tau_k = \eta_{m+k \cdot M}$, Disc computes and executes schedule $S_{\eta(m+k \cdot M)}$. Let $T_{\mathcal{I}}$ be the threshold guaranteed by Lemma 2.4 and we define $K_{\mathcal{I}}$ as the smallest integer satisfying $\eta(K_{\mathcal{I}} \cdot M) \geq T_{\mathcal{I}}$. Note that $K_{\mathcal{I}}$ is a deterministic function of input $\mathcal{I}$.

For any choice of $m \in [M]$, the schedule $S_{\eta(m+K_{\mathcal{I}} \cdot M)}$ completes all jobs. This schedule is executed by Disc in phase $K_{\mathcal{I}} + 1$, and thus Disc terminates latest in phase $K_{\mathcal{I}} + 1$. Summing up, Disc(γ, M, β) executes schedules $S_{\eta(m+M)}, S_{\eta(m+2M)}, \ldots, S_{\eta(m+K_{\mathcal{I}} \cdot M)}$. At the beginning of the first phase, Disc does nothing, but for notational ease, we assume that in the first phase, it also computes and executes a dummy schedule $S_{\eta(m)}$, which does not complete any job. For succinctness, we use $A_q = S_{\eta(q)}$. In these terms, Disc(γ, M, β) executes schedules $A_{m+k \cdot M}$ for $k \in [K_{\mathcal{I}} + 1]$.

Let $Q = K_{\mathcal{I}} \cdot M + (M - 1)$: possible schedule indexes used by Disc range from 0 to Q. For any schedule A_q, we define the set of indexes of *preceding schedules* $P(q) = \{q', q' + M, \ldots, q - M\}$, where $q' = q \mod M$.

Fresh and Stale Requests. We assume that no jobs are completed by the online algorithm while it is resetting the executor, and we assume that the execution of schedule A_q may complete only jobs from set $R(A_q)$. It is however important to note that $R(A_q)$ and $R(A_{q-M})$ may overlap significantly, in which case the execution of schedule A_q serves only these jobs from $R(A_q)$ that have not been served already. To further quantify this effect, similarly to Basic, for $q \in [Q+1]$, we define the set of *fresh* jobs of schedule A_q as

$$R^{\mathrm{F}}(A_q) = R(A_q) \setminus \bigcup_{\ell \in P(q)} R(A_\ell). \tag{2.9}$$

The remaining jobs from $R(A_q)$ are called *stale* and are denoted $R^{\mathrm{S}}(A_q) = R(A_q) \setminus R^{\mathrm{F}}(A_q)$. For succinctness, we define the following shorthand notations for their weights:

$$w_q^{\mathrm{F}} = w(R^{\mathrm{F}}(A_q)), \qquad w_q^{\mathrm{S}} = w(R^{\mathrm{S}}(A_q)), \qquad w_q = w(R(A_q)) = w_q^{\mathrm{F}} + w_q^{\mathrm{S}}. \tag{2.10}$$

Now, again similarly to Basic, we present relations on weights of fresh and stale requests.

Lemma 2.5. *For any $q \in [Q+1]$, it holds that $w_q^{\mathrm{S}} \leq \sum_{\ell \in P(q)} w_\ell^{\mathrm{F}}$. This relation becomes equality for $q \geq K_{\mathcal{I}} \cdot M$.*

Proof. By a simple induction, it can be shown that $\biguplus_{\ell \in P(q)} R^{\mathrm{F}}(A_\ell) = \bigcup_{\ell \in P(q)} R(A_\ell)$ for any $q \in [Q+1]$. Then, using the definition of stale jobs, $R^{\mathrm{S}}(A_q) \subseteq \bigcup_{\ell \in P(q)} R(A_\ell) = \biguplus_{\ell \in P(q)} R^{\mathrm{F}}(A_\ell)$. Applying weight to both sides yields $w_q^{\mathrm{S}} \leq \sum_{\ell \in P(q)} w_\ell^{\mathrm{F}}$.

Next, we show that this relation can be reversed for $q \geq K_{\mathcal{I}} \cdot M$ (i.e., for the schedule executed in the last phase of Disc). For such q, A_q completes all jobs, and thus $\bigcup_{\ell \in P(q)} R(A_\ell) \subseteq R(A_q) = R^{\mathrm{F}}(A_q) \uplus R^{\mathrm{S}}(A_q)$. By the definition of fresh jobs, $R^{\mathrm{F}}(A_q)$ does not contain any job from $\bigcup_{\ell \in P(q)} R(A_\ell)$, and thus $\bigcup_{\ell \in P(q)} R(A_\ell) \subseteq R^{\mathrm{S}}(A_q)$. This implies that $\biguplus_{\ell \in P(q)} R^{\mathrm{F}}(A_\ell) = \bigcup_{\ell \in P(q)} R(A_\ell) \subseteq R^{\mathrm{S}}(A_q)$. After applying weights to both sides, we obtain $w_q^{\mathrm{S}} \geq \sum_{\ell \in P(q)} w_\ell^{\mathrm{F}}$ as desired. □

Jobs Completed in Sub-phases. For further analysis, we refine our notions when a job is completed. For a η_q-schedule A_q, let $R_j(A_q)$ be the set of jobs completed in sub-phase $j \leq q$, i.e., within interval $[\eta_{j-1}, \eta_j)$. As $\eta_{-1} \leq \eta_{m-1} \leq \min(\mathcal{I})$ (cf. (2.8)), no job can be completed within the interval $[0, \eta_{-1})$ (before sub-phase 0). Hence, $R(A_q) = \biguplus_{j=0}^{q} R_j(A_q)$.

We partition sets $R^{\mathrm{F}}(A_q)$ and $R^{\mathrm{S}}(A_q)$ analogously, defining sets $R_j^{\mathrm{F}}(A_q)$ and $R_j^{\mathrm{S}}(A_q)$ (for $0 \leq j \leq q$), such that $R^{\mathrm{F}}(A_q) = \biguplus_{j=0}^{q} R_j^{\mathrm{F}}(A_q)$ and $R^{\mathrm{S}}(A_q) = \biguplus_{j=0}^{q} R_j^{\mathrm{S}}(A_q)$. For succinctness, for $0 \leq j \leq q$, we introduce the following shorthand notations:

- $w_{qj}^{\mathrm{F}} = w(R_j^{\mathrm{F}}(A_q))$, $w_{qj}^{\mathrm{S}} = w(R_j^{\mathrm{S}}(A_q))$, and $w_{qj} = w(R_j(A_q)) = w_{qj}^{\mathrm{F}} + w_{qj}^{\mathrm{S}}$;
- $g_{qj}^{\mathrm{F}} = \textsc{cost}_{A_q}(R_j^{\mathrm{F}}(A_q))$, $g_{qj}^{\mathrm{S}} = \textsc{cost}_{A_q}(R_j^{\mathrm{S}}(A_q))$, and $g_{qj} = \textsc{cost}_{A_q}(R_j(A_q)) = g_{qj}^{\mathrm{F}} + g_{qj}^{\mathrm{S}}$.

Lemma 2.6. *For any $0 \leq q < \ell \leq Q$, it holds that $\sum_{j=0}^{q}(g_{qj} - g_{\ell j}) + \sum_{j=0}^{q} \eta_q \cdot (w_{\ell j} - w_{qj}) \leq 0$.*

Proof. For any η_q-schedule B, it holds that

$$\begin{aligned}\textsc{val}_{\eta(q)}(B) &= \textsc{cost}_B(R(B)) + \eta_q \cdot w\left(\mathcal{I}_{\eta(q)} \setminus R(B)\right) \\ &= \textstyle\sum_{j=0}^{q} \textsc{cost}_B(R_j(B)) + \eta_q \cdot w(\mathcal{I}_{\eta(q)}) - \eta_q \cdot \sum_{j=0}^{q} w(R_j(B)).\end{aligned}$$

Fix any $\ell \leq Q$ and let A_ℓ^q be the η_q-schedule consisting of the first q sub-phases of η_ℓ-schedule A_ℓ. Since A_q is a minimizer of $\textsc{val}_{\eta(q)}$, it holds that $\textsc{val}_{\eta(q)}(A_q) \leq \textsc{val}_{\eta(q)}(A_\ell^q)$. Thus, $\sum_{j=0}^{q} g_{qj} - \eta_q \cdot \sum_{j=0}^{q} w_{qj} \leq \sum_{j=0}^{q} g_{\ell j} - \eta_q \cdot \sum_{j=0}^{q} w_{\ell j}$. □

Costs of DISC and OPT. Finally, we can express costs of Disc and Opt using the newly introduced notions.

Lemma 2.7. *For any input $\mathcal{I}$, parameters M and $\beta \in (0, 1/M]$, it holds that $\mathbf{E}[\textsc{cost}_{\text{Disc}}(\mathcal{I})] = (1/M) \cdot \sum_{q=0}^{Q} \sum_{j=0}^{q} \left(\eta_q \cdot w_{qj}^{\mathrm{F}} + g_{qj}^{\mathrm{F}}\right)$.*

Proof. Recall that Disc chooses random $m \in [M]$ and then at time η_q it executes schedule A_q, for all $q \in \{m, m+M, \ldots, m + K_{\mathcal{I}} \cdot M\}$. When Disc executes A_q, it completes jobs from $R^{\mathrm{F}}(A_q)$. By the delayed execution property of the resettable scheduling (cf. Section 2.2), each job $r \in R^{\mathrm{F}}(A_q)$ is completed at time $\eta_q + s_{A_q}(r)$. Thus, the cost of executing A_q by Disc is equal to

$$\begin{aligned}\textstyle\sum_{r \in R^{\mathrm{F}}(A_q)} w(r) \cdot (\eta_q + s_{A_q}(r)) &= \eta_q \cdot w(R^{\mathrm{F}}(A_q)) + \textsc{cost}_{A_q}(R^{\mathrm{F}}(A_q)) \\ &= \eta_q \cdot w_q^{\mathrm{F}} + \textstyle\sum_{j=0}^{q} g_{qj}^{\mathrm{F}} = \sum_{j=0}^{q}\left(\eta_q \cdot w_{qj}^{\mathrm{F}} + g_{qj}^{\mathrm{F}}\right).\end{aligned}$$

For any $q \in [Q+1]$, the probability that Disc executes A_q is equal to $1/M$, and thus the lemma follows. □

Lemma 2.8. *For any input $\mathcal{I}$ and any $q \in \{Q-M+1, Q-M+2, \ldots, Q\}$, it holds that $\textsc{cost}_{\text{Opt}}(\mathcal{I}) = \sum_{j=0}^{q} g_{qj}$.*

Proof. Recall that for such choice of q, schedules A_q serve all jobs of $\mathcal{I}$ achieving optimal cost. Therefore, $\textsc{cost}_{\text{Opt}}(\mathcal{I}) = \textsc{cost}_{A_q}(R(A_q)) = \sum_{j=0}^{q} \textsc{cost}_{A_q}(R_j(A_q)) = \sum_{j=0}^{q} g_{qj}$. □

2.5.3 Factor-Revealing Linear Program

Now we show that the Disc-to-Opt cost ratio on an arbitrary input $\mathcal{I}$ can be upper-bounded by a value of a linear (maximization) program.

Assume we fixed γ and any input $\mathcal{I}$ to the γ-resettable scheduling problem. We also fix parameters of Disc: an integer M and $\beta \in (0, 1/M]$. These choices imply the values of Q and η_q for any q. This allows us to define the linear program $\mathcal{P}_{\gamma,\mathcal{I},M,\beta}$ whose goal is to maximize

$$\textstyle\sum_{q=0}^{Q} \sum_{j=0}^{q} \eta_q \cdot w^{\mathrm{F}}_{qj} + g^{\mathrm{F}}_{qj} \tag{2.11}$$

subject to the following constraints:

$$\begin{aligned}
\textstyle\sum_{j=0}^{q} g_{qj} &\le 1 && \text{for all } Q - M + 1 \le q \le Q && (2.12)\\
\textstyle\sum_{j=0}^{q} w^{\mathrm{S}}_{qj} - \sum_{\ell \in P(q)} \sum_{j=0}^{\ell} w^{\mathrm{F}}_{\ell j} &\le 0 && \text{for all } 0 \le q \le Q - M && (2.13)\\
\textstyle\sum_{\ell \in P(q)} \sum_{j=0}^{\ell} w^{\mathrm{F}}_{\ell j} - \sum_{j=0}^{q} w^{\mathrm{S}}_{qj} &\le 0 && \text{for all } Q - M + 1 \le q \le Q && (2.14)\\
\textstyle\sum_{j=0}^{q} (g_{qj} - g_{\ell j}) + \sum_{j=0}^{q} \eta_q \cdot (w_{\ell j} - w_{qj}) &\le 0 && \text{for all } 0 \le q < \ell \le Q && (2.15)\\
\eta_{j-1} \cdot w^{\mathrm{S}}_{qj} - g^{\mathrm{S}}_{qj} &\le 0 && \text{for all } 0 \le j \le q \le Q && (2.16)\\
g^{\mathrm{F}}_{qj} - \eta_j \cdot w^{\mathrm{F}}_{qj} &\le 0 && \text{for all } 0 \le j \le q \le Q && (2.17)\\
\eta_{j-1} \cdot w^{\mathrm{F}}_{qj} - g^{\mathrm{F}}_{qj} &\le 0 && \text{for all } 0 \le j \le q \le Q && (2.18)
\end{aligned}$$

and non-negativity of all variables. In (2.15), we treat w_{qj} and g_{qj} not as variables, but as shorthand notations for $w^{\mathrm{F}}_{qj} + w^{\mathrm{S}}_{qj}$ and $g^{\mathrm{F}}_{qj} + g^{\mathrm{S}}_{qj}$, respectively.

The intuition behind this LP formulation is that instead of creating the whole input $\mathcal{I}$, the adversary only chooses the values of variables w^{F}_{qj}, w^{S}_{qj}, g^{F}_{qj} and g^{S}_{qj} that satisfy some subset of inequalities (inequalities that have to be satisfied if these variables were created on the basis of actual input $\mathcal{I}$). This intuition is formalized below.

Lemma 2.9. *Fix any γ, any input $\mathcal{I}$ for γ-resettable scheduling, and parameters of* Disc*: integer M and $\beta \in (0, 1/M]$. Then, $\mathbf{E}[\textsc{cost}_{\textsc{Disc}}(\mathcal{I})] / \textsc{cost}_{\textsc{Opt}}(\mathcal{I}) \le P^*_{\gamma,\mathcal{I},M,\beta}/M$, where $P^*_{\gamma,\mathcal{I},M,\beta}$ is the value of the optimal solution to $\mathcal{P}_{\gamma,\mathcal{I},M,\beta}$.*

Proof. By scaling all variables by the same value, $\mathcal{P}_{\gamma,\mathcal{I},M,\beta}$ is equivalent to the (non-linear) optimization program $\mathcal{P}'_{\gamma,\mathcal{I},M,\beta}$, whose objective is to maximize the ratio $(\sum_{q=0}^{Q} \sum_{j=0}^{q} \eta_q \cdot w^{\mathrm{F}}_{qj} + g^{\mathrm{F}}_{qj}) / \max_{Q-M+1 \le q \le Q} \sum_{j=0}^{q} g_{qj}$, subject to constraints (2.13)–(2.18). In particular, the optimal values of these programs, $P^*_{\gamma,\mathcal{I},M,\beta}$ and $P'^*_{\gamma,\mathcal{I},M,\beta}$ are equal.

Next, we set the values of variables w^{F}_{qj}, w^{S}_{qj}, g^{F}_{qj} and g^{S}_{qj} on the basis of input $\mathcal{I}$, and parameters M and β. (Note that the variables depend on these parameters, but not on the random choices of Disc.) We now show that they satisfy the constraints of $\mathcal{P}'^*_{\gamma,\mathcal{I},M,\beta}$ and we relate $\mathbf{E}[\textsc{cost}_{\textsc{Disc}}(\mathcal{I})] / \textsc{cost}_{\textsc{Opt}}(\mathcal{I})$ to $P^*_{\gamma,\mathcal{I},M,\beta}$.

By Lemma 2.5 and the relations $w^{\mathrm{F}}_q = \sum_{j=0}^{q} w^{\mathrm{F}}_{qj}$ and $w^{\mathrm{S}}_q = \sum_{j=0}^{q} w^{\mathrm{S}}_{qj}$, the variables satisfy (2.13) and (2.14). Next, Lemma 2.6 implies (2.15). Inequalities (2.16), (2.17) and (2.18) follow directly by the definition of costs and weights. Finally, by Lemma 2.7 and Lemma 2.8, for any $q \in \{Q - M + 1, \dots, Q\}$, it holds that $\mathbf{E}[\textsc{cost}_{\textsc{Disc}}(\mathcal{I})] / \textsc{cost}_{\textsc{Opt}}(\mathcal{I}) =$

$(1/M) \cdot (\sum_{q=0}^{Q} \sum_{j=0}^{q} \eta_q \cdot w^{\mathrm{F}}_{qj} + g^{\mathrm{F}}_{qj}) / (\sum_{j=0}^{q} g_{qj})$, and thus $\mathbf{E}[\textsc{cost}_{\textsc{Disc}}(\mathcal{I})] / \textsc{cost}_{\textsc{Opt}}(\mathcal{I}) \leq P'^{*}_{\gamma,\mathcal{I},M,\beta} / M = P^{*}_{\gamma,\mathcal{I},M,\beta} / M$. □

2.5.4 Dual Program and Competitive Ratio.

By Lemma 2.9, the optimal value of $\mathcal{P}_{\gamma,\mathcal{I},M,\beta}$ is an upper bound on the competitive ratio of Disc. By weak duality, an upper-bound is given by any feasible solution to the dual program $\mathcal{D}_{\gamma,\mathcal{I},M,\beta}$ that we present below.

$\mathcal{D}_{\gamma,\mathcal{I},M,\beta}$ uses variables $\xi_q, B_q, C_q, D_{\ell q}, F_{qj}, G_{qj}$, and H_{qj}, corresponding to inequalities (2.12)–(2.18) from $\mathcal{P}_{\gamma,\mathcal{I},M,\beta}$, respectively. In the formulas below, we use $L_q = M \cdot K + (q \mod M)$ and $S(q) = \{q + M, q + 2 \cdot M, \ldots, L_q - M\}$. For succinctness of the description, we introduce two auxiliary variables for any $0 \leq j \leq q \leq Q$:

$$U_{qj} = \textstyle\sum_{\ell=q+1}^{Q} D_{\ell q} - \sum_{\ell=j}^{q-1} D_{q\ell} \quad \text{and} \quad V_{qj} = \sum_{\ell=j}^{q-1} \eta_\ell \cdot D_{q\ell} - \sum_{\ell=q+1}^{Q} \eta_q \cdot D_{\ell q}. \tag{2.19}$$

The goal of $\mathcal{D}_{\gamma,\mathcal{I},M,\beta}$ is to minimize

$$\textstyle\sum_{q=Q-M+1}^{Q} \xi_q \tag{2.20}$$

subject to the following constraints (in all of them, we omitted the statement that they hold for all $j \in \{0, \ldots, q\}$):

$$\begin{aligned}
U_{qj} + G_{qj} - H_{qj} &\geq 1 & &\text{for all } 0 \leq q \leq Q - M & (2.21)\\
U_{qj} - F_{qj} &\geq 0 & &\text{for all } 0 \leq q \leq Q - M & (2.22)\\
U_{qj} + G_{qj} - H_{qj} + \xi_q &\geq 1 & &\text{for all } Q - M + 1 \leq q \leq Q & (2.23)\\
U_{qj} - F_{qj} + \xi_q &\geq 0 & &\text{for all } Q - M + 1 \leq q \leq Q & (2.24)\\
V_{qj} + \eta_{j-1} \cdot H_{qj} - \eta_j \cdot G_{qj} + C_{L_q} - \textstyle\sum_{\ell \in S(q)} B_\ell &\geq \eta_q & &\text{for all } 0 \leq q \leq Q - M & (2.25)\\
V_{qj} + \eta_{j-1} \cdot F_{qj} + B_q &\geq 0 & &\text{for all } 0 \leq q \leq Q - M & (2.26)\\
V_{qj} - \eta_j \cdot G_{qj} + \eta_{j-1} \cdot H_{qj} &\geq \eta_q & &\text{for all } Q - M + 1 \leq q \leq Q & (2.27)\\
V_{qj} + \eta_{j-1} \cdot F_{qj} - C_q &\geq 0 & &\text{for all } Q - M + 1 \leq q \leq Q & (2.28)
\end{aligned}$$

and non-negativity of all variables.

Lemma 2.10. *For any γ, any input $\mathcal{I}$ for γ-resettable scheduling, any positive integer M, and any $\beta \in (0, 1/M]$, there exists a feasible solution to $\mathcal{D}_{\gamma,\mathcal{I},M,\beta}$ of value at most $M + \sum_{j=1}^{M} (2 + \gamma)^{j/M}$.*

We defer the proof to the next subsection, first arguing how it implies the main theorem of this chapter (the competitive ratio of Disc).

Proof of Theorem 2.1. Fix any γ, and consider algorithm $\textsc{Disc}(\gamma, M, \beta)$ for any positive integer M, and any $\beta \in (0, 1/M]$. Fix any input $\mathcal{I}$ to the γ-resettable scheduling problem. Let $P^*_{\gamma,\mathcal{I},M,\beta}$ be the value of an optimal solution to $\mathcal{P}_{\gamma,\mathcal{I},M,\beta}$. By weak duality and Lemma 2.10, $P_{\gamma,\mathcal{I},M,\beta} \le M + \sum_{j=1}^{M}(2+\gamma)^{j/M}$. Hence, by Lemma 2.9, $\mathbf{E}[\text{cost}_{\textsc{Disc}}(\mathcal{I})]/\text{cost}_{\textsc{Opt}}(\mathcal{I}) \le P^*_{\gamma,\mathcal{I},M,\beta}/M \le 1 + (1/M) \cdot \sum_{j=1}^{M}(2+\gamma)^{j/M}$, as desired. □

2.5.5 Proof of Lemma 2.10

Let

$$\Delta_k = \textstyle\sum_{i=0}^{k} \delta^i = \left(\delta^{k+1} - 1\right) / (\delta - 1)\,.$$

In particular $\Delta_{-1} = 0$. We choose the following values of the dual variables:

$$\xi_q = 1 + \delta^{q-Q+M} \quad \text{for } Q - M + 1 \le q \le Q,$$

$$F_{qj} = \begin{cases} \xi_q & \text{for } Q-M+1 \le j \le q \le Q, \\ \delta \cdot \Delta_{M-1} & \text{for } 0 \le j \le Q-M \text{ and } q = j, \\ 1 & \text{for } 0 \le j \le Q-M \text{ and } q \in \{j+1, \dots, j+M\}, \\ 0 & \text{otherwise,} \end{cases}$$

$$G_{qj} = \begin{cases} \Delta_{q-Q+M-1} - \Delta_{q-j} & \text{for } Q-M+1 \le j \le q \le Q, \\ \Delta_{q-j-M-1} & \text{for } j \le q-M, \\ 0 & \text{otherwise,} \end{cases}$$

$$\begin{aligned} B_q &= \eta_{q-M-1} \cdot \left(\delta^{M+1}+1\right) \cdot \left(\delta^{M}-1\right) && \text{for } 0 \le q \le Q-M, \\ C_q &= \eta_{q-M-1} \cdot \left(\delta^{M+1}+1\right) && \text{for } Q-M+1 \le q \le Q, \\ D_{qj} &= F_{q,j+1} - F_{qj} && \text{for } 0 \le j < q \le Q, \\ H_{qj} &= F_{qj} + G_{qj} - 1 && \text{for } 0 \le j \le q \le Q. \end{aligned}$$

The values of F_{qj} and G_{qj} (for $0 \le j \le q \le Q$) are depicted in Figure 2.3 for an easier reference. We will extensively use the property that $\eta_i \cdot \delta^j = \eta_{i+j}$ for any i and j.

Objective Value. With the above assignment of dual variables the objective value of $\mathcal{D}_{\gamma,\mathcal{I},M,\beta}$ is equal to $\sum_{q=Q-M+1}^{Q} \xi_q = M + \sum_{j=1}^{M} \delta^j = M + \sum_{j=1}^{M}(2+\gamma)^{j/M}$ as desired.

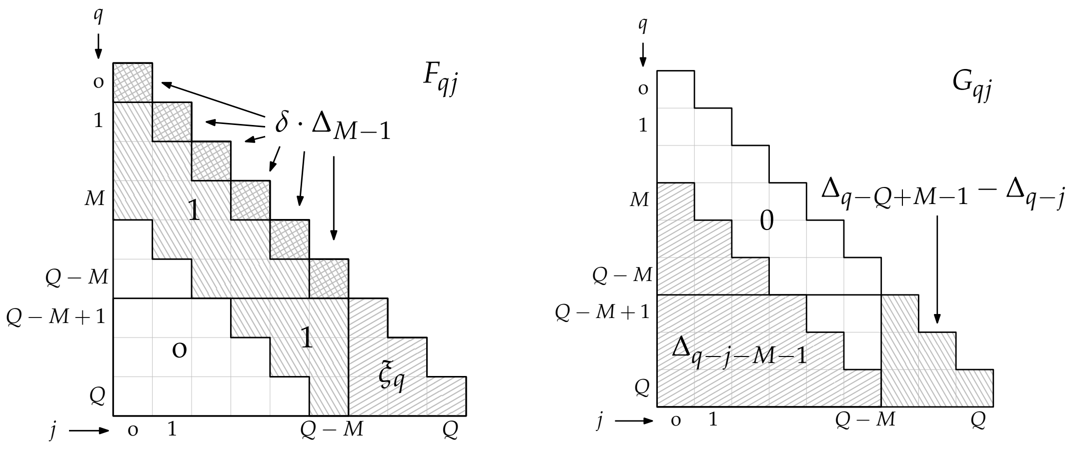


Figure 2.3: Visual presentation of values assigned to dual variables F_{qj} (left) and G_{qj} (right) for $M = 3$ and $Q = 8$.

Non-negativity of Variables. Variables ξ_q, C_q, B_q, F_{qj} and G_{qj} are trivially non-negative (for those q and j for which they are defined). The non-negativity of $D_{qj} = F_{q,j+1} - F_{qj}$ follows as F_{qj} is a non-decreasing function of its second argument (cf. Figure 2.3).

Finally, for showing non-negativity of variable H_{qj}, we consider two cases. If $j \geq q - M$, then $F_{qj} \geq 1$. Otherwise, $j \leq q - M - 1$, and then $G_{qj} = \Delta_{q-j-M-1} \geq 1$. Thus, in either case $H_{qj} = F_{qj} + G_{qj} - 1 \geq 0$.

Helper Bounds. It remains to show that the given values of dual variables satisfy all constraints (2.21)–(2.28) of the dual program $\mathcal{D}_{\gamma,\mathcal{I},M,\beta}$. We define a few helper notions and identities that are used throughout the proof of dual feasibility. For any $q \in [Q+1]$, let

$$R_q = \textstyle\sum_{\ell=q+1}^{Q} D_{\ell q} = \sum_{\ell=q+1}^{Q} \left(F_{\ell,q+1} - F_{\ell q}\right).$$

Lemma 2.11. *$R_q = \delta \cdot \Delta_{M-1}$ for $q \leq Q - M$ and $R_q = 0$ otherwise.*

Proof. We consider three cases.

1. $q \in \{0, \dots, Q-M-1\}$. Then, $R_q = F_{q+1,q+1} + \sum_{\ell=q+1}^{Q} \left(F_{\ell+2,\ell+1} - F_{\ell+1,\ell}\right) - F_{Qq} = \delta \cdot \Delta_{M-1} + \sum_{\ell=q+1}^{Q} 0 - 0 = \delta \cdot \Delta_{M-1}$.

2. $q = Q - M$. Then, $R_q = \sum_{\ell=Q-M+1}^{Q} (\xi_\ell - 1) = \sum_{j=1}^{M} \delta^j = \delta \cdot \Delta_{M-1}$.

3. $q \in \{Q-M+1, \dots, Q\}$. Then, $R_q = \sum_{\ell=q+1}^{Q} (\xi_\ell - \xi_\ell) = 0$. □

Next, we investigate the values of V_{qj} for different q and j. Using its definition (cf. (2.19)),

$$V_{qj} = \textstyle\sum_{\ell=j}^{q-1} \eta_\ell \cdot D_{q\ell} - \sum_{\ell=q+1}^{Q} \eta_q \cdot D_{\ell q} = \sum_{\ell=j}^{q-1} \eta_\ell \cdot \left(F_{q,\ell+1} - F_{q\ell}\right) - \eta_q \cdot R_q. \tag{2.29}$$

Additionally, using $H_{qj} = F_{qj} + G_{qj} - 1$, we obtain

$$\eta_j \cdot G_{qj} - \eta_{j-1} \cdot H_{qj} = (\eta_j - \eta_{j-1}) \cdot G_{qj} + \eta_{j-1} - \eta_{j-1} \cdot F_{qj}. \tag{2.30}$$

Using the chosen values of G_{qj}, we observe that

$$(\eta_j - \eta_{j-1}) \cdot G_{qj} = \begin{cases} \eta_{q+j-Q+M-1} - \eta_q & \text{for } Q-M+1 \le j \le q, \\ \eta_{q-M-1} - \eta_{j-1} & \text{for } j \le q-M-1, \\ 0 & \text{otherwise.} \end{cases} \tag{2.31}$$

Furthermore, in all the cases, it can be verified that

$$(\eta_j - \eta_{j-1}) \cdot G_{qj} + \eta_{j-1} - \eta_{q-M-1} \ge 0. \tag{2.32}$$

Showing inequalities (2.21)–(2.24)

We prove that relations (2.21)–(2.24) hold with equality. In fact, it suffices to show (2.22) and (2.24): inequalities (2.21) and (2.23) follow immediately as we chose $H_{qj} = F_{qj} + G_{qj} - 1$. Using the definition of U_{qj} (cf. (2.19)), we obtain

$$U_{qj} = \textstyle\sum_{\ell=q+1}^{Q} D_{\ell q} - \sum_{\ell=j}^{q-1} D_{q\ell} = R_q - \sum_{\ell=j}^{q-1} \left(F_{q,\ell+1} - F_{q\ell}\right) = R_q - F_{qq} + F_{qj}.$$

Now, we observe that for $q \le Q - M$, it holds that $R_q - F_{qq} = \delta \cdot \Delta_{M-1} - \delta \cdot \Delta_{M-1} = 0$, and thus $U_{qj} - F_{qj} = 0$, which implies (2.22). On the other hand, for $q > Q - M$, it holds that $R_q - F_{qq} = 0 - \xi_q$, and hence $U_{qj} - F_{qj} + \xi_q = 0$, which implies (2.24).

Showing inequalities (2.25)–(2.26)

Within this part, we assume $q \le Q - M$. We start with evaluating some terms that are present in (2.25) and (2.26). First, we observe that

$$B_q = \eta_{q-M-1} \cdot \left(\delta^{M+1} + 1\right) \cdot \left(\delta^M - 1\right) = \eta_{q+M} - \eta_q + \eta_{q-1} - \eta_{q-M-1}. \tag{2.33}$$

Second, we compute the term $C_{L_q} - \sum_{\ell \in S(q)} B_\ell$. Recall that $S(q) = \{q + M, q + 2 \cdot M, \ldots, L_q - M\}$. Thus,

$$\begin{aligned} C_{L_q} - \textstyle\sum_{\ell \in S(q)} B_\ell &= (\delta^{M+1} + 1) \cdot \left[\eta(L_q - M - 1) - (\delta^M - 1) \cdot \eta(-M-1) \cdot \textstyle\sum_{\ell \in S(q)} \delta^\ell\right] \\ &= (\delta^{M+1} + 1) \cdot \left[\eta(L_q - M - 1) - \eta(-M-1) \cdot \left(\delta^{L_q} - \delta^{q+M}\right)\right] \\ &= (\delta^{M+1} + 1) \cdot \eta_{q-1} = \eta_{q+M} + \eta_{q-1}. \end{aligned} \tag{2.34}$$

Lemma 2.12. *Fix any* $0 \le j \le q \le Q - M$. *Then,*

$$V_{qj} = \eta_q - \eta_{q-1} - \eta_{q+M} - \eta_{j-1} \cdot F_{qj} + (\eta_j - \eta_{j-1}) \cdot G_{qj} + \eta_{j-1}.$$

Proof. By the definition, $\Delta_{M-1} = \sum_{i=0}^{M-1} \delta^i$, and therefore $(\eta_{q-1} - \eta_q) \cdot \delta \cdot \Delta_{M-1} = \eta_q - \eta_{q+M}$. Thus, it suffices to show the following relation

$$V_{qj} = \eta_{q-1} \cdot (\delta \cdot \Delta_{M-1} - 1) - \eta_q \cdot \delta \cdot \Delta_{M-1} - \eta_{j-1} \cdot F_{qj} + (\eta_j - \eta_{j-1}) \cdot G_{qj} + \eta_{j-1}.$$

To evaluate V_{qj} using (2.29), it is useful to trace values $F_{qj}, F_{q,j+1}, \ldots, F_{qq}$ (cf. Figure 2.3), noting that only the increases of these values contribute to V_{qj}. We also note that for $q \le Q - M$, possible increases are from 0 to 1 (between $F_{q,q-M-1}$ and $F_{q,q-M}$) and from 1 to $\delta \cdot \Delta_{M-1}$ (between $F_{q,q-1}$ and F_{qq}). We consider three cases, using $R_q = \delta \cdot \Delta_{M-1}$ below.

1. $j \le q - M - 1$. Then, $F_{qj} = 0$ and

$$\begin{aligned} V_{qj} &= \eta_{q-1} \cdot \left(F_{qq} - F_{q,q-1}\right) + \eta_{q-M-1} \cdot \left(F_{q,q-M} - F_{q,q-M-1}\right) - \eta_q \cdot R_q \\ &= \eta_{q-1} \cdot (\delta \cdot \Delta_{M-1} - 1) - \eta_q \cdot \delta \cdot \Delta_{M-1} + \eta_{q-M-1} - \eta_{j-1} + \eta_{j-1} - \eta_{j-1} \cdot F_{qj}. \end{aligned}$$

   The lemma follows as $(\eta_j - \eta_{j-1}) \cdot G_{qj} = \eta_{q-M-1} - \eta_{j-1}$ (see (2.31)).

2. $j \in \{q - M, \ldots, q - 1\}$. Then $F_{qj} = 1$, and

$$\begin{aligned} V_{qj} &= \eta_{q-1} \cdot \left(F_{qq} - F_{q,q-1}\right) - \eta_q \cdot R_q \\ &= \eta_{q-1} \cdot (\delta \cdot \Delta_{M-1} - 1) - \eta_q \cdot \delta \cdot \Delta_{M-1} \\ &= \eta_{q-1} \cdot (\delta \cdot \Delta_{M-1} - 1) - \eta_q \cdot \delta \cdot \Delta_{M-1} - \eta_{j-1} \cdot F_{qj} + \eta_{j-1}. \end{aligned}$$

   The lemma follows as $(\eta_j - \eta_{j-1}) \cdot G_{qj} = 0$ (see (2.31)).

3. $j = q$. Then $F_{qj} = \delta \cdot \Delta_{M-1}$, and thus

$$\begin{aligned} V_{qj} &= -\eta_q \cdot R_q \\ &= \eta_{q-1} \cdot \delta \cdot \Delta_{M-1} - \eta_q \cdot \delta \cdot \Delta_{M-1} - \eta_{j-1} \cdot F_{qj} \\ &= \eta_{q-1} \cdot (\delta \cdot \Delta_{M-1} - 1) - \eta_q \cdot \delta \cdot \Delta_{M-1} - \eta_{j-1} \cdot F_{qj} + \eta_{j-1}. \end{aligned}$$

   As in the previous case, the lemma follows as $(\eta_j - \eta_{j-1}) \cdot G_{qj} = 0$. □

Showing Inequality (2.25)**.** We show that (2.25) holds with equality. Using Lemma 2.12, (2.34), and (2.30) yields

$$\begin{aligned} &V_{qj} + \eta_{j-1} \cdot H_{qj} - \eta_j \cdot G_{qj} + C_{L_q} - \textstyle\sum_{\ell \in S(q)} B_\ell \\ &\quad = \eta_q - \eta_{q-1} - \eta_{q+M} - \eta_{j-1} \cdot F_{qj} + (\eta_j - \eta_{j-1}) \cdot G_{qj} + \eta_{j-1} \\ &\qquad - (\eta_j - \eta_{j-1}) \cdot G_{qj} - \eta_{j-1} + \eta_{j-1} \cdot F_{qj} + \eta_{q+M} + \eta_{q-1} = \eta_q. \end{aligned}$$

Showing Inequality (2.26). Using Lemma 2.12, (2.34), and (2.30) yields

$$\begin{aligned}
&V_{qj} + \eta_{j-1} \cdot F_{qj} + B_q \\
&\quad = \eta_q - \eta_{q-1} - \eta_{q+M} - \eta_{j-1} \cdot F_{qj} + (\eta_j - \eta_{j-1}) \cdot G_{qj} + \eta_{j-1} \\
&\qquad + \eta_{j-1} \cdot F_{qj} + \eta_{q+M} - \eta_q + \eta_{q-1} - \eta_{q-M-1} \\
&\quad = (\eta_j - \eta_{j-1}) \cdot G_{qj} + \eta_{j-1} - \eta_{q-M-1} \geq 0.
\end{aligned}$$

where the last inequality follows by (2.32).

Showing inequalities (2.27)–(2.28)

Within this part, we assume that $q \geq Q - M + 1$.

Lemma 2.13. *Fix any $q \geq Q - M + 1$ and $0 \leq j \leq q$. Then,*

$$V_{qj} = \eta_q + (\eta_j - \eta_{j-1}) \cdot G_{qj} - \eta_{j-1} \cdot F_{qj} + \eta_{j-1}.$$

Proof. As $g \geq Q - M + 1$, it holds that $R_q = 0$, and thus (2.29) reduces to

$$V_{qj} = \textstyle\sum_{\ell=j}^{q-1} \eta_\ell \cdot \left(F_{q,\ell+1} - F_{q\ell}\right).$$

As in the proof of Lemma 2.12, to further evaluate V_{qj}, it is useful to trace values $F_{qj}, F_{q,j+1}, \ldots, F_{qq}$ (cf. Figure 2.3), where the increases of these values contribute to V_{qj}. We also note that for $q \geq Q - M + 1$, the possible increases are from 0 to 1 (between $F_{q,q-M-1}$ and $F_{q,q-M}$) and from 1 to ξ_q (between $F_{q,Q-M}$ and $F_{q,Q-M+1}$). We consider three cases.

1. $j \leq q - M - 1$. Then $F_{qj} = 0$, and

$$\begin{aligned}
V_{qj} &= \eta_{Q-M} \cdot \left(F_{qq} - F_{q,q-1}\right) + \eta_{q-M-1} \cdot \left(F_{q,q-M} - F_{q,q-M-1}\right) \\
&= \eta_{Q-M} \cdot \left(\xi_q - 1\right) + \eta_{q-M-1} \\
&= \eta_q + \eta_{q-M-1} - \eta_{j-1} + \eta_{j-1} - \eta_{j-1} \cdot F_{qj}.
\end{aligned}$$

The lemma follows as $(\eta_j - \eta_{j-1}) \cdot G_{qj} = \eta_{q-M-1} - \eta_{j-1}$ (see (2.31)).

2. $j \in \{q - M, \ldots, Q - M\}$. Then $F_{qj} = 1$, and

$$\begin{aligned}
V_{qj} &= \eta_{Q-M} \cdot \left(F_{qq} - F_{q,q-1}\right) \\
&= \eta_{Q-M} \cdot \left(\xi_q - 1\right) \\
&= \eta_q - \eta_{j-1} \cdot F_{qj} + \eta_{j-1}.
\end{aligned}$$

The lemma follows as $(\eta_j - \eta_{j-1}) \cdot G_{qj} = 0$ (see (2.31)).

3. $j \in \{q-M, \dots, Q-M\}$. Then $F_{qj} = \xi_q = 1 + \delta^{q-Q+M}$, and

$$\begin{aligned} V_{qj} = 0 &= \eta_q + \eta_{q+j-Q+M-1} - \eta_q - \eta_{j-1} \cdot (1 + \delta^{q-Q+M}) + \eta_{j-1} \\ &= \eta_q + \eta_{q+j-Q+M-1} - \eta_q - \eta_{j-1} \cdot F_{qj} + \eta_{j-1}. \end{aligned}$$

The lemma follows as $(\eta_j - \eta_{j-1}) \cdot G_{qj} = \eta_{q+j-Q+M-1} - \eta_q$ (see (2.31)). □

Showing Inequality (2.27). We show that (2.27) holds with equality. Using Lemma 2.13 and (2.30), we obtain

$$\begin{aligned} & V_{qj} + \eta_{j-1} \cdot H_{qj} - \eta_j \cdot G_{qj} \\ & \quad = \eta_q + (\eta_j - \eta_{j-1}) \cdot G_{qj} - \eta_{j-1} \cdot F_{qj} + \eta_{j-1} - (\eta_j - \eta_{j-1}) \cdot G_{qj} - \eta_{j-1} + \eta_{j-1} \cdot F_{qj} \\ & \quad = \eta_q. \end{aligned}$$

Showing Inequality (2.28). Using Lemma 2.13, (2.30), and the definition of C_q, we obtain

$$\begin{aligned} & V_{qj} + \eta_{j-1} \cdot F_{qj} - C_q \\ & \quad = \eta_q + (\eta_j - \eta_{j-1}) \cdot G_{qj} - \eta_{j-1} \cdot F_{qj} + \eta_{j-1} + \eta_{j-1} \cdot F_{qj} - \eta_q - \eta_{q-M-1} \\ & \quad = (\eta_j - \eta_{j-1}) \cdot G_{qj} + \eta_{j-1} - \eta_{q-M-1} \geq 0. \end{aligned}$$

where the last inequality follows by (2.32).

2.6 Tightness of the Analysis

The analysis of our algorithms is tight as proven below. For the deterministic one, we additionally show that choosing ω different from 0 does not help.

Theorem 2.2. *For any γ, there are γ-resettable scheduling problems, such that for any $\omega \in (-1, 0]$, the competitive ratio of* Mimic(γ, ω) *is at least $3 + \gamma$.*

Proof. We fix a small $\varepsilon > 0$ and let $\alpha = 2 + \gamma$. The input $\mathcal{I}$ contains two jobs: the first one of weight ε that arrives at time 1, and second one of weight 1 that arrives at time $\alpha^{1+\omega} + \varepsilon$. We assume that there exists a schedule S_1 that serves the first job at the time of its arrival and a schedule S_2 that serves both jobs at the times of their arrivals. Therefore, $\textsc{cost}_{\textsc{Opt}}(\mathcal{I}) = \varepsilon \cdot 1 + 1 \cdot (\alpha^{1+\omega} + \varepsilon) = \alpha^{1+\omega} + 2 \cdot \varepsilon$.

For analyzing the cost of Mimic, note that at at time 1, Mimic observes the first job and learns the value of $\min(\mathcal{I}) = 1$. This is the sole purpose of the first job: setting $\min(\mathcal{I}) = 1$ makes the algorithm miss the opportunity to serve the second job early.

At time $\tau_1 = \alpha^{1+\omega}$, Mimic executes the τ_1-schedule S_1', which is schedule S_1 prolonged trivially to length τ_1. Next, at time $\tau_2 = \alpha^{2+\omega}$, Mimic executes the τ_2-schedule S_2', which is schedule S_2 prolonged trivially to length τ_2. This way it completes the second job at time $\tau_2 + (\alpha^{1+\omega} + \varepsilon)$, and thus $\text{cost}_{\text{Mimic}}(\mathcal{I}) \geq \tau_2 + \alpha^{1+\omega} + \varepsilon = \alpha^{1+\omega} \cdot (1+\alpha) + \varepsilon$. By taking appropriately small $\varepsilon > 0$, the ratio between $\text{cost}_{\text{Mimic}}(\mathcal{I})$ and $\text{cost}_{\text{Opt}}(\mathcal{I})$ becomes arbitrarily close to $1 + \alpha = 3 + \gamma$. □

Theorem 2.3. *For any γ, there are γ-resettable scheduling problems, such that the competitive ratio of a randomized algorithm that runs* Mimic(γ, ω) *with a random $\omega \in (-1, 0]$ is at least $1 + (1+\gamma)/\ln(2+\gamma)$.*

Proof. Let $\alpha = 2 + \gamma$. The input $\mathcal{I}$ contains a single job of weight 1 arriving at time 1. We also assume that for any $\tau \geq 1$, there exists a τ-schedule S_τ that completes this job at time 1. Clearly, $\text{cost}_{\text{Opt}}(\mathcal{I}) = 1 \cdot 1 = 1$.

At time 1, Mimic observes the only job of $\mathcal{I}$ and learns that $\min(\mathcal{I}) = 1$. It sets $\tau_1 = \alpha^{1+\omega}$ and at time τ_1 it executes schedule S_{τ_1}, thus completing the job at time $\tau_1 + 1$. Therefore, $\text{cost}_{\text{Mimic}}(\mathcal{I}) = \int_{-1}^{0} \tau_1 + 1 \; d\omega = \int_{-1}^{0} \alpha^{1+\omega} + 1 \; d\omega = 1 + (\alpha - 1)/\ln \alpha = 1 + (1+\gamma)/\ln(2+\gamma)$. This implies the desired lower bound. □

2.7 Flaw in the Existing Randomized Lower Bound

The authors of [FKW09] claim a lower bound of 3 for randomized k-DARP (for any $k \geq 1$), see Theorem 4 of [FKW09]. Below we show a flaw in their argument.

The construction given in the proof of their Theorem 4 uses Yao min-max principle and is parameterized with a few variables, in particular with an integer m and with a real number $v \in [0, 1]$. Towards the end of the proof, they show that the competitive ratio of any randomized algorithm for the k-DARP problem is at least

$$L_{m,v} = \frac{3m - 4km - 4km^2 + v^{\frac{m+1}{2-m}} \cdot (3 + (4km^2 + 4km + 6m + 6) \cdot v)}{-4 - m - 2km - 2km^2 + v^{\frac{m+1}{2-m}} \cdot (3 + (2km^2 + 2km + 4m + 4) \cdot v)}$$

and they claim that there exists v, such that $L_{m,v} = 3$ when m tends to infinity. However, for any fixed k and any v (also being a function of m), by dividing numerator and denominator by m^2, we obtain that

$$\lim_{m \to \infty} L_{m,v} = \frac{-4k + v^{\frac{m+1}{2-m}} \cdot 4k \cdot v}{-2k + v^{\frac{m+1}{2-m}} \cdot 2k \cdot v} = 2.$$

That is, the proven lower bound is 2 instead of 3.

Chapter 3

Online Service with Delay

3.1 Introduction

In this chapter, we present the result for the Online Service with Delay, defined in Subsection 1.4.3. Essentially, the OSD problem studies a trade-off between serving requests in batches (and saving because they are located close to each other) and minimizing the delays of particular requests. Similar trade-offs occur naturally in many areas of logistics and planning, scheduling or supply chain management.

For a real-life example modeled by this problem, consider a technician (the server) who needs to respond to repair requests from clients (points in the metric space). The speed of the technician is not restricted and once she arrives at the scene, she fixes the problem immediately. The waiting cost function for a request represents its urgency, e.g., it may depend on the importance of a given client.

3.1.1 Previous Work and Related Problems

The first solution for the OSD problem, given by Azar et al. [AGGP17], was an $O(h^3)$-competitive deterministic algorithm for any hierarchically separated tree of depth h. As any metric space on n points can be randomly approximated by an HST of depth $O(\log n)$ with the expected distance distortion of $O(\log n)$ [BBMN15,FRT04], this result yields a randomized $O(\log^4 n)$-competitive algorithm for any metric space. Later, this bound was improved by Azar and Touitou [AT19] who presented an $O(\log^2 n)$-competitive randomized algorithm.

While the OSD problem has been defined only recently, it is closely related to the reordering buffer management (RBM) problem (see, e.g., [ACER11,AR13,ER17,ERS19]) and the multi-level aggregation (MLA) problem [BBB$^+$16,BFNT17,AT19]. The MLA problem can be seen as a special variant of the OSD problem on a tree. In this variant, the server is initially stored at the root. At any time, the server may choose a set of requests, serve them by navigating along a minimum sub-tree spanning these requests and the root, and then return to the root.

In contrast, in the RBM problem, requests do not have waiting costs, but at any time there may be at most b unserved requests. The currently best (randomized) algorithm for general metrics, due to Englert and Räcke [ER17], is $O(\log n \cdot \log b)$-competitive. An important observation that separates the OSD and the RBM problems is the performance of the following "rent-or-buy" strategy. Assume that an online algorithm waits till there is a subset of requests whose total waiting cost becomes equal to the total cost of serving them. An algorithm for OSD that simply serves only these requests would be $\Omega(n)$-competitive even on simple metrics such as weighted stars [AGGP17], while an analogous algorithm for RBM would be $O(h \cdot \log b)$-competitive for trees of depth h [ERW10].

3.1.2 Line Metric: Our Contribution

In this part, we study the OSD problem on a line consisting of n equidistant points. Apart from the theoretical importance, the original motivation for studying such metric comes from minimizing the movement of a hard disk head: in this scenario, a request is a write demand to a particular cylinder of the disk (where a cylinder is represented by a point on a line) [GS09].

Refined results are known for the line metrics both for the MLA and RBM problems. For the RBM problem, Gamzu and Segev constructed a deterministic $O(\log n)$-competitive algorithm Moving Partition [GS09]. Beating this competitive ratio is a long standing challenge: surprisingly, $O(\log n)$ is also the best known approximation ratio for the offline variant and the best known lower bound on the competitive ratio is only 2.154 [GS09]. The MLA problem is better understood in such spaces: Bienkowski et al. presented a 5-competitive algorithm for a line [BBC+13] and no algorithm can beat the ratio of 4 [BBB+16].

In this chapter we present a deterministic $O(\log n)$-competitive algorithm Bckt (short for Bucket) for the OSD problem on a line, that combines ideas from [GS09] and the approach used in [AGGP17, BBB+16].

A bit surprisingly, our algorithm is *non-clairvoyant*: when a request is presented, the algorithm does not need to know its waiting cost function upfront. (We require that the waiting cost functions are continuous, though.) This stands in contrast to the bound presented by Azar et al. for weighted stars [AGGP17]: they show that for such metrics, the competitive ratio of any non-clairvoyant deterministic algorithm is at least $\Omega(\Delta)$, where Δ is the aspect ratio of the metric. (This lower bound holds even if waiting cost functions are continuous.)

3.1.3 Preliminaries

In the OSD problem on a line, the metric space is a line consisting of n equidistant points (each consecutive pair is connected with an edge of length 1). The server of an algorithm starts at a position chosen by the adversary. An input consists of requests and each request is a triple (τ, p, f), where τ is its arrival time, p is the point an algorithm has to visit to serve the request and $f : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is an arbitrary continuous non-decreasing function, such that $f(0) = 0$. If, at time $\tau' \geq \tau$, an algorithm serves the request (τ, p, f), i.e., moves its server to point p, the request incurs the *waiting cost* of $f(\tau' - \tau)$. The *service cost* of an algorithm is defined as the sum of distances traveled by the server. The goal is to serve all requests and minimize the sum of the service cost and all waiting costs.

3.2 Algorithm Bucket

We present a deterministic algorithm Bckt solving the OSD problem for a line metric comprising n equidistant points. Bckt balances service and waiting costs. It works in phases, each consisting of a *waiting subphase* and a *serving subphase*. In the waiting subphase, Bckt does not move the server and waits until there is a group of requests whose overall waiting cost becomes roughly the distance to this group from the current position of the algorithm's server. In the subsequent serving subphase, Bckt serves this group of requests along with its surrounding (to be defined later). The serving subphase is immediate, i.e., the waiting cost is accrued only in the waiting subphases.

3.2.1 Algorithm Definition

More concretely, at the beginning of the waiting subphase, Bckt (re)numbers the points on the line from left to right with consecutive integers, so that the position of its server is at point 0. Next, Bckt splits the line points into $O(\log n)$ *buckets*. For $i \geq 1$, the i-th right (left) bucket consists of 2^{i-1} points that lie to the right (left) from the server's position and whose distances from server's position are in the range $[2^{i-1}, 2^i - 1]$. By B_{+i} and B_{-i} we denote the i-th right and left bucket, respectively; we say that their *indexes* are equal to i. The *size* $|B|$ of a bucket B is the number of points it contains, i.e., $|B_{+i}| = |B_{-i}| = 2^{i-1}$.

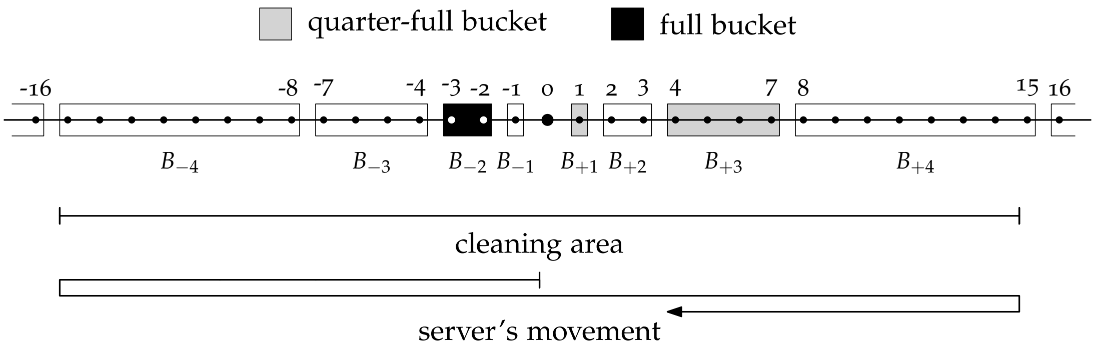


Figure 3.1: An example phase (of label 3) of Bckt. After a waiting subphase, B_{+3} is the critical bucket (the quarter-full bucket with the largest index). Within a serving subphase, Bckt moves its server, so that it visits all the points from the cleaning area and finishes at the closest point of B_{+3}.

Note that the position of the server does not belong to any bucket and all requests arriving at this point are immediately served for free. For each bucket B and time τ, we define its *weight* $w_\tau(B)$ as the total waiting cost incurred by requests in B still pending for Bckt at time τ. We will omit τ and write $w(B)$ whenever it does not lead to ambiguity. We say that a bucket B is *full* if $w(B) \geq |B|$.

Fix any phase S. The waiting subphase lasts until some bucket B becomes full. (As the waiting cost functions are continuous, we may then assume that for such bucket $w(B)$ becomes exactly equal to $|B|$.) At that moment, Bckt considers *quarter-full* buckets, i.e., buckets B' satisfying $w(B') \geq |B'|/4$. Let $B_{\pm r}$ (for $r \geq 1$) be the quarter-full bucket farthest from the server, i.e., the one with the largest index r. We call this bucket *critical* for phase S. (If there are two such buckets, left and right, we pick an arbitrary one of them to be critical.)

On the basis of the critical bucket index r, we define two notions: the *phase label* becomes equal to r and the *cleaning area* $C(S)$ is defined as the region $\biguplus_{j=1}^{r+1}(B_{-j} \uplus B_{+j}) = [-(2^{r+1}-1), 2^{r+1}-1]$. An example is presented in Figure 3.1.

In the serving subphase, Bckt serves all requests pending in the cleaning area $C(S)$. To this end, Bckt chooses its new position to be the closest point of the critical bucket (the point $\pm 2^{r-1}$). Bckt's server follows then the shortest route that visits both endpoints of the cleaning area and ends at point $\pm 2^{r-1}$. For simplicity, we make Bckt visit each point of $C(S)$ even if there is no request waiting at that point. Note that the corresponding service cost is at most twice the size of the cleaning area. A pseudocode of Bckt in a single phase is given in Algorithm 1.

Algorithm 1 Single phase S of the algorithm Bckt.

```
Waiting subphase:
    Number the points relatively to the current server's position.
    Split the line into buckets.
    Wait until there exists a bucket $B$, such that $w(B) = |B|$.

Serving subphase:
    Let $B_{\pm r}$ be the farthest quarter-full bucket.
    /* $B_{\pm r}$ is the critical bucket. Phase $S$ gets label $r$. */
    Let $C(S) = [-(2^{r+1} - 1), 2^{r+1} - 1]$
    if $B_{\pm r}$ is a left bucket then
        /* Go right and then left. */
        Move to $2^{r+1} - 1$ serving all requests on the way.
        Move to $-(2^{r+1} - 1)$ serving all requests on the way.
        Move to $-2^{r-1}$.
    else
        /* Go left and then right. */
        Move to $-(2^{r+1} - 1)$ serving all requests on the way.
        Move to $2^{r+1} - 1$ serving all requests on the way.
        Move to $2^{r-1}$.
    end if
```

3.2.2 Correctness

We start with proving that Bckt is defined properly, i.e., right after a serving phase ends and the algorithm splits the line into new buckets, no bucket is full. Intuitively, buckets that are close to the new server's position are contained in the cleaning area of the serving phase and are now empty. On the other hand, buckets that are far from the server's position are properly contained in two consecutive buckets (of the previous phase) whose weight was small.

Lemma 3.1. *No bucket is full at the beginning of a phase of* Bckt.

Proof. Fix a phase S and let $r \geq 1$ be its label. The cleaning area of the phase is then $C(S) = [-(2^{r+1} - 1), 2^{r+1} - 1]$. Assume, without loss of generality, that the critical bucket of S is the right bucket, $B_{+r} = [2^{r-1}, 2^r - 1]$, i.e., Bckt ends the phase with its server at the point $s = 2^{r-1}$.

We denote the (old) buckets of the serving phase by $B_{\pm i}$ and the buckets of the new waiting phase, i.e., constructed relative to the point s, by $B'_{\pm i}$. We will show that all buckets $B'_{\pm i}$ satisfy $w(B'_{\pm i}) < |B'_{\pm i}|$.

First, we observe that buckets $B'_{-(r+1)}, B'_{-r}, \ldots, B'_{-1}$ and $B'_{+1}, \ldots, B'_{+r}$ are fully contained in the cleaning area $C(S)$. As all the requests in these buckets were served, the weights of these buckets are now equal to 0. Second, all buckets $B'_{\pm i}$ are shifted by 2^{r-1} to the right relative to $B_{\pm i}$. Thus, for any $i \geq r+1$, bucket B'_{+i} is contained in the union $B_{+i} \uplus B_{+(i+1)}$. Since B_{+r} was the farthest quarter-full bucket, the weights of buckets B_{+i} and $B_{+(i+1)}$ are less than a quarter of their sizes. Hence, $w(B'_{+i}) \leq w(B_{+i}) + w(B_{+(i+1)}) < 2^{i-1}/4 + 2^{i}/4 < 2^{i-1} = |B'_{+i}|$. Similarly, for $i \geq r+2$, bucket B'_{-i} is contained in $B_{-i} \uplus B_{-(i-1)}$, i.e., the union of two buckets that are not quarter-full, and therefore $w(B'_{-i}) \leq w(B_{-i}) + w(B_{-(i-1)}) < 2^{i-1}/4 + 2^{i-2}/4 < 2^{i-1} = |B'_{-i}|$. This concludes the proof. □

3.3 Competitiveness

We proceed to prove that Bckt is $O(\log n)$-competitive, where n is the number of points on the line. In our reasoning, we do not aim at minimizing the constants, but rather at the simplicity of the argument.

3.3.1 Waiting and Service Costs

We start by showing that our algorithm balances its waiting and service costs, which allows us to focus only on bounding the latter. By $\text{Bckt}_{\text{wait}}$ and $\text{Bckt}_{\text{serv}}$ we denote, respectively, the waiting and the service costs of Bckt.

Lemma 3.2. *It holds that* $\text{Bckt}_{\text{wait}} \leq \text{Bckt}_{\text{serv}}$.

Proof. Fix any phase S and let $r \geq 1$ be its label, i.e., the cleaning area of the phase is equal to $C(S) = [-(2^{r+1}-1), 2^{r+1}-1]$. As the server starts in the middle of the cleaning area and has to to visit its both endpoints, the service cost incurred in the serving subphase is at least $3 \cdot (2^{r+1}-1) > 2 \cdot 2^{r+1}$.

On the other hand, by the definition of Bckt, the weight of each bucket contained in $C(S)$ is at most the size of this bucket. Hence, the total waiting cost incurred by requests served in phase S is $\sum_{i=1}^{r+1}(w(B_{-i}) + w(B_{+i})) \leq \sum_{i=1}^{r+1}(|B_{-i}| + |B_{+i}|) < 2 \cdot 2^{r+1}$.

Each request is eventually served by Bckt, and therefore summing the waiting and service costs of all phases yields the lemma. □

By Lemma 3.2, it suffices to bound $\text{BCKT}_{\text{serv}}$, the total distance traveled by the BCKT's server. In our proof later we would like to use the following local argument: "if our algorithm traverses an edge, then it moves towards requests (whose total waiting cost was sufficiently large)". Unfortunately, this is not true for all edges from the route traversed by BCKT, especially for the edges near the borders of the cleaning area.

Therefore (for the analysis purposes only), we define the following "virtual" algorithm BCKT_{eff} that operates almost in the same way as BCKT does, and we analyze BCKT_{eff} instead of BCKT. BCKT_{eff} has identical notion of waiting subphases and buckets as BCKT. The only difference is that in the serving subphase, BCKT_{eff} moves the server directly to the closest point of the critical bucket (i.e., to the final position of BCKT's server in the serving subphase).

We will pretend that such server movement of BCKT_{eff} serves all the pending requests in the cleaning area $C(S)$, i.e., serves the same set of requests as BCKT does. Furthermore, we will assume that BCKT_{eff} is charged only for server movement, i.e., it does not pay for the waiting costs. It can be easily observed that replacing BCKT by BCKT_{eff} changes the cost at most by a constant factor.

Lemma 3.3. *It holds that* $\text{BCKT} \leq 32 \cdot \text{BCKT}_{\text{eff}}$.

Proof. Consider a single phase S and let $r \geq 1$ be its label. The cost of BCKT_{eff} on this phase is 2^{r-1} while BCKT pays at most $2 \cdot 2^{r+2}$ for the server movement (the server visits each point of the cleaning area at most twice). This implies that $\text{BCKT}_{\text{serv}} \leq 16 \cdot \text{BCKT}_{\text{eff}}$ for a single phase, and hence also for the entire input sequence. Combining this relation with Lemma 3.2 concludes the proof. □

3.3.2 Critical Requests and Freshness Property

Fix any phase S and let r be its label. The set of requests for phase S which is served in the critical bucket $B_{\pm r}$ is called a *critical set* and denoted $R(S)$. The following properties of critical sets will become useful once we define our charging scheme in Subsection 3.3.4.

Lemma 3.4. *Fix any edge e and consider two phases S and S' of the same label r during which the server of* BCKT_{eff} *moves along e in the same direction. The following two properties hold.*

Weight property. *The total waiting cost of requests from $R(S')$ is at least 2^{r-3}.*

Freshness property. *All requests from $R(S')$ appeared after phase S ended.*

Proof. The first condition of the lemma follows immediately by the definition of the algorithm: $R(S')$ is the set of requests from the bucket $B'_{\pm r}$, which is critical and hence quarter-full. That is, $w(B_{\pm r}) \geq |B_{\pm r}|/4 = 2^{r-3}$.

For the second condition, we assume, without loss of generality, that within both phases $\textsc{Bckt}_{\text{eff}}$ moves to the right. Let B_{+r} and B'_{+r} be the critical buckets in phases S and S', respectively. Observe that $C(S)$, the cleaning area of S, covers all edges that $\textsc{Bckt}_{\text{eff}}$ traverses and also $2^{r+1} - 2^{r-1}$ points to the right of this path. Hence, $C(S)$ covers at least $2^{r+1} - 2^{r-1}$ points to the right of edge e.

In phase S', the bucket B'_{+r} is also to the right of edge e and the distance between e and the farthest point of B'_{+r} is at most 2^r. Hence, $B'_{+r} \subseteq C(S)$, which means that all the requests at points from B'_{+r} are served in phase S. Therefore, all the requests from B'_{+r} that are served by $\textsc{Bckt}_{\text{eff}}$ during phase S' must have arrived after the end of phase S. □

3.3.3 Moving Towards and Away from Opt

From now on, we fix any optimal solution $\textsc{Opt}$ and analyze both $\textsc{Bckt}_{\text{eff}}$ and $\textsc{Opt}$ running on the same input simultaneously. Roughly speaking, we would like to argue that if $\textsc{Bckt}_{\text{eff}}$ traverses an edge e many times, then $\textsc{Opt}$ has to either move along e or pay large waiting costs. However, if $\textsc{Bckt}_{\text{eff}}$ traverses edge e towards $\textsc{Opt}$, we cannot say anything about the relative position of $\textsc{Opt}$ and the requests $\textsc{Bckt}_{\text{eff}}$ serves (it might even happen that $\textsc{Opt}$ serves them for free). Therefore, we split the moves of $\textsc{Bckt}_{\text{eff}}$ into $\textsc{Bckt}_{\text{away}}$— the moves in the direction away from the current position of $\textsc{Opt}$'s server and $\textsc{Bckt}_{\text{toward}}$— the moves towards it. Note that $\textsc{Bckt}_{\text{away}}$ and $\textsc{Bckt}_{\text{toward}}$ are defined only in the analysis. It turns out that by focusing only on $\textsc{Bckt}_{\text{away}}$ we lose only a constant factor in the competitive ratio.

Lemma 3.5. *It holds that* $\textsc{Bckt}_{\text{toward}} \leq \textsc{Bckt}_{\text{away}} + \textsc{Opt}$.

Proof. Fix any edge e. Let $\textsc{Bckt}_{\text{toward}}(e)$ and $\textsc{Bckt}_{\text{away}}(e)$ be the total cost of moves of $\textsc{Bckt}_{\text{eff}}$'s server along edge e in the directions toward and away the $\textsc{Opt}$'s server, respectively. Let $\textsc{Opt}(e)$ be the total costs of $\textsc{Opt}$ traversals of e. It is sufficient to show that

$$\textsc{Bckt}_{\text{toward}}(e) \leq \textsc{Bckt}_{\text{away}}(e) + \textsc{Opt}(e). \tag{3.1}$$

To prove this inequality, we analyze how its both sides evolve in time. Clearly, at the beginning, both sides are equal to zero. Since $\textsc{Bckt}_{\text{eff}}$ and $\textsc{Opt}$ start at the same side of e, before $\textsc{Bckt}_{\text{eff}}$ moves toward $\textsc{Opt}$ for the first time (i.e., the left hand side of (3.1) increases for the first time), either $\textsc{Bckt}_{\text{eff}}$ needs to move away from $\textsc{Opt}$ along e or $\textsc{Opt}$ needs to traverse e (i.e., the right hand side of (3.1) increases for the first time).

Furthermore, between any two consecutive increases of $\text{Bckt}_{\text{toward}}(e)$ (two consecutive increments of the left hand side of (3.1)) either Bckt_{eff} traverses e in the direction away from Opt or Opt traverses e (the right hand side of (3.1) increments). □

3.3.4 Charging Scheme

The plan for the remaining part of the analysis is as follows. We look at any increment of the value $\text{Bckt}_{\text{away}}$ over time. It corresponds to a traversal of some edge e performed by the server of Bckt_{eff} in the direction away from Opt. We call any such movement an *away-traversal of edge e*.

We will map (charge) all away-traversals to some action(s) of Opt. Some away-traversals will be charged directly to the moves of Opt's server. Others will be charged to the waiting of requests; we will show that these requests incurred sufficiently large cost in the solution of Opt. Then, we will estimate, for any action of Opt, how many away-traversals are mapped to them.

To formally define the charging, we now focus entirely on a single edge e. We split the execution of both Bckt_{eff} and Opt into e-epochs using the moves of Opt along e: when Opt traverses edge e in any direction, an e-epoch ends and the next one begins. Fix any such e-epoch. Assume, without loss of generality, that in this e-epoch the server of Opt remains on the left side of edge e. In this case, all away-traversals of e within the e-epoch are movements of Bckt_{eff}'s server to the right. Any such away-traversal of e is a part of a path traveled by Bckt_{eff} in the serving subphase of S. The away-traversal receives the same label that is assigned to phase S.

In a given e-epoch, for each label value r, we will charge away-traversals of e in the following way.

The first away-traversal of e **with label** r**.** We charge this away-traversal either to the movement of Opt's server along e that initiates the current e-epoch or to the movement that finishes it. Such move always exists as Opt eventually serves all requests.

Subsequent away-traversals of e **with label** r**.** Recall that this away-traversal is a part of a movement performed by the Bckt_{eff}'s server within a phase S. We charge the away-traversal to the waiting of the requests from the critical set $R(S)$ (in the solution of Opt).

An example of our charging scheme is given in Figure 3.2. Using the charging scheme, we may finally relate the cost of $\text{Bckt}_{\text{away}}$ to the cost of Opt. Let Opt_{serv} and Opt_{wait} be the total service and waiting costs of Opt, respectively.

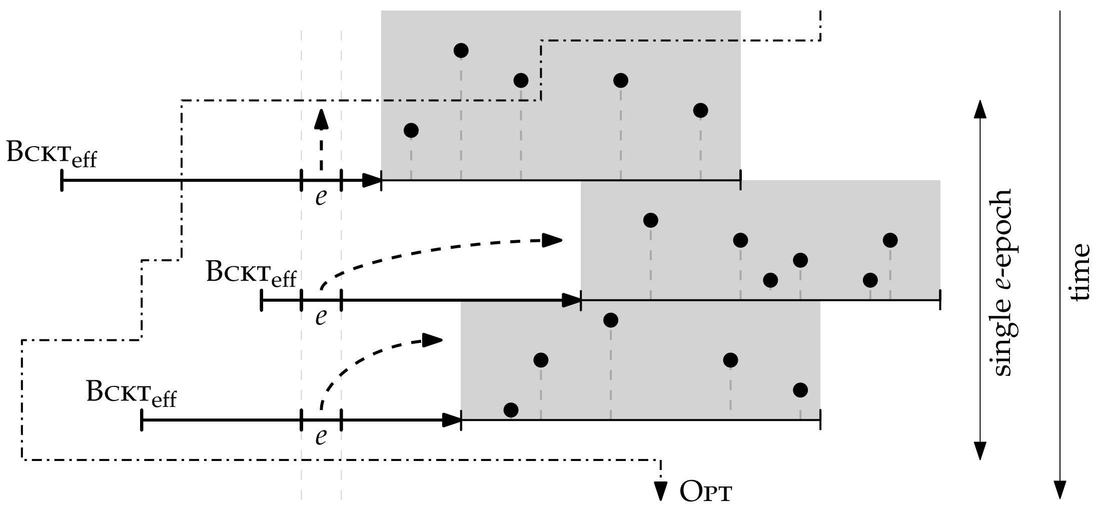


Figure 3.2: Illustration of the charging scheme for away-traversals of edge e performed by the algorithm BCKT$_{\text{eff}}$. The figure contains a single e-epoch defined by the times at which OPT traversed edge e. The route of OPT's server is drawn as a dash-dotted line. All depicted away-traversals of edge e performed by BCKT$_{\text{eff}}$ have the same label r: each corresponding movement of BCKT$_{\text{eff}}$ (thick right arrows) consists of 2^{r-1} away-traversals. The shaded regions contain critical sets of requests (represented by small discs) that are served by BCKT$_{\text{eff}}$ at the ends of the phases in critical buckets (of sizes 2^{r-1}). Dashed arrows denote our charging of away-traversals.

Lemma 3.6. *The number of away-traversals mapped to the movements of* OPT *is at most* $O(\log n) \cdot \text{OPT}_{\text{serv}}$.

Proof. We fix any traversal q of OPT's server of some edge e. Note that q finishes one e-epoch $\mathcal{E}$ and initiates another e-epoch $\mathcal{E}'$. For any label r, only the first away-traversals of edge e with label r from e-epochs $\mathcal{E}$ and $\mathcal{E}'$ may charge to q.

The number of different labels is upper bounded by the number of possible labels for phases, that is, by $O(\log n)$. Hence, the number of different away-traversals that may charge to q is at most $2 \cdot O(\log n)$. The lemma follows by summing over all edge traversals performed by OPT. □

Lemma 3.7. *The number of away-traversals mapped to the waiting of requests in the solution of* OPT *is at most* $4 \cdot \text{OPT}_{\text{wait}}$.

Proof. We now fix any phase S of the algorithm BCKT$_{\text{eff}}$, let r be its label and let $R(S)$ be the corresponding critical set of requests (served in the serving subphase of S). For succinctness, we define the waiting cost of $R(S)$ as the total waiting cost of all requests from $R(S)$. We will now claim that the number of away-traversals that charge to $R(S)$ is at most 4 times larger than the waiting cost of $R(S)$ incurred by OPT. The lemma will then follow by summing over all critical set of requests (they are clearly disjoint).

If no away-traversals charge to $R(S)$, then the claim follows trivially.

We may therefore assume that there exists an away-traversal of some edge e that charges to $R(S)$. Without loss of generality, we assume that $\textsc{Bckt}_{\text{eff}}$ traverses e to the right, and within the corresponding phase $\textsc{Opt}$ remains to the left of edge e. Note that the label of the away-traversal has to be the same as the label of phase S, i.e., equal to r. By the definition of our charging, the preceding away-traversal of edge e with label r belongs to the same e-epoch; let S_p be the phase during which this preceding away-traversal is performed. By the freshness property of Lemma 3.4, all requests from the critical set $R(S)$ arrived after S_p. By the weight property of the same lemma, the waiting cost of $R(S)$ incurred on $\textsc{Bckt}_{\text{eff}}$ is at least 2^{r-3}.

As $\textsc{Opt}$ remains on the left side of edge e for the whole e-epoch, it cannot serve the requests of $R(S)$ earlier than $\textsc{Bckt}_{\text{eff}}$, and therefore the waiting cost of $R(S)$ incurred on $\textsc{Opt}$ is also at least 2^{r-3}. The claim follows by observing that only edges belonging to the $\textsc{Bckt}_{\text{eff}}$'s server movement in phase S (2^{r-1} edges) may charge to $R(S)$. □

Recall that $\textsc{Bckt}_{\text{away}}$ is the set of all away-traversals. As all away-traversals are mapped by our charging scheme, using Lemma 3.6 and Lemma 3.7, we immediately obtain the following corollary.

Corollary 3.1. *It holds that* $\textsc{Bckt}_{\text{away}} \leq 4 \cdot \textsc{Opt}_{\text{wait}} + O(\log n) \cdot \textsc{Opt}_{\text{serv}}$.

3.3.5 The Competitive Ratio

Theorem 3.1. $\textsc{Bckt}$ *is* $O(\log n)$*-competitive.*

Proof. The theorem follows by a straightforward combination of the definitions and lemmas proven in this section.

$$\begin{aligned}
\textsc{Bckt} &\leq 32 \cdot \textsc{Bckt}_{\text{eff}} && \text{(by Lemma 3.3)}\\
&= 32 \cdot (\textsc{Bckt}_{\text{toward}} + \textsc{Bckt}_{\text{away}}) \\
&\leq 32 \cdot (2 \cdot \textsc{Bckt}_{\text{away}} + \textsc{Opt}) && \text{(by Lemma 3.5)}\\
&= O(1) \cdot \textsc{Opt}_{\text{wait}} + O(\log n) \cdot \textsc{Opt}_{\text{serv}} && \text{(by Corollary 3.1)}\\
&= O(\log n) \cdot \textsc{Opt}
\end{aligned}$$

□

Chapter 4

Online Matching with Delays

4.1 Introduction

In this chapter we focus on Min-cost Perfect Matching with Delays problem (MPMD), shortly described in Subsection 1.4.4. One of the motivations for this problem are gaming platforms that host two-player games, such as chess, go or Scrabble, where participants are joining in real time, each wanting to play against another human player. There, players want to compete against opponents with comparable skills, but they do not want to wait for them for a long time. Therefore, a matching mechanism has to balance two conflicting objectives: to minimize the players waiting time and to minimize dissimilarities of skills between matched players.

We also solve the bipartite variant of MPMD, called Min-cost Bipartite Perfect Matching with Delays (MBPMD), where requests have polarities and only requests of different polarities can be matched together. This setting corresponds to a variety of real-life scenarios, e.g., assigning drivers to passengers on ride-sharing platforms or matching patients to donors in kidney transplants. Similarly to the MPMD problem, there is a trade-off between minimizing the waiting time and finding a better match (a closer driver or a more compatible donor).

In this chapter, we present a *deterministic* algorithm that works for both MPMD and MBPMD problems. Our main tool for an algorithm and its analysis is the primal-dual technique (see Subsection 1.5.4 for an overview).

4.1.1 Problem Definition

Formally, both in the MPMD and MBPMD problems, there is a metric space $\mathcal{X}$ equipped with a distance function $\mathsf{dist} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$, both known in advance to an online algorithm. An online part of the input is a sequence of $2m$ requests

$u_1, u_2, \ldots, u_{2m}$. A request (e.g., a player arrival) u is a triple $u = (\mathsf{pos}(u), \mathsf{atime}(u), \mathsf{sgn}(u))$, where $\mathsf{atime}(u)$ is the arrival time of request u, $\mathsf{pos}(u) \in \mathcal{X}$ is the request location, and $\mathsf{sgn}(u)$ is the polarity of the request.

In the bipartite case, half of the requests are positive and $\mathsf{sgn}(u) = +1$ for any such request u; the remaining half are negative and there $\mathsf{sgn}(u) = -1$. In the non-bipartite case, requests do not have polarities, but for technical convenience we set $\mathsf{sgn}(u) = 0$ for any request u.

In applications described above, the function dist measures the dissimilarity of a given pair of requests (e.g., discrepancy between player capabilities in the gaming platform scenario or the physical distance between a driver and a passenger in the ride-sharing platform scenario). For instance, for chess, a player is commonly characterized by her Elo rating (an integer) [Elo78]. In such case, $\mathcal{X}$ may be simply a set of all integers with the distance between two points defined as the difference of their values.

Requests arrive over time, i.e., $\mathsf{atime}(u_1) \leq \mathsf{atime}(u_2) \leq \cdots \leq \mathsf{atime}(u_{2m})$. We note that the integer m is not known beforehand to an online algorithm. At any time τ, an online algorithm may match a pair of requests (players) u and v that

- have already arrived ($\tau \geq \mathsf{atime}(u)$ and $\tau \geq \mathsf{atime}(v)$),
- have not been matched yet,
- satisfy $\mathsf{sgn}(u) = -\mathsf{sgn}(v)$ (i.e., have opposite polarities in the bipartite case; in the non-bipartite case, this condition trivially holds for any pair).

The cost incurred by such *matching edge* is then $\mathsf{dist}(\mathsf{pos}(u), \mathsf{pos}(v)) + (\tau - \mathsf{atime}(u)) + (\tau - \mathsf{atime}(v))$. That is, it is the sum of the *connection cost* defined as $\mathsf{dist}(\mathsf{pos}(u), \mathsf{pos}(v))$ and the *waiting costs* of u and v, defined as $\tau - \mathsf{atime}(u)$ and $\tau - \mathsf{atime}(v)$, respectively. The goal is to eventually match all requests and minimize the total cost of all matching edges.

4.1.2 Previous Work

As already mentioned in the introduction (Subsection 1.4.4), denoting the number of points in the metric space $\mathcal{X}$ by n, the best known competitive ratio for randomized MPMD and MBPMD are equal to $O(\log n)$ [ACK17, AAC+17] and the currently best lower bounds are $\Omega(\log n / \log \log n)$ and $\Omega(\sqrt{\log n / \log \log n})$, respectively [AAC+17]. These lower bounds use $O(n)$ requests. Therefore, no randomized algorithm can achieve a competitive ratio lower than $\Omega(\log m / \log \log m)$ in the non-bipartite case and lower than $\Omega(\sqrt{\log m / \log \log m})$ in the bipartite one. (Recall that $2m$ is the number of requests in the input.)

The status of the achievable performance of *deterministic* solutions is far from being resolved. No better lower bounds than the ones used for randomized settings are known for deterministic algorithms. The first solution that worked for general metric spaces was given by Bienkowski et al. and achieved a quite high competitive ratio of $O(m^{\log_2 5.5}) = O(m^{2.46})$ [BKS17]. Roughly speaking, their algorithm is based on growing spheres around not-yet-paired requests. Each sphere is created upon a request arrival, grows with time, and when two spheres touch, the corresponding requests become matched.

In this chapter, we improve this result presenting an $O(m)$-competitive algorithm using the primal-dual approach. Our result was subsequently improved by Azar and Jacob-Fanani [AJF18], who presented an $O((1/\varepsilon) \cdot m^{\log(3/2+\varepsilon)})$-competitive algorithm, where $\varepsilon > 0$ is a parameter of their algorithm. When ε is small enough, this ratio becomes $O(m^{0.58})$. Their algorithm is also based on growing spheres, but they slightly modify the criteria of request connection and choose a different balance between waiting time and connection cost.

Better deterministic algorithms are known only for simple spaces: Azar et al. [ACK17] gave an $O(\text{height})$-competitive algorithm for trees and Emek et al. [ESW17] constructed a 3-competitive deterministic solution for two-point metrics (the latter competitive ratio is best possible).

4.1.3 Challenges and Used Techniques

In contrast to the previous randomized solutions of these problems [AAC+17, ACK17, EKW16], and similarly to other deterministic solutions [AJF18, BKS17], we do not need the metric space $\mathcal{X}$ to be finite and known in advance by an online algorithm. (All previous randomized solutions started by approximating $\mathcal{X}$ by a random HST (hierarchically separated tree) [FRT04] or a random HST tree with reduced height [BBMN15].) This approach, which can be performed only in the randomized setting, greatly simplifies the task as the underlying tree metric reveals a lot of structural information about the cuts between points of $\mathcal{X}$ and hence about the structure of an optimal solution. In the deterministic setting, such information has to be gradually learned as time passes. For our algorithm, we require only that, together with any request u, it learns the distances from u to all previous requests.

In contrast to the other deterministic algorithms [AJF18, BKS17], we base our algorithm on the moat-growing framework, which is a particular instance of the primal-dual approach described in Subsection 1.5.4. This framework, introduced in [JP95] was later successfully used to give 2-approximation algorithm for the Steiner forest problem [GW95] and other Steiner tree variants.

Glossing over a lot of details, in this framework, the primal program has a constraint (connectivity requirement) for any subset of requests and the dual program has a variable for any such subset. The algorithm maintains a family of active sets, which are initially singletons. In the runtime, dual variables are increased simultaneously, till some dual constraint (corresponding to a pair of requests) becomes tight: in such case an algorithm connects such pair and merges the corresponding sets. At the end, an algorithm usually performs pruning by removing redundant edges.

When one tries to adapt the moat-growing framework to online setting, the main difficulty stems from the irrevocability of the pairing decision: the pruning operation performed at the end is no longer an option. Another difficulty is that an algorithm has to combine the concept of *actual* time that passes in an online instance with the *virtual* time that dictates the growth of dual variables. In particular, dual variables may only start to grow once an online algorithm learns about the request and not from the very beginning as they would do in the offline setting. Finally, requests appear online, and hence both primal and dual programs evolve in time. For instance, this means that for badly defined algorithms, appearing dual constraints may be violated already once they are introduced.

We note that $2m$ (the number of requests) is incomparable with n (the number of different points in the metric space $\mathcal{X}$) and their relation depends on the application. Our algorithm is better suited for applications, where $\mathcal{X}$ is infinite or virtually infinite (e.g., it corresponds to an Euclidean plane or a city map for ride-sharing platforms [LVJ16]) or very large (e.g., for some real-time online games, where player capabilities are represented as multi-dimensional vectors describing their rank, reflex, offensive and defensive skills, etc. [ASvZ13]).

4.1.4 Related Work

Originally, online metric matching problems have been studied in variants where delaying decisions was not permitted. In this variant, m requests with positive polarities are given at the beginning to an algorithm. Afterwards, m requests with negative polarities are presented one by one to an algorithm and they have to be matched *immediately* to existing positive requests. The goal is to minimize the weight of a perfect matching created by the algorithm. For general metric spaces, the best deterministic algorithms achieve the optimal competitive ratio of $2m - 1$ [KP93, KMV94, Rag16] and the best randomized solution is $O(\log^2 m)$-competitive [BBGN14, MNP06]. Better bounds (both deterministic and randomized) of $O(\log m)$ are known for line metrics [ABN+14, GL12, KN03, NR17, Rag18]

Another strand of research concerning online matching problems arose around a non-metric setting where points with different polarities are connected by graph edges and the goal is to maximize the cardinality or the weight of the produced matching. For a comprehensive overview of this type of problems we refer the reader to a survey by Mehta [Meh13].

4.2 Primal-Dual Formulation

We start with introducing a linear program that allows us to lower-bound the cost of an optimal solution. To this end, fix an instance $\mathcal{I}$ of M(B)PMD. Let V be the set of all requests. We call any unordered pair of different requests in $\mathcal{I}$ an edge; let E be the set of all edges that correspond to potential matching pairs, i.e., the set of all edges in the non-bipartite case, and the edges that connect requests of opposite polarities in the bipartite variant. For each set $S \subseteq V$, by $\delta(S)$ we denote the set of all edges from E crossing the boundary of S, i.e., having exactly one endpoint in S.

For any set $S \subseteq V$, we define $\mathsf{sur}(S)$ (*surplus* of set S) as the number of unmatched requests in a maximum cardinality matching of requests within set S.

- In the non-bipartite variant (MPMD), we are allowed to match any two requests. Hence, if S is of even size, then $\mathsf{sur}(S) = 0$. Otherwise, $\mathsf{sur}(S) = 1$ as in any maximum cardinality matching of requests within S exactly one request remains unmatched.

- In the bipartite variant (MBPMD), we can always match two requests of different polarities. Hence, the surplus of a set S is the discrepancy between the number of positive and negative requests inside S, i.e., $\mathsf{sur}(S) = |\sum_{u \in S} \mathsf{sgn}(u)|$.

To describe a matching, we use the following notation. For each edge e, we introduce a binary variable x_e, such that $x_e = 1$ if and only if e is a matching edge. For any set $S \subseteq V$ and any feasible matching (in particular the optimal one), it holds that $\sum_{e \in \delta(S)} x_e \geq \mathsf{sur}(S)$.

Fix an optimal solution Opt for $\mathcal{I}$. If a pair of requests $e = (u, v)$ is matched by Opt, it is matched as soon as both u and v arrive, and hence the cost of matching u with v in the solution of Opt is equal to $\mathsf{opt\text{-}cost}(e) := \mathsf{dist}(\mathsf{pos}(u), \mathsf{pos}(v)) + |\mathsf{atime}(u) - \mathsf{atime}(v)|$. This, together with the preceding observations, motivates the following linear program

$\mathcal{P}$:

$$\begin{aligned}
\text{minimize} \quad & \sum_{e \in E} \mathsf{opt\text{-}cost}(e) \cdot x_e & \\
\text{subject to} \quad & \sum_{e \in \delta(S)} x_e \geq \mathsf{sur}(S) & \forall S \subseteq V \\
& x_e \geq 0 & \forall e \in E.
\end{aligned}$$

As any matching is a feasible solution to $\mathcal{P}$, the cost of the optimal solution of $\mathcal{P}$ lower-bounds the cost of the optimal solution for instance $\mathcal{I}$ of M(B)PMD. Note that there might exist a feasible integral solution of $\mathcal{P}$ that does not correspond to any matching. To exclude all such solutions, we could add constraints $\sum_{e \in \delta(S)} x_e = 1$ for all singleton sets S. The resulting linear program would then exactly describe the matching problem (cf. Chapter 25 of [Sch03]). However, our main concern is not $\mathcal{P}$, but its dual and its current shape is sufficient for our purposes. The program $\mathcal{D}$, dual to $\mathcal{P}$, is then

$$\begin{aligned}
\text{maximize} \quad & \sum_{S \subseteq V} \mathsf{sur}(S) \cdot y_S & \\
\text{subject to} \quad & \sum_{S : e \in \delta(S)} y_S \leq \mathsf{opt\text{-}cost}(e) & \forall e \in E \\
& y_S \geq 0 & \forall S \subseteq V.
\end{aligned}$$

Note that in any solution, the dual variables y_S corresponding to sets S for which $\mathsf{sur}(S) = 0$, can be set to 0 without changing feasibility or objective value.

The following lemma is an immediate consequence of weak duality (cf. Theorem 1.1).

Lemma 4.1. *Fix any instance $\mathcal{I}$ of the M(B)PMD problem. Let $\textsc{Opt}(\mathcal{I})$ be the value of any optimal solution of $\mathcal{I}$ and D be the value of any feasible solution of $\mathcal{D}$. Then $\textsc{Opt}(\mathcal{I}) \geq D$.*

Proof. Let P^* and D^* be the values of optimal solutions for $\mathcal{P}$ and $\mathcal{D}$, respectively. Since any matching is a feasible solution for $\mathcal{P}$, $\textsc{Opt}(\mathcal{I}) \geq P^*$. Hence, $\textsc{Opt}(\mathcal{I}) \geq P^* \geq D^* \geq D$. □

Lemma 4.1 motivates the following approach: We construct an online algorithm Greedy Dual (GD), which, along with its own solution, maintains a feasible solution D for $\mathcal{D}$ corresponding to the already seen part of the input instance. This feasible dual solution not only yields a lower bound on the cost of the optimal matching, but also plays a crucial role in deciding which pair of requests should be matched.

Note that since the requests arrive in an online manner, $\mathcal{D}$ evolves in time. When a request arrives, the number of subsets of V increases (more precisely, it doubles), and hence more dual variables y_S are introduced. Moreover, the newly arrived request creates an edge with every existing request and the corresponding dual constraints are introduced. Therefore, showing the feasibility of the created dual solution is not immediate; we deal with this issue in Section 4.4.

4.3 Algorithm Greedy Dual

The high-level idea of our algorithm is as follows: Greedy Dual (GD) resembles moat-growing algorithms for solving constrained forest problems [GW95]. During its runtime, GD partitions all the requests that have already arrived into *active sets*.[1] If an active set contains any free requests, we call this set *growing*, and *non-growing* otherwise. At any time, for each active growing set S, the algorithm increases continuously its dual variable y_S until a constraint in $\mathcal{D}$ corresponding to some edge (u, v) becomes tight. When it happens, GD makes both active sets (containing u and v, respectively) inactive, and the set being their union active. In addition, if this happened due to two growing sets, GD matches as many pairs of free requests in these sets as possible: in the non-bipartite variant GD matches exactly one pair of free requests, while in the bipartite variant, GD matches free requests of different polarities until all remaining free requests have the same sign.

4.3.1 Algorithm Description

More precisely, at any time, GD partitions all requests that arrived until that time into *active* sets. It maintains mapping $\mathcal{A}$, which assigns an active set to each such request. An active set S, whose all requests are matched is called *non-growing*. Conversely, an active set S is called *growing* if it contains at least one free request. GD ensures that the number of free requests in an active set S is always equal to $\mathsf{sur}(S)$. We denote the set of free requests in an active set S by $\mathsf{free}(S)$; if S is non-growing, then $\mathsf{free}(S) = \emptyset$.

When a request u arrives, the singleton $\{u\}$ becomes a new active and growing set, i.e., $\mathcal{A}(u) = \{u\}$. The dual variables of all active growing sets are increased continuously with the same rate in which time passes. This increase takes place until a dual constraint between two active sets becomes tight, i.e., until there exists at least one edge $e = (u, v)$, such that

$$\mathcal{A}(u) \neq \mathcal{A}(v) \quad \text{and} \sum_{S : e \in \delta(S)} y_S = \mathsf{opt\text{-}cost}(e). \tag{4.1}$$

[1] A reader familiar with the moat-growing algorithm may think that active sets are moats. However, not all of them are growing in time.

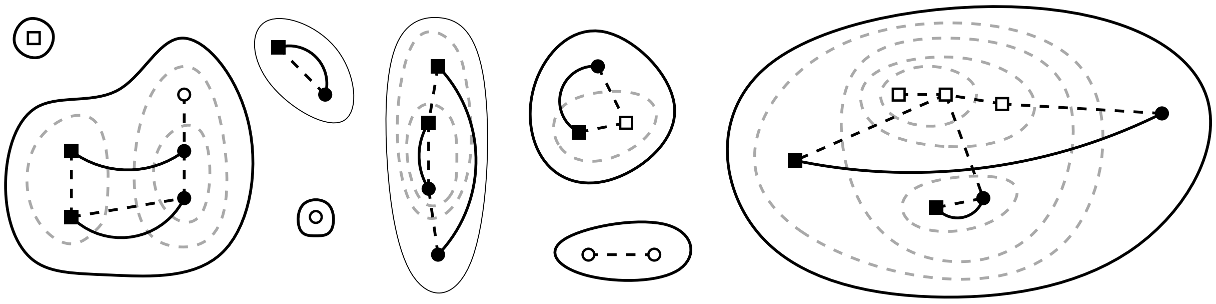


Figure 4.1: A partition of requests into active sets created by GD. Different polarities of requests are represented by discs and squares. Free requests are depicted as empty discs and squares, matched requests by filled ones. Active growing sets have bold boundaries and each of them contains at least one free request. Active non-growing sets contain only matched requests. Dashed lines represent marked edges and solid curvy lines represent matching edges. Dashed gray sets are already inactive; the inactive singleton sets have been omitted.

In such case, while there exists an edge $e = (u, v)$ satisfying (4.1), GD processes such edge in the following way. First, it *merges* active sets $\mathcal{A}(u)$ and $\mathcal{A}(v)$. By merging we mean that the mapping $\mathcal{A}$ is adjusted to the new active set $S = \mathcal{A}(u) \uplus \mathcal{A}(v)$ for each request of S. Old active sets $\mathcal{A}(u)$ and $\mathcal{A}(v)$ become *inactive*.[2] Second, as long as there is a pair of free requests $u', v' \in S$ that can be matched with each other, GD matches them.

In the non-bipartite variant, GD matches at most one pair as each active set contains at most one free request. In the bipartite variant, GD matches pairs of free requests until all unmatched requests in S (possibly zero) have the same polarity. Observe that in either case, the number of free requests after merge is equal to $\mathsf{sur}(S)$. Finally, GD *marks* edge e. Marked edges are used in the analysis, to find a proper charging of the connection cost to the cost of the produced solution for $\mathcal{D}$. The pseudocode of GD is given in Algorithm 2 and an example execution that shows a partition of requests into active sets is given in Figure 4.1.

4.3.2 Greedy Dual Properties

It is instructive to trace how the set $\mathcal{A}(u)$ changes in time for a request u. At the beginning, when u arrives, $\mathcal{A}(u)$ is just the singleton set $\{u\}$. Then, the set $\mathcal{A}(u)$ is merged at least once with another active set. If $\mathcal{A}(u)$ is merged with a non-growing set, the number of requests in $\mathcal{A}(u)$ increases, but its surplus remains intact. After $\mathcal{A}(u)$ is merged with a growing set, some requests inside the new $\mathcal{A}(u)$ may become matched. It is possible that, in effect, the surplus of the new set $\mathcal{A}(u)$ is zero, in which

[2]Note that *inactive* is not the opposite of being *active*, but means that the set was active previously: some sets are never active or inactive.

Algorithm 2 Algorithm Greedy Dual

```
1:  **Request arrival event:**
2:      **if** a request $u$ arrives **then**
3:          $\mathcal{A}(u) \leftarrow \{u\}$
4:          **for all** sets $S$ such that $u \in S$ **do**
5:              $y_S \leftarrow 0$                    ▷ initialize dual variables for sets containing $u$
6:          **end for**
7:      **end if**
8:
9:  **Tight constraint event:**
10:     **while** exists a tight dual constraint for edge $e = (u, v)$ where $\mathcal{A}(u) \neq \mathcal{A}(v)$ **do**
11:         $S \leftarrow \mathcal{A}(u) \uplus \mathcal{A}(v)$                    ▷ merge two active sets
12:         **for all** $w \in S$ **do**                ▷ adjust assignment $\mathcal{A}$ for the new active set $S$
13:             $\mathcal{A}(w) \leftarrow S$
14:         **end for**
15:         **mark** edge $e$
16:         **while** there are $u', v' \in \mathsf{free}(S)$ such that $\mathsf{sgn}(u') = -\mathsf{sgn}(v')$ **do**
17:             **match** $u'$ with $v'$                ▷ match as many pairs as possible
18:         **end while**
19:     **end while**
20:
21: **None of the above events occurs:**
22:     **for all** growing active sets $S$ **do**
23:         increase continuously $y_S$ with the same rate in which time passes
24:     **end for**
```

case the new set $\mathcal{A}(u)$ is non-growing. (In the non-bipartite variant, this is always the case when two growing sets merge.) After $\mathcal{A}(u)$ becomes non-growing, another growing set may be merged with $\mathcal{A}(u)$, and so on. Thus, the set $\mathcal{A}(u)$ can change its state from growing to non-growing (and back) multiple times.

The next observation summarizes the process described above, listing properties of GD that we use later in our proofs.

Observation 4.1. *The following properties hold during the runtime of* GD*.*

1. *For a request u, when time passes, $\mathcal{A}(u)$ refers to different active sets that contain u.*
2. *At any time, every request is contained in exactly one active set. If this request is free, then the active set is growing.*
3. *At any time, an active set S contains exactly $\mathsf{sur}(S)$ free requests.*
4. *Active and inactive sets together constitute a laminar family of sets.*
5. *For any two requests u and v, once $\mathcal{A}(u)$ becomes equal to $\mathcal{A}(v)$, they will be equal forever.*

4.4 Correctness

We now prove that Greedy Dual is defined properly. In other words, we show that the dual values maintained by GD always form a feasible solution of $\mathcal{D}$ (Lemma 4.3) and GD returns a feasible matching of all requests at the end (Lemma 4.4). From now on, we denote the values of a dual variable y_S at time τ by $y_S(\tau)$.

By the definition, the waiting cost of a request is the time difference between the time it arrives and the time it is matched. In the following lemma, we relate the waiting cost of a request to the dual variables for the active sets it belongs to.

Lemma 4.2. *Fix any request u. For any time $\tau \geq \mathsf{atime}(u)$, it holds that*

$$\sum_{S:u\in S} y_S(\tau) \leq \tau - \mathsf{atime}(u).$$

The relation holds with equality if u is free at time τ.

Proof. We show that the inequality is preserved as time passes. At time $\tau = \mathsf{atime}(u)$, request u is introduced and sets S containing u appear. Their y_S values are initialized to 0. Therefore, at that time, $\sum_{S:u\in S} y_S(\tau) = 0$ as desired.

Whenever a merging event or an arrival of any other requests occur, new variables y_S may appear in the sum $\sum_{S:u \in S} y_S(\tau)$, but, at these times, the values of these variables are equal to zero, and therefore do not change the sum value.

It remains to analyze the case when time passes infinitesimally by ε and no event occurs within this period. It is sufficient to argue that the sum $\sum_{S:u \in S} y_S(\tau)$ increases exactly by ε if u is free at τ and at most by ε otherwise. Recall that y_S may grow only if S is an active growing set. By Property 2 of Observation 4.1, the only active set containing u is $\mathcal{A}(u)$. This set is growing if u is free (and then $y_{\mathcal{A}(u)}$ increases exactly by ε) and may be growing or non-growing if u is matched (and then $y_{\mathcal{A}(u)}$ increases at most by ε). $\square$

The following lemma shows that throughout its runtime, GD maintains a feasible dual solution.

Lemma 4.3. *At any time, the values y_S maintained by the algorithm constitute a feasible solution to $\mathcal{D}$.*

Proof. We show that no dual constraint is ever violated during the execution of GD.

When a new request u arrives at time $\tau = \mathsf{atime}(u)$, new sets containing u appear and the dual variables y_S corresponding to these sets are initialized to 0.

Each already existing constraint, corresponding to an edge e not incident to u, is modified: new y_S variables for sets S containing both u and exactly one of endpoints of e appear in the sum. However, all these variables are zero, and hence the feasibility of such constraints is preserved.

Moreover, for any edge $e = (u, v)$ where v is an existing request, a new dual constraint for this edge appears in $\mathcal{D}$. We show that it is not violated, i.e., $\sum_{S:e \in \delta(S)} y_S(\tau) \le \mathsf{opt\text{-}cost}(e)$. As discussed before, $y_S(\tau) = 0$ for the sets S containing u. Therefore,

$$\begin{aligned}
\sum_{S:e \in \delta(S)} y_S(\tau) &= \sum_{S:v \in S \wedge u \notin S} y_S(\tau) + \sum_{S:u \in S \wedge v \notin S} y_S(\tau) \\
&= \sum_{S:v \in S \wedge u \notin S} y_S(\tau) \le \sum_{S:v \in S} y_S(\tau) \\
&\le \mathsf{atime}(u) - \mathsf{atime}(v) && \text{(by Lemma 4.2)} \\
&\le \mathsf{opt\text{-}cost}(e).
\end{aligned}$$

Now, we prove that once a dual constraint for an edge $e = (u, v)$ becomes tight, the involved y_S values are no longer increased. According to the algorithm definition, $\mathcal{A}(u)$ and $\mathcal{A}(v)$ become merged together. By Property 5 of Observation 4.1, from this moment on, any active set S contains either both u and v or neither of them. Hence, there is no active set S, such that $(u,v) \in \delta(S)$, and in particular there is no such active growing set. Therefore, the value of $\sum_{S:e\in\delta(S)} y_S$ remains unchanged, and hence the dual constraint corresponding to edge e remains tight and not violated. □

Finally, we prove that GD returns a proper matching. We need to show that if a pair of requests remains unmatched, then appropriate dual variables increase and they will eventually trigger the matching event.

Lemma 4.4. *For any input for the M(B)PMD problem,* GD *returns a feasible matching.*

Proof. Suppose for a contradiction that GD does not match some request u. Then, by Property 2 of Observation 4.1, $\mathcal{A}(u)$ is always an active growing set and by Property 3, $\mathsf{sur}(\mathcal{A}(u)) > 0$. Therefore, the corresponding dual variable $y_{\mathcal{A}(u)}$ always increases during the execution of GD and appears in the objective function of $\mathcal{D}$ with a positive coefficient. By Lemma 4.3, the solution of $\mathcal{D}$ maintained by GD is always feasible, and hence the optimal value of $\mathcal{D}$ would be unbounded. This would be a contradiction, as there exists a finite solution to the primal program $\mathcal{P}$ (as all distances in the metric space are finite). □

4.5 Cost Analysis

In this section, we show how to relate the cost of the matching returned by Greedy Dual to the value of the produced dual solution. First, we show that the total waiting cost of the algorithm is equal to the value of the dual solution. Afterwards, we bound the connection cost of GD by $2m$ times the dual solution, where $2m$ is the number of requests in the input. This, along with Lemma 4.1, yields the competitive ratio of $2m + 1$.

4.5.1 Waiting Cost

In the proof below, we link the generated waiting cost with the growth of appropriate dual variables. To this end, suppose that a set S is an active set for time period of length Δt. By Property 3 of Observation 4.1, S contains exactly $\mathsf{sur}(S)$ free points, and thus the waiting cost incurred within this time by requests in S is $\Delta t \cdot \mathsf{sur}(S)$. Moreover, in the same time interval, the dual variable y_S increases by Δt, which contributes the same amount, $\mathsf{sur}(S) \cdot \Delta t$, to the growth of the objective function of $\mathcal{D}$. The following lemma formalizes this observation and applies it to all active sets considered by GD in its runtime.

Lemma 4.5. *The total waiting cost of* GD *is equal to* $\sum_{S \subseteq V} \mathsf{sur}(S) \cdot y_S(T)$*, where* T *is the time when* GD *matches the last request.*

Proof. We define $G(\tau)$ as the family of sets that are active and growing at time τ. By Property 2 and Property 3 of Observation 4.1, the number of free requests at time τ, henceforth denoted $\mathsf{wait}(\tau)$, is then equal to $\sum_S \mathsf{sur}(S) \cdot \mathbb{1}[S \in G(\tau)]$. The total waiting cost at time T can be then expressed as

$$\begin{aligned}\int_0^T \mathsf{wait}(\tau)\,\mathrm{d}\tau &= \int_0^T \sum_S \mathsf{sur}(S) \cdot \mathbb{1}[S \in G(\tau)]\,\mathrm{d}\tau \\ &= \sum_S \mathsf{sur}(S) \int_0^T \mathbb{1}[S \in G(\tau)]\,\mathrm{d}\tau = \sum_S \mathsf{sur}(S) \cdot y_S(T),\end{aligned}$$

where the last equality holds as at any time, GD increases y_S value if and only if S is active and growing. □

4.5.2 Connection Cost

Below, we relate the connection cost of GD to the value of the final solution of $\mathcal{D}$, created by GD. We focus on the set of marked edges, which are created by GD in Line 15 of Algorithm 2. We show that for any time, the set of marked edges restricted to an active or an inactive set S forms a "spanning tree" of requests of S. That is, there is a unique path of marked edges between any two requests from S. (Note that this path projected to the metric space may contain cycles as two requests may be given at the same point of $\mathcal{X}$.) We start with a helper observation.

Observation 4.2. *Fix any set* S*. If* S *is active at time* τ*, then its boundary* $\delta(S)$ *does not contain any marked edge at time* τ*.*

Proof. After an edge (u, v) becomes marked, both u and v belong to newly created active set. From now on, by Property 5 of Observation 4.1, they remain in the same active set till the end of the execution. Therefore, this edge will never be contained in a boundary of an active set. □

Lemma 4.6. *At any time, for any active or inactive set* S*, the subset of all marked edges with both endpoints in* S *forms a spanning tree of all requests from* S*.*

Proof. We show that the property holds as time passes. When a new request arrives, a new active growing set containing only one request is created. This set is trivially spanned by an empty set of marked edges.

By the definition of GD, a new active set appears when a dual constraint for some edge $e = (u, v)$ becomes tight. Right before it happens, the active sets containing

u and v are $\mathcal{A}(u)$ and $\mathcal{A}(v)$, respectively. At that time, marked edges form spanning trees of sets $\mathcal{A}(u)$ and $\mathcal{A}(v)$ and, by Observation 4.2, there are no marked edges between these two sets. Hence, these spanning trees together with the newly marked edge e constitute a spanning tree of the requests of $S = \mathcal{A}(u) \uplus \mathcal{A}(v)$. Finally, a set may become inactive only if it was active before, and GD never adds any marked edge inside an already existing active or inactive set. $\square$

Using the lemma above, we are ready to bound the connection cost of one matching edge by the cost of the solution of $\mathcal{D}$.

Lemma 4.7. *The connection cost of any matching edge is at most* $2 \cdot \sum_{S \subseteq V} \mathit{sur}(S) \cdot y_S(T)$*, where T is the time when* GD *matches the last request.*

Proof. Fix a matching edge (u, v) created by GD at time τ. Its connection cost is the distance $\mathsf{dist}(\mathsf{pos}(u), \mathsf{pos}(v))$ between the points corresponding to requests u and v in the underlying metric space.

We consider the state of GD right after it matches u with v. By Lemma 4.6, the active set $S = \mathcal{A}(u) = \mathcal{A}(v)$ containing u and v is spanned by a tree of marked edges. Let P be the (unique) path in this tree connecting u with v. Using the triangle inequality, we can bound $\mathsf{dist}(\mathsf{pos}(u), \mathsf{pos}(v))$ by the length of P projected onto the underlying metric space.

Recall that for any edge $e = (w, w')$, it holds that $\mathsf{dist}(\mathsf{pos}(w), \mathsf{pos}(w')) \leq \mathsf{opt\text{-}cost}(e)$. Moreover, if e is marked, the dual constraint for edge e holds with equality, that is, $\mathsf{opt\text{-}cost}(e) = \sum_{S: e \in \delta(S)} y_S(\tau)$. Therefore,

$$\begin{aligned} \mathsf{dist}(\mathsf{pos}(u), \mathsf{pos}(v)) &\leq \sum_{(w,w') \in P} \mathsf{dist}(\mathsf{pos}(w), \mathsf{pos}(w')) \leq \sum_{e \in P} \mathsf{opt\text{-}cost}(e) \\ &= \sum_{e \in P} \sum_{S: e \in \delta(S)} y_S(\tau) = \sum_{S} |\delta(S) \cap P| \cdot y_S(\tau) \\ &\leq \sum_{S} |\delta(S) \cap P| \cdot \mathsf{sur}(S) \cdot y_S(\tau) \\ &\leq \sum_{S} |\delta(S) \cap P| \cdot \mathsf{sur}(S) \cdot y_S(T). \end{aligned}$$

The penultimate inequality holds because a dual variable y_S can be positive only if $\mathsf{sur}(S) \geq 1$. It is now sufficient to prove that for each (active or inactive) set S, it holds that $|\delta(S) \cap P| \leq 2$, i.e., the path P crosses each such set S at most twice.

For a contradiction, suppose that there exists an (active or inactive) set S, whose boundary is crossed by path P more than twice. We direct all edges on P towards v (we follow P starting from request u and move towards v). Note that u may be inside or outside of S. Let $e_1 = (w_1, w_2)$ be the first edge on P such that $w_1 \in S$ and $w_2 \notin S$, i.e., the first time when path P leaves S. Let $e_2 = (w_3, w_4) \in P$ be the first edge after e_1, such that $w_3 \notin S$ and $w_4 \in S$, that is, the first time when path P returns to S after leaving it with edge e_1. Edge e_2 must exist as we assumed that P crosses the boundary of S at least three times.

By Lemma 4.6, a subset of the marked edges constitutes a spanning tree of S. Hence, there exists a path of marked edges contained entirely in S that connects requests w_1 and w_4. Furthermore, a sub-path of P connects w_2 and w_3 outside of S. These two paths together with edges e_1 and e_2 form a cycle of marked edges. However, by Lemma 4.6 and Observation 4.2, at any time, the set of marked edges forms a forest, which is a contradiction. □

4.5.3 Bounding the Competitive Ratio

Using above results we are able to bound the cost of Greedy Dual.

Theorem 4.1. Greedy Dual *is* $(2m+1)$*-competitive for the M(B)PMD problem.*

Proof. Fix any input instance $\mathcal{I}$ and let $\mathcal{D}$ be the corresponding dual program. Let D be the cost of the solution to $\mathcal{D}$ output by GD. By Lemma 4.5, the total waiting cost of the algorithm is bounded by D and by Lemma 4.7, the connection cost of a single edge in the matching is bounded by $2 \cdot D$. Therefore,

$$\mathrm{GD}(\mathcal{I}) \leq D + m \cdot 2D = (2m+1) \cdot D \leq (2m+1) \cdot \textsc{Opt}(\mathcal{I}),$$

where the first inequality holds as there are exactly m matched edges and the last equality follows by Lemma 4.1. □

4.6 Tightness of the Analysis

We can show that our analysis of Greedy Dual is asymptotically tight, i.e., the competitive ratio of Greedy Dual is $\Omega(m)$.

Theorem 4.2. *Both for MPMD and MBPMD problems, there exists an instance $\mathcal{I}$, such that* $\mathrm{GD}(\mathcal{I}) = \Omega(m) \cdot \textsc{Opt}(\mathcal{I})$.

Proof. Let $m > 0$ be an even integer and $\varepsilon = 1/m$. Let $\mathcal{X}$ be the metric containing two points p and q at distance 2.

In the instance $\mathcal{I}$, requests are released at both points p and q at times $0, 1+\varepsilon, 1+3\varepsilon, 1+5\varepsilon, \ldots, 1+(2m-3)\cdot\varepsilon$. For the MBPMD problem, we additionally specify request polarities: at p, all odd-numbered requests are positive and all even-numbered are negative, while requests issued at q have exactly opposite polarities from those at p.

Regardless of the variant (bipartite or non-bipartite) we solve, GD matches the first pair of requests at time 1, when their active growing sets are merged, forming a new active non-growing set. Every subsequent pair of requests appears exactly ε after the previous pair becomes matched. Therefore, they are matched together ε after their arrival, when their growing sets are merged with the large non-growing set containing all the previous pairs of requests. Hence, the total connection cost of GD is equal to $2m$. On the other hand, observe that the total cost of a solution that matches consecutive requests at each point of the metric space separately is equal to $2 \cdot ((1+\varepsilon) + 2\varepsilon \cdot (m-2)/2) = 2 \cdot (1 + (m-1) \cdot \varepsilon) < 4$. □

4.7 Alternative Deterministic Approaches

In this section, we argue why some alternative solutions fail to yield an improved competitive ratio.

4.7.1 Derandomization Using a Spanning Tree

In this part, we analyze an algorithm that approximates the metric space by a greedily and deterministically chosen spanning tree of requested points and employs the deterministic algorithm for trees of Azar et al. [ACK17]. We show that such algorithm has the competitive ratio of $2^{\Omega(m)}$. For simplicity, we focus on the non-bipartite variant, but the lower bound can be easily extended to the bipartite case.

More precisely, we define a natural algorithm Tree Based (TB). TB internally maintains a spanning tree T of metric space points corresponding to already seen requests. That is, whenever TB receives a request u at point $\mathsf{pos}(u)$, it executes the following two steps.

1. If there was no previous request at $\mathsf{pos}(u)$, TB adds $\mathsf{pos}(u)$ to T, connecting it to the closest point from T. The addition is performed immediately, at the request arrival. This part essentially mimics the behavior of the greedy algorithm for the online Steiner tree problem [IW91].

2. To serve the request u, TB runs the deterministic algorithm of [ACK17] on the tree T.[3]

Theorem 4.3. *The competitive ratio of* Tree Based *is* $2^{\Omega(m)}$.

Proof. The idea of the lower bound is as follows. The adversary first gives $m/2$ requests that force TB to create a tree T, and then gives another $m/2$ requests, so that the initial m requests can be served with a negligible cost by Opt. Afterwards, the adversary consecutively requests a pair of points that are close in the metric space, but far away in the tree T.

Our metric space $\mathcal{X}$ is a continuous ring and we assume that m is an even integer. Let h be the length of this ring and let $\varepsilon = h/(m \cdot 2^{m-1})$.

In the first part of the input, the adversary gives $m/2$ requests in the following way. The first two requests are given at time 0 at antipodal points (their distance is $h/2$). TB connects them using one of two halves of the ring. From now on, the tree T of TB will always cover a contiguous part of the ring. Each of the next $m/2 - 2$ requests is given exactly in the middle of the ring part not covered by T. For $j \in \{3, 4, \ldots, m/2\}$, the j-th request is given at time $(2 \cdot (j-1)/m) \cdot \varepsilon$.

This way, the ring part not covered by T shrinks exponentially, and after $m/2$ initial requests its length is equal to $h/2^{m/2-1}$. Let p and q be the endpoints (the only leaves) of T. Then, $\mathsf{dist}(p,q) = h/2^{m/2-1}$, but the path between p and q in T is of length $h - \mathsf{dist}(p,q)$ and uses an edge of length $h/2$. As T is built as soon as requests appear, its construction is finished right after the appearance of the $(m/2)$-th request, i.e., before time ε.

In the second part of the input, at time ε, the adversary gives $m/2$ requests at the same points as the requests from the first phase. This way, Opt may serve the first m requests paying nothing for the connection cost and paying at most $(m/2) \cdot \varepsilon = h/2^m$ for their waiting cost.

In the third part of the input, the adversary gives $m/2$ pairs of requests, each pair at points p and q. Each pair is given after the previous one is served by TB. Opt may serve each pair immediately after its arrival, paying $\mathsf{dist}(p,q) = h/2^{m/2-1}$ for the connection cost. On the other hand, TB serves each such pair using a path that connects p and q in the tree T. Before matching p with q, TB waits for a time which is at least the length of the longest edge on this path, $h/2$ (see the analysis in [ACK17]). In total, the cost of TB for the last m requests alone is at least $(m/2) \cdot (h/2)$, while the total cost of Opt for the whole input is at most $h/2^m + (m/2) \cdot h/2^{m/2-1}$. This proves that the competitive ratio of TB is $2^{\Omega(m)}$. □

[3]The algorithm must be able to operate on a tree that may be extended (new leaves may appear) in the runtime. The algorithm given by Azar et al. [ACK17] has this property.

4.7.2 Doubling Technique

Another standard deterministic approach that may be applied to the MPMD and MBPMD problems is the *doubling technique* (see, e.g., [CK06]): an online algorithm may trace the cost of an optimal solution Opt and perform a global operation (e.g., match many pending requests) once the cost of Opt increases significantly (e.g., by a factor of two) since the last time when such global operation was performed. This approach does not seem to be feasible here as the total cost of Opt may *decrease* when new requests appear.

Chapter 5

Afterword

In this thesis, we focused on a fresh perspective that delayed decisions recently brought to the world of online computations. While the presented techniques and algorithms hopefully made some modest impact, a lot of open questions remain. Here, we present some of them, highlighting possible further work opportunities.

Traveling Repairperson and Unrelated Machines Scheduling

In Chapter 2, we presented an algorithm that improved competitive ratios for both deterministic and randomized cases of these problems. Our algorithm is phase-based: it partitions the timeline into phases, and in every phase it simulates a solution, that achieves best possible result for the already seen requests. We estimated the competitive ratio of this algorithm using a factor revealing linear program.

Similar phase-based techniques were used by the previous results [KdPPS03, BL19, HJ18]. We conjecture that our result achieved the limits of this approach, and further improvements would require different techniques, because:

- we presented tight examples in Section 2.6 that are very simple and they holds also for other variants of phase-based algorithms,
- we performed extensive computer experiments, with multiple various parameters (such as phase length, different penalties for unserved requests, etc.) and none of them resulted in an improvement.

That said, the gap between achievable lower and upper bounds is substantial. For instance, in case of deterministic TRP, the currently best lower bound is equal to 2.41 [FS01] and the upper bound presented in this thesis is equal to 4. It is plausible that the upper bound can be beaten using non-phase-based approach. In particular, our phase-based algorithm spends a lot of time waiting for the next phase and returning to the origin. A potential improvement could therefore trace the arriving requests while executing the phase, and when the algorithm observes that the simulated solution is far from optimal, it could preempt its current route and start a new phase earlier. However, it is not clear how to analyze such approach using our factor-revealing framework.

Another improvement opportunity lies in considering specific instances. For example, some improvements for the TRP problem were made for the real line, and use properties specific for this metric [BL19]. For machines scheduling better algorithms were given for more restricted settings such as related or identical machines [Sit10].

Online Service with Delay

In Chapter 3, we considered the Online Service with Delay problem on a line metric consisting of n equidistant points. Our deterministic algorithm achieved competitive ratio of $O(\log n)$. For the more general cases, Azar and Touitou recently presented a randomized algorithm for any metric that works in time $O(\log^2 n)$ [AT19].

One of the biggest open problems is to derive a deterministic solution for general metric. Unfortunately, the technique presented in Chapter 3 does not seem to extend to general graphs. They can be slightly extended though, with virtually no changes to algorithm or analysis, to handle non-equidistant points on line metric. The competitive ratio becomes then $O(\log \Delta)$, where Δ is the aspect ratio of the metric (the ratio between the largest and the smallest distances).

Online Matching with Delays

In Chapter 4, we studied the problem of the Online Matching with Delays. The known bounds on the competitive ratios for randomized algorithms are almost tight (between $\Omega(\log n / \log\log n)$ and $O(\log n)$ [ACK17, AAC+17], where n is the number of points in the metric space).

On the other hand, the deterministic case remains widely unresolved. We presented an $O(m)$-competitive deterministic algorithm for the problem, where $2m$ is the number of requests that appeared during algorithm execution. To this end, we used linear programming and moat growing technique. Shortly after our result, Azar and Jacob-Fanani [AJF18] presented a deterministic algorithm with ratio of $O(m^{\log 3/2}) \approx O(m^{0.58})$.

While their algorithm is superior in terms of achieved competitive ratio, it uses a clever variant of a simple greedy routine, and thus can be hard to improve: The lower bound of $\Omega(m^{\log 3/2}) \approx \Omega(m^{0.58})$ for (offline) greedy matching algorithms presented by Reingold and Tarjan [RT81] creates a natural barrier for all greedy approaches in online case as well.

On the other hand, the primal-dual moat growing technique seems to be more promising as there is a wide range of ways to modify the algorithm. For example, one can try to slow down the speed of increment of some moats in certain cases, to create better opportunities for other moats, potentially improving our charging scheme.

Bibliography

[AAC+17] Itai Ashlagi, Yossi Azar, Moses Charikar, Ashish Chiplunkar, Ofir Geri, Haim Kaplan, Rahul M. Makhijani, Yuyi Wang, and Roger Wattenhofer. Min-cost bipartite perfect matching with delays. In *Proc. 20th Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 1:1–1:20, 2017.

[ABC+99] Foto N. Afrati, Evripidis Bampis, Chandra Chekuri, David R. Karger, Claire Kenyon, Sanjeev Khanna, Ioannis Milis, Maurice Queyranne, Martin Skutella, Clifford Stein, and Maxim Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *Proc. 40th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 32–44, 1999.

[ABN+14] Antonios Antoniadis, Neal Barcelo, Michael Nugent, Kirk Pruhs, and Michele Scquizzato. A o(n)-competitive deterministic algorithm for online matching on a line. In *Proc. 12th Workshop on Approximation and Online Algorithms (WAOA)*, pages 11–22, 2014.

[ACER11] Anna Adamaszek, Artur Czumaj, Matthias Englert, and Harald Räcke. Almost tight bounds for reordering buffer management. In *Proc. 43rd ACM Symp. on Theory of Computing (STOC)*, pages 607–616, 2011.

[ACK17] Yossi Azar, Ashish Chiplunkar, and Haim Kaplan. Polylogarithmic bounds on the competitiveness of min-cost perfect matching with delays. In *Proc. 28th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 1051–1061, 2017.

[ACKT20] Yossi Azar, Ashish Chiplunkar, Shay Kutten, and Noam Touitou. Set cover with delay - clairvoyance is not required. In *Proc. 28th European Symp. on Algorithms (ESA)*, pages 8:1–8:21, 2020.

[AFL+94] Giorgio Ausiello, Esteban Feuerstein, Stefano Leonardi, Leen Stougie, and Maurizio Talamo. Serving requests with on-line routing. In *Proc. 4th*

Scandinavian Symposium and Workshops on Algorithm Theory (SWAT), pages 37–48, 1994.

[AFL$^+$95] Giorgio Ausiello, Esteban Feuerstein, Stefano Leonardi, Leen Stougie, and Maurizio Talamo. Competitive algorithms for the on-line traveling salesman. In *Proc. 4th Int. Workshop on Algorithms and Data Structures (WADS)*, pages 206–217, 1995.

[AFL$^+$01] Giorgio Ausiello, Esteban Feuerstein, Stefano Leonardi, Leen Stougie, and Maurizio Talamo. Algorithms for the on-line travelling salesman. *Algorithmica*, 29(4):560–581, 2001.

[AGGP17] Yossi Azar, Arun Ganesh, Rong Ge, and Debmalya Panigrahi. Online service with delay. In *Proc. 49th ACM Symp. on Theory of Computing (STOC)*, pages 551–563, 2017.

[AJF18] Yossi Azar and Amit Jacob-Fanani. Deterministic min-cost matching with delays. In *Proc. 16th Workshop on Approximation and Online Algorithms (WAOA)*, 2018.

[AK03] Sanjeev Arora and George Karakostas. Approximation schemes for minimum latency problems. *SIAM Journal on Computing*, 32(5):1317–1337, 2003.

[AKR00] Norbert Ascheuer, Sven Oliver Krumke, and Jörg Rambau. Online dial-a-ride problems: Minimizing the completion time. In *Proc. 17th Symp. on Theoretical Aspects of Computer Science (STACS)*, pages 639–650, 2000.

[AR13] Noa Avigdor-Elgrabli and Yuval Rabani. An optimal randomized online algorithm for reordering buffer management. In *Proc. 54th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 1–10, 2013.

[ASvZ13] Tetske Avontuur, Pieter Spronck, and Menno van Zaanen. Player skill modeling in Starcraft II. In *Proc. 9th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE-13*, 2013.

[AT19] Yossi Azar and Noam Touitou. General framework for metric optimization problems with delay or with deadlines. In *Proc. 60th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 60–71, 2019.

[AT20] Yossi Azar and Noam Touitou. Beyond tree embeddings - a deterministic framework for network design with deadlines or delay. In *Proc. 61st IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 1368–1379, 2020.

[BBB$^+$16] Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukáš Folwarczný, Łukasz Jeż, Jiří Sgall, Nguyen Kim Thang, and Pavel Veselý. Online algorithms for multi-level aggregation. In *Proc. 24th European Symp. on Algorithms (ESA)*, pages 12:1–12:17, 2016.

[BBC$^+$13] Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Łukasz Jeż, Jiři Sgall, and Grzegorz Stachowiak. Online control message aggregation in chain networks. In *Proc. 13th Int. Workshop on Algorithms and Data Structures (WADS)*, pages 133–145, 2013.

[BBGN14] Nikhil Bansal, Niv Buchbinder, Anupam Gupta, and Joseph Naor. A randomized $O(\log^2 k)$-competitive algorithm for metric bipartite matching. *Algorithmica*, 68(2):390–403, 2014.

[BBM17] Marcin Bienkowski, Jaroslaw Byrka, and Marcin Mucha. Dynamic beats fixed: On phase-based algorithms for file migration. In *Proc. 44th Int. Colloq. on Automata, Languages and Programming (ICALP)*, pages 13:1–13:14, 2017.

[BBMN15] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem. *Journal of the ACM*, 62(5):40:1–40:49, 2015.

[BD20] Alexander Birx and Yann Disser. Tight analysis of the smartstart algorithm for online dial-a-ride on the line. *SIAM Journal on Discrete Mathematics*, 34(2):1409–1443, 2020.

[BDH$^+$17] Antje Bjelde, Yann Disser, Jan Hackfeld, Christoph Hansknecht, Maarten Lipmann, Julie Meißner, Kevin Schewior, Miriam Schlöter, and Leen Stougie. Tight bounds for online TSP on the line. In *Proc. 28th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 994–1005, 2017.

[BDS19] Alexander Birx, Yann Disser, and Kevin Schewior. Improved bounds for open online dial-a-ride on the line. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 21:1–21:22, 2019.

[BE98] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[BFNT17] Niv Buchbinder, Moran Feldman, Joseph (Seffi) Naor, and Ohad Talmon. O(depth)-competitive algorithm for online multi-level aggregation. In *Proc.*

28th ACM-SIAM Symp. on Discrete Algorithms (SODA), pages 1235–1244, 2017.

[BGRS10] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. An improved LP-based approximation for Steiner tree. In *Proc. 42nd ACM Symp. on Theory of Computing (STOC)*, pages 583–592, 2010.

[BKdPS01] Michiel Blom, Sven Oliver Krumke, Willem de Paepe, and Leen Stougie. The online TSP against fair adversaries. *INFORMS Journal on Computing*, 13(2):138–148, 2001.

[BKL21] Marcin Bienkowski, Artur Kraska, and Hsiang-Hsuan Liu. Traveling repairperson, unrelated machines, and other stories about average completion times. In *Proc. 48th Int. Colloq. on Automata, Languages and Programming (ICALP)*, pages 28:1–28:20, 2021.

[BKLS18] Marcin Bienkowski, Artur Kraska, Hsiang-Hsuan Liu, and Pawel Schmidt. A primal-dual online deterministic algorithm for matching with delays. In *Proc. 16th Workshop on Approximation and Online Algorithms (WAOA)*, pages 51–68, 2018.

[BKS17] Marcin Bienkowski, Artur Kraska, and Paweł Schmidt. A match in time saves nine: Deterministic online matching with delays. In *Proc. 15th Workshop on Approximation and Online Algorithms (WAOA)*, pages 132–146, 2017.

[BKS18] Marcin Bienkowski, Artur Kraska, and Paweł Schmidt. Online service with delay on a line. In *Proc. 25th Int. Colloq. on Structural Information and Communication Complexity (SIROCCO)*, pages 237–248, 2018.

[BL19] Marcin Bienkowski and Hsiang-Hsuan Liu. An improved online algorithm for the traveling repairperson problem on a line. In *Proc. 44th Int. Symp. on Mathematical Foundations of Computer Science (MFCS)*, pages 6:1–6:12, 2019.

[Blä16] Markus Bläser. Metric TSP. In *Encyclopedia of Algorithms*, pages 1276–1279. 2016.

[BN09] Niv Buchbinder and Joseph Naor. The design of competitive online algorithms via a primal-dual approach. *Foundations and Trends in Theoretical Computer Science*, 3(2–3):93–263, 2009.

[BS09] Vincenzo Bonifaci and Leen Stougie. Online *k*-server routing problems. *Theory of Computing Systems*, 45(3):470–485, 2009.

[BYCR93] Ricardo A. Baeza-Yates, Joseph C. Culberson, and Gregory J. E. Rawlins. Searching in the plane. *Information and Computation*, 106(2):234–252, 1993.

[CDLW19] Pei-Chuan Chen, Erik D. Demaine, Chung-Shou Liao, and Hao-Ting Wei. Waiting is not easy but worth it: the online TSP on the line revisited. 2019.

[CGRT03] Kamalika Chaudhuri, Brighten Godfrey, Satish Rao, and Kunal Talwar. Paths, trees, and minimum latency tours. In *Proc. 44th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 36–45, 2003.

[CK06] Marek Chrobak and Claire Kenyon-Mathieu. Competitiveness via doubling. *SIGACT News*, 37(4):115–126, 2006.

[CPS$^+$96] Soumen Chakrabarti, Cynthia A. Phillips, Andreas S. Schulz, David B. Shmoys, Clifford Stein, and Joel Wein. Improved scheduling algorithms for minsum criteria. In *Proc. 23rd Int. Colloq. on Automata, Languages and Programming (ICALP)*, pages 646–657, 1996.

[CW09] José R. Correa and Michael R. Wagner. LP-based online scheduling: from single to parallel machines. *Mathematical Programming*, 119(1):109–136, 2009.

[dPLS$^+$04] Willem de Paepe, Jan Karel Lenstra, Jirí Sgall, René A. Sitters, and Leen Stougie. Computer-aided complexity classification of dial-a-ride problems. *INFORMS Journal on Computing*, 16(2):120–132, 2004.

[EKW16] Yuval Emek, Shay Kutten, and Roger Wattenhofer. Online matching: haste makes waste! In *Proc. 48th ACM Symp. on Theory of Computing (STOC)*, pages 333–344, 2016.

[Elo78] Arpad E. Elo. *The rating of chessplayers, past and present*. Arco Publishing, 1978.

[ER17] Matthias Englert and Harald Räcke. Reordering buffers with logarithmic diameter dependency for trees. In *Proc. 28th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 1224–1234, 2017.

[ERS19] Matthias Englert, Harald Räcke, and Richard Stotz. Polylogarithmic guarantees for generalized reordering buffer management. In *Proc. 60th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 38–59, 2019.

[ERW10] Matthias Englert, Harald Räcke, and Matthias Westermann. Reordering buffers for general metric spaces. *Theory of Computing Systems*, 6(1):27–46, 2010.

[ESW17] Yuval Emek, Yaacov Shapiro, and Yuyi Wang. Minimum cost perfect matching with delays for two sources. In *Proc. 10th Int. Conf. on Algorithms and Complexity (CIAC)*, pages 209–221, 2017.

[FKW09] Irene Fink, Sven Oliver Krumke, and Stephan Westphal. New lower bounds for online k-server routing problems. *Information Processing Letters*, 109(11):563–567, 2009.

[FRT04] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004.

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

[GL12] Anupam Gupta and Kevin Lewi. The online metric matching problem for doubling metrics. In *Proc. 39th Int. Colloq. on Automata, Languages and Programming (ICALP)*, pages 424–435, 2012.

[GLLK79] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287–326. Elsevier, 1979.

[GLS88] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988.

[GS09] Iftah Gamzu and Danny Segev. Improved online algorithms for the sorting buffer problem on line metrics. *ACM Transactions on Algorithms*, 6(1):15:1–15:14, 2009.

[GW95] Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.

[HJ18] Dawsen Hwang and Patrick Jaillet. Online scheduling with multi-state machines. *Networks*, 71(3):209–251, 2018.

[HKR00] Dietrich Hauptmeier, Sven Oliver Krumke, and Jörg Rambau. The online dial-a-ride problem under reasonable load. In *Proc. 4th Int. Conf. on Algorithms and Complexity (CIAC)*, pages 125–136, 2000.

[HSSW97] Leslie A. Hall, Andreas S. Schulz, David B. Shmoys, and Joel Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22(3):513–544, 1997.

[HSW01] Han Hoogeveen, Petra Schuurman, and Gerhard J. Woeginger. Non-approximability results for scheduling problems with minsum criteria. *INFORMS Journal on Computing*, 13(2):157–168, 2001.

[IW91] Makoto Imase and Bernard M. Waxman. Dynamic Steiner tree problem. *SIAM Journal on Discrete Mathematics*, 4(3):369–384, 1991.

[JMM$^+$03] Kamal Jain, Mohammad Mahdian, Evangelos Markakis, Amin Saberi, and Vijay V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. *Journal of the ACM*, 50(6):795–824, 2003.

[JMS19] Vinay A. Jawgal, V. N. Muralidhara, and P. S. Srinivasan. Online travelling salesman problem on a circle. In *Proc. 15th Theory and Applications of Models of Computation (TAMC)*, pages 325–336, 2019.

[JP95] Michael Jünger and William R. Pulleyblank. New primal and dual matching heuristics. *Algorithmica*, 13(4):357–386, 1995.

[JW06] Patrick Jaillet and Michael R. Wagner. Online routing problems: Value of advanced information as improved competitive ratios. *Transp. Sci.*, 40(2):200–210, 2006.

[JW08] Patrick Jaillet and Michael R. Wagner. Generalized online routing: New competitive ratios, resource augmentation, and asymptotic analyses. *Oper. Res.*, 56(3):745–757, 2008.

[KdPP$^+$05] Sven Oliver Krumke, Willem de Paepe, Diana Poensgen, Maarten Lipmann, Alberto Marchetti-Spaccamela, and Leen Stougie. On minimizing the maximum flow time in the online dial-a-ride problem. In *Proc. 3rd Workshop on Approximation and Online Algorithms (WAOA)*, pages 258–269, 2005.

[KdPPS03] Sven Oliver Krumke, Willem de Paepe, Diana Poensgen, and Leen Stougie. News from the online traveling repairman. *Theoretical Computer Science*, 295:279–294, 2003.

[KdPPS06] Sven Oliver Krumke, Willem de Paepe, Diana Poensgen, and Leen Stougie. Erratum to "news from the online traveling repairman" [TCS 295 (1-3) (2003) 279-294]. *Theoretical Computer Science*, 352(1-3):347–348, 2006.

[KLL$^+$02] Sven Oliver Krumke, Luigi Laura, Maarten Lipmann, Alberto Marchetti-Spaccamela, Willem de Paepe, Diana Poensgen, and Leen Stougie. Non-abusiveness helps: An O(1)-competitive algorithm for minimizing the maximum flow time in the online traveling salesman problem. In *Proc. 5th Int. Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 200–214, 2002.

[KMV94] Samir Khuller, Stephen G. Mitchell, and Vijay V. Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theoretical Computer Science*, 127(2):255–267, 1994.

[KN03] Elias Koutsoupias and Akash Nanavati. The online matching problem on a line. In *Proc. 1st Workshop on Approximation and Online Algorithms (WAOA)*, pages 179–191, 2003.

[KP93] Bala Kalyanasundaram and Kirk Pruhs. Online weighted matching. *Journal of Algorithms*, 14(3):478–488, 1993.

[LLdP$^+$04] Maarten Lipmann, Xiwen Lu, Willem de Paepe, René Sitters, and Leen Stougie. On-line dial-a-ride problems under a restricted information model. *Algorithmica*, 40(4):319–329, 2004.

[LVJ16] Meghna Lowalekar, Pradeep Varakantham, and Patrick Jaillet. Online spatio-temporal matching in stochastic and dynamic domains. In *Proc. 30th AAAI Conference on Artificial Intelligence*, pages 3271–3277, 2016.

[Meh13] Aranyak Mehta. Online matching and ad allocation. *Foundations and Trends in Theoretical Computer Science*, 8(4):265–368, 2013.

[MNP06] Adam Meyerson, Akash Nanavati, and Laura J. Poplawski. Randomized online algorithms for minimum metric bipartite matching. In *Proc. 7th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 954–959, 2006.

[MS04] Nicole Megow and Andreas S. Schulz. On-line scheduling to minimize average completion time revisited. *Operations Research Letters*, 32(5):485–490, 2004.

[MY11] Mohammad Mahdian and Qiqi Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing LPs. In *Proc. 43rd ACM Symp. on Theory of Computing (STOC)*, pages 597–606, 2011.

[NR17] Krati Nayyar and Sharath Raghvendra. An input sensitive online algorithm for the metric bipartite matching problem. In *Proc. 58th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 505–515, 2017.

[Rag16] Sharath Raghvendra. A robust and optimal online algorithm for minimum metric bipartite matching. In *Proc. 19th Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 18:1–18:16, 2016.

[Rag18] Sharath Raghvendra. Optimal analysis of an online algorithm for the bipartite matching problem on a line. In *Proc. 34th ACM Symp. on Computational Geometry (SoCG)*, pages 67:1–67:14, 2018.

[RSW02] Harald Räcke, Christian Sohler, and Matthias Westermann. Online scheduling for sorting buffers. In *Proc. 10th European Symp. on Algorithms (ESA)*, pages 820–832, 2002.

[RT81] Edward M. Reingold and Robert Endre Tarjan. On a greedy heuristic for complete matching. *SIAM Journal on Computing*, 10(4):676–681, 1981.

[Sch03] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Algorithms and combinatorics. Springer, 2003.

[Sei00] Steven S. Seiden. A guessing game and randomized online algorithms. In *Proc. 32nd ACM Symp. on Theory of Computing (STOC)*, pages 592–601, 2000.

[SG76] Sartaj Sahni and Teofilo F. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23(3):555–565, 1976.

[Sit02] René Sitters. The minimum latency problem is NP-hard for weighted trees. In *Proc. 9th Int. Conf. on Integer Programming and Combinatorial Optimization (IPCO)*, pages 230–239, 2002.

[Sit10] René Sitters. Efficient algorithms for average completion time scheduling. In *Proc. 14th Int. Conf. on Integer Programming and Combinatorial Optimization (IPCO)*, pages 411–423, 2010.

[Sit14] René Sitters. Polynomial time approximation schemes for the traveling repairman and other minimum latency problems. In *Proc. 25th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 604–616, 2014.

[Sit17] René Sitters. Approximability of average completion time scheduling on unrelated machines. *Mathematical Programming*, 161(1-2):135–158, 2017.

[Sku98] Martin Skutella. Semidefinite relaxations for parallel machine scheduling. In *Proc. 39th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 472–481, 1998.

[SS02] Andreas S. Schulz and Martin Skutella. Scheduling unrelated machines by randomized rounding. *SIAM Journal on Discrete Mathematics*, 15(4):450–469, 2002.

[Ves97] Arjen P. A. Vestjens. *On-line Machine Scheduling*. PhD thesis, Eindhoven University of Technology, 1997.

[WH16] Weili Wu and Yaochun Huang. Steiner trees. In *Encyclopedia of Algorithms*, pages 2102–2107. Springer, 2016.

[You16] Neal E. Young. Greedy set-cover algorithms. In *Encyclopedia of Algorithms*, pages 886–889. 2016.