Complexity Results and Competitive Analysis for Vehicle Routing Problems

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Paepe, Willem E. de

Complexity Results and Competitive Analysis for Vehicle Routing Problems / by Willem E. de Paepe. - Eindhoven : Technische Universiteit Eindhoven, 2002.
Proefschrift. - ISBN 90-386-1847-6
NUR 804
Keywords: Combinatorial optimization / Online algorithms / Competitive analysis / Computational complexity / Dial-a-ride / Vehicle routing / Pick up and delivery / Traveling salesman problem / Traveling repairman problem

Printed by Universiteitsdrukkerij Technische Universiteit Eindhoven

Complexity Results and Competitive Analysis for Vehicle Routing Problems

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr. R.A. van Santen, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op maandag 30 september 2002 om 16.00 uur

door

Willem Eduard de Paepe

geboren te Purmerend

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. J.K. Lenstra
en
prof.dr. A.G. de Kok

Copromotor:
dr. L. Stougie

Voor mijn ouders

Preface

From the day I started my Ph.D. period in Eindhoven, I have felt torn up between the department of Technology Management, where my supervisor Peter van Laarhoven and my desk were, and the department of Mathematics and Computer Science, with my other supervisors, Leen Stougie and Jan Karel Lenstra. This feeling became even stronger when Peter van Laarhoven left the university. Being part of two groups had the advantage that I had a lot of direct colleagues to work with, but inevitably had the disadvantage that there was less time to spend with each of them. Therefore the personal contact was sometimes less frequent than one would hope for. Despite this, I always felt very welcome in both groups, and for this I would like to thank my colleagues from OPC (especially the members of the former IDL) and the combinatorial optimization group. In particular I would like to thank Ebbe, Joris, and Kim for spicing up the first years of my Ph.D. period.

The person that has had the most influence on the contents of this thesis is, beyond any doubt, Leen Stougie. We met at the University of Amsterdam, where he supervised my master thesis. Ever since, he has been a great source of inspiration to me. He stimulated me in times of low productivity, and joined in great enthusiasm when results were established. He was always willing to discuss any new idea, even if it turned out to lead nowhere. Next to this, he introduced me to many people in the OR community, leading to fruitful international cooperation. We have spent a lot of time together, not only at the TU and during train rides between Amsterdam and Eindhoven, but also outside working hours. I thank Leen very much for the many pleasant moments we had together.

I am also very grateful to Jan Karel Lenstra. Although we didn't get to work together very often, there where periods in which we had frequent and pleasant contact. In the first year he taught me a lot about proving NP-completeness of problems. At a later stage I learned a lot about writing and editing through his comments on early versions of papers and of this thesis.

I am greatly indebted to all my coauthors, Michiel Blom, Sven Krumke, Luigi Laura, Jan Karel Lenstra, Maarten Lipmann, Xiwen Lu, Alberto Marchetti-Spaccamela, Diana Poensgen, Joerg Rambau, Jiri Sgall, Rene Sitters, Leen Stougie, and Gerhard Woeginger, for allowing me to use our joint results in my thesis. I always considered working together as the best way to get new ideas, and I definitely learned a lot from all of them. Especially Sven has played an important role. We have been working together more or less starting from day one, and he has always been a great inspirator. I would also like to thank Alberto explicitly, for creating the opportunity to visit La Sapienza University of Rome for two months, and for

making it a fruitful and enjoyable period.

Then I would like to thank Alberto, Jiri, Leen, and Jan Karel for their useful comments on a preliminary version of this thesis. I am also grateful for the financial support of BETA, DONET, AMORE, and DFG. Without their funding it would not have been possible to learn so much in an international environment.

There are also many people that are important in my personal life. I want to thank Eric, Frank, Ivar, Jan, Jeroen, Jessica, Joris, Jorrit, Maarten, Marieke, Marietta, Michiel, Pascal, Peter, Remco, Robert, Rogier, Tanja, Thomas, and Yvonne, for reminding me that there is more in life than work, and for making the non-professional part of life extremely pleasant. There are two friends that deserve special attention. I would like to thank Eric in particular for sharing an apartment with me for more than five years, showing interest in my work, and for supporting me in good and in bad times, even during the defense of my thesis, as one of my paranymphs. I also want to thank my other paranymph, Michiel, for the great time we had together, first at the University of Amsterdam as fellow students, then at the University of Eindhoven as colleagues and room mates. Working with Michiel has always been a pleasure, and considering the contents of Chapter 4, also very fruitful.

Finally, I would like to thank Maaike and Gabor for always showing interest in my work and my well-being. I dedicate this thesis to my parents, for their constant support in everything that I choose to do.

Willem de Paepe, July 2002.

Contents

1

Introduction

1.1 Optimization problems

Consider the following simple problem that we will use throughout this introduction. We will refer to this problem as the *grocery shopping problem*, or GSP. In the literature this problem is very well studied and is called the *traveling salesman problem*. Suppose you are asked to do some groceries. You are presented a list of groceries to bring home. On the list there is shampoo, bread, and wine. Your goal is to return home with the groceries as soon as possible. To this end you consider the geographical locations of the drugstore, the bakery, and the winery, and you determine your preferred tour visiting all three shops, starting and ending at your residence.

Without knowing it you are facing an optimization problem, or more specifically a combinatorial optimization problem.

Definition 1.1. [**Optimization problem**]
An optimization problem Π *is a three-tuple* $(\mathcal{I}, S, f)$ *such that*

1. $\mathcal{I}$ *is the set of the instances of* Π*,*
2. *given an instance* $I \in \mathcal{I}$*,* S_I *denotes the set of feasible solutions of* I*,*
3. *given an instance* $I \in \mathcal{I}$*,* f_I *is the objective function that attributes to each feasible solution* x *of* I *a real number* $f_I(x)$*, the so-called objective value of* x*.*

The goal is to determine if, given an instance $I \in \mathcal{I}$*, it has a feasible solution, and if so, to find a feasible solution that has smallest objective function value amongst*

all feasible solutions, that is, a feasible solution x^* such that

$$f_I(x^*) = \min\{f_I(x) : x \in S_I\}.$$

The formulation also allows for a maximization problem. Usually we replace min *by* max *in case of a maximization problem.*

If S is a finite (or countably infinite) set of all instances $I \in \mathcal{I}$, we have a combinatorial optimization problem. In this thesis we concentrate on this class of optimization problems.

The general formulation allows for an enormous variety of problems, each characterized by a specific description of the set S and the function f. For example, an *instance* of the general GSP consists of a set of points with their pairwise distances, where the set S is defined as the set of feasible routes and the function f adds to each route its length (in distance or time). The objective is to find a feasible route with minimum length. The distinction between a problem Π and an instance I of the problem is crucial in this thesis.

In the example with which we started off, the instance is given by the shops to visit and all distances between the shops and from the house and the shops. The set of feasible solutions for our example consists of all tours that visit the designated shops, or equivalently, of all permutations of the shops to be visited, indicating the order in which to visit them. The costs of a specific feasible solution is given by the length of the tour. If we assume that you walk with constant speed and that you know your speed, we can also define the costs in terms of time needed to complete the tour. The goal is to minimize the cost function, i.e. to return home as soon as possible.

Now that you know the problem, you might want to think of a strategy to solve this problem, rather than to make ad-hoc decisions. A possible strategy is to visit the shop closest to your home first, then the shop closest to the first shop, and finally before returning home, the last shop. Such a strategy is called an *algorithm*.

Definition 1.2. [**Algorithm**]
An algorithm A *for a problem* Π *is a general step-by-step procedure that on every instance* I *of* Π *outputs a feasible solution* $x \in S_I$*, or outputs that it cannot find a feasible solution.*

We will denote by $\mathrm{A}(I)$ the value of the objective function corresponding to the output x of algorithm A on instance I, i.e. $\mathrm{A}(I) = f_I(x)$. If algorithm A cannot find a feasible solution on input I, we set $\mathrm{A}(I) = \infty$. Let OPT be a function that maps every instance I of Π to the value of the objective function of an optimal solution. We will call OPT an *optimization algorithm*.

Suppose that your favorite TV show starts 15 minutes after you are asked to do the shopping. You wonder if you can go shopping and return in time for the show. Instead of finding the *best possible* tour, you are interested in the question if there exists a tour such that you return home within 15 minutes.

This question can be answered only with *yes* or *no*. This version of the problem is called a *decision problem* or *recognition problem*. Notice that being able to solve the optimization problem implies being able to answer the decision problem.

Definition 1.3. [**Decision problem**]
With a minimization (maximization) problem Π*, we can associate a decision problem that includes a numerical bound* B *as an additional input parameter, with the question whether or not there exists a feasible solution* $x \in S_I$ *to instance* I *of* Π*, such that* $f_I(x) \leq B$ *(*$f_I(x) \geq B$*).*

When faced with a problem such as the GSP, you might wonder how difficult the problem is. You might be willing to think about your tour for some minutes, but it would be bad if deciding on your tour takes you hours! The three products instance described above might be solved fast, but things change when your list contains 50 items. The complexity of a problem is subject of the next section.

1.2 Complexity theory

When we are faced with a combinatorial problem, we would like to know how difficult it is to solve it. Since difficulty is a subjective term we are interested in having an unambiguous notion of complexity. Complexity theory provides a mathematical framework in which combinatorial problems can be studied so that they can be classified as *easy* or *hard*. The foundations of complexity theory have been laid in the late 1960's by Cook [11]. Since then the field has evolved enormously. In this section, a short introduction to complexity theory is given. For an extensive introduction to this field we refer to the book of Garey and Johnson [15]. For a more recent work, see [43].

In order to determine if a combinatorial problem is easy or hard, we study the time required to *solve* the problem as a function of the length of the encoding of the input I, denoted by $|I|$. For optimization problems, solving means finding a solution, for any given instance, that is optimal in terms of the objective function, and for decision problems solving is deciding correctly, for any given instance, whether or not the instance has a yes-answer. A problem is considered to be *easy* if there exists an algorithm that solves the problem in time bounded by a polynomial function of $|I|$. If a decision problem is easy, then, under a mild restriction on the range of the objective function, one can apply binary search to obtain a polynomial time algorithm for the optimization problem. On the other hand, if an optimization problem is easy, then clearly the decision version of this problem is also easy. Let $\mathcal{P}$ denote the class of decision problems that can be solved in polynomial time.

Let $\mathcal{NP}$ denote the class of decision problem that have the property that for every instance I that has a yes-answer, there exists a certificate y such that $|y|$ is bounded by a polynomial in $|I|$ and such that it can be checked in polynomial time that y is indeed a certificate for a yes-answer of I. Notice that the membership of $\mathcal{NP}$ does not mean that it is easy to find such a certificate.

All decision problems and decision versions of the optimization problems discussed in this thesis are in $\mathcal{NP}$. For example, the GSP of Section 1.1 is a member of $\mathcal{NP}$. If there is a yes-answer to a specific instance of this problem, then the certificate is the order in which the shops, specified in the input, are visited. Given such an order, it can easily be checked that the length of the tour does not exceed the bound specified in the input.

The class of $\mathcal{NP}$ contains an enormous number of problems, including all problems in $\mathcal{P}$. Many problems in $\mathcal{NP}$ are not known to be solvable in polynomial time, however. It is not known whether $\mathcal{P}$ equals $\mathcal{NP}$, but it is widely conjectured that this is not the case.

We can compare the complexity of two problems in $\mathcal{NP}$ by *reducing* one problem to another. We say that Π_1 reduces to Π_2 ($\Pi_1 \propto \Pi_2$) if there is a *polynomial transformation* from every instance of Π_1 to an equivalent instance of Π_2. In this case, the existence of a polynomial algorithm for Π_2 implies the existence of a polynomial algorithm for Π_1. The existence of such a polynomial transformation shows that Π_1 is a special case of Π_2, or that Π_2 is at least as hard as Π_1.

A problem $\Pi \in \mathcal{NP}$ is said to be *NP-complete* if there exists a polynomial reduction from every problem in $\mathcal{NP}$ to Π. If an NP-complete problem would be solvable in polynomial time, then all problems in $\mathcal{NP}$ would be solvable in polynomial time, implying $\mathcal{P}$=$\mathcal{NP}$. Hence, for an NP-complete problem, a polynomial time algorithm is unlikely to exists.

Notice that once at least one problem Π_1 is known to be NP-complete, showing NP-completeness of another problem Π_2 requires only showing that $\Pi_2 \in \mathcal{NP}$, and that $\Pi_1 \propto \Pi_2$. Cook [11] provided the first NP-completeness proof for a problem, by giving a "master reduction" from every problem in $\mathcal{NP}$ to it.

1.3 Online versus offline

Let us go back to the GSP from Section 1.1 again. Suppose that you have chosen to visit the bakery first, then the drugstore, and finally the winery. When you are on your way from the bakery to the drugstore, your mobile phone rings. It is a call from home, asking you to bring also an onion. You are unlucky: the greengrocer has his shop next to the bakery! If you would have known in advance to bring an onion as well, then you would have visited the greengrocer right after the bakery. You face a new decision: should you continue the tour as planned and get the onion last, or are you turning around to get the onion first? Suppose that you decide on the latter. You turn around grumbling to buy the onion, and continue your tour afterwards. When you are about to pay for the wine, the last part of your mission, your phone rings again. It is again about the onions: please bring two...

It is clear that the GSP changes completely if we do not have all information available before we start a shopping tour. We call this new problem the *online grocery shopping problem* as opposed to the former *offline* version of the problem. Most optimization problems have an *offline* and an *online* variant. The difference between the two variants lies in the amount of a priori information that is given to

the problem solver. In the offline variant all information needed to solve a problem is known, whereas in the online variant information becomes available only during the course of problem solving. In Chapter 3 we will explain the concepts of online optimization more extensively.

1.4 Vehicle routing problems

As the title of this thesis suggests, the main focus of this work is on *vehicle routing problems*. Informally, a vehicle routing problem is a problem in which one is asked to transport a number of items from one place to another, using a number of vehicles. Such a transportation of an item is called a *ride*. The execution of the rides can be subject to a number of constraints.

Vehicle routing problems form a substantial class within the class of *combinatorial optimization* problems. There is a wide variety of problems that do not explicitly involve vehicles and rides but that still can be interpreted as members of this class. As a subclass of vehicle routing problems we consider the class of dial-a-ride problems. These problems are simpler in structure than general vehicle routing problems. In dial-a-ride problems the routing itself plays the central role and there are typically no so called *packing* constraints. This is guaranteed by the assumption that all items have unit size.

The GSP is a vehicle routing problem. The vehicle is the person doing the groceries, and the items are the products that have to be transported from the shops back home. If we impose a restriction on the weight the shopping person can carry at the same time, he might have to return home a couple of times to drop the things that he already picked up. In this setting, the problem has a packing element: Suppose that there are four items, with weights 8, 3, 7, and 2 kg respectively, and that the shopping person can carry only 10 kg at the same time. Clearly, in every feasible solution there is at least one intermediate visit to the house. The tour that minimizes the costs depends heavily on the combinations of the weights, and certainly does not need to be the shortest tour that visits the four shops. However, if we assume that all items have unit size there are no packing constraints, and the problem falls in the class of dial-a-ride problems. We note that we do allow capacity constraints in dial-a-ride problems. As long as all items have unit size, such constraints do not lead to packing constraints since any set of items that requires more than one unit of the capacity can be split apart into a set of items requiring only one unit of the capacity each.

The class of dial-a-ride problems is still rather large. For instance, we can impose time window constraints on rides, saying that the winery does not open before a certain time, or that the bakery closes early. In Chapter 2 we elaborate on the class of dial-a-ride problems.

1.5 Outline of the thesis

This thesis consists of two parts. In the first part, offline dial-a-ride problems are addressed. In Chapter 2 we first define a set of dial-a-ride problems and introduce a notation for problems in this set. Then we study the computational complexity of the problems in this set.

In the second part we study online vehicle routing problems. In Chapter 3 we give an introduction to the field of online computation. We describe the models used in the literature and criticism that this field of research has to cope with.

In Chapters 4 to 7 we study a number of specific online dial-a-ride problems. We start with the online traveling salesman problem in Chapter 4. This problem has already been studied in the literature. We contribute to the existing knowledge by analyzing the behaviour of a special class of online algorithms, and by studying the consequences of limiting the power of the adversary. In Chapter 5 we study online one-server dial-a-ride problems with the objective to minimize the makespan. This problem is also well studied in the literature, however, we introduce an unconventional model for the way information becomes available to the server, and we stress the practical relevance of our model. Chapter 6 deals with online one-server dial-a-ride problems with the objective to minimize the sum of completion times. As a special case, we study the online traveling repairman problem. Our results improve the few existing results for this problem and are valid in a more general setting for which no results had been obtained before. Finally, in Chapter 7, we study an online one-server dial-a-ride problem, where we minimize the maximum flow time of the requests. This is an interesting objective in the sense that the maximum flow time can be associated with dissatisfaction of customers. However, for this objective standard competitive analysis fails to distinguish between online algorithms, since no online algorithm can be competitive at all. We restrict the power of the adversary in a natural way to overcome this problem, and obtain the first positive results for dial-a-ride problems under this objective.

In Chapter 8, we consider a vehicle routing problem, called the online bin coloring problem, that is typically not a dial-a-ride problem. It distinguishes itself from a dial-a-ride problem by loading restrictions on the vehicles. The problem is motivated by a case study for Herlitz, a European distributer of office supply, where a robotized assembly environment was studied.

We conclude with an overview of results on the problems studied in this thesis, and point out the interesting open problems left in this field in Chapter 9.

2

Offline dial-a-ride problems

The content of the section is joint work with J.K. Lenstra, J. Sgall, R.A. Sitters, and L. Stougie, and is based on work done in [41].

2.1 Introduction

In a *dial-a-ride* problem, a set of *servers* with given capacities and speeds have to serve a set of *rides*. Every ride is characterized by a source and a destination, which are points in some metric space. A rich variety of dial-a-ride problems emerges from the characteristics of the servers, the rides, the metric space, and the objective function.

Dial-a-ride problems occur frequently in practice. Think for example of a taxi station, which has to execute a set of rides every day. Taxi cabs, the servers, have to be allocated to clients, the rides. In this example the client literally 'dials a ride.' Other examples occur when controlling a set of elevators in a building and planning routes for couriers.

Dial-a-ride problems form a substantial problem class in the field of combinatorial optimization. A large diversity of problem types can be interpreted as members of this class, among which the classical traveling salesman problem.

Graham et al. [20] present a short and unambiguous notation for machine scheduling problems. The idea is based on the notation introduced by Kendall [28] for queueing systems and extended by Conway et al. [10] to a broader class of queueing and scheduling problems. We propose a similar notation for dial-a-ride problems and describe the problem class that arises from it in Section 2.2.

We examine the computational complexity of the problem types in our classification. In particular we aim to expose the boundary between 'easy' and 'hard' problems. A problem is *easy* if it can be solved in polynomial time, and *hard* if it is NP-hard. In Section 2.3 we first identify equivalence classes of problems that can be described in more that one way by our notation. We then define a partial ordering on our problem class. Roughly speaking, problems lower in the partial ordering are easier than problems higher up. In Section 2.4 we introduce the computer program DaRClass, which systematically employs the partial ordering to decide, given initial sets of easy and hard problems, which of the other problems are easy, hard or still open. DaRClass determines minimal classes of easy and hard problems necessary to describe the current state of knowledge and reveals critical points on which further research can focus. Lageweg et al. [33] undertook a similar study for offline machine scheduling problems.

In Sections 2.5 and 2.6 we present the state of the art in complexity used as input for DaRClass. We prove polynomial solvability of some maximal easy problems in Section 2.5 and NP-hardness of some minimal hard problems in Section 2.6. We maintain a website [42], which contains a more elaborate documentation of the results. All proofs that have been omitted from this Chhapter are given there.

In Section 2.7 we summarize and analyze the output of DaRClass. We make some concluding remarks in Section 2.8.

Throughout this Chapter we tacitly assume that $\mathcal{P} \neq \mathcal{NP}$. Finally, we will restrict ourselves to polynomial-time solvability and ordinary NP-hardness. The concepts of solvability in pseudopolynomial time and NP-hardness in the strong sense are beyond the scope of the present discussion.

2.2 Notation

We characterize dial-a-ride problems by the contents of four fields. In those fields, we give information about the servers, the rides, the metric space, and the objective function, respectively.

Servers. The first field consists of two entries. The first entry is an element of $B_1 = \{1, P, Q, R\}$ and specifies the type of server. There can be a single server or multiple servers. A single server, denoted by 1, travels at unit speed. In case of multiple servers, their number is given as part of the input. As in machine scheduling, we use P for identical parallel servers, Q for uniform parallel servers, and R for unrelated parallel servers. Identical parallel servers all travel at unit speed. Uniform parallel servers travel at their own particular speed. Unrelated parallel servers travel at a speed depending on the number of rides they are executing simultaneously. In this case we assume that a server travels at its own particular speed if it is serving no ride at all and that its speed is a non-increasing function of the number of rides executed simultaneously. Notice that this definition differs from that of unrelated parallel machines, where the speed of a machine depends on the job it is processing.

The second entry of the first field is an element of $B_2 = \{cap1, capc, cap\infty, \circ\}$ and specifies the capacity of the servers. The capacity of a server is defined as the number of rides it can execute simultaneously. The default, denoted by $\circ$, is that each server has its own capacity. Common capacities of 1, c or ∞ are denoted by $cap1$, $capc$ and $cap\infty$, respectively. In case of $capc$, the common capacity c is part of the input.

Rides. The second field specifies the features of the rides and consists of four entries. The first entry concerns the location of the sources and destinations of the rides and is an element of $B_3 = \{S, T, s = t, \circ\}$. Each ride j has a source s_j and a destination t_j, both points in some metric space. More information about the metric space is given below. If $s_i = s_j$ for all i, j, the rides have a common source, which is denoted by S. If $t_i = t_j$ for all i, j, the rides have a common destination, denoted by T. We assume that S and T are located in the origin of the metric space. If $s_j = t_j$ for all j, each ride has a coinciding source and destination: the rides have length 0, and the requested points just have to be visited, as in the traveling salesman problem; this case is denoted by $s = t$. The default is that none of the three situations described occurs.

The second entry specifies time window constraints and is an element of $B_4 = \{(r_j, d_j), r_j, d_j, \circ\}$. A ride j cannot be started before its release time r_j and has to be finished by its deadline d_j. We denote the general case by (r_j, d_j), the case that there are no deadlines (all $d_j = \infty$) by r_j, and the case that there are no release dates (all $r_j = 0$) by d_j. The default is that there are neither release dates nor deadlines.

The third entry is an element of $B_5 = \{pmtn, \circ\}$ and indicates whether or not preemption is allowed. If preemption is allowed, a server may start a ride at its source, interrupt it before reaching its destination in order to serve another ride, and resume the ride later at the point where it was interrupted. If there is more than one server, the ride may be resumed by another server than the one that started it; this is called hand-over. No preemption is the default.

The last entry is an element of $B_6 = \{prec, \circ\}$ and indicates if precedence constraints on the rides are to be respected. If so, an acyclic digraph $G = (V, A)$ is given. The vertex set V is the set of rides. For each arc $(i, j) \in A$, ride j cannot start before ride i is finished. We will not distinguish between different structures of the precedence graph. No precedence constraints is the default.

Metric space. The third field specifies the metric space on which the problem is defined. There is only one entry, which is an element of $B_7 = \{line, tree, graph, \mathbb{R}^d, \circ\}$. On the line and on $\mathbb{R}^d$ we use the Euclidean metric. On trees and graphs we assume that the metric space contains all vertices of the graph, and for two such vertices x and y, $d(x, y)$ is the length of the shortest path in the graph from x to y. The default is a general metric space. In every metric space we specify an origin and we assume that each server starts and ends in the origin. Distances are symmetric and satisfy the triangle inequality.

Objective. The last field specifies the objective function and is an element of $B_8 = \{C_{\max}, \sum C_j, \sum w_j C_j\}$. The first objective is to minimize the makespan, i.e. the time all rides have been served and the last server is back in the origin. This will be denoted by $C_{\max}$.

The second objective is to minimize the sum of completion times $\sum C_j$, where C_j is the completion time of ride j. This objective is equivalent to minimizing the average completion time, also referred to as latency. The last objective is to minimize the sum of weighted completion times $\sum w_j C_j$, where every ride j has a given weight w_j. In this field, no default is used. Note that the assumption that the servers have to end in the origin does not influence the solution in latency problems. Note also that $C_{\max}$ is a slight abuse of notation: the makespan is not necessarily equal to the maximum ride completion time, since the servers have to return to the origin. In this chapter we will restrict ourselves to these three objectives, although most of the other objectives used in machine scheduling occur in practical dial-a-ride problems as well.

A dial-a-ride problem is thus represented as $\beta_1, \beta_2 | \beta_3, \beta_4, \beta_5, \beta_6 | \beta_7 | \beta_8$, where $\beta_i \in B_i$, $i = 1, \ldots, 8$. An empty field means that all entries of that field have been given the default value.

For example, $1, cap1|S|line|C_{\max}$ is the problem in which one server has to transport one item at a time from the origin to a number of destinations on a line, minimizing the makespan. $P, capc|(r_j, d_j)|| \sum C_j$ is the pick-up and delivery problem with multiple couriers that all have the same capacity and that have to serve requests for delivery within certain time windows so as to minimize the average completion time of the rides. And $1|s = t||C_{\max}$ is the familiar traveling salesman problem.

2.3 Equivalence classes and partial ordering

Let $\mathcal{S}$ be the class of dial-a-ride problems defined in Section 2.2. The purpose of this section is to define a *partial ordering* $\rightarrow$ on $\mathcal{S}$. This relation should have the property that, for any two problems $\Pi, \Pi' \in \mathcal{S}$,

- if $\Pi \rightarrow \Pi'$ and Π' is easy, then Π is easy, and
- if $\Pi \rightarrow \Pi'$ and Π is hard, then Π' is hard.

For example, $\Pi \rightarrow \Pi'$ may hold because the set of instances of Π is evidently included in the set of instances of Π'. There may be subtler reasons too. In general, $\Pi \rightarrow \Pi'$ implies that Π is *reducible* to Π' ($\Pi \propto \Pi'$).

In attempting to define the relation $\rightarrow$, we encounter a problem that was anticipated by Lageweg et al. [33]. It may happen that, for obvious reasons, both $\Pi \propto \Pi'$ and $\Pi' \propto \Pi$. That is, there may be certain sets of problems of equivalent complexity. We will first identify several such equivalence classes within $\mathcal{S}$ and replace each of them by a single representative. We will then define a proper partial ordering $\rightarrow$ on the reduced class $\mathcal{S}$.

Lemma 2.1. *Any two problems* $1, capc|\beta_3, \beta_4, \beta_5, \beta_6|\beta_7|\beta_8$ *and* $1, \circ|\beta_3, \beta_4, \beta_5, \beta_6|\beta_7|\beta_8$ *are equivalent.*

PROOF. Obvious. □

We represent any such pair of problems by the latter one.

Lemma 2.2. *Any three problems* $\Pi = R, capc|\beta_3, \beta_4, \beta_5, \beta_6|\beta_7|\beta_8$, $\Pi' = R, cap\infty|\beta_3, \beta_4, \beta_5, \beta_6|\beta_7|\beta_8$, *and* $\Pi'' = R, \circ|\beta_3, \beta_4, \beta_5, \beta_6|\beta_7|\beta_8$ *are equivalent.*

PROOF. We will show that $\Pi \propto \Pi''$, $\Pi'' \propto \Pi'$, and $\Pi' \propto \Pi$. Clearly Π is a special case of Π''. An instance of Π'' can be viewed as an instance of Π' in which the speed of a server is equal to the speed of the corresponding server in Π'' if it carries no more items than the capacity of that server, and 0 if its load is higher. Finally, $\Pi' \propto \Pi$ when we give the servers in Π a common capacity equal to the number of rides, which imposes no limitation. □

We will represent any such triple of problems by the third one.

Lemmas 2.1 and 2.2 show that some combinations of server type and capacity are redundant. To exclude these, we merge the sets B_1 and B_2 into the set $B_9 = \{(1, cap1), (1, \circ), (1, cap\infty), (P, cap1), (P, capc), (P, cap\infty), (P, \circ), (Q, cap1), (Q, capc), (Q, cap\infty), (Q, \circ), (R, cap1), (R, \circ)\}$.

Lemma 2.3. *Any two problems* $\Pi = \beta_1, \beta_2|s = t, \beta_4, pmtn, \beta_6|\beta_7|\beta_8$ *and* $\Pi' = \beta_1, \beta_2|s = t, \beta_4, \circ, \beta_6|\beta_7|\beta_8$ *are equivalent.*

PROOF. There is no point in preempting a ride of length 0. □

We will represent any such pair of problems by the non-preemptive version.

Since the combination of coinciding sources and destinations and preemption will not occur, we merge the sets B_3 and B_5 into the set $B_{10} = \{S, T, s = t, (\circ, \circ), (S, pmtn), (T, pmtn), (\circ, pmtn)\}$. Every problem can now be written in the form $\beta_9|\beta_{10}, \beta_4, \beta_6|\beta_7|\beta_8$.

The equivalences proved below involve entries from more than one field.

Lemma 2.4. *Any two problems* $\Pi = \beta_1, \beta_2|s = t, \beta_4, \beta_5, \beta_6|\beta_7|\beta_8$ *and* $\Pi' = \beta_1, \beta_2'|s = t, \beta_4, \beta_5, \beta_6|\beta_7|\beta_8$ *are equivalent.*

PROOF. If sources and destinations coincide, capacities are irrelevant. □

Because of Lemma 2.4, we will not specify β_2 if β_3 is equal to $s = t$.

Lemma 2.5. *Any two problems* $\Pi = Q, \beta_2|s = t, \beta_4, \beta_5, \beta_6|\beta_7|\beta_8$ *and* $\Pi' = R, \beta_2|s = t, \beta_4, \beta_5, \beta_6|\beta_7|\beta_8$ *are equivalent.*

PROOF. The servers always travel empty. □

The problem with uniform parallel servers will represent any such pair.

Lemma 2.6. *Any two problems* $\Pi = \beta_1, \beta_2 | s = t, \beta_4, \beta_5, \circ | \beta_7 | C_{\max}$ *and* $\Pi' = \beta_1, cap\infty | T, \beta_4, \beta_5, \circ | \beta_7 | C_{\max}$ *with* $\beta_1 \in \{1, P, Q\}$ *and* $\beta_4 \in \{\circ, r_j\}$ *are equivalent.*

PROOF. Since the servers in Π' have infinite capacity, they simply collect all items before returning to the origin, which is T. □

Lemma 2.7. *Any two problems* $\Pi = \beta_1, \beta_2 | s = t, \beta_4, \beta_5, \circ | \beta_7 | \beta_8$ *and* $\Pi' = \beta_1, cap\infty | S, \beta_4, \beta_5, \circ | \beta_7 | \beta_8$ *with* $\beta_1 \in \{1, P, Q\}$ *and* $\beta_4 \in \{\circ, d_j\}$ *are equivalent.*

PROOF. Similar to the proof of Lemma 2.6. □

The problem with coinciding source and destination will represent any pair of equivalent problems identified by Lemma 2.6 or 2.7.

Lemma 2.8. *Any two problems* $\Pi = 1, cap\infty | \beta_3, \beta_4, pmtn, \beta_6 | \beta_7 | \beta_8$ *and* $\Pi' = 1, cap\infty | \beta_3, \beta_4, \circ, \beta_6 | \beta_7 | \beta_8$ *are equivalent.*

PROOF. Because there is only one server, hand-over is not possible. Preempting a ride is not useful since the server can carry all rides at no additional cost. □

The non-preemptive problem will represent any such pair.

Lemma 2.9. *Any two problems* $\Pi = \beta_1, \beta_2 | \beta_3, \beta_4, \beta_5, \beta_6 | \beta_7 | C_{\max}$ *and* $\Pi' = \beta_1, \beta_2 | \beta_3', \beta_4', \beta_5, \beta_6 | \beta_7 | C_{\max}$ *with*

$$\beta_3' = \begin{cases} T & \text{if } \beta_3 = S, \\ S & \text{if } \beta_3 = T, \\ \beta_3 & \text{otherwise}, \end{cases}$$

$$\beta_4' = \begin{cases} r_j & \text{if } \beta_4 = d_j, \\ d_j & \text{if } \beta_4 = r_j, \\ \beta_4 & \text{otherwise}, \end{cases}$$

are equivalent.

PROOF. Consider the decision version of both problems, in which, given a value D, we ask if a solution with objective value $C_{\max} \leq D$ exists. For any instance of Π, construct an instance of Π' as follows:
– reverse all rides, i.e., interchange the source and destination of every ride;

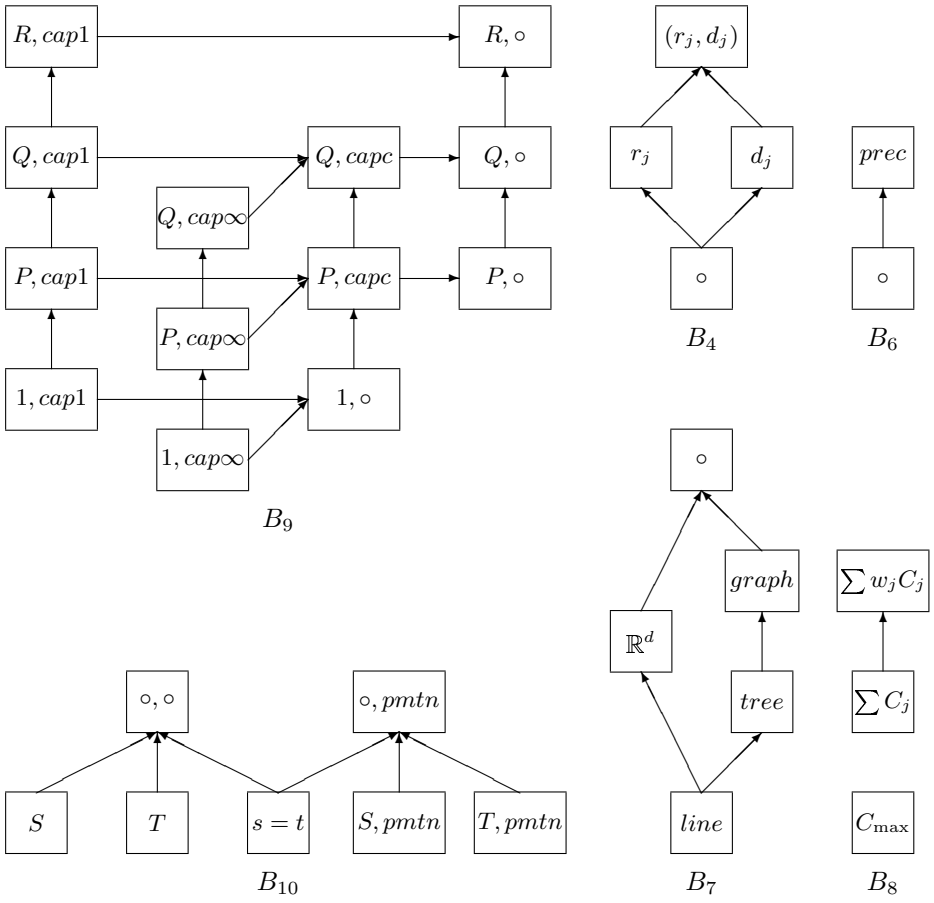

Figure 2.1: The digraphs

– replace every time window (r_j, d_i) by $(D - d_j, D - r_j)$;
– reverse every arc in the precedence graph.
For each feasible solution of Π, we obtain a feasible solution of Π' by reversing the paths of the servers. Hence, $\Pi \propto \Pi'$. $\Pi' \propto \Pi$ follows by symmetry. □

The deadline version will represent any such pair. If there is a choice, the common source version will be used.

By application of the above lemmas, we reduce the number of problem types in the class $\mathcal{S}$ by 48%, from 15,360 to 7,930.

In Section 2.2 we defined the sets $B_1, \ldots, B_8$ of possible values for the entries in every field. In this section we replaced B_1 and B_2 by B_9 and B_3 and B_5 by B_{10} in order to eliminate some of the redundancy in $\mathcal{S}$. The sets $B_4, B_6, \ldots, B_{10}$ and

relations between their elements are illustrated by the six digraphs in Figure 2.1. The nodes are the elements of B_i ($i = 4, 6, \ldots, 10$). There is an arc from one element to another if the latter is a direct generalization of the former.

We use these digraphs to define the partial ordering $\rightarrow$ on $\mathcal{S}$. For two problems $\Pi = \beta_9|\beta_{10}, \beta_4, \beta_6|\beta_7|\beta_8$ and $\Pi' = \beta'_9|\beta'_{10}, \beta'_4, \beta'_6|\beta'_7|\beta'_8$ in $\mathcal{S}$, we have $\Pi \rightarrow \Pi'$ if either $\beta_i = \beta'_i$ or there is a directed path from β_i to β'_i for $i = 4, 6, \ldots, 10$.

We augment the relation $\rightarrow$ by taking into account the following relationship between makespan and latency problems.

Lemma 2.10. *For any two problems* $\Pi = \beta_9|\beta_{10}, \beta_4, \beta_6|\beta_7|C_{\max}$ *and* $\Pi' = \beta_9|\beta_{10}, \beta'_4, \beta_6|\beta_7|\sum C_j$ *with* $\beta'_4 = d_j$ *if* $\beta_4 \in \{d_j, \circ\}$, $\beta'_4 = (r_j, d_j)$ *otherwise, we have* $\Pi \propto \Pi'$.

Proof. Consider the decision version of Π, in which, for a given D, we ask for the existence of a solution with $C_{\max} \leq D$. If a ride in Π' has a deadline larger than D, change it to D. □

2.4 The program DaRClass

The program DaRClass has been written to determine the computational complexity of the members of the class $\mathcal{S}$ of dial-a-ride problems. For any two problems Π, $\Pi' \in \mathcal{S}$ the program knows whether or not $\Pi \rightarrow \Pi'$, where $\rightarrow$ is the partial ordering defined in Section 2.3.

The input of DaRClass consists of a class $\mathcal{S}^*_{in}$ of known easy problems and a class $\mathcal{S}^!_{in}$ of known hard problems. Using the input and the partial ordering, DaRClass partitions $\mathcal{S}$ into three classes $\mathcal{S}^*$, $\mathcal{S}^!$, and $\mathcal{S}^?$ of easy, hard, and open problems, respectively:

$$\begin{aligned}
\mathcal{S}^* &= \{\Pi \in \mathcal{S} \mid \exists \Pi' \in \mathcal{S}^*_{in} : \Pi \rightarrow \Pi'\},\\
\mathcal{S}^! &= \{\Pi \in \mathcal{S} \mid \exists \Pi' \in \mathcal{S}^!_{in} : \Pi' \rightarrow \Pi\},\\
\mathcal{S}^? &= \mathcal{S} \backslash (\mathcal{S}^* \cup \mathcal{S}^!).
\end{aligned}$$

Obviously, $\mathcal{S}^* \cap \mathcal{S}^! = \emptyset$. DaRClass also determines four subclasses of problems that are minimal or maximal within their class with respect to the partial ordering and the current state of knowledge:

$$\begin{aligned}
\mathcal{S}^*_{\max} &= \{\Pi \in \mathcal{S}^* | \nexists \Pi' \in \mathcal{S}^* \backslash \{\Pi\} : \Pi \rightarrow \Pi'\};\\
\mathcal{S}^?_{\min} &= \{\Pi \in \mathcal{S}^? | \nexists \Pi' \in \mathcal{S}^? \backslash \{\Pi\} : \Pi' \rightarrow \Pi\};\\
\mathcal{S}^?_{\max} &= \{\Pi \in \mathcal{S}^? | \nexists \Pi' \in \mathcal{S}^? \backslash \{\Pi\} : \Pi \rightarrow \Pi'\};\\
\mathcal{S}^!_{\min} &= \{\Pi \in \mathcal{S}^! | \nexists \Pi' \in \mathcal{S}^! \backslash \{\Pi\} : \Pi' \rightarrow \Pi\}.
\end{aligned}$$

The maximal easy and minimal hard problems represent the essential results, which imply all others. The minimal and maximal open problems are obvious targets for further research.

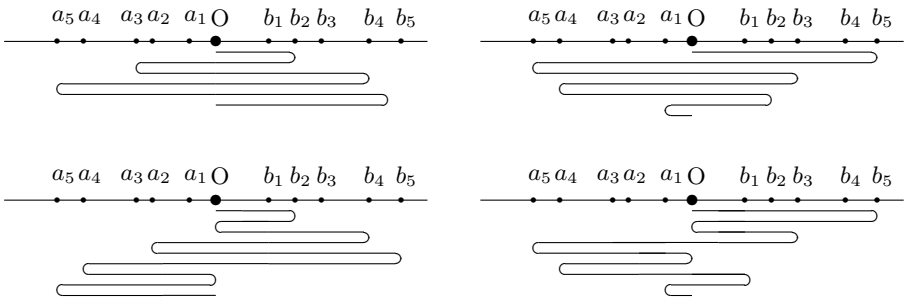

Figure 2.2: Possible spiral forms of optimal solutions

As long as $\mathcal{S}^?$ is not empty, $\mathcal{S}^*_{\max}$ and $\mathcal{S}^!_{\min}$ are subject to change. The maximal easy and minimal hard problems bound the open problems from below and above. Each time an open problem is resolved, the boundaries move closer.

In principle, the status of every problem can be determined by hand, on the basis of $\mathcal{S}^*_{\max}$, $\mathcal{S}^!_{\min}$, and $\rightarrow$. The size of $\mathcal{S}$ and the complexity of $\rightarrow$ make manual operations awkward, however. DARCLASS has been designed to provide automated assistance.

2.5 Maximal easy problems

Our website [42] gives a complete documentation of all of the polynomial-time algorithms and NP-hardness proofs that represent the state of the art. In this section we discuss three characteristic polynomial-time algorithms. The next section gives a representative selection of six NP-hardness proofs.

Many of the easy problems that have the line as their metric space have in common that an optimal solution has a certain 'spiral' structure that allows for solution by dynamic programming. The dynamic program can often be solved in polynomial time, but not always. Afrati et al. [1] show that the problem $1|s = t, d_j|line|\sum C_j$ is NP-hard, although an optimal solution has a spiral structure; they give a fully polynomial-time approximation scheme for this problem.

Figure 2.2 shows various kinds of spiral structures that can occur. The horizontal axis is the line on which the problem is defined. The spiral represents the path of the server. For example, if a solution has the shape of the upper left spiral, the server never turns around at a point between the origin and a point where it turned before. If we know that an optimal solution has one of these spiral structures, we only have to determine the turning points of the server in an optimal solution. In many cases, this can be done in polynomial time.

We will take the problem $1, cap\infty|T|line|\sum w_jC_j$ as an example, show that an optimal solution has the shape of the lower left spiral in Figure 2.2, and give the dynamic program that determines the turning points in an optimal solution. Our algorithm requires time $O(n^3)$, where n is the number of rides.

Theorem 2.11. *$1, cap\infty|T|line|\sum w_jC_j$ can be solved in time $O(n^3)$.*

PROOF. Label the sources of the rides on the left-hand side of the origin from right to left $a_1, \ldots, a_p$, and those on the right-hand side of the origin from left to right $b_1, \ldots, b_q$. The total number of rides is $p + q = n$. Let $a_0 = b_0 = O$. For any x, let $d(O, x)$ be the distance between O and x.

Since the server has infinite capacity, there is an optimal solution in which every item is picked up as soon as its source is visited. Hence, if the server reaches a_i, all rides with source between O and a_i either have been completed already, or will be completed the first time the server is back in the origin. For the rides on the right-hand side of the origin the same argument holds. Therefore, an optimal solution has the shape of the lower left spiral in Figure 2.2.

An optimal solution can be computed by the following recursion. We define $V(a_i, b_j)$ as a lower bound on the optimum solution value, consisting of the minimum cost of serving all rides in $[a_i, b_j]$ plus the current time times the weighted number of unserved rides. Clearly, the optimum is equal to $V(a_p, b_q)$.

We initialize by setting $V(0, 0) = 0$, and compute $V(a_i, b_j)$ recursively by

$$\begin{aligned} &V(a_i, b_j) = \\ &\min\{\min_{0 \le h \le j-1}\{V(a_i, b_h) + 2d(O, b_j)(\textstyle\sum_{i<k\le p} w_{a_k} + \sum_{h<k\le q} w_{b_k})\}, \\ &\qquad \min_{0 \le l \le i-1}\{V(a_l, b_j) + 2d(O, a_i)(\textstyle\sum_{l<k\le p} w_{a_k} + \sum_{j<k\le q} w_{b_k})\}\}. \end{aligned}$$

We have to compute $O(n^2)$ values of V, each requiring time $O(n)$. Thus, the entire dynamic program takes time $O(n^3)$. □

Another basic problem is $1, cap\infty||line|C_{\max}$. An optimal solution has the nice structure shown in Figure 2.3.

Theorem 2.12. *$1, cap\infty||line|C_{\max}$ can be solved in time $O(n \log n)$.*

PROOF. We first show that an optimal solution has the spiral structure of Figure 2.3. Let X and Y be the left and right extreme points, and let x and y be the other turning points, where $X \le x \le 0 \le y \le Y$. First note that the server has to visit X and Y at least once, and that all rides that have a source and destination on the same side of the origin can be served while visiting the extreme points. Hence, turning around without crossing the origin is a waste of time. Suppose without loss of generality that the server visits Y before X. Since the server can pick up all sources on the right-hand side while moving to Y, and since it has to go all the way to X afterwards, it will not visit the right-hand side before the trip to Y in an optimal solution. So an optimal tour starts with $O \to x \to O \to Y \to O$; it may be that $x = 0$. At time $2x + 2Y$ the server is back in the origin. It then only has to visit X and deliver rides that have their source in (X, x). Among these rides, let y be the rightmost destination; if $y < 0$, set $y = 0$. A trivial lower bound on the makespan is $2x + 2Y + 2X + 2y$. This lower bound is attained if the tour has the claimed structure.

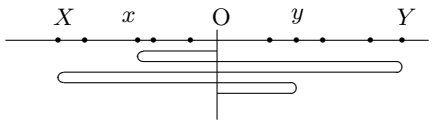

Figure 2.3: Spiral form of an optimal solution

We now show how to determine an optimal tour among the tours of this structure. As shown above, for every x we can determine the corresponding y and the makespan $2x+2Y+2X+2y$. Hence, we have to consider no more than $n+1$ combinations (x, y). After sorting the rides in time $O(n \log n)$, we determine all these combinations in time $O(n)$ and find an optimal solution by taking the minimum over the makespan values. □

Our third easy problem is $1, cap1|S, d_j, prec||C_{\max}$. In solving it, we use the striking similarity that exists between many scheduling and dial-a-ride problems. Celebrated scheduling algorithms such as Smith's shortest processing time rule [51] and Jackson's earliest due date rule [26] can sometimes be used to solve dial-a-ride problems too. The solution method for the problem below is a straightforward extension of the solution method for the scheduling problem $1|prec, d_j|C_{\max}$ due to Lawler and Moore [34].

Theorem 2.13. *$1, cap1|S, d_j, prec||C_{\max}$ can be solved in time $O(|A| + n \log n)$.*

Proof. We first modify the problem by reversing the direction of every ride j and adding $d(O, t_j)$ to the deadline d_j. Since the server has capacity 1, it has to visit the origin between every two rides. The transformation therefore changes neither the set of feasible solutions nor their values.

We now modify the deadlines once more to account for the precedence constraints. For each arc (i, j) of the precedence digraph, we may set

$$d_i = \min\{d_i, d_j - 2d(O, t_j)\}$$

without changing the feasible solution set. The deadline modification can be carried out in time proportional to the number of arcs. Infeasibility is detected as soon as a deadline becomes negative.

We finally sort the rides in order of non-decreasing deadlines. This yields a solution that satisfies the precedence constraints and that, when the server is never idle, has minimum makespan. □

2.6 Minimal hard problems

We next give a selection of the NP-hardness proofs that lead to the current state of the art. The core problems from which the reductions have been made are listed

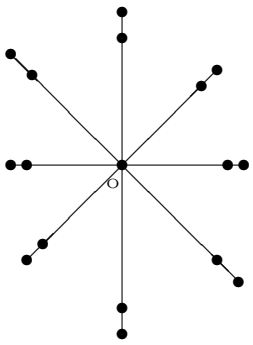

Figure 2.4: Spider graph

below.

PARTITION [27]: Given a finite set $N = \{1, \ldots, n\}$ and a weight $a_j \in \mathbb{Z}^+$ for every $j \in N$ such that $\sum_{j \in N} a_j = 2b$, does there exist a subset $N' \subseteq N$ such that $\sum_{j \in N'} a_j = b$?

3-PARTITION [14]: Given a set $N = \{1, \ldots, 3n\}$ and a weight $a_j \in \mathbb{Z}^+$ for every $j \in N$ such that $\sum_{j \in N} a_j = nb$ and $b/4 < a_j < b/2$ for all $j \in N$, can N be partitioned into n subsets $N_1, \ldots N_n$ such that $\sum_{j \in N_i} a_j = b$ for $1 \leq i \leq n$?

HAMILTONIAN PATH [27]: Given a graph $G = (V, E)$, does G contain a Hamiltonian path?

CIRCULAR ARC COLORING [16]: Given a finite set of intervals on the boundary of a circle and an integer k, can the intervals be colored with at most k colors such that no two overlapping intervals have the same color?

MINIMUM VERTEX FEEDBACK [27]: Given a digraph $D = (V, A)$ and an integer k, does there exist a subset $U \subseteq V$ such that $|U| \leq k$ and the induced subdigraph on the vertex set $V \backslash U$ is acyclic?

SHORTEST COMMON SUPERSEQUENCE [40]: Given a finite set L of equal-length strings over a two-symbol alphabet σ and an integer k, does there exist a string l over σ of length at most k such that each string in L is a subsequence of l?

All of these problems are NP-hard in the strong sense, with the exception of PARTITION, which is solvable in pseudopolynomial time.

Theorem 2.14. PARTITION $\propto$ $1|s = t, d_j|tree|C_{\max}$.

PROOF. For any instance of PARTITION we define an instance of the dial-a-ride problem as follows. Let $c = 2b$. We define a spider graph with n legs; see Figure 2.4. For every element j ($j \in N$), we introduce two zero-length rides A_j and B_j, both on the jth leg of the spider. Ride A_j is at distance ca_j from the root and has deadline $2\sum_{i<j}(c+1)a_i + ca_j$. Ride B_j is at distance $(c+1)a_j$ from the root and has deadline ∞ for $1 \leq j \leq n-1$ and $d_{B_n} = 4cb - (c-1)a_n + 2b$ for $j = n$. The deadlines are chosen such that each ride A_j must be served immediately after A_{j-1} or B_{j-1}. The deadline of A_j allows for serving all B_i ($i = 1, \ldots, j-1$) before A_j. However, the deadline of B_n is chosen such that $2c\sum_{j=1}^{n-1} a_j + (c+1)a_n = d_{B_n} - 2b$. Thus, in any feasible solution, we must serve the rides $A_1, A_2, \ldots, A_n, B_n$ in this order. Moreover, for a subset $N' \subseteq N\backslash\{n\}$ with $\sum_{j\in N'} a_j \leq b$, we can insert the rides B_j ($j \in N'$) in this order, each B_j directly after A_j. Serving the remaining rides B_j ($j \notin N', j \neq n$) after B_n gives a solution of makespan

$$2c\sum_{j=1}^{n-1} a_j + 2\sum_{j\in N'} a_j + 2(c+1)\sum_{j\notin N'} a_j \geq 2c\sum_{j=1}^{n-1} a_j + 2b + 2(c+1)b,$$

with equality if and only if we have a yes-instance of PARTITION. □

Theorem 2.15. SHORTEST COMMON SUPERSEQUENCE $\propto$ $1|s = t, prec|line|C_{\max}$.

PROOF. Take any instance of SHORTEST COMMON SUPERSEQUENCE. Suppose that $\sigma = \{-1, 1\}$ and that each string in L has length n. We define an instance of the dial-a-ride problem as follows. For every string $(l_1, l_2, \ldots, l_n) \in L$ we introduce $2n-1$ zero-length rides: n rides in l_i ($i = 1, \ldots, n$) and $n-1$ rides in O; these rides are subject to a chain-like precedence relation of the form $l_1 \to O \to l_2 \to O \to \ldots \to l_{n-1} \to O \to l_n$.

Suppose that there is a string l of length k that contains each string in L as a subsequence. If we transform l into a chain of $2k-1$ rides by inserting $k-1$ rides in O as above, then we obtain a feasible solution to the dial-a-ride problem of length $2k$; note that the server starts and ends in the origin. Conversely, if the dial-a-ride problem has a solution of length $2k$, then the sequence of visits to the points 1 and -1 defines a string of length k that provides a yes-answer to SHORTEST COMMON SUPERSEQUENCE.

Hence, the precedence-constrained traveling salesman problem on the line is NP-hard, even if the precedence relation is a collection of chains. □

Theorem 2.16. MINIMUM VERTEX FEEDBACK $\propto$ $1, cap\infty||tree|C_{\max}$.

PROOF. Take any instance of MINIMUM VERTEX FEEDBACK. Let $|V| = n$. We define an instance of the dial-a-ride problem as follows. The tree is a star graph

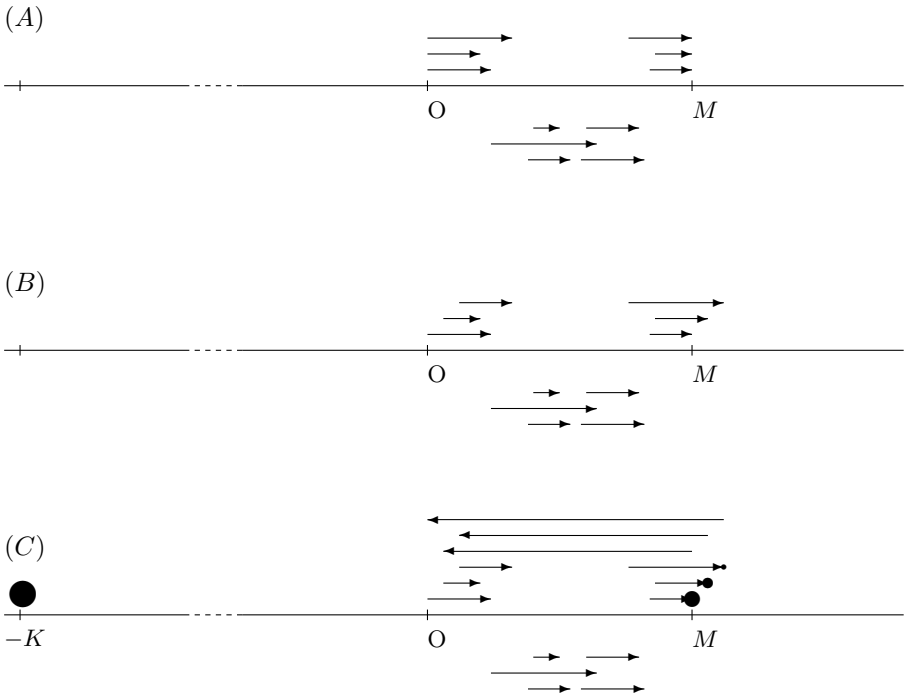

Figure 2.5: Construction of instance of $1, cap1||line|\sum C_j$. *The black dots denote the sets of rides with coinciding source and destination. The bigger the dot, the more rides are located at the same point.*

with vertex set $\{O\} \cup V$ and n unit-length edges $\{O, i\}$ $(i \in V)$. For every arc $(i, j) \in A$ we introduce a ride from i to j.

Suppose there is a subset $U \subseteq V$ of size at most k such that $D' = (V \backslash U, A)$ is acyclic. Consider a solution for the dial-a-ride problem that first visits the vertices in U in any order, then the vertices in $V \backslash U$ according to a topological ordering, and finally the vertices in U again. Such a tour, starting and ending in O, is feasible and has length at most $2n + 2k$. Conversely, suppose that the dial-a-ride problem has a solution of length at most $2n + 2k$ with $k \leq n$. Then at most k vertices are visited more than once. Let U be the set of such vertices. It follows that $D' = (V \backslash U, A)$ is acyclic. Hence, we have a yes-instance of MINIMUM VERTEX FEEDBACK if and only if the dial-a-ride instance has a solution of length at most $2n + 2k$. □

Theorem 2.17. CIRCULAR ARC COLORING $\propto 1, cap1||line|\sum C_j$

PROOF. For any instance of CIRCULAR ARC COLORING we define an instance of the dial-a-ride problem as follows. Let n be the number of intervals on the circle and let X be its perimeter. We cut the circle at a point where k intervals overlap; we may assume without loss of generality that such a point exists. We assume that both parts of every interval cut have length at least k; otherwise we scale the problem appropriately. The point where the circle is cut becomes point O on the line; every other point on the circle gets a coordinate on the line equal to its distance in clockwise direction from the cut point. For each interval on the line thus obtained we introduce a ride from left to right. This gives $n + k$ rides, including k that start in O and k that end in X. We call these the A-rides; see Figure 2.5(i).

Label the rides such that ride A_i has its source in O, ride A_{k+i} has its destination in X, and A_i and A_{k+i} originate from the same interval on the circle, for $1 \leq i \leq k$. Label the other rides arbitrarily A_i, for $2k + 1 \leq i \leq n + k$. Now modify the rides $A_1, \ldots, A_k$ by deleting the initial part of A_i such that it starts in $i - 1$, and modify the rides $A_{k+1}, \ldots, A_{2k}$ by extending A_{k+i} such that it ends in $X + i - 1$; see Figure 2.5(ii).

We next introduce the B-rides and the C-rides. For $i = 1, \ldots, k$, we define $(k - i + 1)b$ zero-length B-rides in point $X + i - 1$, for a value of b to be determined later. For $i = 1, \ldots, k - 1$, we define ride C_i starting in $X + i - 1$ and ending in i, and we define ride C_k starting in $X + k - 1$ and ending in O. Finally, we define d zero-length D-rides, located in $-Y$, for values of d and Y to be determined later. See Figure 2.5(iii).

Suppose that the instance of CIRCULAR ARC COLORING has a k-coloring. The following tour serves all rides. Start in O. Serve A_1 and all rides that correspond to intervals with the same color as the interval corresponding to A_1. By construction, all such rides have been served when we reach the destination of A_{k+1}. Then serve the kb B-rides in that point and serve C_1, which ends where A_2 starts. Next serve A_2 and all rides corresponding to intervals with the same color. Then serve the $(k-1)b$ B-rides in $X + 1$ and serve C_2. Continuing in this way, we serve all rides of type A, B, and C, and we are back in the origin at time $2kX$. We reach the D-rides at time $2kX + Y$.

In this solution, the D-rides contribute $(2kX + Y)d$ to the total completion time, the B-rides contribute $Xkb + (3X - 1)(k - 1)b + (5X - 2)(k - 2)b + \cdots + ((2k - 1)X - (k - 1))b$, and the rides of type A and C contribute at most $2kX(n + 2k)$ together. Hence, the objective value of our solution is no more than

$$Z = (2kX + Y)d + Xkb + (3X - 1)(k - 1)b +$$

$$(5X - 2)(k - 2)b + \cdots + ((2k - 1)X - (k - 1))b + 2kX(n + 2k).$$

Now suppose that no k-coloring exists. In that case, the tour described above cannot serve all A-rides. We will choose the parameters b, d, and Y such that every tour that deviates from this tour has a total completion time larger than Z. As a result, the dial-a-ride problem has a solution of value at most Z if and only if a k-coloring exists.

First set $d = Xkb + (3X-1)(k-1)b + (5X-2)(k-2)b + \cdots + ((2k-1)X - (k-1))b + 2kX(n+2k) + 1$, to make sure that the D-rides are served no later than $2kX + Y$. If they are delayed by one time unit, the total completion time exceeds Z.

Now set $Y = kXd+1$ to make sure that the D-rides will be served only after all other rides have been served. Serving a ride after the D-rides increases its completion time by at least $2Y$. Serving the D-rides earlier yields a gain of at most $2kXd$. So by the choice of Y, serving the D-rides before any other ride increases the total completion time. This in combination with the choice of d implies that the server must be in the origin at time $2kX$ after having served all rides of type A, B, and C. This forces the server to start with ride A_1 and to turn around only if it can start another ride immediately.

To reach our goal, we only have to make sure that $A_{k+1}, \ldots, A_{2k}$ are served in the same order as $A_1, \ldots, A_k$. Notice that this implies that A_i is served before A_j for $1 \le i < j \le k$. Set $b = k(n+2k)$. Serving the rides in any other order such that the server is still back in the origin at time $2kX$ after having served all rides of type A, B, and C increases the total completion time of the B-rides by at least $(2X-1)b = (2X-1)k(n+2k) > 2kX(n+k)$. As only the A-rides can be served earlier in this way, the decrease is less than $2kX(n+k)$. The resulting total completion time is therefore larger than Z. □

Theorem 2.18. HAMILTONIAN PATH $\propto 1, cap1|pmtn|tree|\sum C_j$.

PROOF. For any instance of HAMILTONIAN PATH with $V = \{1, \ldots, n\}$, we define an instance of the dial-a-ride problem as follows. The tree is a star graph with vertex set $\{O, 1, \ldots, n, n+1\}$, n unit-length edges $\{O,1\}, \ldots, \{O,n\}$, and an edge $\{O, n+1\}$ of length $b = 2n^4$. For each leaf vertex i ($i = 1, \ldots, n+1$) we introduce $c = 2b$ zero-length rides in i. For each edge $(i,j) \in E$, we define two rides, one from i to j and one from j to i. We claim that there is a solution to the dial-a-ride problem with total completion time at most $Z = (n^2+2n+b)c + 2(n+b)(2|E| - (n-2)) + n^2$ if and only if a Hamiltonian path exists.

Suppose there is a Hamiltonian path H. We first visit each of the vertices $\{1, \ldots, n\}$ once in the order in which they appear on H, and then vertex $n+1$. In doing so, we serve the $n-1$ rides corresponding to the edges of H. After this, $2|E|-(n-1)$ rides are unserved. The total completion time is at most

$$\begin{aligned}&(1 + 3 + \cdots + (2n-1) + (2n+b))c + 3 + 5 + \cdots + (2n-1) + \\ &+2(n+b)(2|E| - (n-1)) + 3 + 7 + \cdots + 4(2|E| - (n-1)) - 1 \\ &< (n^2 + 2n + b)c + n^2 + 2(n+b)(2|E| - (n-1)) + 4n^4 < Z,\end{aligned}$$

where the last inequality follows from the choice of b.

Suppose no Hamiltonian path exists. If fewer than $n-1$ rides corresponding to edges of E are served before $n+1$ is visited, then at least $2|E| - (n-2)$ rides are

postponed until after time $2n + 2b$, yielding a total completion time larger than Z. To prevent this, at least one leaf must be visited twice before all leaves have been visited once. Compared to the previous situation, this increases the costs by at least $2c$, because at least one of the groups of c zero-length rides is delayed by 2 time units, again yielding a total completion time larger than Z.

Serving more than $n - 1$ rides corresponding to arcs in E before visiting $n + 1$ decreases the total completion time by at most $2(n+b)$ per ride. But for every such ride, at least c requests are delayed by 2 time units, so per ride the net increase of the total completion time is $2c - 2(n + b) > 0$. Therefore, if $\sum C_j \leq Z$, there is a Hamiltonian path.

Clearly, allowing preemption does not change the above argument. □

Theorem 2.19. 3-PARTITION $\propto P, cap1|S|line|\sum w_jC_j$

PROOF. For any instance of 3-PARTITION we define an instance of the dial-a-ride problem with n servers as follows. For every element j ($j \in N$) we introduce a ride (O, a_j) with weight a_j. For $i = 1, \ldots, n$, let $N_i = \{j_1, \ldots, j_{k_i}\}$ be the index set of the rides served by server i, and let $b_i = \sum_{j \in N_i} a_j$ denote their total length. Note that $\sum_{i=1}^{n} b_i = nb$. When server i is never idle, its contribution to the weighted sum of completion times is equal to

$$(a_{j_1})^2 + a_{j_2}(2a_{j_1} + a_{j_2}) + \cdots + a_{j_{k_i}}(2a_{j_1} + 2a_{j_2} + \cdots + a_{j_{k_i}}) = b_i^2.$$

Hence, $\sum w_jC_j = \sum_{i=1}^{n} b_i^2 \geq nb^2$, with equality if and only if $b_1 = \cdots = b_n = b$, i.e., we have a yes-instance of 3-PARTITION. □

2.7 Summary and analysis

Below we list all of the easy and open problems. To keep the list manageable, we use the following shorthand: A/B denotes alternatives within a problem type, (A) is an optional feature, and $*$ is a wildcard for an arbitrary type of metric or for weights in the objective.

In this section we will examine the overall influence of each characteristic of a dial-a-ride problem on its complexity and point out some interesting open problems.

Precedence constraints, release times and deadlines. We have classified all problems with precedence constraints. Only the problems of type $1, cap1|S/T, (pmtn), (d_j), prec|*|C_{\max}$ are easy. The others are hard, including all problems with parallel servers or capacity not equal to 1 or latency objective.

All problems with both release times and deadlines are hard. Almost all problems with either release times or deadlines have been classified as well: only nine problems with deadlines and eight with release times are open.

Easy problems		**180 problems**
s=t rides		**10 problems**
$E1$	$1\|s=t\|tree\|C_{\max}$	1
$E2$	$1/P\|s=t\|line\|\sum *C_j$	4
$E3$	$1/P\|s=t,(d_j)\|line\|C_{\max}$	4
$E4$	$Q\|s=t\|line\|C_{\max}$	1
remaining – deadlines		**61 problems**
$E5$	$1,cap1\|S/T,(pmtn),d_j,(prec)\|*\|\ C_{\max}$	40
$E6$	$1,cap1\|S/T,(pmtn),d_j\|*\|\ \sum C_j$	20
$E7$	$1,cap\infty\|T,d_j\|line\|C_{\max}$	1
remaining – general rides		**4 problems**
$E8$	$1,cap1\|(pmtn)\|line\|C_{\max}$	2
$E9$	$1,cap1\|pmtn\|tree\|C_{\max}$	1
$E10$	$1,cap\infty\|\|line\|C_{\max}$	1
remaining – S/T rides		**105 problems**
$E11$	$1,cap1\|S,(pmtn),(prec)\|*\|C_{\max}$	20
$E12$	$1,cap1\|S/T,(pmtn)\|*\|\ \sum *C_j$	40
$E13$	$1\|S,(pmtn)\|line\|C_{\max}$	2
$E14$	$1\|S,pmtn\|tree\|C_{\max}$	1
$E15$	$P,cap1\|S/T,pmtn\|*\|\ \sum C_j$	10
$E16$	$P/Q/R,cap1\|S/T\|*\|\ \sum C_j$	30
$E17$	$1,cap\infty\|T\|line\|\sum *C_j$	2
Open problems		**77 problems**
s=t rides		**6 problems**
$O1$	$Q\|s=t\|line\|\sum *C_j$	2
$O2$	$Q\|s=t,d_j\|line\|C_{\max}$	1
$O3$	$1/P/Q\|s=t,r_j\|line\|\sum C_j$	3
remaining – release times and deadlines		**13 problems**
$O4$	$1,cap\infty\|(S/T),r_j\|line\|\sum C_j$	3
$O5$	$P/Q,cap\infty\|S,pmtn,r_j\|line\|\sum C_j$	2
$O6$	$P/Q,cap\infty\|T,pmtn,d_j\|line\|C_{\max}$	2
$O7$	$P/Q/R,cap1\|S/T,pmtn,d_j\|line\|C_{\max}$	6
remaining – general rides		**14 problems**
$O8$	$1,(cap1)\|pmtn\|line\|\sum *C_j$	4
$O9$	$1\|pmtn\|line\|C_{\max}$	1
$O10$	$1,cap\infty\|\|line\|\sum *C_j$	2
$O11$	$P/Q,cap\infty\|(pmtn)\|line\|C_{\max}$	4
$O12$	$P/Q/R,cap1\|pmtn\|line\|\sum C_j$	3
remaining – S/T rides		**44 problems**
$O13$	$1\|S/T,(pmtn)\|line\|\sum *C_j$	8
$O14$	$P/Q,cap\infty\|T,(pmtn)\|line\|\sum *C_j$	8
$O15$	$P/Q/R,(cap1/c)\|S,pmtn\|line\|C_{\max}$	8
$O16$	$Q/R,cap1\|S/T,pmtn\|*\|\ \sum C_j$	20

There are easy problems with deadlines. As in the case of precedence constraints these include a class of single-server problems ($E5$ and $E6$) but also one with parallel servers ($E3$). For the latency objective, no problem with release times is known to be easy. (For makespan, some are solvable by symmetry between deadlines and release times.)

The open problems with release times include some of the most challenging open questions in the area: $1|s = t, r_j|line|\sum C_j$ and $1, cap\infty|(S), r_j|line|\sum C_j$.

Metric spaces. It turns out that the distinction between *line* and more general spaces is the main breakpoint in the classification. Most of the open problems have the real line as a metric space, the exception being $O16$. Some problems are easy for all metric spaces. Almost all other problems are hard for the variants of metric spaces different from the line. The only exceptions, besides $O16$, are three problems solvable on trees: $E1$, $E9$ and $E14$.

Sources and destinations. Not surprisingly, the traveling salesman-like problems with rides with coinciding source and destination behave differently than problems with more general rides. Most of them are also hard, only ten are easy and six open. Perhaps the most interesting result in the entire classification is the hardness of $1|s = t|tree|\sum C_j$ [49], which has been a long-standing open problem.

Most of the other easy or open problems have rides with the same source or the same destination. Only four problems with general rides are known to be easy, fifteen of them are open.

Preemptions. As in machine scheduling, allowing preemption makes a problem harder to classify. Among the open problems there are eleven where preemption is irrelevant (with rides of type $s = t$ or a server of type $1, cap\infty$), ten pairs in which both the preemptive and the non-preemptive version are open, and 46 problems where only the preemptive version is open.

The makespan problems on trees $E9$ and $E14$ are, surprisingly, the only problems where the preemptive version is easy and the non-preemptive is hard.

For all open preemptive problems with release times or deadlines or general rides, the non-preemptive version is hard. The remaining problems, $O15$ and $O16$, are interesting in relation to their counterparts from machine scheduling.

The easiest problem of type $O15$, $P, cap1|S, pmtn|line|C_{\max}$, is easy if all the rides are on the same side of the origin, as preemptive scheduling results then apply. With rides on both sides, however, it may be impossible to balance the work among the servers. The non-preemptive version is hard.

The problems $O16$ correspond to scheduling problems that are easy on uniform machines [18] and hard on unrelated ones [48]. The former result is not helpful for the dial-a-ride problem. Note that this is the only instance where a non-preemptive problem is easy and the preemptive version may be hard.

Number, type and capacity of servers. Not surprisingly, parallel servers and

capacities other than 1 tend to make the problems hard. Most of the easy problems involve unit capacity. At present only two problems with general capacity and four with infinite capacity are easy, and they all have a single server.

For parallel servers, there is never a difference between Q and R in our current classification, and there is never a difference between common capacity c and general capacity (which could only matter for P or Q).

The open problems $O1$ and $O2$ exhibit a striking difference between P and Q. For identical servers the problems are trivial: it is optimal to use two servers, one for either direction. For a single server the problems are solved by dynamic programming. For uniform servers one can create complex instances where, for example, any number of fastest servers turns any given number of times.

Another problem with a different classification for P and Q is $O16$ discussed above. For identical servers it is easy, since preemptions do not help. We still have not found a problem, however, where the Q variant is provably hard and the P variant is easy.

Problem $O11$ without preemption is easy for a constant number of servers, as each server has to try only polynomially many reasonable tours.

Objective. As in machine scheduling, makespan and latency have little in common. There are 22 open problems with the makespan objective.

For the latency objective, it is interesting to examine the influence of weights. There are several classes of problems where adding weights to the objective causes hardness: $E6$, $E15$ and $E16$. There are twelve pairs of open problems where both the weighted and unweighted version have not been classified, and 31 problems where the unweighted version is open and the weighted one is hard; these involve a reduction from PARTITION and exponential weights. Weighted problems seem to be easier to classify: we have no case where the unweighted version is easy and the weighted one is open.

2.8 Postlude

Application of DARCLASS to the class $\mathcal{S}$ of 7,930 problem types with the current knowledge as input tells us that 180 problems are easy, 7,673 are hard, and 77 remain open. There are 26 maximal easy problems and 68 minimal hard ones.

We also used DARCLASS to determine two subclasses of the classes of maximal easy and minimal hard problems. An easy problem Π is called *borderline easy* if every Π' for which $\Pi \to \Pi'$ is hard. A hard problem Π is called *borderline hard* if every Π' for which $\Pi' \to \Pi$ is easy. These problems determine part of the exact borderline between the easy and the hard problems. If there are no more open problems left, all maximal easy problems are borderline easy, and all minimal hard problems are borderline hard. At present, 19 problems are borderline easy and 49 are borderline hard. Since the status of these problems is no longer subject to change, their number is in some sense a measure of the progress we have made in closing the gap.

The partial ordering on $\mathcal{S}$, with the exclusion of the augmentation implied by Lemma 2.10, may be helpful in the design and analysis of polynomial-time approximation algorithms for dial-a-ride problems. If $\Pi \rightarrow \Pi'$, then a lower bound on the approximability of Π implies the same lower bound for Π', and a performance bound for Π' implies the same bound for Π.

3

Introduction to online optimization

3.1 The online model

In practice, the assumption that we have complete information about a problem instance is often unnatural. Information can be uncertain or even lacking completely, simply because data becomes available over time. Think for example of a pizza delivery service where at the beginning of the evening almost nothing is known about the addresses where pizzas are to be delivered during the night. When information is lacking, algorithms that are known to output the optimal solution for the offline problem are of little value. The strategy of just waiting until all information about the instance is available with certainty, and then solving it as well as possible, does not lead to satisfactory solutions. Instead, such problems require an algorithm that starts outputting intermediate or partial solutions while information is still incomplete. Optimization problems in which the input arrives in an online manner are called *online optimization problems*.

The input of an online problem is modeled as a *request sequence* $\sigma = \sigma_1, \sigma_2, \ldots$ that is revealed step by step. An *online algorithm* is required to produce an output online, based on partial information only. Since we assume no knowledge at all about the future, it is even unknown to the online algorithm when the input ends. This implies that *every* partial output of an online algorithm could be the final output.

There are several ways in which the request sequence can be revealed to the algorithm. The way this is done depends on the problem under study. For the problems studied in this thesis we use two different models.

The "real-time" model [13]
In this model requests are arriving over time, in the sense that every request σ_j has a release time $r_j \geq 0$ at which all information about the request becomes available. Before this release time the existence of the request is not known to an online algorithm. An online algorithm must therefore determine its behavior at any time t based only on the information about requests with release time smaller than or equal to t. The algorithm is free to determine the order in which the requests are served. It can even choose to remain idle for a while to gather information about possible future requests, at a cost depending on the time elapsing while being idle. Decisions made by an online algorithm are irrevocable in the sense that once an action is being executed or has already been executed it cannot be undone. Note that this model only makes sense for problems where time plays a role in the objective function. For a lot of vehicle routing problems this is the case. Throughout Chapters 4 to 7 we will use the real-time model.

The "making decisions one-by-one" model [13]
In this model time does not play an essential role as in the previous paradigm. Requests are ordered in some list and are presented one-by-one according to this list. Decisions about the order in which the request will be served have to be taken immediately and irrevocably without seeing the next request in the list. After the order in which all requests will be served is determined in an online way, the requests are executed and the cost of the online solution is determined. We will use this model in Chapter 8 where we study the online bin coloring problem.

In the following chapters, we study online vehicle routing problems. Part of the input of a vehicle routing problem is the space in which the servers move. In the following chapters, we only consider problems on metric spaces, i.e. distances are symmetric and satisfy the triangle inequality. Sometimes we refer to the metric space as a line, a graph, or as $\mathbb{R}^d$. In the case of a (half) line and $\mathbb{R}^d$, the space X consists of all elements of $\mathbb{R}_0^+$, $\mathbb{R}$, or $\mathbb{R}^d$, and as metric we use the Euclidean metric. In the case of a graph, X contains all vertices of the graph, and for two vertices $x, y \in X$, $d(x, y)$ is the length of the shortest path in the graph from x to y. Moreover, we assume that for each pair of vertices x, y, there exists a subset of X isomorphic to a line segment of length $d(x, y)$. This additional technical assumption allows servers to change their mind and turn around at any point between two vertices of a graph. However, we do restrict sources and destinations of rides to be located in the vertices. We denote by $\mathcal{M}$ the set of all metric spaces that satisfy the description above.

3.2 Competitive analysis

In this section we discuss a method to measure the quality of online algorithms. In the mid-eighties Sleator and Tarjan introduced the idea of comparing an online solution to the optimal offline solution, thereby formulating the notion of *competitive analysis* [50]. Some online problems like job scheduling, bin packing, and vehicle

routing have a natural interpretation in an offline setting as well. For other online problems the offline variant might be unnatural. Nevertheless, for any online problem we can assume the existence of an offline algorithm that is clairvoyant and obtains an optimal offline solution for each instance [7]. We will refer to such an algorithm as an *optimization algorithm*.

Definition 3.1. [**c-competitiveness for deterministic algorithms**]
A deterministic online algorithm A *is called c-competitive if for all input sequences* σ*,*

$$\mathrm{A}(\sigma) \leq c \cdot \mathrm{OPT}(\sigma),$$

where $\mathrm{A}(\sigma)$ *is the cost incurred by algorithm* A *on input sequence* σ *and* $\mathrm{OPT}(\sigma)$ *is the cost of an optimization algorithm on instance* σ*.*

Sometimes one allows for an additive constant in the righthand side of the inequality to make up for initial differences in the state of the online and the offline algorithm. Throughout this thesis the additive constant will not be used.

The *competitive ratio* of a deterministic online algorithm A is the smallest ρ for which A is ρ-competitive.

Since an online algorithm is judged on its performance on the worst possible instance, we can look at competitive analysis as a game between an online player and an evil *adversary*. The online player chooses a strategy (i.e. an online algorithm), and depending on this strategy the adversary chooses an input sequence such that the online player performs relatively as badly as possible.

So far we assumed that the online player chooses a strategy deterministically. Instead, an online player can base its strategy on some random process. For example, he can choose his strategy randomly out of a set of deterministic strategies. This is called a *mixed strategy*. For our purposes we can assume that mixed strategies are equivalent to other possible non-deterministic strategies; for details, see [7]. Hence we choose to define a randomized algorithm as follows:

Definition 3.2. [**Randomized algorithm**]
A randomized algorithm RA *is defined as a probability distribution over a set of deterministic online algorithms. The cost* $\mathrm{RA}(\sigma)$ *of* RA *on input sequence* σ*, is a random variable.*

In order to define the competitive ratio of a randomized algorithm we have to be precise about the adversary model. In the literature three different kinds of adversaries are used [7]. The *oblivious adversary* has to choose an input sequence in advance, based only on the description of the online algorithm. It cannot adapt the input sequence based on the behaviour of the online algorithm. The *adaptive offline adversary* can build the input sequence online and can base future requests on the actions of the online algorithm on previous requests. The cost of the adaptive offline adversary is the optimal offline cost on the complete sequence that can be determined only in hindsight. The *adaptive online adversary* can build the input sequence online like the adaptive offline adversary, but it must also generate its own

solution online. Clearly, adaptive adversaries are more powerful than the oblivious adversary. In this thesis we will only use the oblivious adversary model when dealing with randomized algorithms.

Definition 3.3. [**c-competitiveness for randomized algorithms**]
A randomized online algorithm RA *distributed over a set* $\{A_x\}$ *of deterministic online algorithms is called c-competitive against an oblivious adversary, if for all input sequences* σ*,*

$$\mathbb{E}_x\left[A_x(\sigma)\right] \leq c \cdot \mathrm{OPT}(\sigma).$$

The *competitive ratio* of a randomized online algorithm RA is the smallest ρ for which RA is ρ-competitive.

For offline problems the running time of an algorithm is usually a crucial item (see Section 1.2). In online optimization, however, running times of algorithms are usually neglected. Typically an online strategy is allowed to compute the optimal solution at any time t over the request sequence released up to time t and decide upon an action depending on this optimal solution, even if this requires solving an NP-hard problem. The only restriction that we impose on the online algorithm concerns the amount of information available.

Since in competitive analysis online algorithms are compared to an optimization algorithm, we can compare the quality of different online algorithms by comparing their competitive ratios. The algorithm with the smallest competitive ratio can be seen as the best online algorithm in terms of worst-case performance.

For an extensive introduction to online optimization we refer to [7] and [13].

3.3 Deriving lower bounds on the competitive ratio

Competitive analysis is a way to express the cost of having incomplete information. Often we can prove a lower bound, say $\bar{c}$, on the competitive ratio of any online algorithm for a problem, stating that no online algorithm can *guarantee* a solution of less than $\bar{c}$ times the optimal solution, where the optimal solution is the solution that we could have found if we would have had complete information. Usually this is done by constructing an input sequence on which all possible algorithms perform badly. For deterministic algorithms this is usually relatively easy. Examples can be found in the following chapters.

For randomized algorithms it is usually difficult to give a lower bound on the expected cost of an arbitrary randomized algorithm on a specific instance. However, the use of *Yao's Principle* [52] can be helpful. Examples of the application of Yao's principle can be found in [7]. As the authors of [7] point out, straightforward application is often not possible. For our purposes, however, we can use a special form of the principle, as stated below [30].

Theorem 3.4. **[Yao's Principle]** *Let* $\{\mathrm{ALG}_y : y \in \mathcal{Y}\}$ *denote the set of deterministic online algorithms for an online minimization problem. Let the set of possible request sequences for this online problem be indexed by the set* $\mathcal{X}$*, that is,*

the set of inputs is represented by $\{ \sigma_x : x \in \mathcal{X} \}$. *If* $\bar{X}$ *is a probability distribution over input sequences* $\{ \sigma_x : x \in \mathcal{X} \}$ *and* $\bar{c} \geq 1$ *is a real number such that*

$$\inf_{y \in \mathcal{Y}} \frac{\mathbb{E}_{\bar{X}} [\mathrm{ALG}_y(\sigma_x)]}{\mathbb{E}_{\bar{X}} [\mathrm{OPT}(\sigma_x)]} \geq \bar{c}, \tag{3.1}$$

then $\bar{c}$ *is a lower bound on the competitive ratio of any randomized algorithm against an oblivious adversary, provided that the left hand side is bounded.*

3.4 Drawbacks of competitive analysis

Competitive analysis has some drawbacks. First of all, the competitive ratio is a worst-case ratio. Like all forms of worst-case analysis, competitive analysis can therefore be viewed as being too pessimistic. A second, and more important argument against competitive analysis is that an algorithm that has to perform well on all sequences is tailored not to fail on the worst-case input sequence, which often leads to a somewhat paranoid behaviour. As a consequence, the performance on most sequences is worse than it could be. A third argument against competitive analysis is that algorithms are not restricted in terms of computation time. An algorithm that performs well with respect to competitive analysis can therefore be very impractical (not to say useless) in practice.

There have been several attempts to overcome these drawbacks. To address the first problem, the pessimistic character of competitive analysis, one could restrict the power of the adversary and compare the performance of online algorithms to the performance of the restricted adversary. The ways to do this should depend on the specific online problem. Examples of this approach can be found in Chapters 4 and 7, where we look at specific online dial-a-ride problems and restrict the adversary in his freedom to move. We show that this restriction helps in getting a more realistic performance measure. Another example can be found in [29], where the paging problem is studied and the adversary is restricted in the sense that he only has a restricted look ahead.

Another idea would be to allow the online algorithm to use more resources than the adversary. This is called *resource augmentation* and examples can be found in [46] and [44]. Resource augmentation is not a new concept: already in 1966, Graham applied resource augmentation for machine scheduling problems [19]. Notice that both in resource augmentation models and in fair adversary models there is no restriction on the input sequence.

Restricting the input sequence is another way to address the drawbacks mentioned above. If we allow the adversary to choose the input sequence only from a set of "reasonable" instances we can rule out extremes. We can even force the adversary to draw specific input parameters randomly according to a probability distribution that is known to the algorithm beforehand. This is done in stochastic optimization (see e.g. [24]). This approach has the disadvantage that in practice the input distribution is not known and the error in guessing the input distribution can be too large to obtain valuable results.

Recently Koutsoupias and Papadimitriou [29] formulated two online paradigms

that capture the above ideas and generalize standard competitive analysis. The first is *comparative analysis*, where two classes of algorithms $\mathcal{A}$ and $\mathcal{B}$ are defined from which the online player and the adversarial player must choose one, respectively. The comparative ratio $\rho(\mathcal{A}, \mathcal{B})$ is defined as

$$\rho(\mathcal{A}, \mathcal{B}) = \max_{\mathrm{B} \in \mathcal{B}} \min_{\mathrm{A} \in \mathcal{A}} \max_{\sigma} \frac{\mathrm{A}(\sigma)}{\mathrm{B}(\sigma)},$$

if it exists. Notice that when $\mathcal{A}$ is the set of all online algorithms and $\mathcal{B}$ is the set of all offline algorithms, this is just the competitive ratio. The fair adversary model described before also fits in this model. When we study the online traveling salesman problem in Chapter 4, we introduce a class of online algorithms that we call *zealous*. This class is a restriction on the set of online algorithms. It fits in the comparative analysis model where the set of adversarial strategies is just the complete set of offline algorithms.

A way to overcome the third objection against competitive analysis, i.e. the unrestricted running time of an algorithm, is to restrict the class of online algorithms to those having polynomial running time. This approach also fits in the comparative analysis framework.

The second model presented in [29] is the *diffuse adversary* model, where the adversary chooses a probability distribution from a set of probability distributions known to the online algorithm, according to which he draws the input. Here the competitive ratio measures the expected online cost under the input distribution, compared to the optimal offline cost, where the adversary chooses the probability distribution such that the ratio is as large as possible.

Which model to use when dealing with uncertainty in the input data depends on the problem under study and is, to a certain extent, a matter of taste. All the approaches described above that go beyond standard competitive analysis have shown to be useful for specific problems. However, the standard performance measure used in the literature is still the competitive ratio.

4

The online traveling salesman problem

The content of this chapter is joint work with M. Blom, S.O. Krumke, and L. Stougie, and has appeared in [5].

4.1 Introduction

The traveling salesman problem is a well-studied problem in combinatorial optimization: Given a set of C of n cities, a distance $d(c_i, c_j) \in \mathbb{Z}^+$ for each pair of cities $c_i, c_j \in C$, and a positive integer B, does there exist a tour of C having length at most B? Here we consider the following online variant of the traveling salesman problem, called the OlTsp in the sequel, which was introduced in [4]. Cities (requests) arrive online over time while the salesman is traveling. The requests are to be handled by a salesman-server who starts and ends his work at a designated origin. The objective is to find a routing for the server that finishes as early as possible, thus minimizing the makespan.

More formally, an instance of the OlTsp consists of a metric space $M = (X, d) \in \mathcal{M}$ (see Section 3.1) with a distinguished origin $O \in X$ and a sequence $\sigma = \sigma_1, \ldots, \sigma_m$ of requests. A server is located at the origin O at time 0 and can move at most at unit speed. In this chapter we are mainly concerned with the special case that M is $\mathbb{R}_0^+$, the non-negative part of the real line. The origin O equals the point 0.

Each *request* is a pair $\sigma_i = (t_i, x_i)$, where $t_i \in \mathbb{R}_0^+$ is the time at which request σ_i is released, and $x_i \in X$ is the point in the metric space requested to be visited. We assume that the sequence $\sigma = \sigma_1, \ldots, \sigma_m$ of requests is given in order of non-decreasing release times. For a real number t we denote by $\sigma_{\leq t}$ the subsequence of

requests in σ released up to time t. Similarly, $\sigma_{<t}$ is the subsequence of σ consisting of those requests with release time strictly smaller than t.

It is assumed that the online algorithm has neither information about the time when the last request is released, nor about the total number of requests.

An online algorithm for the OlTsp must determine the behavior of the server at a certain moment t as a function of all the requests in $\sigma_{\leq t}$ and of the current time t. In contrast, the adversary has information at time 0 about all requests in the whole sequence σ. A feasible solution is a route for the server that serves all requested points, where each request is served not earlier than the time it is released, and which starts and ends in the origin O.

The objective in the OlTsp is to minimize the makespan or total completion time of the server, that is, the time when the server has served all requests and has returned to the origin.

Let $\mathrm{ALG}(\sigma)$ denote the completion time of the server moved by algorithm ALG on the sequence σ of requests. We use OPT to denote an optimal offline algorithm. We use competitive analysis to measure the quality of the algorithms that we propose.

In [4], the authors present a 2-competitive algorithm for the OlTsp that works in general metric spaces. The authors also show that, for general metric spaces, no deterministic algorithm can be c-competitive with $c < 2$. For the special case that the metric space is $\mathbb{R}$, the real line, their best algorithm is 7/4-competitive, whereas a lower bound on the competitive ratio of any algorithm of $(9 + \sqrt{17})/8 \approx 1.64$ is derived. Recently, Lipmann [36] designed an algorithm for the problem on the real line with competitive ratio that matches the just-mentioned lower bound.

We study the effect of restricting the class of algorithms allowed and restricting the power of the adversary in the competitive analysis. We introduce and analyze a new class of online algorithms that we call *zealous algorithms*. Roughly speaking, a zealous algorithm never sits idle while there is work to do. A similar concept was used for scheduling problems in [39]. A precise definition of zealousness is presented in Section 4.2, where we also show that, in general, zealous algorithms are strictly weaker than algorithms that allow waiting time. In particular, we prove that no zealous algorithm can achieve a competitive ratio lower than 7/4 for the OlTsp on the real line. The 7/4-competitive algorithm in [4] is in fact a zealous algorithm and therefore the best possible within this restricted class of algorithms.

We then concentrate on the special case of the OlTsp where the underlying metric space is $\mathbb{R}_0^+$, the non-negative part of the real line. In Section 4.3 we show that an extremely simple and natural zealous strategy is 3/2-competitive and that this result is the best possible for zealous and non-zealous deterministic algorithms on $\mathbb{R}_0^+$.

In Section 4.4, we deal with an objection frequently encountered against competitive analysis concerning the unrealistic power of the adversary against which performance is measured. Indeed, for the OlTsp on the real line the before-mentioned 7/4-competitive algorithm reaches its competitive ratio against an adversary that moves away from previously released requests without giving any information to the online algorithm. We introduce an adversary that is in a natural way restricted in the

context of the OLTSP studied here; we call it a *fair adversary*. It should be seen as a more reasonable adversary model. A fair adversary always keeps its server within the convex hull of the requests released so far. We show that this adversary model indeed allows for lower competitive ratios. For instance, the above-mentioned 3/2-competitive zealous strategy against the conventional adversary is 4/3-competitive against the fair adversary. This result is the best possible for zealous algorithms against a fair adversary.

Table 4.1: Overview of lower bounds (LB) and upper bounds (UB) for the competitive ratio of deterministic algorithms for OLTSP *on* $\mathbb{R}_0^+$.

	Zealous Algorithms	General Algorithms
General Adversary	LB = UB = 3/2	LB = UB = 3/2
Fair Adversary	LB = UB = 4/3	LB = UB = $(1+\sqrt{17})/4 \approx 1.28$

We also present a non-zealous algorithm with competitive ratio $(1+\sqrt{17})/4 \approx 1.28 < 4/3$ competing against the fair adversary. This result is, together with the before-mentioned algorithm in [36], the first result that shows that waiting is actually advantageous in the OLTSP. Such results are known already for online scheduling problems (see, e.g., [25], [9], [45]) and for an online dial-a-ride problem [3]. Our competitiveness result is complemented by a matching lower bound on the competitive ratio of algorithms against the fair adversary. Table 4.1 summarizes our results for the OLTSP on $\mathbb{R}_0^+$. Table 4.2 gives an overview of the corresponding results for the OLTSP on $\mathbb{R}$.

4.2 Zealous algorithms

We introduce a particular class of algorithms for the OLTSP, which we call *zealous algorithms*. Intuitively, the server of a zealous algorithm, a zealous server, should never sit and wait when it could serve unserved requests. A zealous server should also move towards work that has to be done directly without any detours. To translate

Table 4.2: Overview of lower bounds (LB) and upper bounds (UB) for the competitive ratio of deterministic algorithms for OLTSP *on* $\mathbb{R}$*, known from literature.*

	Zealous Algorithms	General Algorithms
General Adversary	LB = 7/4 (Lemma 4.3) UB = 7/4 † [4]	LB = $(9+\sqrt{17})/8 \approx 1.64$ [4] UB = $(9+\sqrt{17})/8 \approx 1.64$ [36]
Fair Adversary	LB = 8/5 (Theorem 4.10) UB = 7/4 ‡ [4]	LB = $(5+\sqrt{57})/8 \approx 1.57$ (Theorem 4.9) UB = $(9+\sqrt{17})/8 \approx 1.64$ ‡ [36]

† Although the work does not use the term "zealous" it can be seen that their 7/4-competitive algorithm is in fact a zealous one.

‡ The upper bounds stated here stem from the upper bounds against a general adversary; improved results that exploit the fairness of the adversary are yet unknown.

this intuition into a rigorous definition some care has to be taken.

Definition 4.1. [**Zealous Algorithm**]
An algorithm ALG *for the* OlTsp *is called* zealous*, if it satisfies the following conditions:*

1. *If there are still unserved requests, then the direction of the server operated by* ALG *changes only if a new request becomes known, or the server is either in the origin or at a request that has just been served.*

2. *At any time when there are unserved requests, the server operated by* ALG *either moves towards an unserved request or the origin at maximum (i.e. unit) speed. (The latter case is only allowed if the server operated by* ALG *is not yet in the origin.)*

We emphasize that a zealous algorithm is allowed to move its server towards an unserved request and change his direction towards another unserved request or to the origin at the moment a new request becomes known.

Theorem 4.2. Ausiello et al. [4]. *Let* ALG *be a deterministic online algorithm for the* OlTsp *on the real line* $\mathbb{R}$*. Then the competitive ratio of* ALG *is at least* $(9+\sqrt{17})/8 \approx 1.64$*.*

Lemma 4.3. *Let* ALG *be a zealous online algorithm for the* OlTsp *on the real line* $\mathbb{R}$*. Then the competitive ratio of* ALG *is at least* $7/4$*.*

Proof. Suppose that ALG is a zealous algorithm for OlTsp on the real line. Consider the following adversarial input sequence. At times 0 and 1/2, two requests $\sigma_1 = (0, 1/2)$ and $\sigma_2 = (1/2, 0)$ are released. There will be no further requests before time 1. Thus, by the zealousness of the algorithm the server will be at the origin at time 1.

At time 1, two new requests at points 1 and -1, respectively, are released. Since the algorithm is zealous, starting at time 1 it must move its server to one of these requests at maximum, i.e., unit, speed. Without loss of generality assume that this is the request at 1. ALG's server will reach this point at time 2. Starting at time 2, ALG will have to move its server either directly towards the unserved request at -1 or towards the origin, which essentially gives the same movement and implies that the server is at the origin at time 3. At that time, the adversary issues another request at 1. Thus, ALG's server will still need at least 4 time units to serve -1 and 1 and return to the origin. Therefore, it will not be able to complete its work before time 7.

The offline adversary handles the sequence by first serving the request at -1, then the two requests at 1, and finally returns to the origin at time 4, yielding the desired result. □

This lower bound shows that the 7/4-competitive algorithm presented in [4], which is in fact a zealous algorithm, is the best possible within the class of zealous algorithms for the OLTSP on the real line.

4.3 The OLTSP on the non-negative part of the real line

We first consider the OLTSP on $\mathbb{R}_0^+$ when the offline adversary is the conventional (omnipotent) opponent.

Theorem 4.4. *Let* ALG *be any deterministic algorithm for the* OLTSP *on* $\mathbb{R}_0^+$. *Then the competitive ratio of* ALG *is at least* 3/2.

PROOF. At time 0 the request $\sigma_1 = (0, 1)$ is released. Let $T \geq 2$ be the time that the server operated by ALG has served the request σ_1 and returned to the origin O. If $T \geq 3$, then no further request is released and ALG is no better than 3/2-competitive since $\mathrm{OPT}(\sigma_1) = 2$. Thus, assume that $T < 3$.

In this case the adversary releases a new request $\sigma_2 = (T, T)$. Clearly, $\mathrm{OPT}(\sigma_1, \sigma_2) = 2T$. On the other hand $\mathrm{ALG}(\sigma_1, \sigma_2) \geq 3T$, yielding a competitive ratio of 3/2. □

The following simple strategy achieves a competitive ratio that matches this lower bound, as we will show below:

> **Algorithm MRIN** ("Move-Right-If-Necessary")
>
> If a new request is released and the request is to the right of the current position of the server operated by MRIN, then the MRIN-server starts to move right at full speed. The server continues to move right as long as there are unserved requests to the right of the server. If there are no more unserved requests to the right, then the server moves toward the origin O at full speed.

It is easy to verify that Algorithm MRIN is in fact a zealous algorithm. The following theorem shows that the strategy has a best possible competitive ratio for the OLTSP on $\mathbb{R}_0^+$.

Theorem 4.5. *Algorithm* MRIN *is a zealous* 3/2*-competitive algorithm for the* OLTSP *on the non-negative part* $\mathbb{R}_0^+$ *of the real line.*

PROOF. We show the theorem by induction on the number of requests in the sequence σ. It clearly holds if σ contains at most one request. The induction hypothesis states that the claim of the theorem holds for any sequence of $m - 1$ requests.

Suppose that request $\sigma_m = (t, x)$ is the last request of $\sigma = \sigma_1, \ldots, \sigma_{m-1}, \sigma_m$. If $t = 0$, then MRIN is obviously 3/2-competitive, so we will assume that $t > 0$.

Let f be the position of the request unserved by the MRIN-server at time t (excluding σ_m), which is furthest away from the origin. If all requests in $\sigma_1, \ldots, \sigma_{m-1}$ have already been served by MRIN at time t then we set $f = 0$.

In case $x \leq f$, MRIN's cost for serving σ is equal to the cost for serving the sequence consisting of the first $m-1$ requests of σ. Since new requests can never decrease the optimal offline cost, the induction hypothesis implies the theorem.

Now assume that $f < x$. Thus, at time t the request in x is the furthest unserved request. If the position of MRIN at time t is to the right of x, then its cost does not increase by the release of σ_m. The claim then follows from the induction hypothesis as above. On the other hand, if at time t MRIN is to the left of x, then MRIN will complete its work no later than time $t + 2x$. The optimal offline cost $\mathrm{OPT}(\sigma)$ is bounded from below by $\max\{t+x, 2x\}$. Therefore,

$$\frac{\mathrm{MRIN}(\sigma)}{\mathrm{OPT}(\sigma)} \leq \frac{t+x}{\mathrm{OPT}(\sigma)} + \frac{x}{\mathrm{OPT}(\sigma)} \leq \frac{t+x}{t+x} + \frac{x}{2x} = \frac{3}{2}.$$

□

The result established above can be used to obtain competitiveness results for the generalization of the OlTsp on the real line when there are $k \geq 2$ servers, and the goal is to minimize the time when the last of its servers returns to the origin O after all requests have been served.

Lemma 4.6. *There is an optimal offline strategy for the* OlTsp *on the real line with $k \geq 2$ servers such that no server ever crosses the origin.*

Proof. Let σ be any request sequence and let (t_0^-, x_0^-) and (t_0^+, x_0^+) respectively be the leftmost and rightmost request in σ. For any time t we denote by $\sigma_{>t}$ the subsequence of requests released strictly after time t.

Now consider the following offline strategy for routing the servers. Only two of the servers are used. At time 0 the first server moves to the right until point x_0^+, and the second moves to the left until it reaches x_0^-. We now describe the strategy for the server at x_0^+ for moving left. The situation for the other server is symmetric.

The server waits at x_0^+ until time $T := \max\{x_0^+, t_0^+\}$. Then it moves left until it reaches the position of x_T^+, the rightmost request in the subsequence $\sigma_{>T}$, with release time t_T^+. It waits there until time $S := \max\{T + x_0^+ - x_T^+, t_T^+\}$. The time parameter T is updated to $T := S$ and the left movement is continued as above. If $\sigma_{>T}$ becomes empty the server moves back to the origin.

It is easy to see that the parameter T maintained by the right server always has the property that $T + x_T^+$ is a lower bound for the optimum offline completion time. Thus, the strategy described above yields in fact an optimal offline solution. □

Corollary 4.7. *There is a 3/2-competitive algorithm for the* OlTsp *with $k \geq 2$ servers on the real line.*

PROOF. The online algorithm uses two servers. One server handles all requests on $\mathbb{R}_0^+$, and the other one serves the requests on $\mathbb{R}_0^-$. Each server uses the MRIN-strategy. It follows from Theorem 4.5 and Lemma 4.6 that this strategy is 3/2-competitive. □

4.4 Fair adversaries

The adversaries used in the bounds of the previous section are abusing their power in the sense that they can move to points where they know a request will pop up without revealing the request to the online server before reaching the point. As an alternative we propose the following more reasonable adversary that we called *fair adversary*. We show that one can obtain better competitive ratios for the OLTSP on $\mathbb{R}_0^+$ with this model. We will also see that, with this adversary model, there exists a distinction in competitiveness between zealous and non-zealous algorithms.

Recall that $\sigma_{<t}$ is the subsequence of σ consisting of those requests with release time strictly smaller than t.

Definition 4.8. [**Fair adversary**]
An offline adversary for the OLTSP *in the Euclidean space* $(\mathbb{R}^n, \|.\|)$ *is* fair*, if at any moment* t*, the position of the server operated by the adversary is within the convex hull of the origin* O *and the requested points from* $\sigma_{<t}$*.*

In the special case of $\mathbb{R}_0^+$, a fair adversary must always keep its server in the interval $[0, F]$, where F is the position of the request with the largest distance to the origin O among all requests released so far.

The following lower bound result shows that the OLTSP on the real line against fair adversaries is still a non-trivial problem.

Theorem 4.9. *Let* ALG *be any deterministic algorithm for the* OLTSP *on* $\mathbb{R}$*. Then the competitive ratio of* ALG *against a fair adversary is at least* $(5+\sqrt{57})/8 \approx 1.57$*.*

PROOF. Suppose there exists an online algorithm ALG that is ρ-competitive. The adversarial sequence starts with two requests at time 0, $\sigma_1 = (0, 1)$ and $\sigma_2 = (0, -1)$. Without loss of generality, we assume that the first request served by ALG is σ_1. At time 2 the online server cannot have served both requests. We distinguish two main cases divided into subcases.

Case 1: None of the requests has been served at time 2.
If at time 3 request σ_1 is still unserved, let t' be the first time the server crosses the origin after serving the request. Clearly, $t' \geq 4$. At time t' the online server still has to visit the request in -1.

If $t' > 4\rho - 2$, the server cannot be ρ-competitive because the fair adversary can serve the sequence and be back in the origin at time 4.

Thus, suppose that $4 \leq t' \leq 4\rho - 2$. At time t' a new request $\sigma_3 = (t', 1)$ is released. The online server cannot finish the complete sequence before $t'+4$, whereas the adversary needs time $t'+1$. Therefore, $\rho \geq (t'+4)/(t'+1)$. For $4 \leq t' \leq 4\rho-2$, $(t'+4)/(t'+1)$ is decreasing in t'. Thus

$$\rho \geq \frac{(4\rho - 2) + 4}{(4\rho - 2) + 1} = \frac{4\rho + 2}{4\rho - 1},$$

implying $\rho \geq (5 + \sqrt{57})/8$.

If at time 3 the request σ_1 has already been served, the online server cannot be to the left of the origin at time 3, given the fact that at time 2 no request had been served. The adversary now gives a new request $\sigma_3 = (3, 1)$. There are two possibilities: either σ_2, the request in -1, is served before σ_3, or the other way round.

If the server decides to serve σ_2 before σ_3, then it cannot complete before time 7. Since the adversary completes the sequence in time 4, the competitive ratio is at least $7/4$.

If the online server serves σ_3 first, then again, let t' be the time that the server crosses the origin after serving σ_3. As before, we must have $4 \leq t' \leq 4\rho - 2$. At time t' the fourth request $\sigma_4 = (t', 1)$ is released. The same arguments as above apply to show that the algorithm is at least $(5 + \sqrt{57})/8$-competitive.

Case 2: σ_1 has been served at time 2 by the online server.
At time 2 the third request $\sigma_3 = (2, 1)$ is released. In fact, we are now back in the situation in which at time 2 none of the two requests are served. In case the movements of the online server are such that no further request is released by the adversary, the latter will complete at time 4. In the other cases the last released requests are released after time 4 and the adversary can still reach them in time. □

Recently Lipmann [35] designed an algorithm that has competitive ratio $(5 + \sqrt{57})/8$ against a fair adversary. For comparison, the best possible algorithm for the OLTSP in $\mathbb{R}$ against an adversary that is not restricted to be fair is $(9 + \sqrt{17})/8$-competitive [36]. On $\mathbb{R}_0^+$, the picture is also complete (see Theorems 4.12 and 4.14 for zealous algorithms and Theorems 4.11 and 4.15 for non-zealous algorithms below). Before we study the competitive ratio of algorithms on $\mathbb{R}_0^+$ against fair adversaries, we will establish a lower bound for zealous algorithms in $\mathbb{R}$.

Theorem 4.10. *Let* ALG *be any deterministic zealous algorithm for the* OLTSP *on* $\mathbb{R}$. *Then the competitive ratio of* ALG *against a fair adversary is at least* $8/5$.

PROOF. At time 0 the adversary releases two requests, $\sigma_1 = (0, -1)$ and $\sigma_2 = (0, 1)$. There will be no further requests before time 2. By the zealousness of the algorithm, starting at time 0 ALG will move at full speed to either -1 or 1, and then again

at full speed towards the other request. We assume that ALG first serves σ_1; the other case is symmetric.

As noted above, ALG's server will be at the origin at time 2 (after having served σ_1). At time 2 the adversary releases two new requests $\sigma_3 = (2, -1)$ and $\sigma_4 = (2, 1)$. ALG is now in a similar situation as at time 0: there are unserved requests both at -1 and at 1. We assume that ALG first serves the request at 1 (again, the other case is symmetric to this one).

At time 4 the server moved by ALG is at 0. At this time the adversary releases the last request $\sigma_5 = (4, 1)$. Hence, since ALG still has to serve requests both at -1 and 1, it follows that ALG cannot complete before time 8.

The fair adversary can serve the sequence as follows: At time 0 it moves to -1 serving request σ_1. Then, it waits at -1 until time 2, when σ_3 and σ_4 are released. From this point on, the fair adversary moves at full speed to 1, which is reached at time 4, when σ_5 is released, and then back to the origin. This gives a completion time of 5 for the fair adversary. □

Theorem 4.11. *Let* ALG *be any deterministic algorithm for the* OLTSP *on* $\mathbb{R}_0^+$. *Then the competitive ratio of* ALG *against a fair adversary is at least* $(1+\sqrt{17})/4 \approx 1.28$.

PROOF. Suppose that ALG is ρ-competitive for some $\rho \geq 1$. At time 0 the adversary releases request $\sigma_1 = (0, 1)$. Let T denote the time that the server operated by ALG has served this request and is back at the origin. For ALG to be ρ-competitive, we must have that $T \leq \rho \cdot \mathrm{OPT}(\sigma_1) = 2\rho$. At time T the adversary releases a second request $\sigma_2 = (T, 1)$. The completion time of ALG becomes then at least $T+2$.

On the other hand, starting at time 0 the fair adversary moves its server to 1, lets it wait there until time T, and then goes back to the origin O, yielding a completion time of $T + 1$. Therefore,

$$\frac{\mathrm{ALG}(\sigma)}{\mathrm{OPT}(\sigma)} \geq \frac{T+2}{T+1} \geq \frac{2\rho+2}{2\rho+1} = 1 + \frac{1}{2\rho+1},$$

given the fact that $T \leq 2\rho$. Since by assumption ALG is ρ-competitive, we have that $1 + 1/(2\rho + 1) \leq \rho$, implying that $\rho \geq (1 + \sqrt{17})/4$. □

For zealous algorithms we can show a higher lower bound against a fair adversary.

Theorem 4.12. *Let* ALG *be any deterministic zealous algorithm for the* OLTSP *on* $\mathbb{R}_0^+$. *Then the competitive ratio of* ALG *against a fair adversary is at least* $4/3$.

PROOF. Consider the adversarial sequence consisting of the three requests $\sigma_1 = (0, 1)$, $\sigma_2 = (1, 0)$, and $\sigma_3 = (2, 1)$.

By its zealousness, the online algorithm will start to travel to 1 at time 0, back to 0 at time 1, arriving there at time 2. Then its server has to visit 1 again, so that he will finish no earlier than time 4. Obviously, the optimal fair offline solution is to leave 1 not before time 2, and finishing at time 3. □

Lemma 4.13. *The optimal fair offline strategy for the* OLTSP *on* $\mathbb{R}_0^+$ *is as follows: If there is an unserved released request to the right of the server, move to the right. Move to the left only if there is no (unreleased) request in the remaining request sequence to the right of the server. Otherwise remain at the current position.*

PROOF. Trivial. □

We now show that the algorithm MRIN presented before has a better competitive ratio against the fair adversary than the ratio of 3/2 against a conventional adversary. In fact we show the ratio matches the lower bound for zealous algorithms proved in Theorem 4.12.

Theorem 4.14. *Algorithm* MRIN *is a* 4/3*-competitive algorithm for the* OLTSP *on* $\mathbb{R}_0^+$ *against a fair adversary.*

PROOF. Again we use induction on the number of requests in the sequence σ to establish the claim of the theorem. The claim clearly holds if σ contains at most one request. The induction hypothesis states that the claim of the theorem holds for any sequence of $m-1$ requests.

Let $\sigma = \sigma_1, \ldots, \sigma_m$ be any sequence of requests. We consider the time $t = t_m$ when the last set of requests $\sigma_{=t}$ is released. If $t = 0$, then the claim obviously holds, so we will assume for the remainder of the proof that $t > 0$. Let $\sigma_m = (t, x)$ be that request of $\sigma_{=t}$ that is furthest away from the origin.

In the sequel we denote by $s(t)$ and $s^*(t)$ the positions of the MRIN-server and the fair adversary server at time t, respectively.

Let $\sigma_f = (t_f, f)$ be the furthest unserved request by MRIN of the subsequence $\sigma_{<t}$ at time t, that is, the unserved request from $\sigma_{<t}$ most remote from the origin O. Finally, let $\sigma_F = (t_F, F)$ be the furthest request in $\sigma_{<t}$. Notice that by definition $f \leq F$.

We now distinguish cases depending on the position x of the request σ_m relative to f and F.

Case 1: $x \leq f$ (see Figure 4.1)
Since the MRIN-server still has to travel to f, all the requests in $\sigma_{=t}$ will be served on the way back to the origin and the total completion time of the MRIN-server will not increase by releasing the requests $\sigma_{=t}$. Since new requests can never decrease the optimal offline solution value, the claim follows from the induction hypothesis.

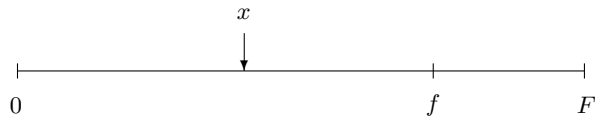

Figure 4.1: Case 1 of the Proofs of Theorem 4.14 and Theorem 4.15

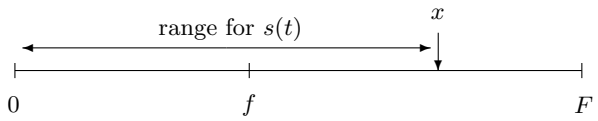

Figure 4.2: Case 2 of the Proof of Theorems 4.14 and 4.15

Case 2: $f \leq x < F$
If $s(t) \geq x$, again MRIN's completion time does not increase compared to the situation before the requests in $\sigma_{=t}$ were released, so we can assume that $s(t) \leq x$ (see Figure 4.2). The MRIN-server will now travel to point x, which needs time $d(s(t), x)$, and then return to the origin. Thus, $\mathrm{MRIN}(\sigma) = t + d(s(t), x) + x$. On the other hand $\mathrm{OPT}(\sigma) \geq t + x$. It follows that

$$\frac{\mathrm{MRIN}(\sigma)}{\mathrm{OPT}(\sigma)} \leq 1 + \frac{d(s(t), x)}{\mathrm{OPT}(\sigma)}. \tag{4.1}$$

We now show that $\mathrm{OPT}(\sigma)$ is at least 3 times $d(s(t), x)$; this will establish the claimed ratio of 4/3. Notice that $f < F$ (since $f \leq x < F$) and the fact that f is the furthest unserved request at time t implies that the MRIN-server must have already visited F at time t, otherwise the furthest unserved request would be at F and not at $f < F$. Therefore, $t \geq F + d(F, s(t))$, and

$$\mathrm{OPT}(\sigma) \geq t + x \geq F + d(F, s(t)) + x. \tag{4.2}$$

Clearly, each of the terms on the right-hand side of (4.2) is at least $d(s(t), x)$.

Case 3: $f \leq F \leq x$ (see Figure 4.3)
First remember that the MRIN-server always moves to the right if there are yet unserved requests to the right of his present position available.

Since the last request (t, x) is at least as far away from the origin as F, the optimal offline server as described in Lemma 4.13 will only move left after it has served the furthest request in σ, in this case at x.

This implies that at any time in the interval $[0, t]$ the fair adversary's server is to the right of the MRIN-server (or at the same position).

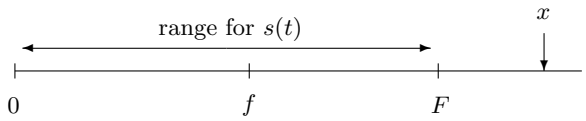

Figure 4.3: Case 3 of the Proof of Theorem 4.14 and Theorem 4.15.

Because the offline server does not return to the origin as long as there will be new requests released to the right of its current position, the distance between the MRIN-server and the offline server increases only if the offline server is waiting at some point. Let $W^*(t)$ be the total waiting time of the offline server at the moment t when the last request x is released. Then we know that

$$d(s(t), s^*(t)) \leq W^*(t). \tag{4.3}$$

Moreover, the following relation between the current time and the waiting time holds:

$$t = d(o, s^*(t)) + W^*(t). \tag{4.4}$$

Since the adversary is fair, its position $s^*(t)$ at time t cannot be to the right of F. Thus, $d(s^*(t), x) = d(s^*(t), F) + d(F, x)$, which gives us

$$\begin{aligned} \mathrm{OPT}(\sigma) &\geq t + d(s^*(t), F) + d(F, x) + x \qquad (4.5)\\ &\overset{(4.4)}{=} d(o, s^*(t)) + W^*(t) + d(s^*(t), F) + d(F, x) + x \\ &= W^*(t) + F + d(F, x) + x \\ &= W^*(t) + 2x \\ &\geq W^*(t) + 2d(s(t), s^*(t)) \\ &\overset{(4.3)}{\geq} 3d(s(t), s^*(t)) \qquad (4.6) \end{aligned}$$

At time t MRIN's server has to move from its current position $s(t)$ to x and from there move to the origin:

$$\begin{aligned} \mathrm{MRIN}(\sigma) &= t + d(s(t), x) + x \\ &= t + d(s(t), s^*(t)) + d(s^*(t), F) + d(F, x) + x. \end{aligned}$$

Hence,

$$\begin{aligned} \frac{\mathrm{MRIN}(\sigma)}{\mathrm{OPT}(\sigma)} &= \frac{t + d(s^*(t), F) + d(F, x) + x}{\mathrm{OPT}(\sigma)} + \frac{d(s(t), s^*(t))}{\mathrm{OPT}(\sigma)} \\ &\overset{(4.5)}{\leq} 1 + \frac{d(s(t), s^*(t))}{\mathrm{OPT}(\sigma)} \\ &\overset{(4.6)}{\leq} \frac{4}{3}. \end{aligned}$$

This proves the claim. □

Given the lower bound for general non-zealous algorithms in Theorem 4.11 we conclude that online algorithms that may obtain better competitive ratios against a fair adversary will have to be non-zealous, i.e., incorporate waiting times.

The problem with Algorithm MRIN is that shortly after it starts to return towards the origin from the furthest previously unserved request, a new request to the right of its server arrives (becomes known). In this case the MRIN-server has to return to a position it just left. Algorithm WS presented below avoids this pitfall.

Algorithm WS ("Wait Smartly")

The WS-server moves right if there are unserved requests to the right of his present position. Otherwise, he takes the following actions. Suppose he arrives at his present position $s(t)$ at time t.

1. Compute the optimal offline solution value $\mathrm{OPT}(\sigma_{\leq t})$ for all requests released up to time t.
2. Determine a waiting time $W := \alpha\,\mathrm{OPT}(\sigma_{\leq t}) - s(t) - t$, with $\alpha = (1 + \sqrt{17})/4$.
3. Wait at point $s(t)$ until time $t + W$ and then start to move back to the origin O.

We note that when the server is moving back to the origin and no new requests are released until time $t + W + s(t)$, then the WS-server reaches the origin O at time $t + W + s(t) = \alpha\,\mathrm{OPT}(\sigma_{\leq t})$ having served all requests released so far. If a new request is released at time $t' \leq W + t + s(t)$ and the request is to the right of $s(t')$, then the WS-server interrupts its waiting and starts to move to the right immediately until it reaches the furthest unserved request.

Theorem 4.15. *Algorithm* WS *is α-competitive for the* OlTsp *on the non-negative part $\mathbb{R}_0^+$ of the real line against a fair adversary, where $\alpha = (1 + \sqrt{17})/4 \approx 1.28$.*

Proof. By the definition of the waiting time it is sufficient to prove that at any point where a waiting time is computed this waiting time is non-negative. In that case the server will always return at 0 no later than time $\alpha\,\mathrm{OPT}(\sigma)$. This is clearly true if the sequence σ contains at most one request. We make the induction hypothesis that WS is α-competitive for any sequence of at most $m-1$ requests.

Let $\sigma = \sigma_1, \ldots, \sigma_m$ be any sequence of requests. As in the proof of Theorem 4.14 we consider the time $t = t_m$ when the last set of requests $\sigma_{=t}$ is released, and let $\sigma_m = (t, x)$ be the request of $\sigma_{=t}$ that is furthest away from the origin. If $t = 0$, then the claim obviously holds, so we will assume for the remainder of the proof that $t > 0$.

We denote by $s(t)$ and $s^*(t)$ the positions of the WS-server and the fair adversary's server at time t, respectively. As before we let $\sigma_f = (t_f, f)$ be the furthest

unserved request by WS at time t of $\sigma_{<t}$. Finally, let $\sigma_F = (t_F, F)$ be the furthest released request in $\sigma_{<t}$.

We now distinguish three different cases depending on the position of x relative to f and F. Recall that $f \leq F$.

Case 1: $x \leq f$ (see Figure 4.1)
First note that the optimal offline completion time cannot decrease by an additional request. Since the WS-server has to travel to f anyway, and by the induction hypothesis there was a non-negative waiting time in f or $s(t)$ (depending on whether $s(t) > f$ or $s(t) \leq f$) before requests $\sigma_{=t}$ were released, the waiting time in f or $s(t)$ cannot decrease.

Case 2: $f \leq x < F$
If $s(t) \geq x$, then again by the induction hypothesis and the facts that the route length of WS's server does not increase and the optimal offline completion time does not decrease, the possible waiting time at $s(t)$ is non-negative.

Thus we can assume that $s(t) < x$ (see Figure 4.2). The WS-server will now travel to point x, arrive there at time $t + d(s(t), x)$, and possibly wait there some time W before returning to the origin, with

$$W = \alpha \mathrm{OPT}(\sigma) - (t + d(s(t), x) + x).$$

Inserting the obvious lower bound $\mathrm{OPT}(\sigma) \geq t + x$ yields

$$W \geq (\alpha - 1)\mathrm{OPT}(\sigma) - d(s(t), x). \tag{4.7}$$

To bound $\mathrm{OPT}(\sigma)$ in terms of $d(s(t), x)$ consider the time t' when the WS-server had served the request at F and started to move left. It must be that $t' < t$ since otherwise $s(t)$ could not be smaller than x as assumed. Thus, the subsequence $\sigma_{\leq t'}$ of σ does not contain (t, x). By the induction hypothesis, WS is α-competitive for the sequence $\sigma_{\leq t'}$. At time t' when it left F it would have arrived at α times the optimal offline cost $\mathrm{OPT}(\sigma_{\leq t'})$ on that subsequence:

$$t' + F = \alpha \cdot \mathrm{OPT}(\sigma_{\leq t'}). \tag{4.8}$$

Notice that $t \geq t' + d(F, s(t))$. Since $\mathrm{OPT}(\sigma_{\leq t'}) \geq 2F$ we obtain from (4.8) that

$$t \geq \alpha 2F - F + d(F, s(t)) = (2\alpha - 1)F + d(s(t), F). \tag{4.9}$$

Since by assumption we have $s(t) < x < F$, we get that $d(s(t), x) \leq d(s(t), F)$ and $d(s(t), x) \leq F$, which inserted in (4.9) yields

$$t \geq (2\alpha - 1)d(s(t), x) + d(s(t), x) = 2\alpha d(s(t), x). \tag{4.10}$$

We combine this with the previously mentioned lower bound $\mathrm{OPT}(\sigma) \geq t + x$ to obtain

$$\mathrm{OPT}(\sigma) \geq 2\alpha d(s(t), x) + x \geq (2\alpha + 1)d(s(t), x). \tag{4.11}$$

Using inequality (4.11) in (4.7) gives

$$\begin{aligned} W &\geq (\alpha-1)(2\alpha+1)d(s(t),x) - d(s(t),x) \\ &= (2\alpha^2-\alpha-2)d(s(t),x) \\ &= \left(\frac{9+\sqrt{17}}{4} - \frac{1+\sqrt{17}}{4} - 2\right) d(s(t),x) \\ &= 0. \end{aligned}$$

This completes the proof for the second case.

Case 3: $f \leq F \leq x$ (see Figure 4.3)
Starting at time t the WS-server moves to the right until it reaches x, and after waiting there an amount W returns to 0, with

$$W = \alpha \mathrm{OPT}(\sigma) - (t + d(s(t),x) + x). \tag{4.12}$$

Again we will show that $W \geq 0$, i.e., that also in this case the computed waiting time at x for WS is non-negative. At time t the adversary's server still has to travel at least $d(s^*(t),x)+x$ units. This results in

$$\mathrm{OPT}(\sigma) \geq t + d(s^*(t),x) + x.$$

Since the offline adversary is fair, its position $s^*(t)$ at time t cannot be strictly to the right of F. This yields

$$\mathrm{OPT}(\sigma) \geq t + d(F,x) + x. \tag{4.13}$$

Insertion into (4.12) yields

$$\begin{aligned} W &\geq (\alpha-1)\mathrm{OPT}(\sigma) + \mathrm{OPT}(\sigma) - (t + d(s(t),x) + x) \\ &\geq (\alpha-1)\mathrm{OPT}(\sigma) + d(F,x) - d(s(t),x) \\ &= (\alpha-1)\mathrm{OPT}(\sigma) - d(s(t),F) \end{aligned} \tag{4.14}$$

since $s(t) \leq F$ by definition of the algorithm WS.

The rest of the arguments is similar to those used in the previous case. Again suppose that WS's server started to move to the left from F at some time $t' \leq t$, where t' is chosen to be maximal. By the induction hypothesis the WS-server would have returned to the origin at time $\alpha\,\mathrm{OPT}(\sigma_{<t'})$ if the requests in $\sigma_{=t}$ had not been released. Hence

$$t' + F = \alpha\,\mathrm{OPT}(\sigma_{<t'}). \tag{4.15}$$

Notice that $t \geq t' + d(s(t),F)$ and $\mathrm{OPT}(\sigma_{<t'}) \geq 2F$, by the fact that $\sigma_{<t'}$ must contain at least one request at F since otherwise WS would not have moved its server to F. Hence we obtain from (4.15) that

$$t \geq \alpha 2F - F + d(s(t),F) = (2\alpha-1)F + d(s(t),F) \geq 2\alpha d(s(t),F).$$

We combine this with (4.13) and the fact that $x \geq F \geq d(s(t), F)$ to achieve

$$\mathrm{OPT}(\sigma) \geq 2\alpha d(s(t), F) + d(F, x) + x \geq (2\alpha + 1)d(s(t), F).$$

Using this inequality in (4.14) gives

$$\begin{aligned} W &\geq (\alpha - 1)(2\alpha + 1)d(s(t), F) - d(s(t), F) \\ &= (2\alpha^2 - \alpha - 2)d(s(t), F) \\ &= \left(\frac{9 + \sqrt{17}}{4} - \frac{1 + \sqrt{17}}{4} - 2\right) d(s(t), F) \\ &= 0. \end{aligned}$$

This completes the proof. □

5

Online dial-a-ride problems under restricted information models

The content of this chapter is joint work with M. Lipmann, X. Lu, R.A. Sitters, and L. Stougie, and has appeared in [38].

5.1 Introduction

An instance of the online single server dial-a-ride problem (OLDARP) is specified by a metric space $M = (X, d) \in \mathcal{M}$ (see Section 3.1) with a distinguished origin $O \in X$, a sequence $\sigma = \sigma_1, \ldots, \sigma_m$ of requests for rides, and a capacity for the server. A server is located at the origin O at time 0 and can move at most at unit speed. Each *ride* is a triple $\sigma_i = (t_i, s_i, d_i)$, where $t_i \in \mathbb{R}_0^+$ is the time at which ride σ_i is released, $s_i \in X$ is the source of the ride, and $d_i \in X$ is the destination of the ride. Every ride $\sigma_i \in \sigma$ has to be executed (served) by the server, that is, the server has to visit the source, start the ride, and end it at the destination. The *capacity* of the the server is an upper bound on the number of rides the server can execute simultaneously. We consider unit capacity, constant capacity $c \geq 2$, and infinite capacity for the server. The objective in the OLDARP problem is to minimize the makespan of the server, that is, the time when the server has served all rides and has has returned to the origin. We assume that the sequence $\sigma = \sigma_1, \ldots, \sigma_m$ of rides is revealed in order of non-decreasing release times, and that the online server has neither information about the time when the last ride is released, nor about the total number of rides. For $t \geq 0$ we denote by $\sigma_{\leq t}$ the set of rides in σ released no later than time t. An

online algorithm for the OLDARP must determine the behavior of the server at any moment t as a function of t and $\sigma_{\leq t}$, whereas the offline algorithm knows σ at time 0. A feasible solution is a route for the server that starts and ends in the origin O and serves all requested rides such that each ride is picked up at the source not earlier than the time it is released.

Online dial-a-ride problems have been studied in literature before [3], [12]. In these papers it is assumed that the rides are specified completely upon presentation, i.e., both the source and the destination of the ride become known at the same time. We diverge from this setting here, because in many practical situations complete specification of the rides is not realistic. Think for example of the problem to schedule an elevator. Here, a ride is the transportation of a person from one floor (the source) to another (the destination), and the release time of the ride is the moment the button on the wall outside the elevator is pressed. The destination of such a ride is revealed only at the moment the person enters the elevator and presses the button inside the elevator.

We feel that lack of information is often a *choice*, rather than inherent to the problem: additional information *can* be obtained, but this requires investments in information systems. In this chapter we give mathematical evidence that for the problem under study it pays to invest.

We study the online single server dial-a-ride problem in which only the source of a ride is presented at the release time of the ride. The destination of a ride is revealed at visiting its source. We call this model the *incomplete ride information model* and refer to the model used in [3] and [12] as the *complete ride information model*.

We distinguish two versions of the online dial-a-ride problem under the incomplete ride information model. In the first version the server is allowed to preempt any ride at any point, and resume the ride later. In particular the server is allowed to visit the source of a ride and learn its destination without executing the ride immediately. This version we call the *preemptive version*. In the second version, the *non-preemptive* version, a ride has to be executed as soon as the ride has been picked up in the source. In this version we do allow the server to pass a source without starting the ride, in which case he does not learn the destination of the ride at passing the source. We study each version of the problem under various *capacities* of the server.

In [3] a 2-competitive deterministic algorithm is given for the online dial-a-ride problem under the complete ride information model, independent of the capacity of the server. In this paper preemption of rides is not allowed. However, the lower bound of 2 comes from a sequence of rides with zero length, an instance of the online traveling salesman problem [4], hence the bound also holds for the problem with preemption. We show that under the incomplete ride information model, no deterministic algorithm can have a competitive ratio smaller than 3, even if preemption is allowed, and independent of the capacity of the server. For the preemptive version, we design an algorithm with competitive ratio matching the lower bound of 3, independent of the capacity of the server. These results are presented in Section 5.2.

If preemption is not allowed, we derive a lower bound of $\max\{c, 1+\frac{3}{2}\sqrt{2}\}$ on the competitive ratio of any deterministic algorithm, where c is a given fixed capacity of the server. We present a $(2c+2)$-competitive algorithm for the non-preemptive version. These results are presented in Section 5.3.

We notice that there is no difference between the preemptive version and the non-preemptive version of the problem if the server has infinite capacity, hence we inherit the matching lower and upper bound of 3 of the preemptive version for this case. An overview of the results obtained in this chapter is given in Table 5.1.

Table 5.1: Overview of lower bounds (LB) and upper bounds (UB) on the competitive ratio of deterministic algorithms for online dial-a-ride problems.

	capacity	LB	UB
complete ride information			
preemption	$1, c, \infty$	2 [4]	2 [3]
no preemption	$1, c, \infty$	2 [4]	2 [3]
incomplete ride information			
preemption	$1, c, \infty$	3	3
no preemption	1	$1+\frac{3}{2}\sqrt{2}$	4
	c	$\max\{1+\frac{3}{2}\sqrt{2}, c\}$	$2c+2$
	∞	3	3

The results in this chapter, combined with those from [3], show the effect of having complete knowledge about rides on worst-case performance for online dial-a-ride problems. This is an important issue, since in practice complete information is often lacking. Investments in information systems can help to obtain more complete information, and mathematical support is essential in justifying such investments.

We conclude this introduction by referring back to the elevator scheduling problem. We have seen that the typical elevator with only a request button at the wall outside the elevator fits our incomplete ride information model. In an alternative construction of an elevator, the destination buttons could be built outside the elevator, fitting the complete ride information model. Notice that minimizing the latest completion time is not the most natural objective for an elevator.

5.2 The preemptive version

We describe our algorithm SNIFFER, which preempts rides only immediately at the source, just to learn the destinations of the rides: it "sniffs" the rides. Upon visiting the source of a ride for the second time, the ride is completed right away. The algorithm is an adaption of the 2-competitive algorithm for the online traveling salesman problem (OLTSP), described in [4]. The proof of 3-competitiveness of SNIFFER borrows parts of the proof in the latter paper. The algorithm is described completely by the actions it takes at any moment t at which the server either arrives in the origin or receives a new request. When SNIFFER computes an optimal tour over a set of requests or rides, we always assume that this tour includes the origin.

We use $|T|$ to denote the length of a tour T.

Algorithm SNIFFER

(1) **The server is in the origin at t.**
If the set S of yet unvisited sources is non-empty, compute the optimal traveling salesman tour $T_{\text{TSP}}(S)$ on the points in S, and start following $T_{\text{TSP}}(S)$. Just learn the destinations of the rides with sources in S, without starting to execute any of these rides.
If $S = \emptyset$ and the set R of rides yet to be executed is non-empty, compute the optimal dial-a-ride tour $T_{\text{DAR}}(R)$ on the rides in R. Also compute the optimal dial-a-ride tour $T_{\text{DAR}}(\sigma_{\leq t})$ on all rides requested in $\sigma_{\leq t}$. If $t = 2|T_{\text{DAR}}(\sigma_{\leq t})|$, start following $T_{\text{DAR}}(R)$. If $t < 2|T_{\text{DAR}}(\sigma_{\leq t})|$, remain idle. If no new requests arrive before time $2|T_{\text{DAR}}(\sigma_{\leq t})|$ start following $T_{\text{DAR}}(R)$ at time $2|T_{\text{DAR}}(\sigma_{\leq t})|$.

(2) **The server is on a tour $T_{\text{TSP}}(S)$ at t when a new ride is released.**
Let p_t denote the location of the server at time t. If the new ride, say $\sigma_k = (t, s_k, ?)$ (the question mark indicating the unknown destination), is such that $d(s_k, O) > d(p_t, O)$, then return to the origin via the shortest path, ignoring all rides released while traveling to the origin.
If $d(s_k, O) \leq d(p_t, O)$, ignore the new ride until the origin is reached again and proceed on $T_{\text{TSP}}(S)$.

(3) **The server is on a tour $T_{\text{DAR}}(R)$ at t when a new ride is released.**
Return to the origin as soon as possible via the shortest path, and ignore rides released in the mean time. If the server is executing a ride, the ride is finished before returning to the origin.

Theorem 5.1. *Algorithm* SNIFFER *is* 3*-competitive for the preemptive* OLDARP *problem under the incomplete ride information model, independent of the capacity of the server.*

PROOF. Let $T_{\text{DAR}}(\sigma)$ be the optimal tour over all rides of σ. It is sufficient to prove that for any sequence σ the server can always be in the origin at time $2|T_{\text{DAR}}(\sigma)|$ to start the final tour on the yet unserved rides: He will then always finish this tour before time $3|T_{\text{DAR}}(\sigma)| \leq 3\text{OPT}(\sigma)$. This is obviously true for any sequence σ consisting of only one ride. We assume it holds for any sequence of $m-1$ rides, and prove that then it also holds for any sequence σ of m rides. Let $\sigma_m = (t_m, s_m, d_m)$ be the last ride in σ (notice that the destination d_m is not given to the online algorithm until the moment the source s_m is visited).

(1) Suppose the server is in O at t_m, and $S \neq \emptyset$. He starts tour $T_{\text{TSP}}(S)$ and returns to O at time $t_m + |T_{\text{TSP}}(S)| \leq 2|T_{\text{DAR}}(\sigma)|$.

(2) Suppose the server is in p_{t_m} following $T_{\text{TSP}}(S)$. If $d(O, s_m) \leq d(O, p_{t_m})$, σ_m is added to a set Q of rides ignored since the last time the server was in O. Let $s_q \in Q$ be the source of a ride visited first in an optimal solution. Since this ride

was ignored, the server was at least $d(O, s_q)$ away from the origin at time t_q, and hence had moved at least this distance on tour $T_{\text{TSP}}(S)$. Thus, the server returns in O before $t_q + |T_{\text{TSP}}(S)| - d(O, s_q)$. Back in O the server commences on $T_{\text{TSP}}(Q)$. Let $P_q(Q)$ be the path of minimum length that starts in s_q, ends in O, and visits all sources in Q. Obviously, $|T_{\text{TSP}}(Q)| \leq d(O, s_q) + |P_q(Q)|$. Hence the server is back in the origin after visiting all sources no later than $t_q + |T_{\text{TSP}}(S)| - d(O, s_q) + d(O, s_q) + |P_q(Q)| = t_q + |T_{\text{TSP}}(S)| + |P_q(Q)| \leq 2|T_{\text{DAR}}(\sigma)|$, since, clearly, $|T_{\text{DAR}}(\sigma)| \geq t_q + |P_q(Q)|$, and $|T_{\text{DAR}}(\sigma)| \geq |T_{\text{TSP}}(S)|$.

If $d(O, s_m) > d(O, p_{t_m})$, the server returns to O immediately, arriving there at $t_m + d(O, p_{t_m}) < t_m + d(O, s_m) \leq |T_{\text{DAR}}(\sigma)|$. Back in O the server computes and starts following an optimal TSP tour over the yet unvisited sources, which has a length of at most $|T_{\text{DAR}}(\sigma)|$. Hence the server is back in O again before time $2|T_{\text{DAR}}(\sigma)|$.

If the server was already moving towards the origin because a ride was released before σ_m that was further away from the origin than the online server, then the arguments above remain valid.

(3) Suppose the server is on a tour $T_{\text{DAR}}(R)$ at time t_m, or moving towards O because of another ride released before t_m. Let $t(R)$ be the time at which the server started $T_{\text{DAR}}(R)$. Then $R \subset \sigma_{\leq t(R)}$ and $t(R) = 2|T_{\text{DAR}}(\sigma_{\leq t(R)})|$, by induction. Thus, the server is back in O before time $3|T_{\text{DAR}}(\sigma_{\leq t(R)})|$. There, it starts a tour $T_{\text{TSP}}(S)$ over a set S of unvisited sources, being again back in O before time $3|T_{\text{DAR}}(\sigma_{t(R)})| + |T_{\text{TSP}}(S)| = \frac{3}{2}t(R) + |T_{\text{TSP}}(S)|$. We need to show that this is not greater than $2|T_{\text{DAR}}(\sigma)|$.

Let s_q be the first ride from S served in an optimal solution and $P_q(S)$ the shortest path starting in $s_q \in S$, ending in O, and visiting all sources in S. Clearly, $|T_{\text{DAR}}(\sigma)| \geq t_q + |P_q(S)|$ and $|T_{\text{TSP}}(S)| \leq 2|P_q(S)|$. Since all rides in S are released after $t(R)$, $t_q \geq t(R)$. Therefore, $|T_{\text{DAR}}(\sigma)| \geq t(R) + |P_q(S)|$ and

$$\frac{3}{2}t(R) + |T_{\text{TSP}}(S)| \leq 2t(R) + 2|P_q(S)| \leq 2|T_{\text{DAR}}(\sigma)|.$$

□

We show that SNIFFER is a best possible deterministic algorithm for the preemptive version of the OLDARP problem, even if SNIFFER uses preemption only at the source of rides.

Theorem 5.2. *No deterministic algorithm can have a competitive ratio smaller than $3 - \epsilon$ for the* OLDARP *problem under the incomplete ride information model, independent of the capacity of the server, where ϵ is arbitrarily small.*

PROOF. For the proof of this theorem we use a commonly applied setting of a two-person game, with an adversary providing a sequence of rides, and an online algorithm serving the rides (see [7]). Typically, the outcome of the algorithm is compared by the solution value the adversary achieves himself on the sequence, which

usually is the optimal offline solution value. We consider the OLDARP problem under the incomplete ride information model where the online server has infinite capacity. Let ALG be a deterministic online algorithm for this problem. We will construct an adversarial sequence σ of requests for rides. We restrict the adversary by giving his server capacity 1. We will prove that ALG can not be better than $3-\epsilon$-competitive for this restricted adversary model, where ϵ is arbitrary small.

The metric space $M = (X, d)$ is a graph with vertex set $X = \{x_1, x_2, \ldots, x_{n^2}\} \cup O$ and the distance function d, where $d(O, x_i) = 1$ and $d(x_i, x_j) = 2$ for all $x_i, x_j \in X \setminus O$. To facilitate the exposition we denote point x_i by i.

At time 0 there is one ride in each of the n^2 points in $X \setminus \{O\}$. If the online server visits the source i of a ride at time t with $t \leq 2n^2 - 1$, then the destination turns out to be i as well, and at time $t+1$, a new ride with source i is released.

In this way, the situation remains basically the same for the online server until time $2n^2$. We may assume that at some moment t^*, with $2n^2 - 1 < t^* \leq 2n^2$, there is exactly one ride $\sigma_i = (t_i, i, d_i)$ in each of the points i. Without loss of generality we assume that the vertices i are labeled in such a way that $t_1 \leq \cdots \leq t_{n^2}$.

Thus, at time t^* the online server still has to complete exactly n^2 rides. We partition the set of n^2 vertices into n sets: $I_k = \{(n-1)k+1, \ldots, nk\}$, $k = 1, \ldots, n$. Within each of these sets we order the vertices by the online server's first visit to them after time t^*. Let b_{kj}, $j \in \{1, \ldots, n\}$ be the jth vertex in this order in I_k. Now we define for all $k \in \{1, \ldots, n\}$ the destination of the ride in vertex b_{kj} as b_{k1} for $j = 1$ and $b_{k,j-1}$ for all $j \in \{2, \ldots, n\}$. Notice that the destination of ride σ_i only depends on the tour followed by the online server until he picks up the ride to look at its destination. For the online server this means that n of the n^2 rides can be served immediately since the source equals the destination. For the other $n^2 - n$ rides the server finds out that the destination of the rides he just picked up is another point that he already visited after time t^*. Therefore, $n^2 - n$ points will have to be visited by the online server at least twice after time t^*. Hence, the completion time for the online server is at least $t^* + 4(n^2 - n) - 1 + 2n > 6n^2 - 2n - 2$.

We will now describe the tour made by the adversary. Given our definition of t^* we have that $t_{n^2} \leq t^* \leq 2n^2$. Since the online server needs at least 2 time units to move from a point i to another point i', it follows that $t_i \leq 2i$, for all $i \in \{1, \ldots, n^2\}$. The adversary waits until time $2n$ and then starts to serve the rides $\sigma_1, \ldots, \sigma_n$, by visiting the sources in reversed order of $b_{11}, \ldots, b_{1n}$. The rides with equal source and destination are served immediately at arrival in the point. This takes the adversary $2n$ time units. At time $4n$ the adversary starts serving the rides $\sigma_{n+1}, \ldots, \sigma_{2n}$, and then at time $6n$ the rides $\sigma_{2n+1}, \ldots, \sigma_{3n}$, etc. Continuing like this the server has completed all the rides and is back in the origin at time $2n^2 + 2n$.

Hence, the competitive ratio is bounded from below by $(6n^2 - 2n - 2)/(2n^2 + 2n)$, which can be made arbitrarily close to 3 by choosing n large enough. □

5.3 The non-preemptive version

For the non-preemptive version we design an algorithm, called BOUNCER, because the server always "bounces" back to the source, once a ride is completed. This algorithm uses as a subroutine the 2-competitive algorithm for the OLTSP problem from [4].

Algorithm BOUNCER

Perform the OLTSP algorithm on the sources of the rides. This algorithm outputs a tour T. The BOUNCER server follows tour T, until a source is visited. There he executes the ride, and returns to the source via the shortest path. As soon as the server arrives in the source again, he continues to follow T.

Theorem 5.3. *Algorithm* BOUNCER *is* $(2c+2)$*-competitive for the* OLDARP *problem under the incomplete ride information model, where* c *is the capacity of the server.*

PROOF. Consider any request sequence σ. Since the OLTSP algorithm is 2-competitive, and in any solution for the OLDARP problem all sources have to be visited, $\mathrm{OPT}(\sigma) \geq |T|/2$. Let $D = \sum_{i:\sigma_i \in \sigma} d(s_i, d_i)$, then also $\mathrm{OPT}(\sigma) \geq D/c$. The completion time of the BOUNCER server is at most $T + 2D \leq 2\mathrm{OPT}(\sigma) + 2c\mathrm{OPT}(\sigma)$. □

Corollary 5.4. *Algorithm* BOUNCER *is* 4*-competitive for the* OLDARP *problem under the incomplete ride information model, if the capacity of the server is* 1.

Theorem 5.5. *No non-preemptive deterministic online algorithm can have a competitive ratio smaller than* $c - \epsilon$ *for the* OLDARP *problem under the incomplete ride information model, where the capacity of the server* c *is a constant, and where* $\epsilon > 0$ *is arbitrarily small.*

PROOF. Consider an instance of the OLDARP problem on a star graph with $K >> c$ leaves, where the origin is located in the center of the star, and each leaf has distance 1 to the origin. At time 0, cK rides are released, all with their source in the origin, and each of the leaves being destination of c rides. Thus, there are K sets of c identical rides each, hence the instance has an optimal solution value of $2K$.

Any online server can carry at most c rides at a time. The instance is constructed in such a way that, until time $2(cK - c^2)$, any time the online server is in the origin he has all different rides (rides with different destinations). It is clear that this can indeed be arranged, given that the online server can not distinguish between the rides until he picks them up at the source and he has no possibility to preempt, not even in the source. At time $2(cK - c^2)$ the online server can have served at most $cK - c^2$ rides, and hence at least c^2 rides remain yet to be served, requiring an extra

of at least $2c$ time units. Hence the completion time of any online server is at least $2(cK - c^2) + 2c$.

Therefore, the competitive ratio is bounded from below by

$$\frac{2(cK - c^2) + 2c}{2K} = \frac{cK - c^2 + c}{K}.$$

For any $\epsilon > 0$ we can choose K large enough for the theorem to hold. □

Together with Theorem 5.1 this theorem shows that for servers with capacity greater than 3, the best possible deterministic online algorithm for the non-preemptive version of the problem has a strictly higher competitive ratio than SNIFFER for the preemptive problem, in which the server can have any capacity. The following theorem shows that this phenomenon also occurs for lower capacities of the server.

Theorem 5.6. *No non-preemptive deterministic algorithm can have a competitive ratio smaller than* $1+\frac{3}{2}\sqrt{2}-\epsilon \approx 3.12$ *for the* OLDARP *problem under the incomplete ride information model, where the capacity of the server* c *is a constant and* ϵ *is arbitrarily small.*

PROOF. First we consider the OLDARP problem under the incomplete ride information model when the online server has capacity 1. Then we will sketch how to extend the proof for any capacity c.

Let ALG be a non-preemptive deterministic online algorithm for this problem. The metric space is a star graph with $2n$ leaves. All leaves have length 1. The center of the star is the origin O and the leaf vertices are denoted by a_i $(i = 1 \dots n)$, and b_i $(i = 1 \dots n)$. On every leaf a_i and b_i there is an additional vertex a_i' or b_i' at distance $1 - \alpha$ $(0 < \alpha < 1)$ from the origin, where α is a fixed number that we choose appropriately later.

We give the following sequence σ of rides. At time zero there are three rides in each point a_i and b_i, $i = 1 \dots n$. If the online server visits a source, then the destination turns out to be the same as the source. This kind of rides are called empty rides. One time unit later the ride is replaced by a new ride with the same source. Every source that is visited by the online server before time $4n$ is handled in this way. Sources visited after this time are not replaced.

Let t_i^a (resp. t_i^b) be the last moment before time $4n - 4$ that the online server is in point a_i (resp. b_i). We set $t_i^a = 0$ (resp. $t_i^b = 0$) if a_i (resp. b_i) is not visited before $4n - 4$. Without loss of generality we assume that $t_i^a \le t_j^a$, $t_i^b \le t_j^b$, $t_i^a \le t_j^b$, and $t_i^b \le t_j^a$, for all $1 \le i < j \le n$. Any server needs at least 2 time units to travel from one leaf to another, implying that $t_i^a \le 4(i-1)$ and $t_i^b \le 4(i-1)$, for all $i \in \{1, \dots, n\}$.

We refer to the three rides that were released latest in a leaf as the *decisive rides* and define them as follows. In each point b_i two of the decisive rides are empty and one has destination a_i. In point a_i one of the decisive rides is empty, one has

destination a_i', and one is either empty or has destination O. With these rides, the online server is unable to distinguish between the points a_i and b_i, and since we did not distinguish between these points before, we may assume that, after time $4n-4$, the online server visits point a_i before point b_i. The first ride that the server picks up in point a_i is the ride to a_i'. The first ride the server picks up in point b_i is the ride to a_i. We distinguish between two cases. In the first case the online server executes the ride from b_i to a_i before it picks up the second ride in a_i. In this case the second ride is a ride to the origin. Otherwise, the second ride being picked up in a_i is empty.

In the first case the online server needs at least 10 time units (from origin to origin) to serve all the rides connected to each pair a_i, b_i, whereas in the optimal offline solution only $4+2\alpha$ time units are required. In the second case the online server needs at least $8+2\alpha$ time units, whereas in the optimal offline solution 4 time units are required. The optimal offline strategy starts a tour at time 0, first serving a_1 and b_1, then serving a_2 and b_2, etc. Empty rides are served without taking any extra time.

The online server cannot start with the decisive rides until time $4n-4$. Assume that he is in the origin at time $4n-5$ and then moves to point a_1. Now we consider the contribution of a pair a_i, b_i in the total time needed for the online and the offline server and we take the ratio of the two. For fixed α ($0 < \alpha < 1$) this ratio becomes at least

$$\min\left\{\tfrac{10+(4n-5)/n}{4+2\alpha}, \tfrac{8+2\alpha+(4n-5)/n}{4}\right\} \quad > \quad \min\left\{\tfrac{14}{4+2\alpha}, \tfrac{12+2\alpha}{4}\right\} - \tfrac{2}{n}.$$

Optimizing over α yields $\alpha = -4+3\sqrt{2}$. Hence, for the competitive ratio we find

$$\frac{\mathrm{ALG}(\sigma)}{\mathrm{OPT}(\sigma)} > 1 + \frac{3}{2}\sqrt{2} - \frac{2}{n}.$$

For any ϵ we can choose n large enough for the theorem to hold in the case of a unit capacity server. If the capacity of the server is c, with $c \geq 1$, we just give c copies of the same sequence σ simultaneously. An online server cannot benefit from this extra capacity in combining rides from different pairs a_i, b_i. The online server will have to do the rides in a specific point in the same order as before. For example the first c rides that the online server picks up in a_i are rides to a_i'. Hence, the completion time for the online server cannot be smaller than in the capacity 1 case, and the offline server can complete in exactly the same time. □

Corollary 5.7. *No non-preemptive deterministic online algorithm can have a competitive ratio smaller than* $\max\{1+\frac{3}{2}\sqrt{2}, c\} - \epsilon$ *for the* OlDarp *problem under the incomplete ride information model, where the capacity of the server c is a constant and $\epsilon > 0$ is arbitrarily small.*

5.4 Discussion

In [3] and [12] the competitive ratio measures the cost of having no information about the release times of future rides. In this discussion we show how we can measure the cost of having no information about the destinations of the rides through the competitive ratio.

Suppose that at time 0 the release times and the location of the sources of the rides are given, but the information about the destinations is again revealed only at visiting the sources.

Both SNIFFER and BOUNCER use the online algorithm of Aussiello et al. [4] for a TSP tour along the sources. In case all sources of the rides and the release times are known, an optimal TSP tour over the sources, that satisfies the release time constraints, can be computed (disregarding complexity issues). In this way SNIFFER and BOUNCER gain an additive factor of 1 on their competitive ratio, making SNIFFER 2-competitive and BOUNCER $2c+1$-competitive.

Notice that the lower bound of $c - \epsilon$ on the competitive ratio for the non-preemptive problem in Theorem 5.5 is obtained through a sequence of rides all with release time 0, thus this lower bound is completely due to the lack of information about the destinations of the rides.

The rides in the sequence giving the lower bound of $1 + \frac{3}{2}\sqrt{2}$ for the non-preemptive problem in Theorem 5.6 have release times no larger than $4n-5$. Taking the unserved rides at time $4n-5$ as an instance given at time 0, shows that the competitive ratio is at least $\min\{\frac{10}{4+2\alpha}, \frac{8+2\alpha}{4}\}$. Optimizing over α yields a lower bound of $\frac{1}{2} + \frac{1}{2}\sqrt{11} \approx 2,15$. Thus due to the lack of information about destinations only, any algorithm will not be able to attain a ratio of less than $\max\{\frac{1}{2}+\frac{1}{2}\sqrt{11}, c-\epsilon\}$.

In the lower bound construction for the preemptive problem in Theorem 5.2 the adversary stops giving requests at time $2n^2$. Take the set of rides unserved by any online algorithm at that time as an instance with release time 0. Following the proof of Theorem 5.2 any online algorithm will need $4n^2 - 2n$, whereas an optimal tour takes $2n^2$, yielding a lower bound of 2.

Notice that the above lower bounds are established on sequences where all rides have release time 0. For the preemptive version of the problem this is clearly sufficient, since the performance of SNIFFER matches the lower bound. However, for the non-preemptive version higher lower bounds might be obtained using diverse release times of rides.

6

Online dial-a-ride problems minimizing latency

The content of this section is joint work with S.O. Krumke, D. Poensgen and L. Stougie, and has appeared in [31].

6.1 Introduction

In this chapter we study two online dial-a-ride problems, where the objective is to minimize the weighted sum of completion times, or the latency. The first problem is the online version of the *traveling repairman problem* (TRP) [1], where a server must visit a set of m points $p_1, \ldots, p_m$ in a metric space. The server starts in a designated point 0 of the metric space, called the *origin*, and travels at most at unit speed. Given a tour through the m points, the completion time C_j of point p_j is defined as the time traveled by the server on the tour until it reaches p_j ($j = 1, \ldots, m$). Each point p_j has a weight w_j, and the objective of the TRP is to find the tour that minimizes the total weighted completion time $\sum_{j=1}^{m} w_j C_j$. This objective is also referred to as the *latency*.

Here, we consider the online version of the TRP called the *online traveling repairman problem* (OLTRP). Requests for visits to points are released over time while the repairman (the server) is traveling. In the online setting the completion time of a request σ_j at point p_j with release time t_j is the first time at which the repairman visits p_j after the release time t_j. The online model allows the server to wait. However, waiting yields an increase in the completion times of the points still to be

served. Decisions are revocable as long as they have not been executed.

The second problem that we consider is a generalized version of the OlTrp, that we will call the L-OlDarp (for "latency online dial-a-ride problem"). In this problem each request specifies a ride from one point in the metric space, its source, to another point, its destination. Given a tour that serves all rides, the completion time C_j of ride j is defined as the time traveled by the server on the tour until it reaches the destination of ride j. We assume that the server can serve only one ride at a time, and preemption of rides is not allowed: once a ride is started it has to be finished without interruption.

Definition 6.1. [L-OlDarp, OlTrp]
Given a request sequence $\sigma = \sigma_1, \ldots, \sigma_m$ the problem L-OlDarp *is to find a feasible route S minimizing $\sum_{j=1}^{m} w_j C_j^S$. The online traveling repairman problem (*OlTrp*) is the special case of* L-OlDarp *in which for each request σ_j source and destination coincide, i.e., $s_j = d_j$.*

An instance of the online dial-a-ride problem OlDarp consists of a metric space $M = (X, d) \in \mathcal{M}$ (see Section 3.1) with a distinguished origin $O \in X$ and a sequence $\sigma = \sigma_1, \ldots, \sigma_m$ of requests. Each request $\sigma_j = (t_j, s_j, d_j, w_j)$ specifies a ride, defined by two points: $s_j \in X$, the ride's source, and $d_j \in X$, its destination, and a weight $w_j \geq 0$. Each request σ_j is released at a non-negative time $t_j \geq 0$, its release time.

For $t \geq 0$ we denote by $\sigma_{\leq t}$ ($\sigma_{=t}$) the set of requests in σ released no later than time t (exactly at time t). A server is located at the origin $O \in X$ at time 0 and can move at most at unit speed. We consider the case in which the server can serve only one ride at a time, and in which preemption of rides is not allowed.

Given a sequence σ of requests, a feasible route for σ is a sequence of moves of the server such that the following conditions are satisfied: (a) The server starts in the origin O, (b) each ride requested in σ is served, and (c) the repairman does not start to serve any ride (in its source) before its release time. Let C_j^S denote the completion time of request σ_j on a feasible route S. The length of a route S, denoted by $l(S)$, is defined as the difference between the time when S is completed and its start time.

We denote by $\mathrm{ALG}(\sigma)$ the objective function value of the solution produced by an algorithm ALG on input σ. We use OPT to denote an optimization algorithm.

In [12] Feuerstein and Stougie presented a 9-competitive algorithm for the OlTrp on the real line and a 15-competitive algorithm for the L-OlDarp on the real line for the special case that the server has infinite capacity. In the same paper lower bounds of $1+\sqrt{2}$ and 3 on the competitive ratio of any deterministic algorithm for the OlTrp and the L-OlDarp, respectively, are proved.

The offline traveling repairman problem is known to be $\mathcal{NP}$-hard [1]. In the special case when the metric space is the real line, the Trp can be solved in polynomial time [1]. Recently, Sitters [49] showed that the Trp on weighted trees is $\mathcal{NP}$-hard. Approximation algorithms for the Trp have been studied in [17], [6], and [2].

We study competitive algorithms and randomized lower bounds for the OlTrp

and the L-OlDarp. Our algorithms improve the competitive ratios given in [12], and, moreover, the results are valid for any metric space and not just the real line. For the case of the L-OlDarp our algorithms are the first competitive algorithms. The randomized lower bounds are the first ones for the OlTrp and the L-OlDarp.

Our algorithms are adaptations of the GREEDY-INTERVAL algorithm for online-scheduling presented in [22], [21] and of the randomized version given in [8]. Our lower bound results are obtained by applying Yao's principle (see Section 3.3) in conjunction with a technique of Seiden [47]. An overview of the results is given in Tables 6.1 and 6.2.

Table 6.1: Deterministic upper and lower bounds for the OlTrp *and the* L-OlDarp.

	Deterministic UB	Previous best UB	Deterministic LB
OlTrp	$(1+\sqrt{2})^2 < 5.8285$ (Corollary 6.6)	9, see [12] (real line)	$1+\sqrt{2}$, see [12]
L-OlDarp	$(1+\sqrt{2})^2 < 5.8285$ (Theorem 6.5)	15, see [12] (real line, server cap ∞)	3, see [12]

Table 6.2: Randomized upper and lower bounds for the OlTrp *and the* L-OlDarp.

	Randomized UB	Randomized LB
OlTrp	$\frac{4}{\ln 3} < 3.6410$ (Corollary 6.10)	general: $\frac{7}{3}$ (Theorem 6.13) real line: 2 (Theorem 6.14)
L-OlDarp	$\frac{4}{\ln 3} < 3.6410$ (Theorem 6.9)	general: $\frac{4e-5}{2e-3} > 2.4104$ (Theorem 6.11) real line: $\frac{\ln 16+1}{\ln 16-1} > 2.1282$ (Theorem 6.12)

Section 6.2 gives the deterministic algorithm INTERVAL_α and the proof of its competitive ratio. In Section 6.3 we present the randomized algorithm $\text{RANDINTERVAL}_\alpha$ which achieves a better competitive ratio. In Section 6.4 we prove lower bounds on the competitive ratio of randomized algorithms against an oblivious adversary.

6.2 A deterministic algorithm

Our deterministic strategy is an adaption of the GREEDY-INTERVAL algorithm presented in [22], [21] for online scheduling. The proof of competitiveness borrows concepts of the proofs in [22], [21].

Algorithm INTERVAL$_\alpha$

Phase 0: Initialization:

Set L to be the earliest time that any request can be completed by OPT. We can assume that $L > 0$, since $L = 0$ means that there are requests released at time 0 with source and destination O which are served at no cost. For $i = 1, 2, \ldots$, define $B_i := \alpha^{i-1}L$, where $\alpha \in [1, 3]$ is fixed.

Phase i, for $i = 1, 2, \ldots$: At time B_i compute a transportation schedule S_i for the set of yet unserved requests released up to time B_i with the following properties:

(i) Schedule S_i starts at the origin O.

(ii) Schedule S_i ends at a point x_i with an empty server such that $d(O, x_i) \leq B_i$.

(iii) The length of schedule S_i, denoted by $l(S_i)$, satisfies $l(S_i) \leq B_i$.

(iv) The transportation schedule S_i maximizes the sum of the weights of requests served among all schedules satisfying (i)–(iii).

Starting at time βB_i, follow schedule S_i, where $\beta := \frac{2}{\alpha-1}$. As soon as schedule S_i is finished, return to the origin.

To justify the correctness of the algorithm first notice that $\beta \geq 1$ for any $\alpha \in (1, 3]$. Moreover, it holds that the algorithm can finish transportation schedule S_i computed at time B_i and return to the origin before time βB_{i+1}, the time when transportation schedule S_{i+1}, computed at time B_{i+1}, needs to be started: by requirements (ii) and (iii), following S_i and going back immediately to the origin can take at most $2B_i$ time units. Hence, starting at time βB_i, the algorithm is back at the origin latest at time $\beta B_i + 2B_i = (\frac{2}{\alpha-1} + 2)B_i = \frac{2\alpha}{\alpha-1}B_i = \beta B_{i+1}$.

Lemma 6.2. *Let R_i be the set of requests served by schedule S_i computed at time B_i, $i = 1, 2, \ldots$, and let R_i^* be the set of requests in the optimal offline solution which are completed in the time interval $(B_{i-1}, B_i]$. Then*

$$\sum_{i=1}^{k} w(R_i) \geq \sum_{i=1}^{k} w(R_i^*) \quad \text{for } k = 1, 2, \ldots \ .$$

PROOF. We first argue that for any $k \geq 1$ we can obtain from the optimal offline solution S^* a schedule $S^{(k)}$ which starts in the origin, has length at most B_k, ends with an empty server at a point with distance at most B_k from the origin, and which serves all requests in $\bigcup_{i=1}^{k} R_i^*$.

Consider the optimal offline transportation schedule S^*. Start at the origin and follow S^* for the first B_k time units, and serve the rides that can be picked up and delivered on S^* before time B_k. The rides on S^* that can be picked up but

not finished before time B_k are left unserved. Observe that this implies that the server is empty at the end of this schedule. We thereby obtain a schedule $S^{(k)}$ of length at most B_k which serves all requests in $\bigcup_{i=1}^{k} R_i^*$. Since the server moves at unit speed, it follows that $S^{(k)}$ ends at a point with distance at most B_k from the origin.

We now consider phase k and show that by the end of phase k, at least requests of weight $\sum_{i=1}^{k} w(R_i^*)$ have been scheduled by INTERVAL$_\alpha$. For each $k \geq 1$, the transportation schedule $S^{(k)}$ obtained as outlined above satisfies all conditions (i)–(iii) required by INTERVAL$_\alpha$.

Recall that schedule $S^{(k)}$ serves all requests in $\bigcup_{i=1}^{k} R_i^*$. Possibly, some of the requests from $\bigcup_{i=1}^{k} R_i^*$ have already been served by INTERVAL$_\alpha$ in previous phases. As omitting requests can never increase the length of a transportation schedule, in phase k, INTERVAL$_\alpha$ can schedule at least all requests from

$$\left(\bigcup_{i=1}^{k} R_i^*\right) \setminus \left(\bigcup_{i=1}^{k-1} R_i\right).$$

Consequently, the weight of all requests served in schedules $S_1, \ldots, S_k$ of INTERVAL$_\alpha$ is at least $w(\bigcup_{i=1}^{k} R_k^*) = \sum_{i=1}^{k} w(R_k^*)$ as claimed. □

The previous lemma gives us the following bound on the number of phases that INTERVAL$_\alpha$ uses to process a given input sequence σ.

Corollary 6.3. *Suppose that the optimum offline schedule is completed in the interval $(B_{p-1}, B_p]$ for some $p \geq 1$. Then the number of phases of the Algorithm* INTERVAL$_\alpha$ *is at most p. Schedule S_p computed at time B_p by* INTERVAL$_\alpha$ *is completed no later than time $(\beta+1)B_p$.*

Proof. By Lemma 6.2 the weight of all requests scheduled in the first p phases equals the total weight of all requests. Hence all requests must be scheduled within the first p phases. Since, by construction of INTERVAL$_\alpha$, schedule S_p computed in phase p is started at time βB_p and has length at most B_p, it completes by time $(\beta+1)B_p$. □

To prove competitiveness of INTERVAL$_\alpha$ we need an elementary lemma which can be proven by induction.

Lemma 6.4. *Let $a_i, b_i \in \mathbb{R}_0^+$ for $i = 1, \ldots, p$, for which*

$\sum_{i=1}^{p} a_i = \sum_{i=1}^{p} b_i$, *and*

$\sum_{i=1}^{p'} a_i \geq \sum_{i=1}^{p'} b_i$ *for all $1 \leq p' \leq p$.*

Then $\sum_{i=1}^{p} \tau_i a_i \leq \sum_{i=1}^{p} \tau_i b_i$ for any nondecreasing sequence $0 \leq \tau_1 \leq \cdots \leq \tau_p$.

Theorem 6.5. *Algorithm* INTERVAL$_\alpha$ *is* $\frac{\alpha(\alpha+1)}{\alpha-1}$*-competitive for the* L-OLDARP *for any* $\alpha \in (1,3]$*. For* $\alpha = 1+\sqrt{2}$*, this yields a competitive ratio of* $(1+\sqrt{2})^2 < 5.8285$*.*

PROOF. Let $\sigma = \sigma_1, \ldots, \sigma_m$ be any sequence of requests. By definition of INTERVAL$_\alpha$, each request served in schedule S_i completes no later than time $(\beta+1)B_i = (\frac{2}{\alpha-1}+1)B_i = \frac{\alpha+1}{\alpha-1}B_i$. Summing over all phases $1, \ldots, p$ yields

$$\text{INTERVAL}_\alpha(\sigma) \leq \frac{\alpha+1}{\alpha-1}\sum_{i=1}^{p} B_i w(R_i) = \alpha \cdot \frac{\alpha+1}{\alpha-1}\sum_{i=1}^{p} B_{i-1} w(R_i). \tag{6.1}$$

From Lemma 6.2 we know that $\sum_{i=1}^{k} w(R_i) \geq \sum_{i=1}^{k} w(R_i^*)$ for $k = 1, 2, \ldots,$ and from Corollary 6.3 we know that $\sum_{i=1}^{p} w(R_i) = \sum_{i=1}^{p} w(R_i^*)$. Therefore, application of Lemma 6.4 to the sequences $a_i := w(R_i)$ and $b_i := w(R_i^*)$ with the weighing sequence $\tau_i := B_{i-1}$, $i = 1, \ldots, p$, gives

$$\alpha \cdot \frac{\alpha+1}{\alpha-1}\sum_{i=1}^{p} B_{i-1} w(R_i) \leq \alpha \cdot \frac{\alpha+1}{\alpha-1}\sum_{i=1}^{p} B_{i-1} w(R_i^*). \tag{6.2}$$

Denote by C_j^* the completion time of request σ_j in the optimal offline solution OPT(σ). For each request σ_j denote by $(B_{\phi_j}, B_{\phi_{j+1}}]$ the interval that contains C_j^*. Then

$$\alpha \cdot \frac{\alpha+1}{\alpha-1}\sum_{i=1}^{p} B_{i-1} w(R_i^*) = \alpha \cdot \frac{\alpha+1}{\alpha-1}\sum_{j=1}^{m} B_{\phi_j} w_j \leq \alpha \cdot \frac{\alpha+1}{\alpha-1}\sum_{j=1}^{m} w_j C_j^*. \tag{6.3}$$

(6.1), (6.2), and (6.3) together yield

$$\text{INTERVAL}_\alpha(\sigma) \leq \alpha \cdot \frac{\alpha+1}{\alpha-1} \cdot \text{OPT}(\sigma).$$

The value $\alpha = 1+\sqrt{2}$ minimizes the function $f(\alpha) := \frac{\alpha(\alpha+1)}{\alpha-1}$ in the interval $(1,3]$, yielding $(1+\sqrt{2})^2 < 5.8285$ as competitive ratio. □

Corollary 6.6. *For* $\alpha = 1+\sqrt{2}$*, algorithm* INTERVAL$_\alpha$ *is* $(1+\sqrt{2})^2$*-competitive for the* OLTRP*.*

6.3 An improved randomized algorithm

In this section we use techniques developed in [8] to devise a randomized algorithm RANDINTERVAL$_\alpha$. At the beginning, RANDINTERVAL$_\alpha$ chooses a random number $\delta \in (0,1]$ according to the uniform distribution. From this moment on, the

algorithm is completely deterministic, working in the same way as the deterministic algorithm INTERVAL$_\alpha$ presented in the previous section. For $i \geq 1$ define $B'_i := \alpha^{i-1+\delta}L$, where again L is the earliest time that a request could be completed by OPT. As stated before in the case of INTERVAL$_\alpha$ we can assume that $L > 0$.

The difference between RANDINTERVAL$_\alpha$ and INTERVAL$_\alpha$ is that all phases are defined using $B'_i := \alpha^{i-1+\delta}L$ instead of $B_i := \alpha^{i-1}L$, $i \geq 1$. To justify the accuracy of RANDINTERVAL$_\alpha$, note that in the correctness proof for the deterministic version, we only made use of the fact that $B_{i+1} = \alpha B_i$ for $i \geq 0$. This also holds for the B'_i. Hence, any choice of the parameter $\alpha \in (1, 3]$ yields a correct version of RANDINTERVAL$_\alpha$. We will show later that the optimal choice for RANDINTERVAL$_\alpha$ is $\alpha = 3$.

The proof of Lemma 6.2 also holds also with B_i replaced by B'_i for each $i \geq 0$. We thus obtain the following lemma.

Lemma 6.7. *Let R_i be the set of requests scheduled in phase $i \geq 1$ of Algorithm* RANDINTERVAL$_\alpha$ *and denote by R^*_i the set of requests that are completed by* OPT *in the time interval $(B'_{i-1}, B'_i]$. Then*

$$\sum_{i=1}^{k} w(R_i) \geq \sum_{i=1}^{k} w(R^*_i) \quad \text{for } k = 1, 2, \ldots \ .$$

We can now use the proof of Theorem 6.5 with Lemma 6.2 replaced by Lemma 6.7. This enables us to conclude that for a sequence $\sigma = \sigma_1, \ldots, \sigma_m$ of requests the expected objective function value of RANDINTERVAL$_\alpha$ satisfies

$$\begin{aligned} \mathbb{E}\left[\text{RANDINTERVAL}_\alpha(\sigma)\right] &\leq \mathbb{E}\left[\alpha \cdot \frac{\alpha+1}{\alpha-1} \sum_{j=1}^{m} B'_{\phi_j} w_j\right] \\ &= \alpha \cdot \frac{\alpha+1}{\alpha-1} \sum_{j=1}^{m} w_j \mathbb{E}\left[B'_{\phi_j}\right], \end{aligned} \tag{6.4}$$

where $(B'_{\phi_j}, B'_{\phi_{j+1}}]$ is the interval containing the completion time C^*_j of request σ_j in the optimal solution OPT(σ).

To prove a bound on the performance of RANDINTERVAL$_\alpha$ we compute $\mathbb{E}\left[B'_{\phi_j}\right]$. Notice that B'_{ϕ_j} is the largest value $\alpha^{k+\delta}L$, $k \in \mathbb{Z}$, which is strictly smaller than C^*_j.

Lemma 6.8. *Let $z \geq L$ and $\delta \in (0, 1]$ be a random variable uniformly distributed on $(0, 1]$. Define B by $B := \max\{\, \alpha^{k+\delta}L : \alpha^{k+\delta}L < z \text{ and } k \in \mathbb{Z} \,\}$. Then, $\mathbb{E}[B] = \frac{\alpha-1}{\alpha \ln \alpha} \cdot z$.*

Proof. Suppose that $\alpha^k L \leq z < \alpha^{k+1}L$ for some $k \geq 0$. Observe that

$$B = \begin{cases} \alpha^{k-1+\delta}L, & \text{if } \delta \geq \log_\alpha \frac{z}{\alpha^k L}; \\ \alpha^{k+\delta}L, & \text{otherwise.} \end{cases}$$

Hence

$$\begin{aligned}\mathbb{E}[B] &= \int_0^{\log_\alpha \frac{z}{\alpha^k L}} \alpha^{k+\delta} L \, d\delta + \int_{\log_\alpha \frac{z}{\alpha^k L}}^1 \alpha^{k-1+\delta} L \, d\delta \\ &= \alpha^k L \left[\frac{1}{\ln \alpha} \alpha^\delta\right]_0^{\log_\alpha \frac{z}{\alpha^k L}} + \alpha^{k-1} L \left[\frac{1}{\ln \alpha} \alpha^\delta\right]_{\log_\alpha \frac{z}{\alpha^k L}}^1 = \frac{\alpha - 1}{\alpha \ln \alpha} \cdot z.\end{aligned}$$

This completes the proof. □

From Lemma 6.8 we can conclude that $\mathbb{E}\left[B'_{\phi_j}\right] = \frac{\alpha-1}{\alpha \ln \alpha} \cdot C_j^*$. Using this result in inequality (6.4), we obtain

$$\mathbb{E}[\text{RANDINTERVAL}_\alpha(\sigma)] \leq \alpha \cdot \frac{\alpha+1}{\alpha-1} \cdot \frac{\alpha-1}{\alpha \ln \alpha} \cdot \sum_{j=1}^m w_j C_j^* = \frac{\alpha+1}{\ln \alpha} \cdot \text{OPT}(\sigma).$$

Minimizing the function $g(\alpha) := \frac{\alpha+1}{\ln \alpha}$ over the interval $[(1, 3]$, we conclude that the best choice is $\alpha = 3$.

Theorem 6.9. *Algorithm* $\text{RANDINTERVAL}_\alpha$ *is* $\frac{\alpha+1}{\ln \alpha}$*-competitive for the* L-OlDarp *against an oblivious adversary, where* $\alpha \in (1, 3]$*. Choosing* $\alpha = 3$ *yields a competitive ratio of* $\frac{4}{\ln 3} < 3.6410$ *for* $\text{RANDINTERVAL}_\alpha$ *against an oblivious adversary.*

Corollary 6.10. *For* $\alpha = 3$*, algorithm* $\text{RANDINTERVAL}_\alpha$ *is* $\frac{4}{\ln 3}$*-competitive for the* OlTrp *against an oblivious adversary.*

6.4 Lower bounds

In this section we show lower bounds for the competitive ratio of any randomized algorithm against an oblivious adversary for the problem L-OlDarp. We use a method explained in [47] to compute a suitable distribution once our ground set of request sequences has been fixed.

A general lower bound for the L-OlDarp

We provide a general lower bound where the metric space is a star that consists of three rays of length 2 each. The center of the star is the origin, denoted by O. The server capacity equals one. Let the rays of the star be named A, B and C. Let x_A be the point on A with distance x to O, $0 < x \leq 2$. Points on B and C will be denoted in the same manner. Figure 6.1 contains an illustration of the star-shaped metric space.

Let $k \in \mathbb{N}$, $k \geq 2$ be arbitrary. At time 0 there is one request from O to 1_A with weight 1, denoted by $\sigma_1 = (0, O, 1_A, 1)$. With probability θ there are no further

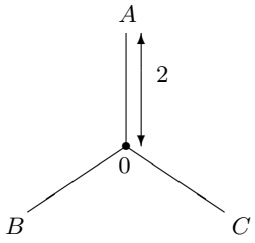

Figure 6.1: The star with three rays used in the lower bound construction.

requests. With probability $1-\theta$ there are k requests at time $2x$, where $x \in (0,1]$ is chosen according to a density function p. The density p satisfies $\theta + \int_0^1 p(x)\,dx = 1$ and will be determined suitably in the sequel. Each of the k requests released at time $2x$ has weight 1. With probability $\frac{1}{2}(1-\theta)$ these requests will be given from $2x_B$ to $2x_B$ and with probability $\frac{1}{2}(1-\theta)$ from $2x_C$ to $2x_C$. This yields a probability distribution X over the set $\Sigma = \{\, \sigma_{x,R} : x \in [0,1], R \in \{B,C\} \,\}$ of request sequences where $\sigma_{0,R} = \sigma_1$ for $R \in \{B,C\}$, and

$$\sigma_{x,R} = \sigma_1, \underbrace{(2x, 2x_R, 2x_R, 1), \ldots, (2x, 2x_R, 2x_R, 1)}_{k \text{ times}}$$

for $0 < x \leq 1$ and $R \in \{B,C\}$.

We first calculate the expected cost $\mathbb{E}_X[\mathrm{OPT}(\sigma_{x,R})]$ of an optimal offline algorithm with respect to the distribution X on Σ. With probability θ there is only request σ_1 to be served, and in this case the offline cost is 1. Now consider the situation where there are k additional requests at position $2x_B$ or $2x_C$. Clearly, the optimal offline cost is independent of the ray on which the requests arrive. First serving request σ_1 and then the k requests yields the objective function value $1 + k(2 + 2x)$, whereas first serving the set of k requests and then σ_1 results in a total cost of $2kx+4x+1$. Hence, for $k \geq 2$, we have

$$\mathbb{E}_X[\mathrm{OPT}(\sigma_{x,R})] = \theta + \int_0^1 (2kx + 4x + 1)p(x)\,dx. \tag{6.5}$$

The strategy of a generic deterministic online algorithm ALG_y can be cast into the following framework: ALG_y starts serving request σ_1 at time $2y$ where $y \geq 0$, unless further requests are released before time $2y$. If the sequence ends after σ_1, the online costs are $2y+1$. Otherwise, two cases have to be distinguished.

If $2x > 2y$, that is, the set of k requests is released after the time at which ALG_y starts the ride requested in σ_1, the algorithm must first finish σ_1 before it can serve the k requests. In this case, the cost of ALG_y is at least $2y+1+k(2y+2+2x)$.

If $2x \leq 2y$, the server of ALG_y has not yet started σ_1 and can serve the k requests before σ_1. To calculate the cost it incurs in this case, let l denote the distance of

ALG_y's server to the origin at time $2x$. Then $0 \le l \le y$, since otherwise the server cannot start σ_1 at time $2y$. We may assume that ALG_y is either on ray B or C, since moving onto ray A without serving σ_1 is clearly not advantageous.

Now, with probability $\frac{1}{2}$, ALG_y's server is on the "wrong" ray. This yields cost of at least $(2x+(l+2x))k+6x+l+1$ for ALG_y. Being on the "right" ray will cost $(2x+(2x-l))k+6x-l+1$. Putting this together, we get that for $y \le 1$

$$\begin{aligned}\mathbb{E}_X\left[\mathrm{ALG}_y(\sigma_{x,R})\right] \ge \theta(2y+1) &+ \int_y^1 (2y+1+k(2y+2+2x))\,p(x)\,dx \\ &+ \frac{1}{2}\int_0^y (4kx+6x+kl+l+1)p(x)\,dx \\ &+ \frac{1}{2}\int_0^y (4kx+6x-kl-l+1)p(x)\,dx.\end{aligned}$$

This results in

$$\begin{aligned}\mathbb{E}_X\left[\mathrm{ALG}_y(\sigma_{x,R})\right] \ge &\theta(2y+1) + \int_y^1 (2y+1+k(2y+2+2x))\,p(x)\,dx \\ &+ \int_0^y (4kx+6x+1)p(x)\,dx \\ &=: F(y).\end{aligned} \tag{6.6}$$

Observe that for $y \ge 1$ we have that

$$\mathbb{E}_X\left[\mathrm{ALG}_y(\sigma_{x,R})\right] \ge \theta(2y+1) + \int_0^1 (4kx+6x+1)p(x)\,dx \ge F(1).$$

Hence in what follows it suffices to consider the case $y \le 1$. To maximize the expected cost of any deterministic online algorithm on our randomized input sequence, we wish to choose θ and a density function p such that $\min_{y\in[0,1]} F(y)$ is maximized. We use the following heuristic approach (cf. [47]): Assume that θ and the density function p maximizing the minimum have the property that $F(y)$ is constant on $[0,1]$. Hence $F'(y)=0$ and $F''(y)=0$ for all $y \in (0,1)$. Differentiating we find that

$$\begin{aligned}F'(y) &= 2\left(\theta + (1+k)\int_y^1 p(x)\,dx - (k-2y)p(y)\right) \\ \text{and}\quad F''(y) &= -2(k-1)p(y) - 2(k-2y)p'(y).\end{aligned}$$

From the condition $F''(y)=0$ for all $y \in (0,1)$ we obtain the differential equation

$$-2(k-1)p(x) - 2(k-2x)p'(x) = 0,$$

which has the general solution

$$p(x) = \beta(k-2x)^{\frac{1}{2}(k-1)}. \tag{6.7}$$

The value of $\beta > 0$ is obtained from the initial condition $\theta + \int_0^1 p(x)\,dx = 1$ as

$$\beta = \frac{1-\theta}{\int_0^1 (k-2x)^{\frac{1}{2}(k-1)}\,dx} = \frac{(1+k)\ (\theta-1)}{(k-2)^{\frac{1+k}{2}} - k^{\frac{1+k}{2}}}. \tag{6.8}$$

It remains to determine θ. Recall that we attempted to choose θ and p in such a way that F is constant over the interval $[0,1]$. Hence in particular we must have $F(0) = F(1)$. Using

$$F(0) = \theta + \int_0^1 (1 + k(2+2x))p(x)\,dx$$
$$\text{and} \quad F(1) = 3\theta + \int_0^1 (4kx + 6x + 1)p(x)\,dx,$$

and substituting p and β from (6.7) and (6.8), respectively, we obtain

$$\theta = \frac{(k-2)^{\frac{1+k}{2}}(1+k)}{(k-2)^{\frac{1+k}{2}}k + k^{\frac{1+k}{2}}}.$$

We now use the distribution obtained this way in (6.6) and (6.5). This results in

$$\mathbb{E}_X\left[\mathrm{ALG}_y(\sigma_{x,R})\right] \geq \frac{(1+k)\left(-5(k-2)^{\frac{2+k}{2}}\sqrt{k} + \sqrt{k-2}\,k^{\frac{k}{2}}\,(3+4k)\right)}{\sqrt{k-2}\,(3+k)\left((k-2)^{\frac{1+k}{2}}\sqrt{k} + k^{\frac{k}{2}}\right)}$$

and

$$\mathbb{E}_X\left[\mathrm{OPT}(\sigma_{x,R})\right] = \frac{\sqrt{k-2}\,k^{\frac{1+k}{2}}(1+k)(3+2k) - (k-2)^{\frac{2+k}{2}}(1+k)(4+3k)}{\sqrt{k-2}\left((k-2)^{\frac{1+k}{2}}k(3+k) + k^{\frac{1+k}{2}}(3+k)\right)}.$$

Hence we conclude that

$$\frac{\mathbb{E}_X\left[\mathrm{ALG}_y(\sigma_{x,R})\right]}{\mathbb{E}_X\left[\mathrm{OPT}(\sigma_{x,R})\right]} \geq \frac{-5(k-2)^{1+\frac{k}{2}}\,k + \sqrt{k-2}\,k^{\frac{1+k}{2}}\,(3+4k)}{\sqrt{k-2}\,k^{\frac{1+k}{2}}(3+2k) - (k-2)^{\frac{2+k}{2}}\,(4+3k)}. \tag{6.9}$$

For $k \to \infty$, the right hand side of (6.9) converges to $\frac{4e-5}{2e-3}$. Hence by Yao's Principle (see Section 3.3) we obtain the following result.

Theorem 6.11. *Any randomized algorithm for the* L-OlDarp *has a competitive ratio greater or equal to* $\frac{4e-5}{2e-3} > 2.4104$ *against an oblivious adversary.*

A lower bound on the real line

The lower bound construction on the star uses the fact that the online server does not know on which ray to move if he wants to anticipate the arrival of the k requests

at time $2x$. If the metric space is the real line, there are only two rays, and this argument is no longer valid. The server can move towards the point $2x$ (of course still at the risk that there will be no requests at all on this ray) in anticipation of the set of k requests. Essentially the same construction therefore leads to a slightly worse lower bound.

Theorem 6.12. *Any randomized algorithm for the* L-OlDarp *on the real line has competitive ratio greater or equal to* $\frac{\ln 16+1}{\ln 16-1} > 2.1282$ *against an oblivious adversary.*

Proof. We use the following request set: At time 0 there is one request $\sigma_1 = (0, 0, -1, 1)$. With probability θ there are no further requests.

With probability $1-\theta$ there are k requests with weight 1 at time $2x$ from $2x$ to $2x$ where $x \in (0, 1]$ is chosen according to the density function $p(x)$, where $\theta + \int_0^1 p(x)\,dx = 1$. Again, this yields a probability distribution X over a set $\Sigma = \{\, \sigma_x : x \in [0, 1] \,\}$ of request sequences where

$$\sigma_x = \begin{cases} \sigma_1 & \text{for } x = 0 \\ \sigma_1, \underbrace{(2x, 2x, 2x, 1), \ldots, (2x, 2x, 2x, 1)}_{k \text{ times}} & \text{for } 0 < x \le 1. \end{cases}$$

By a similar argumentation as in the previous section we obtain

$$\mathbb{E}_X\left[\mathrm{OPT}(\sigma_x)\right] = \theta + \int_0^1 (2kx + 4x + 1)p(x)\,dx$$

and

$$\mathbb{E}_X\left[\mathrm{ALG}_y(\sigma_x)\right] \ge \theta(2y+1) + \int_0^y (2kx + 4x + 1)p(x)\,dx + \int_y^1 (2y + 1 + k(2y + 2 + 2x))\,p(x)\,dx.$$

By choosing

$$p(x) = \beta(k - x + k\,x)^{-\frac{2\,k}{-1+k}},$$

$$\beta = \frac{1-\theta}{\int_0^1 (k - x + kx)^{-\frac{2\,k}{-1+k}}\,dx} = \frac{(1+k)\;(1-\theta)}{k^{\frac{1+k}{1-k}} - (-1+2\,k)^{\frac{1+k}{1-k}}},$$

$$\text{and} \quad \theta = \frac{-1+k+2\,k^2}{k\left(-1+2\,k+k^{-\frac{2\,k}{-1+k}}(-1+2\,k)^{\frac{2\,k}{-1+k}}\right)},$$

we obtain

$$\frac{\mathbb{E}_X\left[\mathrm{ALG}_y(\sigma_x)\right]}{\mathbb{E}_X\left[\mathrm{OPT}(\sigma_x)\right]} \ge 1 + \frac{2}{(1+k)\left(-2 + k^{\frac{1+k}{1-k}}\,(-1+2\,k)^{\frac{1+k}{-1+k}}\right)}. \tag{6.10}$$

For $k \to \infty$, the right hand side of (6.10) converges to $\frac{\ln 16+1}{\ln 16-1}$. The theorem now follows from Yao's Principle (see Section 3.3). □

Lower bounds for the OlTrp

For the OlTrp we provide a general lower bound, again using a star with three rays of length 2 as a metric space.

Theorem 6.13. *Any randomized algorithm for the* OlTrp *has competitive ratio greater or equal to* $7/3$ *against an oblivious adversary.*

Proof. Let $k \in \mathbb{N}$ be arbitrary. With probability $\theta = \frac{k+1}{k+2}$, the input sequence consists of only one request σ_1 at distance 1 from the origin, released at time 1, and with weight 1. The ray on which this request occurs is chosen uniformly at random among the three rays.

With probability $1-\theta$ there will be an additional series of k requests at distance 2 from the origin, each with weight equal to 1. These requests are released at time 2, and the ray on which they occur is chosen uniformly among the two rays that do not contain σ_1.

The cost for the adversary is given by

$$\mathbb{E}_X\left[\mathrm{OPT}(\sigma_x)\right] = \theta + (1-\theta)(2k+5) = \frac{3k+6}{k+2}.$$

It is easy to show that no online algorithm can do better than one whose server is in the origin at time 1 and, at time 2, at distance $0 \leq y \leq 1$ from the origin on the ray where σ_1 is located. Using this fact, we get

$$\mathbb{E}_X\left[\mathrm{ALG}_y(\sigma_x)\right] \geq \theta(3-y) + (1-\theta)((4+y)k+7+y) = \frac{7k+10}{k+2}.$$

This leads to

$$\frac{\mathbb{E}_X\left[\mathrm{ALG}_y(\sigma_x)\right]}{\mathbb{E}_X\left[\mathrm{OPT}(\sigma_x)\right]} \geq \frac{7k+10}{3k+6}. \tag{6.11}$$

By letting $k \to \infty$ and applying Yao's Principle once more, the theorem follows. □

On the real line, a lower bound is obtained by the following very simple randomized request sequence. With probability $1/2$ we give a request at time 1 in -1, and with probability $1/2$ we give a request at time 1 in $+1$.

Theorem 6.14. *Any randomized algorithm for the* OlTrp *on the real line has competitive ratio greater or equal to* 2 *against an oblivious adversary.*

7

Online dial-a-ride problems minimizing maximum flow time

The content of this chapter is joint work with L. Laura, M. Lipmann, A. Marchetti-Spaccamela, S.O. Krumke, D. Poensgen, and L. Stougie, and has appeared in [37].

7.1 Introduction

An instance of the *online traveling salesman problem minimizing the maximum flow time* ($F_{\max}$-OLTSP) consists of a metric space $M = (X, d) \in \mathcal{M}$ (see Section 3.1) with a distinguished origin $O \in X$ and a sequence $\sigma = \sigma_1, \ldots, \sigma_m$ of requests. A server is located at the origin O at time 0 and can move at most at unit speed. In this chapter we are concerned with the special case that M is $\mathbb{R}$, the real line. The origin O equals the point 0.

Each *request* is a pair $\sigma_i = (t_i, x_i)$, where $t_i \in \mathbb{R}^+$ is the time at which request σ_i is revealed, and $x_i \in X$ is the point in the metric space to be visited. We assume that the sequence $\sigma = \sigma_1, \ldots, \sigma_m$ of requests is given in order of non-decreasing release times. As before, for a real number t we denote by $\sigma_{\leq t}$ the subsequence of requests in σ released up to time t. Similarly, $\sigma_{<t}$ is the subsequence of σ consisting of those requests with release time strictly smaller than t.

An online algorithm learns from the existence of request σ_j only at its release time t_j. In particular, it has neither information about the release time of the last request nor about the total number of requests. Hence, at any moment in time t, an online algorithm must make its decisions only knowing the requests in $\sigma_{\leq t}$.

Given a sequence σ of requests, a feasible route for σ is a route for the server that starts in the origin O and serves all requested points, where each request is served not earlier than the time it is released.

Let C_j^{ALG} denote the completion time of request σ_j by the server of algorithm ALG, and let $F_j^{\mathrm{ALG}} = C_j^{\mathrm{ALG}} - t_j$ be the corresponding *flow time* of σ_j.

We denote by $\mathrm{ALG}(\sigma)$ the maximum flow time of the solution produced by an algorithm ALG on input σ. We use OPT to denote an optimization algorithm.

The online traveling salesman problem has been studied in the literature for the objectives of minimizing the makespan (see [4], [3], and [12]), the weighted sum of completion times (see [12], and Chapter 6), and the maximum and average flow time (see [23]). For applications, the maximum flow time is of particular interest. For instance, it can be identified with the maximal dissatisfaction of customers. Unfortunately, there can be no competitive algorithm, neither deterministic nor randomized [23]. Moreover, in contrast to scheduling [46], resource augmentation, e.g. providing the online algorithm with a faster server, does not help. The crucial difference with scheduling is that in the $F_{\max}$-OLTSP the server moves in space.

The only hope to overcome the weaknesses of standard competitive analysis in the context of the $F_{\max}$-OLTSP is to restrict the power of the adversary. In Chapter 4 we introduced the concept of a *fair adversary*, who is in a natural way restricted in the context of the OLTSP. A fair adversary always keeps his server within the convex hull of the requests released so far. In Chapter 4 we showed that against a fair adversary lower competitive ratios can be obtained when minimizing the makespan. However, against a fair adversary, still a constant competitive ratio for the $F_{\max}$-OLTSP is out of reach (see Theorem 7.2).

In this chapter we refine the concept of a fair adversary for the $F_{\max}$-OLTSP on the real line. A *non-abusive adversary* may only move his server in a direction of a released but yet unserved requests. This reasonable restriction on the adversary strongly improves the situation. Our main result is an online algorithm with a constant competitive ratio against a non-abusive adversary for the $F_{\max}$-OLTSP in $\mathbb{R}$.

Definition 7.1. [**Non-abusive adversary**]
An adversary for the OLTSP *on* $\mathbb{R}$ *is* non-abusive*, if at any moment in time his server is idle, or moving in the direction of a request that has been released but not yet served.*

In the sequel we slightly abuse notation and denote by $\mathrm{OPT}(\sigma)$ the maximal flow time in an optimal *non-abusive* solution for the sequence σ.

7.2 Lower bounds

In [23] the authors show that there can be no competitive algorithm for the $F_{\max}$-OLTSP, neither deterministic nor randomized. The following theorem shows that even against a fair adversary (see Chapter 4) no constant competitive ratio can be achieved.

Theorem 7.2. *No randomized algorithm for the $F_{\max}$-OLTSP on $\mathbb{R}$ can achieve a constant competitive ratio against an oblivious fair adversary.*

PROOF. Let $\varepsilon > 0$ and $k \in \mathbb{N}$ be arbitrary, but fixed. We give one of the two request sequences $\sigma_1 := (\varepsilon, \varepsilon), (2\varepsilon, 2\varepsilon), \ldots, (k\varepsilon, k\varepsilon), (T, O)$ and $\sigma_1 := (\varepsilon, \varepsilon), (2\varepsilon, 2\varepsilon), \ldots, (k\varepsilon, k\varepsilon), (T, k\varepsilon)$, each with probability $1/2$, where $T = 4k\varepsilon$.

The expected cost of an optimal fair offline solution is at most ε, while any deterministic online algorithm has cost at least $k\varepsilon/2$. The claim now follows by applying Yao's Principle (see Section 3.3). □

The following result shows that the $F_{\max}$-OLTSP is still non-trivial against a non-abusive adversary.

Theorem 7.3. *No deterministic algorithm for the $F_{\max}$-OLTSP on $\mathbb{R}$ can achieve a competitive ratio less than 2 against a non-abusive adversary.*

PROOF. Let ALG be any deterministic online algorithm. The adversary first presents the following $2m$ requests: $(0, \pm 1), (3, \pm 2), \ldots, (\sum_{k=1}^{m-1}(1 + 2k), \pm m)$. W.l.o.g., let ALG serve the request in $-m$ later than the one in $+m$, and let T be the time it reaches $-m$. Clearly, $T \geq \sum_{k=1}^{m}(1 + 2k)$. At time T, the adversary presents one more request in $+(3m+1)$ which results in a flow time of at least $4m+1$ for ALG. On the other hand, a non-abusive offline algorithm can serve all of the first $2m$ requests with maximum flow time $2m + 1$ by time $\sum_{k=1}^{m}(1 + 2k)$, ending with the request at $+m$. From there it can easily reach the last request with flow time $2m + 1$. The theorem follows by letting $m \to \infty$. □

7.3 The algorithm DETOUR

We denote by $p^{\mathrm{D}}(t)$ the position of the server operated by DETOUR at time t. The terms *ahead of* and *in the back of* the server refer in an obvious way to positions on the line with respect to the direction in which the server is currently moving.

Given a point $p \in \mathbb{R}$, we call request $\sigma_i = (t_i, x_i)$ *more critical* than request $\sigma_j = (t_j, x_j)$ w.r.t. p if both x_i and x_j are on the same side of p, and $d(p, x_i) - t_i > d(p, x_j) - t_j$. If at time t, request σ_i is more critical than σ_j w.r.t. DETOUR's position $p^{\mathrm{D}}(t)$, then $F_i^{\mathrm{DTO}} > F_j^{\mathrm{DTO}}$. Moreover, σ_i remains more critical than σ_j after time t as long as both requests are unserved. Conversely, we have the following observation.

Observation 7.4. *If, at time t, request σ_i is more critical than σ_j w.r.t. $p^D(t)$, and* DETOUR *moves straight ahead to the more distant request after having served the one closer to $p^D(t)$, then* DETOUR*'s flow time for σ_j is at most the flow time it achieves for σ_i.*

The *critical region* $\vee(\sigma_j, p, G)$ w.r.t. a request σ_j, a point $p \in \mathbb{R}$, and a bound G, contains all those pairs $(t, x) \in \mathbb{R}_+ \times \mathbb{R}$ with the property that (i) (t, x) is more critical than σ_j w.r.t. p, and (ii) $t + d(x, x_j) - t_j \leq G$.

In the setting of DETOUR, p will be the position of the online server at a certain time t'. Condition (ii) has the meaning that a request in (t, x) could be served before σ_j in an offline tour with maximal flow time at most G.

DETOUR's decisions at time t are based on an approximation, called the *guess* $G(t)$, of $\mathrm{OPT}(\sigma_{\leq t})$. Roughly, DETOUR's strategy is to serve all requests in a first-come-first-serve (FCFS) manner. However, blindly following an FCFS-scheme makes it easy for the adversary to fool the algorithm. DETOUR enforces the offline cost in a malicious sequence to increase by making a detour on his way to the next "target", the request that he wants to serve first: he first moves his server in the "wrong direction" as long as he can serve the target with flow time thrice the guess. If the guess changes, the detour, and possibly the target, are adjusted accordingly (this requires some technicalities in the description of the algorithm).

Our algorithm DETOUR can assume three modes:

idle In this mode, DETOUR's server has served all unserved requests, and is waiting for new requests at the point at which he served the last request.

focus Here, the server is moving in one direction serving requests until a request in his back becomes the oldest unserved one or all requests have been served.

detour In this case, the server is moving away from his current target (possibly serving requests on the way), thus making a "detour".

At any time, at most one unserved request is marked as a *target* by DETOUR. Moreover, DETOUR keeps at most one critical region, denoted by $\vee$. Before we formalize the behavior of DETOUR we specify important building blocks for the algorithm.

- *Guess Update:* Given the current time t, replace the current guess value G by G', defined as follows: If $G = 0$, then $G' := \mathrm{OPT}(\sigma_{\leq t})$. If $G > 0$, then $G' := 2^a G$, where a is the smallest integer k such that $\mathrm{OPT}(\sigma_{\leq t}) \leq 2^k G$.

- *Target Selection:* Given a candidate set C, that is determined by the algorithm before, and the current time t, let $s_0 = (t_0, x_0)$ be the most critical request from C w.r.t. $p^{\mathrm{D}}(t)$ with the property that s_0 is *feasible* in the following sense:

  Let X_0 be the point ahead of the server such that $t + d(p^{\mathrm{D}}(t), X_0) + d(X_0, x_0) = t_0 + 3G$, provided such a point exists, otherwise let $X_0 := p^{\mathrm{D}}(t)$. Define the *turning point* $\mathrm{TP}_0 = (T_0, X_0)$, where $T_0 := t + d(p^{\mathrm{D}}(t), X_0)$. There is no unserved request ahead of the server further away from $p^{\mathrm{D}}(t)$ than TP_0 and older than s_0.

  If necessary, unmark the current target and turning point. Mark s_0 as a target and set TP_0 to be the current turning point.

- *Mode Selection:* If $\mathrm{TP}_0 \neq p^{\mathrm{D}}(t)$, then set $\vee := \vee(s_0, p^{\mathrm{D}}(t), G)$ and enter the detour mode. Otherwise, set $\vee := \emptyset$ and unmark s_0 as a target. Change the direction and enter the focus mode.

We now specify for each of the three states how DETOUR reacts to possible events. All events not mentioned are ignored. In the beginning, $G := 0$, $\vee := \emptyset$ and the algorithm is in the idle mode.

Idle Mode In the idle mode DETOUR waits for the next request to arrive.

- A new request is released at time t.

  The pair $(t, p^{\mathrm{D}}(t))$ is called a *selection point*. DETOUR performs a *guess update*. The direction of the server is defined to be such that the oldest unserved request is in his back. In case that there is one oldest request on both sides of the server chooses to have the most critical one in the server's back.

  DETOUR defines C to be the set of unserved requests in the back of the server and performs a *target selection*, followed by a *mode selection*.

Detour Mode In this mode, DETOUR has a current target $s_m = (t_m, x_m)$, a critical region $\vee \neq \emptyset$ and a turning point TP. Let T be the time DETOUR entered the detour mode and s_0 the then chosen target. The server moves towards TP until one of following two events happens, where the former one has a higher priority than the later one (i.e., the former event is always checked for first).

- A new request is released at time t or an already existing request has just been served.

  DETOUR performs a *guess update*, and enlarges the critical region to $\vee := \vee(s_0, p^{\mathrm{D}}(T), G)$ where G is the updated guess value. Replace the old turning point by a new point TP which satisfies $t + d(p^{\mathrm{D}}(t), \mathrm{TP}) + d(\mathrm{TP}, x_m) = t_m + 3G$ for the updated guess value G.

  DETOUR defines C to be the set of unserved requests which are in $\vee$ and more critical than the current target s_m. If $C \neq \emptyset$, it executes the *target selection* and the *mode selection*. If $C = \emptyset$, DETOUR remains in the detour mode.

- The turning point TP is reached at time t.

  DETOUR unmarks the current target, sets $\vee := \emptyset$ and clears the turning point. The server reverses direction and enters the focus mode.

Focus Mode When entering the focus mode, DETOUR's server has a direction. He moves in this direction, reacting to the following events:

- A new request is released at time t.

  A *guess update* is performed, and the server remains in the focus mode.

- The last unserved request has been served.

  The server stops, forgets his direction and enters the idle mode.

- A request in the back of the server becomes the oldest unserved request.

  If this happens at time t, the pair $(t, p^{\mathrm{D}}(t))$ is also called a *selection point*. DETOUR defines C to be the set of unserved requests in the back of the server and performs a *target selection*, followed by a *mode selection*, exactly as in the idle mode.

7.4 Analysis of DETOUR

For the analysis of DETOUR it is convenient to compare intermediate solutions $\mathrm{DETOUR}(\sigma_{\leq t})$ not only to the optimal non-abusive solution on $\sigma_{\leq t}$, but to a whole class of offline solutions $\mathcal{ADV}(t)$ which depend on the guess value $G(t)$ maintained by DETOUR. Notice that for any time t we have $\mathrm{OPT}(\sigma_{\leq t}) \leq G(t) \leq 2\mathrm{OPT}(\sigma_{\leq t})$.

Definition 7.5. *By $\mathcal{ADV}(t)$, we denote the set of all non-abusive offline solutions for the sequence $\sigma_{\leq t}$ with the property that the maximum flow time of any request in $\sigma_{\leq t}$ is bounded from above by $G(t)$.*

For $\sigma_i \in \sigma_{\leq t}$, the smallest achievable flow time *$\alpha_i(t)$ is defined to be minimum flow time of σ_i taken over all solutions in $\mathcal{ADV}(t)$.*

By definition, $\alpha_i(t) \leq G(t)$. Possibly, $\alpha_i(t) \geq \alpha_i(t')$ for $t' > t$, as an increase in the allowed flow time can help an adversary to serve a request σ_i earlier. However, we make the following observation.

Observation 7.6. *If $t \leq t'$ and $G(t) = G(t')$, then $\alpha_i(t) \leq \alpha_i(t')$ for any request $\sigma_i = (t_i, x_i)$ with $t_i \leq t$.*

Definition 7.7. [**Served in time**]
Given $\sigma_i = (t_i, x_i)$, we define $\tilde{T}_i := \max\{t_i, T\}$, where T is the last time DETOUR *reverses direction before serving σ_i. We say that σ_i is* served in time *if $C_i^{\mathrm{DTO}} \leq t_i + 3G(\tilde{T}_i) + \alpha_i(\tilde{T}_i)$.*

Notice that any request σ_i served in time is served with a flow time of at most $4G(\tilde{T}_i) \leq 4G(C_i^{\mathrm{DTO}})$.

Lemma 7.8. *Let $\sigma_i = (t_i, x_i)$ and $\sigma_j = (t_j, x_j)$ be two requests such that:*

(i) *σ_i is served in time,*

(ii) $C_j^{\mathrm{DTO}} \leq C_i^{\mathrm{DTO}} + d(x_i, x_j)$,

(iii) $\tilde{T}_i \leq \tilde{T}_j$,

(iv) *σ_j is served after σ_i by all* ADV $\in \mathcal{ADV}(\tilde{T}_j)$.

Then, σ_j is served in time.

PROOF. By (iv) we have

$$t_j + \alpha_j(\tilde{T}_j) \geq t_i + \alpha_i(\tilde{T}_j) + d(x_i, x_j). \tag{7.1}$$

In particular, this means that

$$t_i + d(x_i, x_j) \leq t_j + G(\tilde{T}_j). \tag{7.2}$$

By (i), (ii), and the definition of in time, we get:

$$C_j^{\mathrm{DTO}} \leq t_i + \alpha_i(\tilde{T}_i) + 3G(\tilde{T}_i) + d(x_i, x_j). \tag{7.3}$$

If $G(\tilde{T}_i) = G(\tilde{T}_j)$, we have that $\alpha_i(\tilde{T}_i) \leq \alpha_i(\tilde{T}_j)$ by Observation 7.6. In this case, inequality (7.3) yields

$$\begin{aligned} C_j^{\mathrm{DTO}} &\leq t_i + \alpha_i(\tilde{T}_j) + 3G(\tilde{T}_j) + d(x_i, x_j) \\ &\leq t_j + \alpha_j(\tilde{T}_j) + 3G(\tilde{T}_j) && \text{by (7.1).} \end{aligned}$$

If $G(\tilde{T}_i) < G(\tilde{T}_j)$, it must hold that $2G(\tilde{T}_i) \leq G(\tilde{T}_j)$, and inequality (7.3) yields

$$\begin{aligned} C_j^{\mathrm{DTO}} &\leq t_i + 4G(\tilde{T}_i) + d(x_i, x_j) \\ &\leq t_j + G(\tilde{T}_j) + 4G(\tilde{T}_i) && \text{by (7.2)} \\ &\leq t_j + 3G(\tilde{T}_j). \end{aligned}$$

□

A useful observation is that assumption (iv) of Lemma 7.8 is ensured if $t_j + d(x_j, x_i) > t_i + G(t)$. This implies the following observation which will be used frequently in order to apply Lemma 7.8:

Observation 7.9. 1 *If $d(x_i, x_j) > G(t)$ and $t_i \leq t_j \leq t$, then $C_i^{\mathrm{ADV}} \leq C_j^{\mathrm{ADV}}$ for all* ADV $\in \mathcal{ADV}(t)$.

2 *If a request σ_j is outside the critical region $\vee(\sigma_i, p, G(t))$ valid at time t, then $C_i^{\mathrm{ADV}} \leq C_j^{\mathrm{ADV}}$ for all* ADV $\in \mathcal{ADV}(t)$.

We define a *busy period* to be the period between the moment in time when DETOUR leaves the idle mode and the next time the idle mode is entered again.

Lemma 7.10. *Suppose that at the beginning of a busy period at time t each request σ_j served in one of the preceeding busy periods was served by* DETOUR *with a flow time at most $4\,G(C_j^{\mathrm{DTO}})$. Then, $d(p^D(t), p^{\mathrm{ADV}}(t)) \leq 5/2G(t)$ for any* ADV $\in \mathcal{ADV}(t)$.

PROOF. The claim of the lemma is trivially true for the first busy period, since a non-abusive adversary must keep his server in the origin until the first request is released. Hence, it suffices to consider the later busy periods.

Let σ_l be last request served by DETOUR in the preceeding busy period, so DETOUR enters the idle mode at time C_l^{DTO}. Consider that ADV $\in \mathcal{ADV}(t)$ in which the adversary's server is furthest away from $p^{\mathrm{D}}(t) = x_l$.

Case 1: At time t, ADV has served all requests in $\sigma_{<t}$.

Since ADV is non-abusive, his server satisfies $p^{\mathrm{ADV}}(t) = x_k$ for the request σ_k he served last. DETOUR must have served σ_k in the preceeding busy period, hence no later than σ_l. Thus, $C_k^{\mathrm{DTO}} \leq C_l^{\mathrm{DTO}} - d(x_k, x_l)$, and

$$t_k \leq C_l^{\mathrm{DTO}} - d(x_k, x_l) \leq t_l + 4G(C_l^{\mathrm{DTO}}) - d(x_k, x_l),$$

because σ_l was served with a flow time of at most $4G(C_l^{\mathrm{DTO}})$. On the other hand, ADV serves σ_l no later than σ_k, which implies that $t_l + d(x_k, x_l) \leq t_k + G(C_l^{\mathrm{DTO}})$. The two inequalities together yield

$$d(x_k, x_l) \leq t_k - t_l + G(C_l^{\mathrm{DTO}}) \leq 5G(C_l^{\mathrm{DTO}}) - d(x_k, x_l).$$

Hence $d(p^{\mathrm{D}}(t), p^{\mathrm{ADV}}(t)) = d(x_k, x_l) \leq 5/2G(C_l^{\mathrm{DTO}})$.

Case 2: At time t, there is a request from $\sigma_{<t}$ which has not been served yet by ADV.

If σ_l has not been served by ADV at time t, then the distance $d(p^{\mathrm{ADV}}(t), x_l) = d(p^{\mathrm{ADV}}(t), p^{\mathrm{D}}(t))$ is at most $G(t)$, because otherwise the adversary's flow time for σ_l would be greater than $G(t)$.

Otherwise, σ_l has been served, but another request in $\sigma_{<t}$ is yet unserved by ADV. Let σ_k be the request in $\sigma_{<t}$ which is furthest away from σ_l and yet unserved by ADV. The same reasoning as in Case 1 shows that $d(x_k, x_l) \leq \frac{5}{2}G(C_l^{\mathrm{DTO}})$. So, if the adversary's server is between σ_k and σ_l, the claim holds true. Assume that the adversary's server is further away from σ_l than σ_k. Since the adversary is non-abusive, there must be a request σ_j even further away from σ_l than $p^{\mathrm{ADV}}(t)$, which ADV served last before (or at) time t. In particular, ADV served σ_l before σ_j. Thus, the same arguments as in Case 1 apply to σ_j in place of σ_k, showing that $d(x_j, x_l) \leq \frac{5}{2}G(C_l^{\mathrm{DTO}})$. □

We further subdivide each busy period into *phases*, where a phase is defined to be the time between two subsequent selection points of DETOUR. Remember that DETOUR reaches a selection point whenever it leaves the idle mode, and each time at which a request in the server's back becomes the oldest unserved one. The following statement is the key theorem of our analysis.

Theorem 7.11. *The following is true for any phase $\rho \geq 1$:*

(a) *At any time t in phase ρ at which* DETOUR *is in the detour mode, it holds that $d(X_i, x_i) \geq G(t)$ for the turning point $TP_i = (X_i, T_i)$ valid at that time and its corresponding target $\sigma_i = (t_i, x_i)$. Moreover, if at some time t during phase ρ, a request σ_j failed to become a new target only because it was infeasible, the above inequality holds as well for σ_j and its hypothetical turning point TP_j.*

(b) *Any request σ_j served in phase ρ is served with a flow time of at most $4G(C_j^{\mathrm{DTO}})$.*

(c) *The last request served in phase ρ is served in time.*

Before we give the proof of Theorem 7.11, we show how this theorem implies 8-competitiveness of algorithm DETOUR.

Theorem 7.12. DETOUR *is 8-competitive against a non-abusive adversary for the $F_{\max}$-*OLTSP.

PROOF. By Theorem 7.11 we have that any request σ_i is served with flow time at most $4G(C_i^{\mathrm{DTO}})$. If $C_{\mathrm{last}}^{\mathrm{DTO}}$ is the time at which the last request is served by DETOUR, all requests are served with flow time at most $4G(C_{\mathrm{last}}^{\mathrm{DTO}})$, which, by construction, is bounded by $2\mathrm{OPT}(\sigma)$, hence the claim. □

PROOF OF THEOREM 7.11. The proof is divided into a proof for Statement (a) and a proof for Statement (b) and (c). We prove the statement by induction on the total number of phases. In the inductive step we distinguish whether phase ρ is the first phase of a busy period or not. The former case includes the induction base ($\rho = 1$), i.e., the first phase of the first busy period, as a special case. Since the proof for the induction base is similar to the proof of the induction step, we choose not to separate them. However, proper understanding of the proof requires first reading the proof for the base case of Statement (a), then for the base case of Statements (b) and (c), and afterwards for the induction step of all three statements.

We call the turning point $\mathrm{TP}^\rho = (T^\rho, X^\rho)$ at which the server actually reverses direction the *realized turning point* of phase ρ. Note that both the realized turning point TP^ρ and its corresponding target $s^\rho = (t^\rho, x^\rho)$ are reached by DETOUR in the same phase. Throughout the whole proof we assume without loss of generality that the realized turning point is to the right of the final target, that is, in phase ρ, DETOUR moves to the right while in the detour mode and to the left after entering the focus mode. We also assume without loss of generality that at time 0 a request appears in the origin since this request does not increase the offline cost.

Proof of Statement (a):

Let $\mathrm{TP}_0 = (T_0, X_0)$ be the first turning point chosen in phase ρ, TP_1 the next one, etc. until TP^ρ, the realized turning point of the phase. Let $s_i = (t_i, x_i)$ be the target corresponding to TP_i (s_i is released at time t_i). Part (a) is proven by induction on

the number of turning points in the considered phase ρ.

1. Phase ρ is the first phase of a busy period.

The first target $s_0 = (t_0, x_0)$ must be among the requests whose release initiates the start of the busy period at time t_0, and $\mathrm{TP}_0 = (T_0, X_0)$ is chosen such that $t_0 + d(p^{\mathrm{D}}(t_0), X_0) + d(x_0, X_0) \geq t_0 + 3G(t_0)$. Since $d(p^{\mathrm{D}}(t_0), X_0) < d(x_0, X_0)$, it follows that $d(X_0, x_0) > \frac{3}{2} G(t_0)$.

Assume that (a) holds for the turning points $\mathrm{TP}_0, \ldots, \mathrm{TP}_{i-1}$ of phase ρ. Consider $\mathrm{TP}_i = (T_i, X_i)$ replacing TP_{i-1} at time t of phase 1, and let $s_i = (t_i, x_i)$ be TP_i's corresponding target.

If s_i was released at time t_0, then TP_i is exactly the same as if the guess value at time t_0 had already been $G(t)$ and s_i had been selected as a target at time t_0. As before we can conclude that $d(X_i, x_i) \geq \frac{3}{2} G(t)$.

If s_i was released later than t_0, the detour taken by DETOUR is only longer as the one chosen if s_i had been released already at time t_0. Hence, the arguments above apply again and $d(X_i, x_i) \geq \frac{3}{2} G(t)$.

2. Phase ρ is not the first phase of a busy period.

First consider $\mathrm{TP}_0 = (T_0, X_0)$, the turning point planned first in phase ρ. Both s_0 and T_0 are determined in the selection point SP which marks the end of phase $\rho-1$ and the start of phase ρ. It must hold that $\mathrm{SP} = (C_l^{\mathrm{DTO}}, x_l)$ for some request σ_l. When DETOUR serves request σ_l, the oldest unserved request, call it σ_z, is in his back. Observe that SP cannot be reached before the final target $s^{\rho-1}$ of phase $\rho-1$ is served, otherwise there would be an unserved request in the back of DETOUR's server before $s^{\rho-1}$ is reached which is older than $s^{\rho-1}$, and $s^{\rho-1}$ would have been infeasible at the time it became a target, which is a contradiction.

Assume first that s_0 is located between $X^{\rho-1}$ and x_l. Then

$$t_0 \geq C_l^{\mathrm{DTO}} - d(x_0, x_l), \tag{7.4}$$

because otherwise s_0 would have been served on the way to σ_l. Since TP_0 is chosen at time C_l^{DTO} such that s_0 is not served before $t_0 + 3G(C_l^{\mathrm{DTO}})$, we have

$$C_l^{\mathrm{DTO}} + d(x_l, X_0) + d(X_0, x_0) \geq t_0 + 3G(C_l^{\mathrm{DTO}}).$$

It follows that

$$\begin{aligned} d(X_0, x_0) &\geq t_0 + 3G(C_l^{\mathrm{DTO}}) - C_l^{\mathrm{DTO}} - d(x_l, X_0) \\ &\geq 3G(C_l^{\mathrm{DTO}}) - d(x_0, x_l) - d(x_l, X_0) \qquad \text{by (7.4)} \\ &= 3G(C_l^{\mathrm{DTO}}) - d(x_0, X_0). \end{aligned}$$

Hence $d(x_0, X_0) \geq \frac{3}{2} G(C_l^{\mathrm{DTO}})$.

We now have to cover the case that $d(x_0, x_l) > d(X^{\rho-1}, x_l)$. Notice that σ_l must be older than s_0, otherwise the oldest unserved request would have been in DETOUR's back before reaching σ_l, and $(C_l^{\mathrm{DTO}}, x_l)$ would not have been the selection

point. Observe also that $d(X^{\rho-1}, x_l)$ is at least the distance between the realized turning point $\mathrm{TP}^{\rho-1}$ and the corresponding target, as the final target of a phase is always served within that phase, as shown above. From the inductive hypothesis for phase $\rho-1$, Statement (a), we obtain that

$$d(x_l, x_0) > G(T^{\rho-1}). \tag{7.5}$$

As $d(X_0, x_0) \geq d(x_l, x_0)$, we only need to consider $G(T^{\rho-1}) < G(C_l^{\mathrm{DTO}})$. Let $2^a G(T^{\rho-1}) = G(C_l^{\mathrm{DTO}})$ for some integer $a \geq 1$. We distinguish two cases.

Case 1: $t_0 \geq T^{\rho-1}$. In this case we have

$$T^{\rho-1} + d(X^{\rho-1}, X_0) + d(X_0, x_0) \geq t_0 + 3G(C_l^{\mathrm{DTO}}),$$

as DETOUR's server started from $X^{\rho-1}$ at time $T^{\rho-1}$ and chooses the turning point TP_0 at time C_l^{DTO} such that s_0 is not served with a smaller flow time than $3G(C_l^{\mathrm{DTO}})$. Since $d(x_0, X_0) \geq d(X^{\rho-1}, X_0)$, this yields

$$2d(x_0, X_0) \geq t_0 - T^{\rho-1} + 3G(C_l^{\mathrm{DTO}}) \geq 3G(C_l^{\mathrm{DTO}}).$$

Case 2: $t_0 < T^{\rho-1}$. Here we use $t_l \leq t_0 < T^{\rho-1}$ since σ_l is older than s_0, as reasoned above. By (7.5) and Observation 7.9 (1), request s_0 is served after σ_l by every $\mathrm{ADV} \in \mathcal{ADV}(T^{\rho-1})$. Hence,

$$t_0 + \alpha_0(T^{\rho-1}) \geq t_l + \alpha_l(T^{\rho-1}) + d(x_0, x_l). \tag{7.6}$$

From the assumption that Statement (c) holds true for phase $\rho - 1$, we know that σ_l is served in time, i.e.,

$$C_l^{\mathrm{DTO}} \leq t_l + \alpha_l(T^{\rho-1}) + 3G(T^{\rho-1}), \tag{7.7}$$

because $T^{\rho-1}$ was the last time DETOUR turned around before he served σ_l, and because of $t_l \leq T^{\rho-1}$. Hence, if DETOUR's server turned around immediately after serving σ_l, it would hold that

$$\begin{aligned} C_0^{\mathrm{DTO}} &= C_l^{\mathrm{DTO}} + d(x_0, x_l) \\ &\leq t_l + \alpha_l(T^{\rho-1}) + 3G(T^{\rho-1}) + d(x_0, x_l) && \text{by (7.7)} \\ &\leq t_0 + \alpha_0(T^{\rho-1}) + 3G(T^{\rho-1}) && \text{by (7.6)} \\ &\leq t_0 + 4G(T^{\rho-1}) \leq t_0 + 2^{-a+2} G(C_l^{\mathrm{DTO}}). \end{aligned}$$

Thus, s_0 would be served with a flow time of at most $2\,G(C_l^{\mathrm{DTO}})$, because $a \geq 1$. Since DETOUR never plans his turning point in such a way that the target is reached with a flow time of less than three times the current guess value, we can deduce that the server does in fact not turn around, but has time of at least $(3 - 2^{-a+2})G(C_l^{\mathrm{DTO}}) > 0$ to spend on a detour, starting at time C_l^{DTO}. Thus, the

distance between x_l and the turning point TP_0 planned at time C_l^{DTO} is at least $\frac{1}{2}(3-2^{-a+2})G(C_l^{\mathrm{DTO}})$, and we conclude that

$$\begin{aligned} d(X_0,x_0) &= d(X_0,x_l) + d(x_l,x_0) \\ &\geq \frac{1}{2}\left(3-2^{-a+2}\right) G(C_l^{\mathrm{DTO}}) + d(x_l,x_0) \\ &\geq \left(1-2^{-a}\right) G(C_l^{\mathrm{DTO}}) + G(T^{\rho-1}) \qquad \text{by (7.5)} \\ &= \left(1-2^{-a}\right) G(C_l^{\mathrm{DTO}}) + 2^{-a} G(C_l^{\mathrm{DTO}}) \\ &\geq G(C_l^{\mathrm{DTO}}). \end{aligned}$$

Assume now that (a) holds for the turning points $\mathrm{TP}_0,\ldots,\mathrm{TP}_{i-1}$ of the considered phase ρ. Consider $\mathrm{TP}_i=(T_i,X_i)$ replacing TP_{i-1} at time t of phase ρ, and let $t'<t$ be the time when TP_{i-1} was valid for the first time. Denote by $s_i=(t_i,x_i)$ the target corresponding to TP_i. Recall that we assumed w.l.o.g. that TP_i is to the right of s_i.

If $s_i=s_0$, we can deduce that $G(t)\geq 2G(t')$ because the turning point changes but the target does not. That is, in order to serve the target with a flow time of $3G(t)$ instead of $3G(t')$, DETOUR has now at least $3(G(t)-G(t'))$ additional time units to spend on the way to the target. Hence, he can enlarge his detour by

$$d(X_i,X_{i-1}) \geq \frac{3}{2}(G(t)-G(t')) \geq G(t)-G(t').$$

By the inductive assumption, $d(X_{i-1},x_0)\geq G(t')$. Consequently,

$$\begin{aligned} d(X_i,x_0) &= d(X_i,X_{i-1}) + d(X_{i-1},x_0) \\ &\geq G(t)-G(t')+G(t') = G(t). \end{aligned}$$

If $s_i\neq s_0$ and $t_i > T^{\rho-1}$, then we can conclude, as in the proof for TP_0, that $d(X_i,x_i)\geq \frac{3}{2}G(t)$.

If $s_i\neq s_0$ and $t_i\leq T^{\rho-1}$, it follows that s_i must have been infeasible at time C_l^{DTO} when the initial target was chosen, as s_i was more critical than s_0 at that time: otherwise it could not have become a target later on. Hence, there exists a request σ_j older than s_i and to the right of s_i's hypothetical turning point $\mathrm{TP}_i'=(T_i',X_i')$ considered at time C_l^{DTO}.

By exactly the same arguments used for TP_0 above, we can deduce for the hypothetical turning point TP_i' considered at time C_l^{DTO} that

$$d(X_i',x_i) \geq G(C_l^{\mathrm{DTO}}). \tag{7.8}$$

If $G(C_l^{\mathrm{DTO}})=G(t)$, we obtain that

$$d(X_i,x_i) > d(X_i',x_i) \geq G(C_l^{\mathrm{DTO}}) = G(t).$$

On the other hand, if $G(t)\geq 2G(C_l^{\mathrm{DTO}})$, DETOUR has at least $3(G(t)-G(C_l^{\mathrm{DTO}}))$ time units more to spend on his detour to x_i than he would have had if s_i had become

the target at time C_l^{DTO}. Hence, the distance of the hypothetical turning point TP_i' considered at time C_l^{DTO} and the one chosen at time t, namely TP_i, is at least half of that additional time. More precisely, we have that

$$d(X_i', X_i) \geq \frac{3}{2}(G(t) - G(C_l^{\mathrm{DTO}})) \geq G(t) - G(C_l^{\mathrm{DTO}}).$$

Together with (7.8), we obtain that

$$\begin{aligned} d(X_i, x_i) &= d(X_i, X_i') + d(X_i', x_i) \\ &\geq G(t) - G(C_l^{\mathrm{DTO}}) + G(C_l^{\mathrm{DTO}}) = G(t), \end{aligned}$$

which proves the inductive step.

Notice that exactly the same arguments apply whenever a request is not made a target because it is not feasible: its hypothetical turning point considered at that time t must be at least at distance $G(t)$ to the corresponding target.

Proof of Statements (b) and (c):

Let SP denote the selection point which defines the end of the previous phase. If no previous phase exists, we define $\mathrm{SP} = (0, 0)$. By definition, $\mathrm{SP} = (C_l^{\mathrm{DTO}}, x_l)$ for some request σ_l. We distinguish two cases: in Case I we consider the situation that DETOUR's server immediately turns around in the selection point, thus not entering the detour mode, in Case II, we assume that he enters the detour mode at the selection point. Furthermore, we partition the set of requests served in phase ρ into three classes, depending on which part of DETOUR's route they are served in: Class 1 contains all requests served between the realized turning point and its corresponding target. Class 2 consists of those requests served in phase ρ after the target, whereas all requests served between the selection point and the realized turning point belong to Class 3.

Let σ_z be the oldest unserved request at time C_l^{DTO}. Denote by $\mathrm{TP}_0 = (T_0, X_0)$ be the turning point chosen at time C_l^{DTO}, and by $s_0 = (t_0, x_0)$ its corresponding target.

Case I: DETOUR turns around in the selection point.

1. Phase ρ is the first phase of a busy period.

Hence, all requests served in that phase are released at time t_0 or later, and since DETOUR immediately enters the focus mode at the beginning of the phase, they are all served without any detour. By Lemma 7.10, we know that $d(p^{\mathrm{D}}(t_0), p^{\mathrm{ADV}}(t_0)) \leq \frac{5}{2}G(t_0)$. Therefore, DETOUR reaches all requests served in phase 1 at most $\frac{5}{2}G(t_0)$ time units later than the adversary, so all requests are served in time.

2. Phase ρ is not the first phase of a busy period.

We have $\mathrm{TP}_0 = \mathrm{SP} = (C_l^{\mathrm{DTO}}, x_l)$, because the server turns around immediately as he cannot serve s_0 with a flow time of $3G(C_l^{\mathrm{DTO}})$ or less. Thus, TP_0 equals the realized turning point $\mathrm{TP}^\rho = (T^\rho, X^\rho)$, request s_0 is the final target, and $C_0^{\mathrm{DTO}} =$

$C_l^{\mathrm{DTO}} + d(x_0, x_l)$. Note that Class 3 is empty in this case. First of all, we know that s_0 cannot be older than σ_l, otherwise the oldest request would have been in the servers back before reaching σ_l. Furthermore, by Statement (a), $d(x_l, x_0) = d(X^\rho, x_0) \geq G(T^\rho) = G(C_l^{\mathrm{DTO}})$.

Therefore, by Observation 7.9 (i), in all offline solutions in $\mathcal{ADV}(T^\rho)$, request σ_l must be served before s_0. It is easy to see that $\tilde{T}_l \leq \tilde{T}_0 = T^\rho$, and since Statement (c) for phase $\rho-1$ tells us that σ_l is served in time, we can apply Lemma 7.8 and conclude that s_0 is also served in time. Notice that exactly the same arguments apply to all requests in Class 2. This ensures Statement (c).

It remains to consider all other requests of Class 1. To this end, let σ_j be a request served between σ_l and s_0 by DETOUR. If σ_j was more critical than s_0, it must be at least as old as s_0, because it is to the right of s_0. Since σ_j was not chosen as target at time C_l^{DTO}, it must have been infeasible, that is, there is a request σ_b older than σ_j and to the right of σ_j's hypothetical turning point TP'_j. But as σ_j is more critical than s_0, and DETOUR turns around immediately for s_0, we have that $\mathrm{TP}'_j = \mathrm{TP}^\rho$, which implies that also s_0 cannot have been feasible at time C_l^{DTO}, a contradiction. Thus, σ_j must be less critical than s_0. This means that it is served with the same or a smaller flow time than s_0, hence with flow time of at most $4G(T^\rho) \leq 4G(C_j^{\mathrm{DTO}})$.

Case II: DETOUR enters the detour mode at the selection point.

Since in this case, the proof is in large parts the same for whether phase ρ is the first phase of a busy period or not, we only make the distinction when needed.

We start with the requests in Class 1. Let us first consider an arbitrary request s_i which was ever marked as target during the current phase ρ but did not become the final target. We prove the stronger statement that s_i is served with a flow time of at most $3G(T^\rho)$. Consider the time T^ρ at which DETOUR reverses direction. If s_i was still the target at T^ρ, and $\mathrm{TP}_i = (T_i, X_i)$ the corresponding turning point valid at that time, s_i would be served with a flow time of $3G(T^\rho)$. As s_i was not the target at time T^ρ anymore, the then valid target s^ρ must be more critical than s_i. Hence, the realized turning point $\mathrm{TP}^\rho = (T^\rho, X^\rho)$ must be closer to the selection point $\mathrm{SP} = (C_l^{\mathrm{DTO}}, x_l)$ than the point $\mathrm{TP}_i = (T_i, X_i)$ defined above, as DETOUR must turn around earlier for the more critical request s^ρ. Consequently, s_i is reached even earlier than in the case that it was not replaced. We can conclude that in both cases, s_i is served with a flow time of at most $3G(T^\rho)$.

Conclusion 7.1. *Let S be the set of all requests which were ever marked as target during the current phase except for the final target. All requests in S are served with a flow time at most $3G(T^\rho)$.*

From this we can conclude that the oldest request σ_z is also served with a flow time of at most $3G(T^\rho)$, since it is less critical than the initial target s_0 of this phase: if it was more critical, it would have been selected as target at time C_l^{DTO} instead of s_0 (as the oldest unserved request overall σ_z cannot be infeasible).

Conclusion 7.2. *The oldest unserved request σ_z is served with a flow time at most $3G(T^\rho)$.*

Before we consider the final target and requests from Class 1 which are less critical than the final target, let us show that any request $\sigma_i = (t_i, x_i)$ in Class 1 which is more critical than the final target is served in time. As a member of Class 1, request σ_i must lie between the turning point and the final target. Since it is also more critical than s^ρ, it must be older than s^ρ. Consider the time $t \leq T^\rho$ at which the candidate setup was performed last during phase ρ. By definition, s^ρ was either made a target at time t or it remained the target valid at that time. Hence, $s^\rho \in \vee(s_0, p^{\mathrm{D}}(C_l^{\mathrm{DTO}}), G(t))$, the critical region valid at time t. Since σ_i is more critical than s^ρ and also closer to the selection point, it must also be inside $\vee(s_0, p^{\mathrm{D}}(C_l^{\mathrm{DTO}}), G(t))$. Hence, the only reason why σ_i was not made a target at time t is that it was infeasible. Notice that this also holds true if $t = C_l^{\mathrm{DTO}}$ and no critical region had yet been defined when s^ρ was marked as a target.

As no further candidate setup takes place, the guess value does not change anymore after time t, so σ_i is still infeasible at time T^ρ. This means that there exists a request $\sigma_b = (t_b, x_b)$ between the hypothetical turning point $\mathrm{TP}'_i = (T'_i, X'_i)$ corresponding to σ_i and the actually chosen turning point TP^ρ, which is older than σ_i. By Part (a), this implies that $d(X'_i, x_i) \geq G(T^\rho)$. Hence, as σ_b is even further away from x'_i than X'_i, we deduce that $d(x_b, x_i) \geq G(T^\rho)$, which by Observation 7.9 (i) implies that σ_b must be served before σ_i in any offline solution in $\mathcal{ADV}(T^\rho)$. Observe that the oldest request σ_z must lie between σ_b and σ_i as σ_i is more critical and younger than σ_z. Hence, σ_i is served after σ_z in any such offline solution. Clearly, $C_i^{\mathrm{DTO}} = C_z^{\mathrm{DTO}} + d(x_z, x_i)$, and as $t_i \leq T^\rho$, we have that $\tilde{T}_i = \tilde{T}_z = T^\rho$. Hence, we can apply Lemma 7.8 to deduce from Conclusion 7.2 that also σ_i is served in time.

Conclusion 7.3. *All requests σ_i of Class 1 which are more critical than the final target are served in time.*

Before continuing the proof for Class 1, let us briefly consider a subclass of Class 2. To this end, let σ_j be a request of Class 2 which is released by time T^ρ and more critical than the final target s^ρ. Again, consider the last time $t \leq T^\rho$ at which a candidate setup was performed. As above, we need to investigate why σ_j was not made a target at time t. In this case, it was either infeasible or outside the critical region $\vee(s_0, p^{\mathrm{D}}(C_l^{\mathrm{DTO}}), G(t))$ valid at time t. If it was infeasible, the same argument as above proves that σ_j is served in time (note that again, $\tilde{T}_j = \tilde{T}_z = T^\rho$). It remains to consider the case that σ_j is outside $\vee(s_0, p^{\mathrm{D}}(C_l^{\mathrm{DTO}}), G(t))$. By definition of t, we have that $\vee(s_0, p^{\mathrm{D}}(C_l^{\mathrm{DTO}}), G(t))$ is still valid at time T^ρ, so by Observation 7.9 (ii), σ_j must be served after s_0 in all offline solutions in $\mathcal{ADV}(T^\rho)$. Since DETOUR serves σ_j immediately after s_0, Conclusion 7.1 and the fact that $\tilde{T}_0 = \tilde{T}_j = T^\rho$ allow us to apply Lemma 7.8 and deduce that σ_j is served in time.

Conclusion 7.4. *All requests σ_j of class 2 which are more critical than the final target and released no later than T^ρ are served in time.*

Now consider the final target s^ρ of the phase. Clearly, if DETOUR does not turn around immediately when marking s^ρ as a target and switch to the focus mode, s^ρ is served with flow time $3G(T^\rho)$.

So assume that DETOUR enters the focus mode at the time he marks s^ρ as target. We need to distinguish two subcases: (i) $t^\rho < T^\rho$, and (ii) $t^\rho = T^\rho$.

Consider first case (i). Let $t' < T^\rho$ be the last time before T^ρ at which a candidate setup was performed. Hence, $t^\rho \leq t'$. Let $\mathrm{TP}' = (T', X')$ be the turning point chosen at time t'. Since s^ρ was not made a target at time t', it was either infeasible at time t' or feasible but outside the critical region valid at time t'.

If it was infeasible, there must be a request $\sigma_b = (t_b, x_b)$ older than s^ρ yet unserved at time t' and which lies to the right of s^ρ's hypothetical turning point considered at time t'. Since a target candidate setup is also performed whenever DETOUR serves a request while in the detour mode, form the definition of t' and by the assumption that σ_b is yet unserved at time t', it follows that $C_b^{\mathrm{DTO}} \geq T^\rho$. On the other hand, we assumed that the time T^ρ at which DETOUR turns around is the time at which he marks s^ρ as a target. Since s^ρ must be feasible at that time, request σ_b must have been served by then, which means that $C_b^{\mathrm{DTO}} \leq T^\rho$. Hence, we obtain that $(C_b^{\mathrm{DTO}}, x_b) = (T^\rho, X^\rho)$. From part (a) it follows that $d(x_b, x^\rho) \geq G(T^\rho)$, so by Observation 7.9 (i), we have that s^ρ is served after σ_b in all offline solutions in $\mathcal{ADV}(T^\rho)$. But then, s^ρ must be served after σ_z in all such offline solutions as well, since σ_z lies between σ_b and s^ρ and is older than both. Since $\tilde{T}^\rho = \tilde{T}_z = T^\rho$, Conclusion 7.2 and Lemma 7.8 let us deduce that s^ρ is served in time in that case.

We now consider the case that s^ρ was feasible but outside the critical region at time t'. Since s^ρ was marked as target at time T^ρ, it was inside the critical region valid at time T^ρ, and we can deduce that $2G(t') \leq G(T^\rho)$. Now since s^ρ was feasible at time t', it would have satisfied the preconditions of Conclusion 7.3 or 7.4 if the sequence had ended at time t'. Consequently, the turning point TP' chosen at time t' was chosen in such way that s^ρ would be served with a flow time of at most $4G(t')$.

Thus, it holds for all times $t \in [t', T^\rho)$ that

$$t + d(p^{\mathrm{D}}(t), X') + d(X', x^\rho) \leq t^\rho + 4G(t').$$

This implies that

$$T^\rho + d(p^{\mathrm{D}}(T^\rho), x^\rho) \leq t^\rho + 4G(t') \leq t^\rho + 2G(T^\rho),$$

contradicting the assumption that DETOUR had to turn around immediately at time t^ρ. Notice that we showed that in the case that $t^\rho < T^\rho$, the final target s^ρ must have been infeasible at the last time $t' < T^\rho$ at which a target candidate setup was performed.

It remains to consider case (ii), where s^ρ is made a target as soon as it is released: $t^\rho = T^\rho$.

Let us first investigate the length of the detour made by DETOUR. I.e., we first prove a bound on $d(x_l, X^\rho) = T^\rho - C_l^{\mathrm{DTO}} = t^\rho - C_l^{\mathrm{DTO}}$. To this end, we make use of the assumption that DETOUR can not serve s^ρ with a flow time of $3G(T^\rho)$. Hence, $d(X^\rho, x^\rho) > 3G(T^\rho)$. On the other hand, $d(x^\rho, x_0) \leq G(T^\rho) - (t^\rho - t_0)$ since s^ρ is inside the critical region valid at time T^ρ and younger than s_0. Putting the two inequalities together, we obtain

$$d(x_0, X^\rho) = d(x^\rho, X^\rho) - d(x^\rho, x_0) > 2G(T^\rho) + t^\rho - t_0. \tag{7.9}$$

Making use of the fact that DETOUR serves s_0 with a flow time of at most $3G(T^\rho)$ (Conclusion 7.1), we have

$$C_l^{\mathrm{DTO}} + d(x_l, X^\rho) + d(X^\rho, x_0) \leq t_0 + 3G(T^\rho).$$

Therefore,

$$\begin{aligned}
d(x_l, X^\rho) &\leq t_0 + 3G(T^\rho) - C_l^{\mathrm{DTO}} \\
&\quad - d(X^\rho, x_0) \\
&< t_0 + 3G(T^\rho) - C_l^{\mathrm{DTO}} \\
&\quad - 2G(T^\rho) - t^\rho + t_0 && \text{by (7.9)} \\
&= G(T^\rho) + (t_0 - C_l^{\mathrm{DTO}}) - t^\rho + t_0 \\
&\leq G(T^\rho) - t^\rho + C_l^{\mathrm{DTO}} && \text{as } t_0 \leq C_l^{\mathrm{DTO}} \\
&= G(T^\rho) - d(x_l, X^\rho).
\end{aligned}$$

We thus obtain that

$$d(x_l, X^\rho) = T^\rho - C_l^{\mathrm{DTO}} \leq \frac{1}{2} G(T^\rho). \tag{7.10}$$

Recall that S is the set of requests which contains all requests in Class 1 that were ever marked as target, except for s^ρ itself. We showed before that each request in $S \cup \{\sigma_z\}$ is served with a flow time of at most $3G(T^\rho)$ (Conclusions 7.1 and 7.2), in particular in time. Furthermore, each such request has been released before time T^ρ, which is the last time DETOUR reverses direction before serving that request. Hence, if for every offline solution in $\mathcal{ADV}(T^\rho)$ there exists a request from $S \cup \{\sigma_z\}$ which is served before s^ρ, then Lemma 7.8 yields that s^ρ is served in time.

Now consider an arbitrary, but fixed $\textsc{adv} \in \mathcal{ADV}(T^\rho)$ in which s^ρ is served before all requests in $S \cup \{\sigma_z\}$. Recall that we assumed that the turning point's position X^ρ is to the right of x^ρ. At time T^ρ, ADV's server must be located left of all requests in $S \cup \{\sigma_z\}$, and all requests in $S \cup \{\sigma_z\}$ are yet unserved by ADV. Our aim is to show for the respective completions times of s^ρ that

$$C_\rho^{\mathrm{DTO}} \leq C_\rho^{\textsc{adv}} + 3G(T^\rho) \tag{7.11}$$

holds. Alas, in one subcase we will only prove the weaker claim $C_\rho^{\mathrm{DETOUR}} \leq t^\rho + 4G(T^\rho)$. However, we will deduce in that subcase that there is a request in Class 2

which is served in time by DETOUR. In all other cases, (7.11) holds, i.e., the final target is reached by DETOUR no later than $3G(T^\rho)$ time units after the adversary reaches it in the considered offline solution, and as we considered an arbitrary ADV $\in \mathcal{ADV}(T^\rho)$, we can deduce that DETOUR serves s^ρ in time. Hence if Class 2 is empty, (7.11) yields statement (c).

Consider $p^{\mathrm{ADV}}(T^\rho)$, which by our assumption is further to the left than any point in $S \cup \{\sigma_z\}$. Since the adversary is non-abusive, there must be a request σ_k left of his position at time T^ρ which he just served or which he is heading to. This request σ_k must have been released strictly before time T^ρ, i.e., $t_k < T^\rho$. Another case distinction is needed.

- σ_k was already served by DETOUR by time T^ρ.

  Notice that this situation can not occur if we are in the first phase of the first busy period. Hence we may assume that $\rho \geq 2$.

  We show that in this case,

  $$d(p^{\mathrm{ADV}}(T^\rho), x_l) \leq \frac{5}{2}\, G(T^\rho). \tag{7.12}$$

  This, together with (7.10) then yields $d(p^{\mathrm{D}}(T^\rho), p^{\mathrm{ADV}}(T^\rho)) \leq 3G(T^\rho)$, from which (7.11) easily follows, as DETOUR immediately heads to serve s^ρ at time T^ρ, while ADV cannot proceed to serve $s^\rho = (t^\rho, x^\rho)$ before $t^\rho = T^\rho$.

  By assumption, σ_k is served before σ_l by DETOUR, and by Statement (b) for phase $\rho - 1$ applied to σ_l we have the inequality

  $$t_k + d(x_k, x_l) \leq t_l + 4G(C_l^{\mathrm{DTO}}) \leq t_l + 4G(T^\rho). \tag{7.13}$$

  If the adversary serves σ_k after σ_l, then

  $$t_l + d(x_k, x_l) \leq t_k + G(T^\rho),$$

  and together with (7.13), we obtain

  $$\begin{aligned} d(x_k, x_l) &\leq t_k + G(T^\rho) - t_l \\ &\leq t_l + 4G(T^\rho) - d(x_k, x_l) + G(T^\rho) - t_l \\ &= 5G(T^\rho) - d(x_k, x_l). \end{aligned}$$

  This yields (7.12).

  Now consider the case that ADV serves σ_k before σ_l. Since the adversary is heading to or just coming from σ_k at time T^ρ and is still on the left of the set S, this means that he hasn't served σ_l yet at time $T^\rho \geq C_l^{\mathrm{DTO}} \geq t_l$. Hence, at time T^ρ, his server must be within range $G(T^\rho)$ from x_l, so in particular (7.12) holds.

  Notice that, in the previous line of reasoning, we did neither make use of the assumption that DETOUR serves the final target with flow time more than $3G(T^\rho)$, nor that $t^\rho = T^\rho$.

- σ_k has not been served yet by DETOUR at time T^ρ.

  Recall that we are in the situation that the final target s^ρ was made a target by DETOUR at its release time $t^\rho = T^\rho$, and that the server turns around immediately at that time. Let $t' < T^\rho$ be the last time before time T^ρ at which a target candidate setup is performed by DETOUR. As $t_k < T^\rho$, and since a target candidate setup is performed whenever a new request is released, we have that $t_k \leq t'$. Note that request σ_k must be served by DETOUR in the current phase ρ. If it wasn't served in phase ρ, there would be an older request which remains unserved at least until time T^ρ and which is in the server's back after he turned in TP^ρ. But then, this request must be older than s^ρ and would have caused s^ρ to be infeasible.

  Let s_m be the target valid at time t' (after the target selection), and TP_m the corresponding turning point. In the proof for Conclusions 7.1 and 7.2 we showed that the oldest unserved request σ_z, and all requests in S are served with a flow time of at most $3G(t')$ if TP_m is also the realized turning point. We are in the situation that TP_m is replaced at time T^ρ by TP^ρ. But as DETOUR turns around immediately when replacing TP_m, he turns earlier than planned and hence serves the requests in $S \cup \{\sigma_z\}$ are even served earlier, in particular with a flow time of at most $3G(t')$.

  Now consider request σ_k. There are three possible reasons why σ_k is not selected as target at time t': (i) it is less critical than s_m, (ii) σ_k is more critical than s_m but infeasible at time t', or (iii) it is more critical than s_m but outside the critical region valid at that time.

  In case (i), i.e. if σ_k is less critical than s_m, it is also served with a flow time of at most $3G(t') \leq 3G(T^\rho)$, which yields in particular $C_k^{\mathrm{DTO}} \leq C_k^{\mathrm{ADV}} + 3G(T^\rho)$ for the considered $\mathrm{ADV} \in \mathcal{ADV}(T^\rho)$. Furthermore, s^ρ is more critical than σ_k and younger. Therefore, it must be to the left of σ_k, which in turn was left of $p^{\mathrm{ADV}}(T^\rho)$. Therefore, s^ρ is served after σ_k by ADV, and we conclude

$$\begin{aligned} C_\rho^{\mathrm{DTO}} &= C_k^{\mathrm{DTO}} + d(x^\rho, x_k) \\ &\leq C_k^{\mathrm{ADV}} + 3G(T^\rho) + d(x_k, x^\rho) \\ &\leq C_\rho^{\mathrm{ADV}} + 3G(T^\rho), \end{aligned}$$

  which was our claim (7.11).

  Now let us investigate case (ii), in which σ_k is infeasible at time t'. That means that there exists a request $\sigma_b = (t_b, x_b)$ older than σ_k and to the right of the hypothetical turning point corresponding to σ_k at time t'. If ADV served σ_b before time T^ρ, he must have served σ_z on his way from σ_b to his current position, as σ_z is older than σ_b and located between σ_b and $p^{\mathrm{ADV}}(T^\rho)$. This contradicts our assumption that ADV has not served any of the requests in $S \cup \{\sigma_z\}$ by time T^ρ. Consequently, σ_b must be yet unserved by ADV at time T^ρ, which yields $d(p^{\mathrm{ADV}}(T^\rho), x_b) \leq G(T^\rho)$. Since σ_b must be

to the right of the selection point $(C_l^{\mathrm{DTO}}, x_l)$, we obtain in particular that $d(p^{\mathrm{ADV}}(T^\rho), x_l) \leq G(T^\rho)$, which together with (7.10) implies

$$d(p^{\mathrm{ADV}}(T^\rho), p^{\mathrm{D}}(T^\rho)) \leq \frac{3}{2} G(T^\rho).$$

Hence, DETOUR reaches s^ρ no more $\frac{3}{2}G(T^\rho)$ time units later than ADV, which means that s^ρ is served in time.

Finally, consider Case (iii): request σ_k is outside the critical region valid at time t'. Consequently, σ_k is served after s_0 in all offline solutions in $\mathcal{ADV}(t')$. Recall that we showed before that in the current case, all requests in $S \cup \{\sigma_z\}$ are served with a flow time of at most $3G(t')$. Hence, we obtain that

$$\begin{aligned} T^\rho + d(p^{\mathrm{D}}(T^\rho), x_k) = C_k^{\mathrm{DTO}} &= C_0^{\mathrm{DTO}} + d(x_0, x_k) \\ &\leq t_0 + 3G(t') + d(x_0, x_k) \\ &\leq t_k + \alpha_k(t') + 3G(t'). \end{aligned}$$

If $2G(t') \leq G(T^\rho)$, we obtain together with $t_k < T^\rho$ that

$$d(p^{\mathrm{D}}(T^\rho), x_k) \leq 4G(t') \leq 2G(T^\rho),$$

which lets us conclude that $d(p^{\mathrm{D}}(T^\rho), p^{\mathrm{ADV}}(T^\rho)) \leq 2G(T^\rho)$. Thus, DETOUR serves s^ρ at most $2G(T^\rho)$ time units later than ADV.

If $G(t') = G(T^\rho)$, we obtain by Property 7.6 that

$$\alpha_k(t') \leq \alpha_k(T^\rho).$$

Thus, we have that σ_k is served in time, since $\tilde{T}_k = T^\rho$. If s^ρ is served after σ_k by ADV, we conclude with Lemma 7.8 that (7.11) holds. Otherwise, s^ρ must be to the right of σ_k, is therefore, as the younger one, less critical than σ_k and served with a flow time of at most $4G(T^\rho)$ by DETOUR.

Note that we proved the stronger statement that s^ρ is served in time for all cases except for the case that all of the following statements hold simultaneously true:

- $t^\rho = T^\rho$ and DETOUR turns around immediately upon s^ρ's release,
- there exists ADV $\in \mathcal{ADV}(T^\rho)$ in which s^ρ is served before all requests in $S \cup \{\sigma_z\}$,
- there is a request σ_k to the left of $p^{\mathrm{ADV}}(T^\rho)$ with $t_k < T^\rho$ which is outside the critical region at the last time $t' < T^\rho$ at which a candidate setup was performed by DETOUR,
- σ_k is served after s^ρ by ADV.

In that special case, s^ρ is shown to be served with flow time $4G(T^\rho)$, and σ_k is shown to be served in time by DETOUR.

Conclusion 7.5. *The final target s^ρ is served either in time, or it is served with a flow time of at most $4G(T^\rho)$ and there exists a request σ_k in Class 2 which is released before T^ρ and which is served in time by* DETOUR.

Note that this implies Statement (c) for the case that Class 2 is empty.

Finally, let $\sigma_i = (t_i, x_i)$ be an arbitrary request of Class 1 which was never marked as target and which is less critical than the final target s^ρ. As it is less critical, it is served with the same or a smaller flow time than s^ρ, which is at most $4G(T^\rho) \leq 4G(C_i^{\mathrm{DTO}})$, which was our claim.

Conclusion 7.6. *Any request σ_i in Class 1 which does not belong to any of the sets of requests covered by Conclusions 7.1–7.3 or by Conclusion 7.5 is served with flow time at most $4G(T^\rho)$.*

We now consider Class 2. To this end, let $\sigma_j = (t_j, x_j)$ be a request served in phase ρ after s^ρ by DETOUR. First consider the case that σ_j is released no later than time T^ρ, that is, $t_j \leq T^\rho$. We proved already that σ_j is served in time if it is more critical than the final target s^ρ (Conclusion 7.4). Consider the case that σ_j is less critical than s^ρ. If s^ρ was served with a flow time of $3G(T^\rho)$, then also σ_j is served with that flow time, hence in time. So assume that s^ρ is served with a bigger flow time. Since σ_j is less critical and to the left of s^ρ, it must be strictly younger. Consequently, $t^\rho < t_j \leq T^\rho$. We showed before that in this case, s^ρ must have been infeasible at time t', the last time before T^ρ at which a candidate setup was performed, and that there exists a request σ_b older than s^ρ for which $(C_b^{\mathrm{DTO}}, x_b) = (T^\rho, X^\rho)$. By part (a), we deduce that

$$d(x_b, x_j) \geq d(x_b, x^\rho) = d(X^\rho, x^\rho) \geq G(T^\rho).$$

As σ_b is older than σ_j, Observation 7.9 (i) implies that it must be served before σ_j in all offline solutions in $\mathcal{ADV}(T^\rho)$. As reasoned before, then also σ_z must be served before σ_j in all such offline solutions. Since DETOUR serves σ_j immediately after σ_z and as $\tilde{T}_z = \tilde{T}_j = T^\rho$, we can apply Lemma 7.8 to conclude that σ_j is served in time.

It remains to consider those requests $\sigma_j \in$ Class 2 for which $t_j > T^\rho$. Note that $\tilde{T}_j = t_j > T^\rho$ in this case. Let

$$A := \{\sigma_z\} \cup S \cup \{\sigma_i \in \text{Class 2 } : t_i \leq T^\rho\}.$$

It is easy to see that each request $\sigma_a \in A$ is released by time T^ρ, so we have that $\tilde{T}_a = T^\rho < \tilde{T}_j$. Furthermore, we showed before that all requests in A are served in time by DETOUR. Consider an arbitrary ADV $\in \mathcal{ADV}(\tilde{T}_j)$. Assume that there exists a request $\sigma_a \in A$ which is served by ADV before σ_j. No matter whether x_j is left of x_a or not, we have that $C_j^{\mathrm{DTO}} \leq C_a^{\mathrm{DTO}} + d(x_j, x_a)$. Hence, we can apply Lemma 7.8 in that case and deduce that σ_j is served in time.

Therefore, we can restrict our attention to the case that ADV serves all requests in A after σ_j. In particular, this means that is has not served any of the requests

in A at at time T^ρ yet. We distinguish two subcases: (i) there is a request $\sigma_a \in A$ which is left of $p^{\text{ADV}}(T^\rho)$, and (ii) ADV's position at time T^ρ is to the left of all requests in A.

In case (i), σ_j cannot be to the left of σ_a, since otherwise it would be served after σ_a by ADV, contradicting our assumption in this case. Hence, it suffices to show that σ_j is served with flow time at most $4G(C_j^{\text{DTO}})$, because it cannot be the last request served by DETOUR in the current phase. As an element of the set A, request σ_a is served latest at time $t_a + 4G(T^\rho)$. Since σ_a was released before T^ρ, we know that $t_a < t_j$, and because DETOUR serves σ_j on the way to σ_a, we have that

$$\begin{aligned} C_j^{\text{DTO}} \leq C_a^{\text{DTO}} &\leq t_a + 4G(T^\rho) \\ &\leq t_j + 4G(T^\rho) \leq t_j + 4G(C_j^{\text{DTO}}), \end{aligned}$$

which was our claim.

Now consider case (ii): ADV's position at time T^ρ is to the left of all requests in A. Since the adversary is non-abusive, there must be a request σ_k to his left which he just served or where he is heading to at time T^ρ. In particular, it holds that $t_k < T^\rho$, from which we can deduce that σ_k must have been served already by DETOUR at time T^ρ: If it was served after T^ρ in phase ρ, it would belong to the set A, contradicting that it is left of ADV which in turn is left of all requests in A. Furthermore, σ_k cannot be served by DETOUR in a later phase: if it was, there would be an older request in the server's back, and DETOUR would have to turn around before reaching σ_j, thus serving σ_j also in a later phase, contradicting the assumption that σ_j is served in the current phase.

Exactly as in the proof of inequality (7.12) (used for the final target), we can in this case deduce for the distance of ADV's position at time T^ρ to the selection point $\text{SP} = (C_l^{\text{DTO}}, x_l)$ that

$$d(p^{\text{ADV}}(T^\rho), x_l) \leq \frac{5}{2}\, G(T^\rho). \tag{7.14}$$

Let $L := d(X^\rho, x_l) = T^\rho - C_l^{\text{DTO}}$ be the length of the detour made by DETOUR at the beginning of the current phase. If $L \leq G(T^\rho)/2$, we obtain from (7.14) that

$$d(p^{\text{ADV}}(T^\rho), p^{\text{D}}(T^\rho)) \leq 3G(T^\rho),$$

which implies that σ_j is served in time as ADV cannot serve σ_j before time $t_j > T^\rho$, and since DETOUR proceeds towards σ_j without any detour after time T^ρ.

So assume that $L = T^\rho - C_l^{\text{DTO}} > G(T^\rho)/2$. Since DETOUR serves σ_z with a flow time of at most $3G(T^\rho)$, we have that

$$C_l^{\text{DTO}} + L + d(X^\rho, x_z) \leq t_z + 3G(T^\rho),$$

which implies

$$d(X^\rho, x_z) \leq t_z + 3G(T^\rho) - L - C_l^{\text{DTO}}. \tag{7.15}$$

Furthermore, by assumption, ADV serves $\sigma_z \in A$ after σ_j, and we have that

$$T^\rho + d(p^{\text{ADV}}(T^\rho), x_j) + d(x_j, x_z) \leq t_z + G(T^\rho),$$

so in particular,

$$T^\rho + d(x_j, x_z) \leq t_z + G(T^\rho). \tag{7.16}$$

Note that $t_z \leq C_l^{\text{DTO}} \leq T^\rho < t_j$. We obtain the following estimate on DETOUR's completion time for σ_j.

$$\begin{aligned}
C_j^{\text{DTO}} &= T^\rho + d(X^\rho, x_z) + d(x_z, x_j) \\
&\leq t_z + 3G(T^\rho) - L - C_l^{\text{DTO}} + t_z + G(T^\rho) \\
&\quad \text{by (7.15) and (7.16)} \\
&= t_j + 3G(T^\rho) + [G(T^\rho) + 2t_z - C_l^{\text{DTO}} - t_j - L] \\
&\leq t_j + 3G(T^\rho) + [G(T^\rho) + C_l^{\text{DTO}} - T^\rho - L] \\
&\quad \text{as } t_z \leq C_l^{\text{DTO}} \text{ and } t_j \geq T^\rho \\
&= t_j + 3G(T^\rho) + [G(T^\rho) - 2L] \\
&\quad \text{as } L = T^\rho - C_l^{\text{DTO}} \\
&< t_j + 3G(T^\rho) \\
&\quad \text{by the assumption that } L > G(T^\rho)/2.
\end{aligned}$$

Hence, in this case, σ_j is even served with a flow time of at most $3G(T^\rho)$, in particular in time.

Conclusion 7.7. *Each request σ_j in Class 2 is either served in time, or there exists a request σ_k served later in the phase which is served in time, while σ_j is served with a flow time of $4G(C_j^{\text{DTO}})$.*

Note that this implies Statement (c) for Case II if Class 2 is non-empty.

Finally, we consider the requests in Class 3: Let σ_k be served between SP and TP^ρ. Since the oldest request σ_z lies in the server's back at time C_l^{DTO}, request σ_k is younger than σ_z. At the time C_k^{DTO} at which σ_k is served, DETOUR is either in the detour mode and has a valid turning point TP_m, or he has already turned in TP^ρ. In the first case, σ_z would be served latest at time $t_z + 3G(C_k^{\text{DTO}})$ if TP_m wasn't replaced, as shown before. Therefore, it must hold that

$$\begin{aligned}
C_k^{\text{DTO}} + d(x_k, \text{TP}_m) + d(\text{TP}_m, x_z) &\leq t_z + 3G(C_k^{\text{DTO}}) \\
&\leq t_k + 3G(C_k^{\text{DTO}}).
\end{aligned}$$

In the second case, $C_k^{\text{DTO}} \geq T^\rho$, and as σ_z is served with flow time at most $3G(T^\rho)$, we conclude that $C_k^{\text{DTO}} + d(x_k, x_z) \leq t_z + 3G(T^\rho) \leq t_k + 3G(C_k^{\text{DTO}})$.

Conclusion 7.8. *Each request σ_k in Class 3 is served with flow time at most $3G(C_k^{\mathrm{DTO}})$.*

This completes the proof of Theorem 7.11. □

8

Online bin coloring

The contents of this chapter is joint work with S.O. Krumke, L. Stougie, and J. Rambau, and has appeared in [32].

8.1 Introduction

Suppose a distributor delivers items to his customers. On demand, an item is taken from stock and brought to the loading platform immediately. Every item is labeled with the address of the customer. Items are transported by trucks and the trucks are loaded at the loading platform. The number of places at the loading platform is limited and due to space restrictions on the platforms, items that come out of stock have to be loaded immediately and cannot be exchanged from one truck to another at a later time. Each truck can carry a fixed number of items at a time and we assume that all items have unit size. A restriction is that all trucks must be fully loaded before they leave the loading platform.

The problem of the distributor is to decide which item is loaded in which truck. The combination of items in a truck defines the tour that the truck will have to make. Every tour starts and ends at the distributor. We assume that due to the geographical positions of the customers and the cost function, the most cost-efficient solution is one in which the number of customers in the tour that visits the most customers is as small as possible.

The above example is an application of the Bin Coloring Problem (Bcp). In the Bcp, one receives a finite sequence of unit size items $\sigma = r_1, r_2, \ldots$ where each item has a *color* $r_i \in \mathbb{N}$, and is asked to pack them into bins of size B. The goal is to pack the items into the bins "most uniformly", that is, to minimize the maximum

number of different colors assigned to a bin. The packing process is subject to the constraint that at any moment in time, at most $q \in \mathbb{N}$ bins may be partially filled. Bins may only be closed if they are filled completely. Notice that without these strict bounded space constraints the problem is trivial since in this case each item can be packed into a separate bin.

Theorem 8.1. BCP *is NP-complete.*

PROOF. We reduce 3-PARTITION to the BCP. Take any instance of 3-PARTITION, $a_1, \ldots, a_{3t}$ with $\sum_i a_i = tB$, $B/4 < a_i < B/2$ for all i, and $a_i \in \mathbb{Z}^+$ for all i. We can assume that $B \geq 4$. We can make an instance of BCP with t bins of size B as follows. First there are a_i items of color i for all $i = 1, \ldots, 3t$. Then there are $3t$ items, each with a different color of $3t+1, \ldots, 6t$. We will show that the BCP has a solution of at most 3 if and only if 3-PARTITION has a yes-answer.

Suppose that there is a yes-answer to 3-PARTITION. Let $S_1, \ldots, S_t$ be sets such that $\sum_{i:a_i \in S} a_i = B$. We put all items corresponding to $a_i \in S_j$ in bin j, for all $i = 1, \ldots, 3t$. After this, all bins contain exactly B items of at most three different colors. Consequently, all bins will be closed, and t empty bins can be used for placing the last $3t$ items. Assigning three of the remaining items to every bin will give a solution of 3.

If there is no yes-answer for 3-PARTITION, one of the following is true. There is at least one non-empty bin after the first tB items, or there is one bin that contains items of at least four different colors. In the latter case, we cannot have a solution of at most 3, so we can assume that all bins contain items of at most three colors, and there is at least one non-empty bin. Since $B/4 < a_i < B/2$ for all i, we know that the non-empty bins that contain items of only one different color have at least $B/2 + 1 \geq 3$ empty places, and that the non-empty bins that contain items of two different colors have at least two empty places. Non-empty bins containing items of three different colors must have at least one empty place, otherwise the bin would be replaced by an empty bin. If we want to establish a solution of at most 3, it follows that no bin can be completely filled (and replaced) by assigning the last $3t$ items, since these items all have a different color. As a consequence, all $3t$ items must be distributed over the t currently open bins. Since at least one bin in non-empty, at least one bin ends up with four or more different colors. □

In the online version of the BCP, the ONLINE BIN COLORING PROBLEM (OLBCP), each item must be packed without knowledge of any future items. Trivially, any online algorithm for the OLBCP is B-competitive, where B denotes the size of the bins.

Definition 8.2. [**Online Bin Coloring Problem**]
In the online bin coloring problem ($\text{OLBCP}_{B,q}$*) with parameters* $B, q \in \mathbb{N}$ *(*$B, q \geq 2$*), one is given a sequence* $\sigma = r_1, \ldots, r_m$ *of unit size items (requests), each with a color* $r_i \in \mathbb{N}$*, and is asked to pack them into bins with size* B*, that is, each bin can accommodate exactly* B *items. The packing is subject to the following constraints:*

1. *The items must be packed according to the order of their appearance, that is, item i must be packed before item k for all $i < k$.*

2. *At most q partially filled bins may be open to further items at any point in time during the packing process.*

3. *A bin may only be closed if it is filled completely, i.e., if it has been assigned exactly B items.*

The objective is to minimize the maximum number of different colors assigned to a bin.

An online algorithm for the OLBCP$_{B,q}$ *must pack each item r_i (irrevocably) without knowledge of requests r_k with $k > i$.*

Notice that the way the items are presented to the online algorithm, and the way an online algorithm has to react to the input, are according to the "making decisions one-by-one" model, as described in Section 3.1. Since the order in which items are present in a bin does not influence the objective value we can assume that an item is packed immediately at its presentation.

In the sequel it will be helpful to use the following view on the bins used to process an input sequence σ. Each open bin has an *index* x, where the number x satisfies $x \in \{1, \ldots, q\}$. Each time a bin with index x is closed (since it is filled completely) and a new bin is opened the new bin will also have index x. If no confusion can occur, we will refer to a bin with index x as *bin* x.

We investigate which competitive ratios are achievable by online algorithms for the OLBCP. Our results reveal a curiosity of competitive analysis: a truly stupid algorithm achieves essentially a (non-trivial) best possible competitive ratio for the problem, whereas a seemingly reasonable algorithm performs provably worse in terms of competitive analysis.

In Section 8.2 we analyze a natural "greedy-type" strategy, called GREEDY-FIT, and show that this strategy has a competitive ratio no greater than $3q$ but no smaller than $2q$, where q is the maximum number of bins that may be partially filled (open) at the same time. We show that a trivial strategy, called ONEBIN, which only uses one open bin at any time, has a strictly better competitive ratio of $2q - 1$ in Section 8.3. Then we show that surprisingly no deterministic algorithm can be substantially better than the trivial strategy. More specifically, in Section 8.4 we prove that no deterministic algorithm can, in general, have a competitive ratio less than q. Even more surprisingly, the general lower bound of q for the competitive ratio continues to hold for randomized algorithms against an oblivious adversary, as shown in Section 8.5. Finally, not even "resource augmentation", which means that the online algorithm is allowed to use a fixed number $q' \geq q$ of open bins instead of q, can help to overcome the lower bound of $\Omega(q)$ on the competitive ratio (see Section 8.4).

8.2 The algorithm GREEDYFIT

In this section we introduce a natural greedy-type strategy, which we call GREEDYFIT, and show that the competitive ratio of this strategy is at most $3q$ but no smaller than $2q$ (provided the capacity B is sufficiently large).

Algorithm GREEDYFIT

If upon the arrival of request r_i the color r_i is already contained in one of the currently open bins, say bin b, then put r_i into bin b. Otherwise put item r_i into a bin that contains the least number of different colors (which means opening a new bin if currently fewer than q bins are non-empty). Ties are broken arbitrarily.

Lemma 8.3. *Let $\sigma = r_1, \ldots, r_m$ be any request sequence for the* OLBCP$_{B,q}$ *and let $\sigma' = r_i, \ldots, r_{i+\ell}$ be any contiguous subsequence of σ. Then any algorithm packs the items of σ' into at most $2q + \lfloor(\ell - 2q)/B\rfloor$ different bins.*

PROOF. Let ALG be any algorithm and let $b_1, \ldots, b_t$ be the set of open bins for ALG just prior to the arrival of the first item of σ'. Denote by $f(b_j) \in \{1, \ldots, B-1\}$ the empty space in bin b_j at that stage. To close an open bin b_j, ALG requires $f(b_j)$ items. Opening and closing an additional new bin needs B items. To achieve the maximum number of bins ($\geq 2q$), ALG must first close each open bin and put at least one item into each newly opened bin. From this moment in time, opening a new bin requires B new items. □

Theorem 8.4. *Algorithm* GREEDYFIT *is c-competitive for the* OLBCP$_{B,q}$ *with $c = \min\{2q + \lfloor(qB - 3q + 1)/B\rfloor, B\}$.*

PROOF. Let σ be any request sequence and suppose GREEDYFIT$(\sigma) = w$. It suffices to consider the case $w \geq 2$. Let s be the smallest integer such that GREEDYFIT$(r_1, \ldots, r_{s-1}) = w - 1$ and GREEDYFIT$(r_1, \ldots, r_s) = w$. By the construction of GREEDYFIT, after processing $r_1, \ldots, r_{s-1}$ each of the currently open bins must contain exactly $w - 1$ different colors. Moreover, since $w \geq 2$, after packing additionally request r_s, GREEDYFIT has exactly q open bins (where as an exception we count here the bin where r_s is packed as open even if by this assignment it is just closed). Denote those bins by $b_1, \ldots, b_q$.

Let bin b_j be the bin among $b_1, \ldots, b_q$ that has been opened last by GREEDYFIT. Let $r_{s'}$ be the first item that was assigned to b_j. Then, the subsequence $\sigma' = r_{s'}, \ldots, r_s$ consists of at most $qB - (q - 1)$ items, since between $r_{s'}$ and r_s no bin is closed and at the moment $r_{s'}$ was processed, $q - 1$ bins already contained at least one item. Moreover, σ' contains items with at least w different colors. By Lemma 8.3 OPT distributes the items of σ' into at most $2q + \lfloor(qB - 3q + 1)/B\rfloor$ bins. Consequently,

$$\mathrm{OPT}(\sigma) \geq \frac{w}{2q + \lfloor(qB - 3q + 1)/B\rfloor}.$$

□

Corollary 8.5. *Algorithm* GREEDYFIT *is c-competitive for the* OLBCP$_{B,q}$ *with* $c = \min\{3q - 1, B\}$.

We continue by proving a lower bound on the competitive ratio of GREEDYFIT. This lower bound shows that the analysis of the previous theorem is essentially tight.

Theorem 8.6. GREEDYFIT *has a competitive ratio larger or equal to* $2q$ *for the* OLBCP$_{B,q}$ *if* $B \geq 2q^3 - q^2 - q + 1$.

PROOF. We construct a request sequence σ that consists of a finite number M of phases in each of which qB requests are given. The sequence is constructed in such a way that after each phase the adversary has q empty bins.

Each phase consists of two steps. In the first step q^2 items are presented, each with a new color that has not been used before. In the second step $qB - q^2$ items are presented, all with a color that has occurred before. We will show that we can choose the items given in Step 2 of every phase such that the following properties hold for the bins of GREEDYFIT:

> **Property 1:** The bins with indices $1, \ldots, q-1$ are never closed.
>
> **Property 2:** The bins with indices $1, \ldots, q-1$ contain only items of different colors.
>
> **Property 3:** There is an $M \in \mathbb{N}$ such that during Phase M GREEDYFIT assigns for the first time an item with a new color to a bin that already contains items with $2q^2 - 1$ different colors.
>
> **Property 4:** There is an assignment of the items of σ such that no bin contains items with more than q different colors.

We analyze the behavior of GREEDYFIT by distinguishing between the items assigned to the bin with index q and the items assigned to bins with indices 1 through $q-1$. Let L_k be the set of colors of the items assigned to bins $1, \ldots, q-1$ and let R_k be the set of colors assigned to bin q during Step 1 of Phase k.

We now describe a general construction of the request sequence given in Step 2 of a phase. During Step 1 of Phase k there are items with $|R_k|$ different colors assigned to bin q. For the moment, suppose that $|R_k| \geq q$ (see Lemma 8.9 (iv)). We now partition the at most q^2 colors in $|R_k|$ into q disjoint non-empty sets $S_1, \ldots, S_q$. We give $qB - q^2 \geq 2q^2$ items with colors from $|R_k|$ such that the number of items with colors from S_j is $B-q$ for every j, and the last $|R_k|$ items all have a different color.

By Lemma 8.9 (iii), GREEDYFIT will pack all items given in Step 2 into bin q. Hence bins $1, \ldots, q-1$ only get assigned items during Step 1, which implies the properties 1 and 2.

The adversary assigns the items of Step 1 such that every bin receives q items, and the items with colors in the color set S_j go to bin j. Clearly, the items in every bin have no more than q different colors. The items given in Step 2 can by construction of the sequence be assigned to the bins of the adversary such that all bins are completely filled, and the number of different colors per bin does not increase. This ensures that property 4 is satisfied.

Lemma 8.7. *At the end of Phase k where $k = 1, \ldots, M-1$, bin q of* GREEDYFIT *contains exactly $B - \sum_{j \le k} |L_j|$ items, and this number is at least q^2.*

PROOF. After Phase k, exactly kqB items have been given. Moreover, after k phases bins 1 through $q-1$ contain exactly $\sum_{j \le k} |L_j|$ items because the items of Step 2 are always packed into bin q by GREEDYFIT. Thus, the number of items in bin q of GREEDYFIT equals

$$kqB - \sum_{j \le k} |L_j| \bmod B \equiv \underbrace{B - \sum_{j \le k} |L_j|}_{<B} \bmod B.$$

We show that $B - \sum_{j \le k} |L_j| \ge q^2$. This implies that $B - \sum_{j \le k} |L_j| \bmod B = B - \sum_{j \le k} |L_j|$.

Since $k < M$ we know that each of the bins 1 through $q-1$ contains at most $2q^2 - 1$ colors. Thus, $\sum_{j \le k} |L_j| \le (2q^2 - 1)(q-1) = 2q^3 - 2q^2 - q + 1$. It follows from the assumption on B that $B - \sum_{j \le k} |L_j| \ge q^2$. □

Corollary 8.8. *For any Phase $k < M$, bin q is never closed by* GREEDYFIT *before the end of Step 1 of Phase k.*

PROOF. The claim clearly holds for the first phase. Hence for the remainder we consider the case $k > 1$.

Since there are exactly q^2 items presented in Step 1 of any phase, the claim is true by Lemma 8.7 as soon as $|\sum_{j \le k} L_j| \ge q^2$ at the beginning of Phase k: in that case, there is even enough space in bin q to accommodate all items given in Step 1. We show that $|L_1| + |L_2| \ge q^2$, which implies that $|\sum_{j \le k} L_j| \ge q^2$ for $k \ge 2$.

After Phase 1, each bin of GREEDYFIT contains q colors, which yields $|L_1| = q(q-1)$. It is easy to see that all items presented in Step 2 of the first phase are packed into bin q by GREEDYFIT: All these items have colors from R_1 where $|R_1| = q$. Either a color from R_1 is currently already present in bin q or bin q has less than q different colors, while all other bins contain q colors. In either case, GREEDYFIT packs the corresponding item into bin q.

By Lemma 8.7, at the end of Phase 1 bin q contains at least q^2 items. Since the last $|R_1| = q$ items presented in Step 2 of the first phase have all different colors (and all of these are packed into bin q as shown above) we can conclude that at the beginning of Phase 2 bin q of GREEDYFIT already contains q colors. Thus,

in Step 1 of Phase 2 GREEDYFIT again puts q items into each of its bins. At this point, the total number of distinct colors in the first $q-1$ bins is at least $(q-1)q + (q-1)q = 2q^2 - 2q \geq q^2$ for $q > 1$, so that $|L_1| + |L_2| \geq q^2$. As noted above, this implies the claim. □

The success of our construction heavily relies on the fact that at the beginning of each phase, bin q of GREEDYFIT contains at least q colors. We show that this is indeed true.

Lemma 8.9. *For $k \geq 1$ the following statements are true:*

(i) *At the beginning of Phase k bin q of* GREEDYFIT *contains exactly the colors from R_{k-1} (where $R_0 := \emptyset$).*

(ii) *After Step* 1 *of Phase k, each of the bins $1, \ldots, q-1$ of* GREEDYFIT *contains at least $|R_k| + |R_{k-1}| - 1$ different colors.*

(iii) *In Step* 2 *of Phase k* GREEDYFIT *packs all items into bin q.*

(iv) $|R_k| \geq q$.

Proof. The proof is by induction on k. All claims are easily seen to be true for $k = 1$. Hence, in the inductive step we assume that statements (i)–(iv) are true for some $k \geq 1$ and we consider Phase $k+1$.

(i) By the induction hypothesis (iii) all items from Step 2 presented in Phase k were packed into bin q by GREEDYFIT. Since at the end of Phase k bin q contains at least $q^2 \geq |R_k|$ items (see Lemma 8.7) and the last $|R_k|$ items presented in Phase k had different colors, it follows that at the beginning of Phase $k+1$, bin q contains *at least* all colors from R_k. On the other hand, since all the $Bq - q^2 > B$ items from Step 2 of phase k were packed into bin q by GREEDYFIT, this bin was closed during this process and consequently can only contain colors from R_k.

(ii) By Corollary 8.8 bin q is not closed before the end of Step 1. After Step 1 all colors from R_{k+1} are already in bin q by construction. Since by (i) before Step 1 also all colors from R_k were contained in bin q, it follows that bin q contains at least $|R_k| + |R_{k+1}|$ different colors at the end of Step 1.

By construction of GREEDYFIT each of the bins $1, \ldots, q-1$ must then contain at least $|R_k| + |R_{k+1}| - 1$ different colors.

(iii) When Step 2 starts, all colors from R_{k+1} are already in bin q by construction. Therefore, GREEDYFIT will initially pack items with colors from R_{k+1} into bin q as long as this bin is not yet filled up. We have to show that after bin q has been closed the number of colors in any other bin is always larger than in bin q. This follows from (ii), since by (ii) each of the bins $1, \ldots, q-1$ has at

least $|R_k| + |R_{k+1}| - 1$ colors after Step 2 of Phase $k+1$ and by the induction hypothesis (iv) the estimate $|R_k| \geq q$ holds, which gives

$$|R_k| + |R_{k+1}| - 1 \geq |R_{k+1}| + q - 1 > |R_{k+1}|.$$

(iv) At the beginning of Phase $k+1$, bin q contains exactly $|R_k|$ colors by (i). By the induction hypothesis (ii) and (iii) each of the bins $1, \ldots, q-1$ contains at least $|R_k| + |R_{k-1}| - 1 \geq |R_k|$ colors. Hence, at the beginning of Phase $k+1$, the minimum number of colors in bins $1, \ldots, q-1$ is at least the number of colors in bin q. It follows from the definition of GREEDYFIT that during Step 1 of Phase $k+1$, bin q is assigned at least the $q^2/q = q$ colors. In other words, $|R_{k+1}| \geq q$.

□

To this point we have shown that we can actually construct the sequence as suggested, and that the optimal offline cost on this sequence is no more than q. Now we need to prove that there is a number $M \in \mathbb{N}$ such that after M phases there is a bin from GREEDYFIT that contains items with $2q^2$ different colors. We will do this by establishing the following lemma:

Lemma 8.10. *In every two subsequent Phases k and $k+1$, either $|L_k \cup L_{k+1}| > 0$ or bin q contains items with $2q^2$ different colors during one of the two phases.*

Proof. Suppose that there is a Phase k in which $|L_k| = 0$. This means that all q^2 items given in Step 1 are assigned to bin q ($|R_k| = q^2$). By Lemma 8.9 (i), at the beginning of Phase $k+1$, bin q still contains q^2 different colors. If in Step 1 of Phase $k+1$ again all q^2 items are assigned to bin q, bin q contains items with $2q^2$ different colors. Recall that bin q is never closed before the end of Step 1 by Corollary 8.8. If less than q^2 items are assigned to bin q then one of the other bins gets at least one item, and $|L_{k+1}| > 0$. □

We can conclude from Lemma 8.10 that at least once every two phases the number of items in the bins 1 through $q-1$ grows. Since these bins are never closed (property 1), and all items have a unique color (property 2), after a finite number M of phases, one of the bins of GREEDYFIT must contain items with $2q^2$ different colors. This completes the proof of the Theorem 8.6. □

8.3 The trivial algorithm ONEBIN

This section is devoted to a very simple algorithm for the OlBcp, which has a better competitive ratio than GREEDYFIT. Moreover, as we will see later this algorithm achieves essentially the best competitive ratio for the problem.

Algorithm ONEBIN The algorithm uses only one open bin at any point in time. The next item r_i is packed into the open bin. A new bin is opened only if the previous item has closed the bin by filling it up completely.

The proof of the upper bound on the competitive ratio of ONEBIN is along the same lines as that of GREEDYFIT.

Lemma 8.11. *Let $\sigma = r_1, \ldots, r_m$ be any request sequence. Then for $i \geq 0$ any algorithm packs the items $r_{iB+1}, \ldots, r_{(i+1)B}$ into at most $\min\{2q-1, B\}$ bins.*

PROOF. It is trivial that the B items $r_{iB+1}, \ldots, r_{(i+1)B}$ can be packed into at most B different bins. Hence we can assume that $2q - 1 \leq B$, which means $q \leq (B-1)/2 \leq B$.

Consider the subsequence $\sigma' = r_{iB+1}, \ldots, r_{(i+1)B}$ of σ. Let ALG be any algorithm and suppose that just prior to the arrival of the first item of σ', algorithm ALG has t open bins. If $t = 0$, the claim of the lemma trivially follows, so we can assume for the rest of the proof that $t \geq 1$. Denote the open bins by $b_1, \ldots, b_t$. Let $f(b_j) \in \{1, \ldots, B-1\}$ be the number of empty places in bin b_j, $j = 1, \ldots, t$. Notice that

$$\sum_{j=1}^{t} f(b_j) \equiv 0 \bmod B. \tag{8.1}$$

Suppose that ALG uses at least $2q$ bins to distribute the items of σ'. By arguments similar to those given in Lemma 8.3, ALG can maximize the number of bins used only by closing each currently open bin and put at least one item into each of the newly opened bins. To obtain at least $2q$ bins at least $\sum_{j=1}^{t} f(b_j) + (q-t) + q$ items are required. Since σ' contains B items and $t \leq q$ it follows that

$$\sum_{j=1}^{t} f(b_j) + q \leq B. \tag{8.2}$$

Since by (8.1) the sum $\sum_{j=1}^{t} f(b_j)$ is a multiple of B and $q \geq 1$, the only possibility that the left hand side of (8.2) can be bounded from above by B is that $\sum_{j=1}^{t} f(b_j) = 0$. However, this is a contradiction to $f(b_j) \geq 1$ for $j = 1, \ldots, t$. □

As a consequence of the previous lemma we obtain the following bound on the competitive ratio of ONEBIN.

Theorem 8.12. *Algorithm* ONEBIN *is c-competitive for the* OLBCP$_{B,q}$ *where $c = \min\{2q-1, B\}$.*

PROOF. Let $\sigma = r_1, \ldots, r_m$ be any request sequence for the OLBCP$_{B,q}$ and suppose that ONEBIN$(\sigma) = w$. Let $\sigma' = r_{iB+1}, \ldots, r_{(i+1)B}$ of σ be the subsequence on

which ONEBIN gets w different colors. Clearly, σ' contains items with exactly w colors. By Lemma 8.11 OPT distributes the items of σ' into at most $\min\{2q-1, B\}$ different bins. Hence, one of those bins must be filled with at least $\frac{w}{\min\{2q-1,B\}}$ colors. □

The competitive ratio proved in the previous theorem is tight as the following example shows. Let $B \geq 2q-1$. First we give $(q-1)B$ items. The items have q different colors, every color but one occurs $B-1$ times, one color occurs only $q-1$ times. After this, in a second step q items with all the different colors used before are requested. Finally, in the third step $q-1$ items with new (previously unused) colors are given.

After the first $(q-1)B$ items by definition ONEBIN has only empty bins. The adversary assigns all items of the same color to the same bin, using one color per bin. When the second set of of q items arrives, the adversary can now close $q-1$ bins, still using only one color per bin. ONEBIN ends up with q different colors in its bin.

The adversary can assign every item given in the third step to an empty bin, thus still having only one different color per bin, while ONEBIN puts these items in the bin where already q different colors where present.

8.4 A general lower bound for deterministic algorithms

In this section we prove a general lower bound on the competitive ratio of any deterministic online algorithm for the OLBCP. We establish a lemma which immediately leads to the desired lower bound but which is even more powerful. In particular, this lemma will allow us to derive essentially the same lower bound for randomized algorithms in Section 8.5.

In the sequel we will have to refer to the "state" of (the bins managed by) an algorithm ALG after processing a prefix of a request sequence σ. To this end we introduce the notion of a *$\mathcal{C}$-configuration*.

Definition 8.13. [**$\mathcal{C}$-configuration**]
Let $\mathcal{C}$ be a set of colors. A $\mathcal{C}$-configuration is a packing of items with colors from $\mathcal{C}$ into at most q bins. More formally, a $\mathcal{C}$-configuration can be defined as a mapping $K\colon \{1,\ldots,q\} \to \mathcal{S}_B$, where

$$\mathcal{S}_B = \{\, S : S \text{ is a multiset over } \mathcal{C} \text{ containing at most } B \text{ elements from } \mathcal{C} \,\}$$

with the interpretation that $K(j)$ is the multiset of colors contained in bin j. We omit the reference to the set $\mathcal{C}$ if it is clear from the context.

We are now ready to prove the key lemma which will be used in our lower bound constructions.

Lemma 8.14. *Let $B, q, s \in \mathbb{N}$ be numbers such that $s \geq 1$ and the inequality $B/q \geq s-1$ holds. There exists a finite set $\mathcal{C}$ of colors and a constant $L \in \mathbb{N}$ with the following property: For any deterministic algorithm* ALG *and any $\mathcal{C}$-configuration K there exists an input sequence $\sigma_{\mathrm{ALG},K}$ of* OLBCP$_{B,q}$ *such that*

(i) *The sequence $\sigma_{\mathrm{ALG},K}$ uses only colors from $\mathcal{C}$ and $|\sigma_{\mathrm{ALG},K}| \leq L$, that is, $\sigma_{\mathrm{ALG},K}$ consists of at most L requests.*

(ii) *If* ALG *starts with initial C-configuration K then* $\mathrm{ALG}(\sigma_{\mathrm{ALG},K}) \geq (s-1)q$.

(iii) *If* OPT *starts with the empty configuration (i.e., all bins are empty), then* $\mathrm{OPT}(\sigma_{\mathrm{ALG},K}) \leq s$. *Additionally,* OPT *can process the sequence in such a way that at the end again the empty configuration is attained.*

Moreover, all of the above statements remain true even in the case that the online algorithm is allowed to use $q' \geq q$ bins instead of q (while the offline adversary still only uses q bins). In this case, the constants $|\mathcal{C}|$ and K depend only on q' but not on the particular algorithm ALG.

PROOF. Let $\mathcal{C}$ be a set of $(s-1)^2q^2q'$ colors and ALG be any deterministic online algorithm which starts with some initial C-configuration K. The construction of the request sequence $\sigma_{\mathrm{ALG},K}$ works in *phases*, where at the beginning of each phase the offline adversary has all bins empty.

During the run of the request sequence, a subset of the currently open bins of ALG will be *marked*. We will denote by P_k the subset of marked bins at the beginning of Phase k. Then, $P_1 = \emptyset$ and we are going to show that during some Phase M, one bin in P_M will contain at least $(s-1)q$ colors. In order to assure that this goal can in principle be achieved, we keep the invariant that each bin $b \in P_k$ has the property that the number of different colors in b plus the free space in b is at least $(s-1)q$. In other words, each bin $b \in P_k$ could potentially still be forced to contain at least $(s-1)q$ different colors. For technical reasons, P_k is only a subset of the bins with this property.

For bin j of ALG we denote by $n(j)$ the number of different colors currently in bin j and by $f(j)$ the space left in bin j. Then every bin $j \in P_k$ satisfies $n(j) + f(j) \geq (s-1)q$. By $\min P_k := \min_{j \in P_k} n(j)$ we denote the minimum number of colors in a bin from P_k.

The idea of the construction is the following (cf. Claim 8.2 below): We will force that in each phase either $|P_k|$ or $\min P_k$ increases. Hence, after a finite number of phases we must have $\min P_k \geq (s-1)q$. On the other hand, we will ensure that the optimal offline cost remains bounded by s during the whole process.

We now describe Phase k with $1 \leq k \leq q(s-1)q'$. The adversary selects a set of $(s-1)q$ new colors $C_k = \{c_1, \ldots, c_{(s-1)q}\}$ from $\mathcal{C}$ not used in any phase before and starts to present one item of each color in the order

$$c_1, c_2, \ldots, c_{(s-1)q}, c_1, c_2, \ldots, c_{(s-1)q}, c_1, c_2, \ldots \tag{8.3}$$

until one of the following cases appears:

Case 1 ALG puts an item into a bin $p \in P_k$.

In this case we let $Q := P_k \setminus \{ j \in P_k : n(j) < n(p) \}$, that is, we remove all bins from P_k which have less than $n(p)$ colors.

Notice that $\min_{j \in Q} n(j) > \min P_k$, since the number of different colors in bin p increases.

Case 2 ALG puts an item into some bin $j \notin P_k$ which satisfies

$$n(j) + f(j) \geq (s-1)q. \tag{8.4}$$

In this case we set $Q := P_k \cup \{j\}$ (that is, we tentatively add bin j to the set P_k).

Notice that after a finite number of requests one of these two cases must occur: Let $b_1, \ldots, b_t$ be the set of currently open bins of ALG. If ALG never puts an item into a bin from P_k then at some point all bins of $\{b_1, \ldots, b_t\} \setminus P_k$ are filled and a new bin, say bin j, must be opened by ALG by putting the new item into bin j. But at this moment bin j satisfies satisfies $n(j) = 1$, $f(j) = B - 1$ and hence $n(j) + f(j) = B \geq (s-1)q$ which gives (8.4).

Since the adversary started the phase with all bins empty and during the current phase we have given no more than $(s-1)q$ colors, the adversary can assign the items to bins such that no bin contains more than $s-1$ different colors. We will describe below how this is done precisely. Notice that due to our stopping criterions from above (Case 1 and Case 2) it might be the case that in fact we have presented less than $(s-1)q$ colors so far.

In the sequel we imagine that each currently open bin of the adversary has an index x, where $1 \leq x \leq q$. Let $\varphi\colon C_k \to \{1, \ldots, q\}$ be any mapping of the colors from C_k to the offline bin index such that $|\varphi^{-1}(\{x\})| \leq s-1$ for $j = 1, \ldots, q$. We imagine color c_r to "belong" to the bin with index $\varphi(c_r)$ even if no item of this color has been presented (yet). For those items presented already in Phase k, each item with color c_r goes into the currently open bin with index $\varphi(c_r)$. If there is no open bin with index $\varphi(c_r)$ when the item arrives a new bin with index $\varphi(c_r)$ is opened by the adversary to accommodate the item.

Our goal now is to clear all open offline bins so that we can start a new phase. During our clearing loop the offline bin with index x might be closed and replaced by an empty bin multiple times. Each time a bin with index x is replaced by an empty bin, the new bin will also have index x. The bin with index x receives a color not in $\varphi^{-1}(\{x\})$ at most once, ensuring that the optimum offline cost still remains bounded from above by s. The clearing loop works as follows:

1. (Start of clearing loop iteration)

   Choose a color $c^* \in C_k$ that is not contained in any bin from Q. If there is no such color, go to the "good end" of the clearing loop (Step 4).

2. Let $F \leq qB$ denote the current total empty space in the open offline bins. Present items of color c^* until one of the following things happens:

Case (a): At some point in time ALG puts the ℓth item with color c^* into a bin $j \in Q$ for some $1 \leq \ell < F$. Notice that the number of different colors in j increases. Let

$$Q' := Q \setminus \{b \in Q : n(b) < n(j)\},$$

in other words, we remove all bins b from Q which currently have less than $n(j)$ colors. This guarantees that

$$\min_{b \in Q'} n(b) > \min_{b \in Q} n(b) \geq \min P_k. \tag{8.5}$$

The adversary puts all ℓ items with color c^* into bins with index $\varphi(c^*)$. Notice that during this process the open bin with index $\varphi(c^*)$ might be filled up and replaced by a new empty bin with the same index.

Set $Q := Q'$ and go to the start of the next clearing loop iteration (Step 1). Notice that the number of colors from C_k which are not contained in Q decreases by one, but $\min_{b \in Q} n(b)$ increases.

Case (b): F items of color c^* have been presented, but ALG has not put any of these items into a bin from Q.

In this case, the offline adversary processes these items differently from Case (a): The F items of color c^* are used to fill up the exactly F empty places in all currently open offline bins. Since up to this point, each offline bin with index x had received colors only from the $s-1$ element set $\varphi^{-1}(\{x\})$, it follows that no offline bin has contained more than s different colors.

We close the clearing loop by proceeding as specified at the "standard end" (Step 3).

3. (Standard end of clearing loop iteration)

   In case we have reached this step, we are in the situation that all offline bins have been cleared, since we can originate only from Case (b) above. We set $P_{k+1} := Q$ and end the clearing loop and the current Phase k.

4. (Good end of clearing loop iteration)

   Stop the current phase and issue additional requests such that all offline bins are closed without increasing the offline cost. After this, end the sequence.

We analyze the different possible endings of the clearing loop. First we show that in case of a "good end" we have successfully constructed a sufficiently bad sequence for ALG.

Claim 8.1. *If the clearing loop finishes with a "good end", then one bin in Q contains at least $(s-1)q$ different colors.*

Proof. If the clearing loop finishes with a "good end", then we have reached the point that all colors from C_k are contained in a bin from Q. Before the first iteration,

exactly one color from C_k was contained in Q. The number of colors from C_k which are contained in bins from Q can only increase by one (which is in Case (a) above) if $\min_{b\in Q} n(b)$ increases. Hence, if all colors from C_k are contained in bins from Q, $\min_{b\in Q} n(b)$ must have increased $(s-1)q-1$ times, which implies $\min_{b\in Q} n(b) = (s-1)q$. In other words, one of ALG's bins in Q contains at least $(s-1)q$ different colors. □

What happens if the clearing loop finishes with a "standard end"?

Claim 8.2. *If the clearing loop of Phase k completes with a "standard end", then* $\min P_{k+1} > \min P_k$ *or* $|P_{k+1}| > |P_k|$.

Before we prove Claim 8.2, let us show how this claim implies the result of the lemma. Since the case $|P_{k+1}| > |P_k|$ can happen at most q' times, it follows that after at most q' phases, $\min P_k$ must increase. On the other hand, since $\min P_k$ never decreases by our construction and the offline cost always remains bounded from above by s, after at most $q(s-1)q'$ phases we must be in the situation that $\min P_k \geq (s-1)q$, which implies a "good end". Since in each phase at most $(s-1)q$ new colors are used, it follows that our initial set $\mathcal{C}$ of $(s-1)^2q^2q'$ colors suffices to construct the sequence $\sigma_{\mathrm{ALG},K}$. Clearly, the length of $\sigma_{\mathrm{ALG},K}$ can be bounded by a constant L independent of ALG and K.

Proof of Claim 8.2. Suppose that the sequence (8.3) given at the beginning of the phase was ended because Case 1 occurred, i.e., ALG put one of the new items into a bin from P_k. In this case $\min_{b\in Q} n(b) > \min P_k$. Since during the clearing loop $\min_{b\in Q} n(b)$ can never decrease and P_{k+1} is initialized with the result of Q at the "standard end" of the clearing loop, the claim follows.

The remaining case is that the sequence (8.3) was ended because of a Case 2-situation. Then $|Q| = |P_k \cup \{j\}|$ for some $j \notin P_k$ and hence $|Q| > |P_k|$. During the clearing loop Q can only decrease in size if $\min_{i\in Q} n(i)$ increases. It follows that either $|P_{k+1}| = |P_k| + 1$ or $\min P_{k+1} > \min P_k$ which is what we claimed. □

This completes the proof of Lemma 8.14. □

As an immediate consequence of Lemma 8.14 we obtain the following lower bound result for the competitive ratio of any deterministic algorithm:

Theorem 8.15. *Let $B, q, s \in \mathbb{N}$ such that $s \geq 1$ and the inequality $B/q \geq s-1$ holds. No deterministic algorithm for* OLBCP$_{B,q}$ *can achieve a competitive ratio less than $\frac{s-1}{s} \cdot q$. The competitive ratio of any deterministic algorithm for fixed B and q is at least $\left(1 - \frac{q}{B+q}\right) q$. For the general case with no restrictions on the relation of the capacity B to the number of bins q, there can be no deterministic algorithm for* OLBCP$_{B,q}$ *that achieves a competitive ratio less than q. Even if the online algorithm*

is allowed to use an arbitrary (but fixed) number $q' \geq q$ of open bins, these lower bounds are valid.

8.5 A general lower bound for randomized algorithms

In this section we show lower bounds for the competitive ratio of any randomized algorithm against an oblivious adversary for the $\textsc{OlBcp}_{B,q}$.

Theorem 8.16. *Let $B, q, s \in \mathbb{N}$ such that $s \geq 1$ and the inequality $B/q \geq s-1$ holds. Then no randomized algorithm for $\textsc{OlBcp}_{B,q}$ can achieve a competitive ratio less than $\frac{s-1}{s}q$ against an oblivious adversary.*

In particular for fixed B and q, the competitive ratio against an oblivious adversary is at least $\left(1 - \frac{q}{B+q}\right) q$.

All of the above claims remain valid, even if the online algorithm is allowed to use an arbitrary (but fixed) number $q' \geq q$ of open bins.

Proof. Let $\mathcal{A} := \{ \mathrm{ALG}_y : y \in \mathcal{Y} \}$ be the set of deterministic algorithms for the $\textsc{OlBcp}_{B,q}$. We will show that there is a probability distribution X over a certain set of request sequences $\{ \sigma_x : x \in \mathcal{X} \}$ such that for any algorithm $\mathrm{ALG}_y \in \mathcal{A}$

$$\mathbb{E}_X \left[\mathrm{ALG}_y(\sigma_x)\right] \geq (s-1)q,$$

and, moreover,

$$\mathbb{E}_X \left[\mathrm{OPT}(\sigma_x)\right] \leq s.$$

The claim of the theorem then follows by Yao's Principle (see Section 3.3).

Let us recall the essence of Lemma 8.14. The lemma establishes the existence of a finite color set $\mathcal{C}$ and a constant L such that for a fixed $\mathcal{C}$-configuration K, any deterministic algorithm can be "fooled" by one of at most $|\mathcal{C}|^L$ sequences. Since there are no more than $|\mathcal{C}|^{qB}$ configurations, a *fixed finite* set of at most $N := |\mathcal{C}|^{L+qB}$ sequences $\Sigma = \{\sigma_1, \ldots, \sigma_N\}$ suffices to "fool" *any* deterministic algorithm provided the initial configuration is known.

Let X be a probability distribution over the set of finite request sequences

$$\{ \sigma_{i_1}, \sigma_{i_2}, \ldots, \sigma_{i_k} : k \in \mathbb{N}, 1 \leq i_j \leq N \}$$

such that σ_{i_j} is chosen from Σ uniformly and independently of all previous subsequences $\sigma_{i_1}, \ldots, \sigma_{i_{j-1}}$. We call subsequence σ_{i_k} the *kth phase*.

Let $\mathrm{ALG}_y \in \mathcal{A}$ be arbitrary. Define ϵ_k by

$$\epsilon_k := Pr_X \left[\begin{array}{c} \mathrm{ALG}_y \text{ has at least one bin with} \\ \text{at least } (s-1)q \text{ colors during Phase } k \end{array} \right].$$

The probability that ALG_y has one bin with at least $(s-1)q$ colors on any given phase is at least $1/N$, whence $\epsilon_k \geq 1/N$ for all k. Let

$$p_k := Pr_X[\mathrm{ALG}_y(\sigma_{i_1} \ldots \sigma_{i_{k-1}} \sigma_{i_k}) \geq (s-1)q].$$

Then the probabilities p_k satisfy the following recursion:

$$p_0 = 0 \tag{8.6}$$

$$p_k = p_{k-1} + (1 - p_{k-1})\epsilon_k \tag{8.7}$$

The first term in (8.7) corresponds to the probability that ALG_y has already cost at least $(s-1)q$ after Phase $k-1$, the second term accounts for the probability that this is not the case but cost at least $(s-1)q$ is achieved in Phase k. By construction of X, these events are independent. Since $\epsilon_k \geq 1/N$ we get that

$$p_k \geq p_{k-1} + (1 - p_{k-1})/N. \tag{8.8}$$

It is easy to see that any sequence of real numbers $p_k \in [0, 1]$ satisfying (8.6) and (8.8) must converge to 1. Hence, also the expected cost $\mathbb{E}_X[\mathrm{ALG}_y(\sigma_x)]$ converges to $(s-1)q$. On the other hand, the offline costs remain bounded by s by the choice of the σ_{i_j} according to Lemma 8.14. □

8.6 Discussion

In this chapter we showed that algorithm ONEBIN has a better worst case performance than algorithm GREEDYFIT. We concluded that this is surprising, suggesting that GREEDYFIT is the superior algorithm of the two. However, we have no evidence of any kind that this is indeed the case. It would be very interesting to compare the average performance of these algorithms. So far we have not been able to get any results in this direction.

9

Overview and open problems

A major part of this thesis is concerned with variants of the online traveling salesman problem, exceptions being the introductory Chapters 1 and 3, Chapter 2 where dial-a-ride problems are introduced and thoroughly investigated from an offline point of view, and Chapter 8 where a different kind of vehicle routing problem is studied. We studied the OlTsp under the objective functions of minimizing the makespan, the sum of completion times, and the maximum flow time. In this chapter, we give an overview of the results that are known for each objective, and point out some interesting open problems in this field.

For the OlTsp with the objective to minimize the makespan, a 2-competitive algorithm exists for general metric spaces, and no algorithm can have a competitive ratio of less than 2 [4]. For the dial-a-ride generalization, the same bounds hold [3]. Special metric spaces allow for lower competitive ratios: on the line there is a $9 + \sqrt{17}/8$-competitive algorithm [36] and a matching lower bound [4], and on the half line we have a tight ratio of 3/2 (Chapter 4).

For some years there had been a gap between the best known lower bound of $9 + \sqrt{17}/8 \approx 1.64$ for the OlTsp on the line due to Ausiello et al. [4], and the competitive ratio of 7/4 of their best algorithm. To understand better the intuition that, in order to close this gap, an algorithm had to be designed that sometimes waits patiently for more information to be released, we studied a class of algorithms that we called *zealous*. Algorithms in this class are not allowed to be idle when there are still unserved requests left. The 7/4 competitive algorithm of Ausiello et al. [4] belongs to this class. We showed that no zealous algorithm can have a better competitive ratio than 7/4, and therefore that closing the gap between the

upper and lower bound indeed required a non-zealous algorithm (Chapter 4). In [36], Lipmann gives a non-zealous algorithm that has a competitive ratio equal to the lower bound.

As mentioned in Chapter 3, one of the disadvantages of using competitive analysis is its pessimistic character. Algorithms are only judged on their performance on worst case instances, and are therefore often tailored not to fail on such evil instances, rather than to perform good on average. In the OLTSP, the adversary exhibits very unfair behaviour in his solution for the best known worst case input sequence: he moves away from all previously released requests, and when his distance to the online server is big enough, he releases a request right at his feet. In an attempt to rule out extremes in the input, we designed the *fair adversary* for the OLTSP, who is restricted in his freedom to move: at any moment, he is not allowed to move outside the convex hull of the requests released so far. Requests on which even a smart online algorithm cannot anticipate will also increase the cost of a fair adversary. It turned out that sometimes this additional restriction on the adversary indeed helps to get better competitive ratios. In Chapter 4, we show that for the OLTSP on the half line a strictly smaller competitive ratio can be obtained against a fair adversary than against the traditional adversary.

All together we have a good understanding of the OLTSP when minimizing the makespan. We know what algorithms have best possible worst case performance for the problem on a number of metric spaces, we know the importance of waiting for information rather than behaving zealous, and we know that, at least for the problem on the half line, all worst case instances have in common that the adversary behaves unfairly. This, however, does not mean that there are no more interesting questions left. First of all, virtually nothing is known about the performance of randomized algorithms. There are only a few trivial lower bounds, but since these are weaker than the deterministic lower bounds, we can still hope for randomized algorithms with better competitive ratios than their deterministic counterparts. Secondly, there is nothing known about the OLTSP with more than one server. It is unclear if the results for the one-server problem can be generalized. Finally, for practical use it would be interesting to study the problem where the class of online algorithms is restricted to those running in polynomial time. In [4], the authors give 3-competitive polynomial algorithms for the OLTSP, both in the setting where the server has to return to the origin after serving all requests, and in the setting that this is not required. There are no lower bound results when restricting to polynomial time online algorithms, better than the general lower bounds. It would be interesting to find out if the current lower bounds can also be attained by polynomial time algorithms.

The situation is a bit different when we minimize the sum of completion times. For a long time, the best known algorithm was 9-competitive [12], and this algorithm only worked on the line. For the dial-a-ride generalization, this algorithm also works, but the competitive ratio for this problem is 15 [12]. The gaps with the best known lower bounds of $1 + \sqrt{2}$ and 3 for the problem with coinciding sources and destinations and the dial-a-ride version, respectively [12], are rather

large. In Chapter 6 we improve on the competitive ratio. We use techniques from scheduling to obtain a $(1+\sqrt{2})^2 \approx 5.83$-competitive deterministic algorithm and a $4/\ln 3 \approx 3.64$-competitive randomized algorithm for the dial-a-ride problem. These algorithms also work for the dial-a-ride generalization, even if there are weights on the requests. However, the gap with the lower bound is still quite large, and the algorithms are somewhat unsatisfactory: they do not use all the information that is available, they just compute and follow optimal tours at given times, ignoring all requests that are released along the way. It seems necessary to design a more sophisticated algorithm to get closer to the lower bound. The biggest challenge on this problem is therefore to analyze such an algorithm. This is particularly interesting, since it is closely related to finding and analyzing a good approximation algorithm for the offline TRP problem. The best known result for this problem is from Goemans and Kleinberg [17].

Finally we study the objective of minimizing the maximum flow time of requests for the problem on the real line in Chapter 7. This is a very interesting objective, since it can be associated with dissatisfaction of customers. Trivially, there can be no competitiveness results for this problem, even when the metric space is the line and requests consist of points only. This is the reason that the problem has not been studied from a worst case point of view. However, if we impose a rather strict but in a sense minimal restriction on the power of the adversary, things look different. If we require the adversary to be fair, as we did for the problem of minimizing the makespan, still no constant competitive algorithm can exist. Therefore we inflict a stronger restriction on the adversary, namely that he is not allowed to move in a direction where no unserved request is waiting. We call such an adversary a *non-abusive adversary*, and show that against such an adversary a constant competitive algorithm exists, when requests consist of points only. For this seemingly simple problem, we need a rather complicated algorithm, and we need a very technical analysis to show that the algorithm is 8-competitive. Although this is still far from the lower bound of 2 (Chapter 7), this result is encouraging: apparently, this reasonable restriction on the adversary allows for constant competitive algorithms. We note that the dial-a-ride generalization of this problem can be seen as a model of an elevator. Algorithms that have a good competitive ratio for this problem can be very useful in practice. Especially when this problem is combined with the information model introduced in Chapter 5, we obtain a very realistic elevator model. Following the line of research of Chapter 5 can lead to interesting results.

Bibliography

[1] F. Afrati, S. Cosmadakis, C.H. Papadimitriou, G. Papageorgiou, and N. Papakonstantinou, *The complexity of the traveling repairman problem*, Theoretical Informatics and Applications **20** (1986), 79–87.

[2] S. Arora and G. Karakostas, *Approximation schemes for minimum latency problems*, Proceedings of the 31st Annual ACM Symposium on the Theory of Computing, 1999, pp. 688–693.

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

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

[5] M. Blom, S.O. Krumke, W.E. de Paepe, and L. Stougie, *The online-*TSP *against fair adversaries*, INFORMS Journal on Computing **13** (2001), 138–148.

[6] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan, *The minimum latency problem*, Proceedings of the 26th Annual ACM Symposium on the Theory of Computing, 1994, pp. 163–171.

[7] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*, Cambridge University Press, Cambridge, 1998.

[8] S. Chakrabarti, C.A. Phillips, A.S. Schulz, D.B. Shmoys, C. Stein, and J. Wein, *Improved scheduling algorithms for minsum criteria*, Proceedings of the 23rd International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science, vol. 1099, 1996, pp. 646–657.

[9] B. Chen, A.P.A. Vestjens, and G.J. Woeginger, *On-line scheduling of two-machine open shops where jobs arrive over time*, Journal of Combinatorial Optimization **1** (1997), 355–365.

[10] R.W. Conway, W.L. Maxwell, and L.W. Miller, *Theory of scheduling*, Addison-Wesley, Reading, 1967.

[11] S.A. Cook, *The complexity of theorem-proving procedures*, Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing, 1971, pp. 151–158.

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

[13] A. Fiat and G.J. Woeginger (eds.), *Online algorithms: The state of the art*, Lecture Notes in Computer Science, vol. 1442, Springer, 1998.

[14] M.R. Garey and D.S. Johnson, *Complexity results for multiprocessor scheduling under resource constraints*, SIAM Journal on Computing **4** (1975), 397–411.

[15] M.R. Garey and D.S. Johnson, *Computers and intractability: A guide to the theory of* NP*-completeness*, W.H. Freeman Company, San Francisco, 1979.

[16] M.R. Garey, D.S. Johnson, G.L. Miller, and C.H. Papadimitriou, *The complexity of coloring circular arcs and chords*, SIAM Journal on Algebraic and Discrete Methods **1** (1980), 216–227.

[17] M. Goemans and J. Kleinberg, *An improved approximation ratio for the minimum latency problem*, Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms, 1996, pp. 152–158.

[18] T. Gonzalez, *Optimal mean finish time preemptive schedules*, Technical Report 220, Computer Science Department, Pennsylvania State University, 1977.

[19] R.L. Graham, *Bounds for certain multiprocessor anomalies*, Bell System Technical Journal **45** (1966), 1563–1581.

[20] 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*, Annals of Discrete Mathematics **5** (1979), 287–326.

[21] L.A. Hall, A.S. Schulz, D.B. Shmoys, and J. Wein, *Scheduling to minimize average completion time: Off-line and on-line approximation algorithms*, Mathematics of Operations Research **22** (1997), 513–544.

[22] L.A. Hall, D.B. Shmoys, and J. Wein, *Scheduling to minimize average completion time: Off-line and on-line algorithms*, Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms, 1996, pp. 142–151.

[23] D. Hauptmeier, S. O. Krumke, and J. Rambau, *The online dial-a-ride problem under reasonable load*, Theoretical Computer Science (2002), to appear.

[24] D.P. Heyman and M.J. Sobel, *Stochastic models in operations research, Volume II: Stochastic Optimization*, McGraw Hill, New York, 1984.

[25] J.A. Hoogeveen and A.P.A. Vestjens, *Optimal on-line algorithms for single-machine scheduling*, Proceedings of the 5th Mathematical Programming Society Conference on Integer Programming and Combinatorial Optimization, Lecture Notes in Computer Science, vol. 1084, 1996, pp. 404–414.

[26] J.R. Jackson, *Scheduling a production line to minimize maximum tardiness*, Management science research project, research report 43, University of California, Los Angeles, 1955.

[27] R.M. Karp, *Reducibility among combinatorial problems*, Complexity of Computer Computations (R.E. Miller and J.W. Thatcher, eds.), Plenum Press, New York, 1972, pp. 85–103.

[28] D.G. Kendall, *Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded markov chain*, Annals of Mathematical Statistics **24** (1953), 338–354.

[29] E. Koutsoupias and C. Papadimitriou, *Beyond competitive analysis*, Proceedings of the 35th Annual IEEE Symposium on the Foundations of Computer Science, 1994, pp. 394–400.

[30] S.O. Krumke, *Online optimization: Competitive analysis and beyond*, Habilitationsschrift, Technische Universität Berlin, 2001.

[31] S.O. Krumke, W.E. de Paepe, D. Poensgen, and L. Stougie, *News from the online traveling repairman*, Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science, vol. 2136, 2001, pp. 487–499.

[32] S.O. Krumke, W.E. de Paepe, L. Stougie, and J. Rambau, *Online bin coloring*, Proceedings of the 9th Annual European Symposium on Algorithms, Lecture Notes in Computer Science, vol. 2161, 2001, pp. 74–85.

[33] B.J. Lageweg, J.K. Lenstra, E.L. Lawler, and A.H.G. Rinnooy Kan, *Computer-aided complexity classification of combinatorial problems*, Communications of the ACM **25** (1982), 817–822.

[34] E.L. Lawler and J.M. Moore, *A functional equation and its application to resource allocation and sequencing problems*, Management Science **16** (1969), 77–84.

[35] M. Lipmann, personal communication.

[36] ———, *The online traveling salesman problem on the line*, Master's thesis, University of Amsterdam, Faculty of Economics and Econometrics, 1999.

[37] M. Lipmann, L. Laura, A. Marchetti-Spaccamela, S.O. Krumke, W.E. de Paepe, D. Poensgen, and L. Stougie, *Non-abusiveness helps: An o(1)-competitive algorithm for minimizing the maximum flow time in the online traveling salesman problem*, Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization, Lecture Notes in Computer Science, 2002, to be published.

[38] M. Lipmann, X. Lu, W.E. de Paepe, R. Sitters, and L. Stougie, *Online dial-a-ride problems under a restricted information model*, Proceedings of the 10th Annual European Symposium on Algorithms, Lecture Notes in Computer Science, 2002, to be published.

[39] J.W.S. Liu and C.L. Liu, *Performance analysis of heterogenous multi-processor computing systems*, Computer architectures and networks (E. Gelenbe and R. Mahl, eds.), North Holland, Amsterdam, 1974, pp. 331–343.

[40] M. Middendorf, *More on the complexity of common superstring and supersequence problems*, Theoretical Computer Science **125** (1994), 205–228.

[41] W.E. de Paepe, *Computer-aided complexity classification of dial-a-ride problems*, Master's thesis, University of Amsterdam, Faculty of Economics and Econometrics, 1998.

[42] W.E. de Paepe, J.K. Lenstra, J. Sgall, R.A. Sitters, and L. Stougie, Website, http://www.win.tue.nl/darclass.

[43] C.H. Papadimitriou, *Computational complexity*, Addison Wesley, Reading, 1994.

[44] C. Philips, C. Stein, E. Torng, and J. Wein, *Optimal time-critical scheduling via resource augmentation*, Proceedings of the 29th Annual ACM Symposium on the Theory of Computing, 1997, pp. 140–149.

[45] C. Philips, C. Stein, and J. Wein, *Minimizing average completion time in the presence of release dates*, Proceedings of the 4th Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science, vol. 955, 1995, pp. 86–97.

[46] K. Pruhs and B. Kalyanasundaram, *Speed is as powerful as clairvoyance*, Proceedings of the 36th Annual IEEE Symposium on the Foundations of Computer Science, 1995, pp. 214–221.

[47] S. Seiden, *A guessing game and randomized online algorithms*, Proceedings of the 32nd Annual ACM Symposium on the Theory of Computing, 2000, pp. 592–601.

[48] R.A. Sitters, *Two* NP*-hardness results for preemptive minsum scheduling of unrelated parallel machines*, Proceedings of the 8th Mathematical Programming Society Conference on Integer Programming and Combinatorial Optimization, Lecture Notes in Computer Science, 2001, pp. 396–405.

[49] ———, *The minimum latency problem is* NP*-hard for weighted trees*, Proceedings of the 9th Mathematical Programming Society Conference on Integer Programming and Combinatorial Optimization, Lecture Notes in Computer Science, 2002, to appear.

[50] D.D. Sleator and R.E. Tarjan, *Amortized efficiency of list update and paging rules*, Communications of the ACM **28** (1985), no. 2, 202–208.

[51] W.E. Smith, *Various optimizers for single-stage production*, Naval Research Logistics Quarterly **3** (1956), 59–66.

[52] A.C-C. Yao, *Probabilistic computations: Towards a unified measure of complexity*, Proceedings of the 17th Annual Symposium on Foundations of Computer Science, 1977, pp. 222–227.

Samenvatting

Dit proefschrift staat in het teken van voertuigrouteringsproblemen. Dit zijn problemen waarin gevraagd wordt goederen te transporteren van een startpunt naar een bestemming, door één of meer voertuigen. De uitvoering van deze ritten is onderhevig aan eventuele randvoorwaarden, zoals restricties op de volgorde waarin de ritten worden uitgevoerd of restricties op de starttijden en eindtijden van de ritten. Tegelijkertijd moet een doelfunctie die de kosten van een oplossing weergeeft worden geminimaliseerd. De klasse van voertuigrouteringsproblemen is erg breed. Ook problemen waarbij niet expliciet voertuigen en goederen betrokken zijn, kunnen binnen deze klasse gemodelleerd worden.

In dit proefschrift ligt de nadruk op een deelklasse van de voertuigrouteringsproblemen, de *dial-a-ride* problemen. Deze problemen zijn relatief eenvoudig van structuur en worden in hoofdstuk 2 geïntroduceerd. Alleen in hoofdstuk 8 wordt een voertuigrouteringsprobleem bestudeerd dat niet tot de dial-a-ride problemen behoort.

Voertuigrouteringsproblemen kunnen zowel een *offline* als een *online* karakter hebben. Offline problemen zijn statisch: alle gegevens over het probleem zijn bekend op het moment dat een oplossing gezocht moet gaan worden, en de oplossing kan in een keer worden gegeven. Online problemen zijn dynamisch: slechts een deel van de gegevens is beschikbaar op het moment dat begonnen wordt met het zoeken naar een oplossing van het probleem. Terwijl aan een oplossing wordt gewerkt, komt aanvullende informatie beschikbaar. De oplossing komt gaandeweg tot stand.

In hoofdstuk 2 bestuderen we de complexiteit van offline dial-a-ride problemen. We voeren een notatie in voor dial-a-ride problemen en genereren aan de hand hiervan een brede klasse van problemen. Hierop definiëren we vervolgens een partiële ordening. De gedachte achter deze partiële ordening is dat problemen hoger in de ordening moeilijker zijn dan problemen lager in de ordening. Het doel van dit onderzoek is om de grens te bepalen tussen polynomiaal oplosbare problemen en NP-moeilijke problemen. Zodra deze grens bepaald is, kan aan de hand van de partiële ordening voor elk probleem in de probleemklasse bepaald worden of het gemakkelijk of moeilijk is.

In het tweede deel van dit proefschrift bestuderen we de kwaliteit van online algoritmen voor specifieke online voertuigrouteringsproblemen. Deze kwaliteit wordt bepaald aan de hand van competitiviteitsanalyse. Daarbij wordt de oplossing van een online algoritme vergeleken met de oplossing van een optimaliseringsalgoritme. Een algoritme heeft een competitiviteitsratio van c wanneer in het slechtste geval de online oplossing ten hoogste c keer zo slecht is als een optimale oplossing. De competitiviteitsratio moet gezien worden als een manier om de kwaliteit van ver-

schillende algoritmen met elkaar te vergelijken. Naast de competitiviteitsratio's van online algoritmen zijn we geïnteresseerd in ondergrenzen op competitiviteitsratio's. Deze ondergrenzen geven de kosten aan die het niet hebben van informatie over de input met zich meebrengt, ongeacht het gebruikte online algoritme.

De problemen die we bestuderen in hoofdstuk 4 tot en met 7 zijn varianten van het online handelsreizigersprobleem. We bekijken dit probleem onder verschillende doelstellingen. In hoofdstuk 4 en 5 minimaliseren we de tijd waarop alle ritten zijn uitgevoerd en de voertuigen weer terug zijn in de positie waarin ze begonnen zijn. In hoofdstuk 6 minimaliseren we de som van de completeringstijden van de ritten, en in hoofdstuk 7 de maximale tijd tussen het bekend worden van een rit en zijn completeringstijd. Soms beperken we ons tot een speciale metrische ruimte waarin de voertuigen bewegen, tot speciale klassen van online algoritmen of tot speciale klassen van optimaliseringsalgoritmen. De reden hiervoor varieert. Soms kunnen er zonder deze restricties aantoonbaar geen positieve resultaten worden behaald, en soms is er over het algemene probleem al veel bekend of is het algemene probleem simpelweg te moeilijk. De studie van een gerestricteerd probleem kan van belang zijn voor een toepassing of het kan het inzicht in de structuur van het algemene probleem vergroten. De competitiviteitsratio onder deze restricties geeft andere informatie dan standaard competitiviteitsanalyse.

Tenslotte bestuderen we in hoofdstuk 8 het online bin coloring probleem. Dit probleem behoort niet tot de klasse van dial-a-ride problemen, maar kan wel worden gezien als een voertuigrouteringsprobleem.

Curriculum vitae

Willem Eduard de Paepe was born on September 6th 1975, in Purmerend, the Netherlands. In 1993 he received his V.W.O. diploma at the St. Ignatius College in Purmerend. From September 1993 to May 1998 he was a student at the University of Amsterdam, where he received his Master's degree in Operations Research and Management. During the last three years of this period he was a student assistant at the University of Amsterdam. In July 1998 he started as a Ph.D. student at the department of Technology Management of the Technische Universiteit Eindhoven on a joint project with the department of Mathematics and Computer Science, under the supervision of dr. L. Stougie and prof.dr. J.K. Lenstra. The results of his research are presented in this thesis. In 2001, he visited the department of Computer Science of La Sapienza University of Rome for two months.

Stellingen

behorende bij het proefschrift

Complexity Results and Competitive Analysis for Vehicle Routing Problems

door

W.E. de Paepe

I

Voor het online handelsreizigersprobleem bestaat er geen deterministisch algoritme met een competitiviteitsratio van minder dan 2 op algemene metrische ruimten.[1] Wanneer de onderliggende metrische ruimte echter de reële lijn is, bestaat er een $(9 + \sqrt{17})/8$-competitief deterministisch algoritme.[2] Uit het bewijs van stelling 5.2 van dit proefschrift volgt dat de ondergrens van 2 ook geldt wanneer de onderliggende metrische ruimte een boom is.

II

Voor het taartsnijdprobleem met n eters waarbij een taart precies $n - 1$ keer gesneden wordt, bestaat een "$1/(2n-2)$-eerlijk" protocol voor $n \geq 2$. Er bestaat geen β-eerlijk protocol met $\beta > 1/(2n - 2)$.[3]

III

Beschouw K_n, de volledige graaf op n knopen, met $d(i, j) = 1$ voor de punten i en j met $i \neq j$. Op deze ruimte bestaat geen constant competitief deterministisch algoritme voor het online dial-a-ride probleem met de doelstelling de maximale tijd van vrijgeven tot voltooien over alle ritten te minimaliseren wanneer de capaciteit van het voertuig gelijk is aan 1. Dit resultaat blijft gelden, zelfs wanneer $n = 3$, de ritten ongericht zijn, en de tegenstander "non-abusive" is, zoals gedefinieerd in paragraaf 7.1 van dit proefschrift. In het speciale geval echter dat voor elke rit het startpunt en het eindpunt van de rit samen valt, is de first-come-first-served strategie 2-competitief.

IV

Voor het online dial-a-ride probleem uit stelling III met de reële lijn als onderliggende metrische ruimte bestaat geen constant competitief deterministisch algoritme wanneer het voertuig capaciteit 1 heeft, zelfs niet wanneer de ritten ongericht zijn. Wanneer het voertuig echter onbegrensde capaciteit heeft en de ritten ongericht zijn, is het in hoofdstuk 7 beschreven algoritme DETOUR 8-competitief.

V

Haastige spoed is zelden goed.[4]

VI

"Life is what happens while you are making other plans" – John Lennon.

VII

Het formuleren van vragen bij een proefschrift zou een goed alternatief zijn voor het formuleren van stellingen.

VIII

Het behalen van een middelbare-schooldiploma in wiskunde B is geen noodzakelijke voorwaarde voor een promotie aan een faculteit wiskunde en informatica.

IX

Grote smurf is een communist.

X

Roken voorkomt RSI.

Literatuur

1. G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, en M. Talamo, *Algorithms for the on-line traveling salesman*, Algorithmica **29** (2001), 560–581.

2. M. Lipmann, *The online traveling salesman problem on the line*, Scriptie, Universiteit van Amsterdam, Faculteit der Economische Wetenschappen en Econometrie, 1999.

3. S.O. Krumke, M. Lipmann, W.E. de Paepe, D. Poensgen, J. Rambau, L. Stougie, en G.J. Woeginger. *How to cut a cake almost fairly.* Proceedings of the 13th Annual Symposium on Discrete Algorithms, 2002.

4. Dit proefschrift, tabel 4.1 en 4.2.