Online Optimization
Competitive Analysis and Beyond

Sven O. Krumke

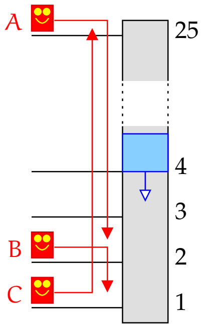
HABILITATIONSSCHRIFT TECHNISCHE UNIVERSITÄT BERLIN 2001

Contents

0

Introduction

A Simple Problem?



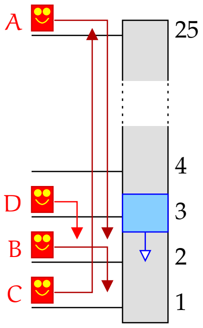
The initial situation: A, B, and C request transportations.

A situation which many of us know: You overslept, you are already late for the morning meeting, all traffic lights are on red, and once you finally reach the office building it takes ages for the elevator to arrive. Who on earth designed this elevator control system? There must be a way to craft a perfect elevator control with a little bit of mathematics!

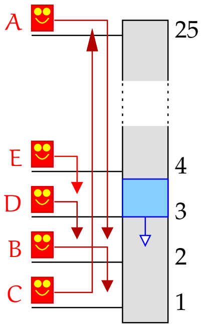
This thesis deals with the question whether the seemingly simple problem is really so simple.

Let us consider the situation of an elevator that runs in a 25 floor building and which currently is waiting at the fourth floor. Person A requests a transportation from the penthouse in the 25th floor down to the second floor, B wants to be moved from the second floor to the first, and C wants to go upwards from the first floor to the penthouse.

Our current task is easy: we obtain a shortest transportation, if we first pick up B, then C, and finally take care of A.



D appears . . .



Is this the future?

However, just a second after we pass the third floor on our way down, suddenly D appears at the third floor and wants to be carried to the second floor. Hmm. . . , what shall we do? Should we first complete our initial transportation schedule and care about D later? Or should we reverse direction, and pick up D first? In any case we waste valuable time since we travel unnecessary distance which we could have avoided *if we had known in advance* that (and when) D showed up.

We have just discovered the *online aspect* of the elevator problem. We are facing incomplete information, and even if every time a new request becomes known we compute a new "optimal" schedule this does not necessarily lead to an overall optimal solution. Suppose that in our particular case D were actually the last transportation request and our goal would be to finish serving requests as early as possible. Then, *in hindsight* (or *with clairvoyance*) the best solution would have been to *wait* at the third floor for one second until D arrived.

At the moment our most promising option looks like reversing gear and handling D first (after all, we have just lost one second right now compared to a clairvoyant controller). But what if E shows up on the fourth floor, just before we pick up D? Should we then serve E first?

Perhaps the elevator problem is not so trivial!

Online Computation

online optimization

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

request sequence

online paradigm

In online optimization the input is modeled as a finite *request sequence* $r_1, r_2, \dots$ which must be served and which is revealed step by step to an online algorithm. How this is done exactly, depends on the specific *online paradigm*. The two most common models are the *sequence model* and the *time stamp model*.

sequence model

Let ALG be an online algorithm. In the *sequence model* requests must be served in the order of their occurrence. More precisely, when serving request r_j, the online algorithm ALG does not have any knowledge of requests r_i with $i > j$ (or the total number of requests). When request r_j is presented it must be served by ALG according to the specific rules of the problem. The serving of r_j incurs a "cost" and the overall goal is to minimize the total service cost.[1] The decision by ALG of how to serve r_j is irrevocable. Only after r_j has been served, the next request r_{j+1} becomes known to ALG.

time stamp model

In the *time stamp model* each request has a *arrival* or *release time* at which it becomes available for service. The release time $t_j \geq 0$ is a nonnegative real number and specifies the time at which request r_j is released (becomes known to an online algorithm). An online algorithm ALG must determine its behavior at a certain moment t in time as a function of all the requests released up to time t and of the current time t. Again, we are in the situation that an online algorithm ALG is confronted with a finite sequence $r_1, r_2, \dots$ of requests which is given in order of non-decreasing release times and the service of each request incurs a cost for ALG. The difference to the sequence model is that the online algorithm is allowed to wait and to revoke decisions and that requests need not be served in the order of their occurrence.

[1] It is also possible to define online profit-maximization problems. For those problems, the serving of each request yields a profit and the goal is to maximize the total profit obtained.

Waiting incurs additional costs, typically depending on the elapsed time. Previously made decisions may, of course, only be revoked as long as they have not been executed.

Analysis of Online Algorithms

Online problems had been studied already explicitly or implicitly during the nineteen-seventies and nineteen-eighties. However, broad systematic investigation only started when Sleator and Tarjan [ST85] suggested comparing an online algorithm to an *optimal offline algorithm*, thus laying the foundations of *competitive analysis*. The actual term "competitive analysis" was coined in the paper [KM$^+$88].

competitive analysis

An online algorithm ALG is called c-*competitive* if the objective function value of the solution produced by ALG on any input sequence is at most c times that of an optimal offline algorithm on the same input. Here, the "optimal offline algorithm" has complete knowledge about the whole input sequence.

c-competitive

Observe that in the above definition there is no restriction on the computational resources of an online algorithm. The only scarce resource in competitive analysis is information. Competitive analysis of online algorithms can be imagined as a game between an online player and a malicious offline *adversary*. The online player uses an online algorithm to process an input which is generated by the adversary. If the adversary knows the (deterministic) strategy of the online player, he can construct a request sequence which maximizes the ratio between the player's cost and the optimal offline cost. For randomized algorithms we have to define what kind of information about the online player is available to the adversary. This leads to different adversary models which we discuss in Chapter 1.

adversary

The "Simple Problem" Revisited

Let us return to the initial elevator problem. This problem is most adequately formulated within the time stamp model: at time $t \geq 0$ all transportation requests with release time at most t are known to an online algorithm. We assume for the moment that our objective function is to minimize the time when the last request has been served. This objective is usually referred to as the *makespan*.

makespan

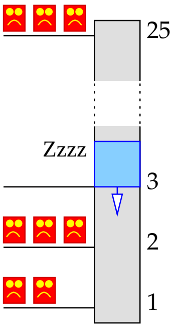
We put ourselves into the situation of an online algorithm which controls the elevator. In the initial situation where A, B, and C requested transportation we do not have any knowledge whether additional requests will be released in the future. If we start to move the elevator downwards immediately, then the advent of request D makes us lose some time compared to the overall best possible solution for $\sigma = A, B, C, D$ (if this is the entire sequence). On the other hand, if we wait at our current position in the fourth floor, this waiting time is wasted in case D never shows up. This argument shows that, in general, no online algorithm is capable of guaranteeing an optimal solution. Expressed in terms of competitive analysis this means that no online algorithm can be 1-competitive for the elevator problem.

But how well can "the best online algorithm" perform compared to the all-knowing optimal offline algorithm? Can we show that there is a constant $c > 1$ such that, no matter what the elevator control looks like, there is always a bad situation where this online strategy yields costs at least c-times the cost of an optimal offline algorithm? This brings us to one of the central issues of competitive analysis, which we are going to investigate in this thesis in the context of a number of online problems:

How much does one lose by not having complete information?

Real Time Issues

A natural approach for an online algorithm in our particular problem is the following strategy: As soon as a new request arrives, the elevator completes the current carrying move (if it is performing one), then it "replans": it computes a new shortest transportation which starts at the current position of the elevator and takes care of all known but yet unserved requests. As we will see in Chapter 2, the "replan"-strategy is in fact competitive for the minimization of the makespan, although it is not "the best" algorithm from a competitive analysis point of view.



Long computation stalls the elevator system.

Up to this point we have neglected the time complexity of algorithms. Competitive analysis approaches online computation from an information theoretic point of view. However, for instance the implementation of the REPLAN-strategy in a real elevator system raises a new issue. If recomputation of a new transportation schedule takes very long time, then the whole system will be stalled, since during the computation the elevator is idle.

real-time problem

A *real-time problem* (or real-time system) is a problem where an online algorithm is required to deliver the next piece of the solution within a very tight time bound, i.e., it is required to react in real-time. The correct behavior of a real-time system depends on the quality of the solution as well as on the time needed for producing the solution. A solution provided too late may be useless or, in some cases, even dangerous because it does not fit to the current system parameters which may vary over time.

Unfortunately, determining an optimal solution in real-time seems to be intractable for many problems due to the problems' computational complexity. This raises the interest in algorithms that *work fast* (at least in polynomial time) but nevertheless *provide a guarantee* on the quality of the delivered solution.

The competitive ratio has an analogue in offline optimization, the *approximation ratio* for *approximation algorithms*. An algorithm has performance guarantee ρ, if the objective value of the solution produced by the algorithm is at most ρ times the optimal solution for each instance. Polynomial time approximation algorithms offer a way to trade solution quality for computation time.

approximation ratio

approximation algorithm

Let us return to the elevator problem once more. The "replan"-approach required to repeatedly solve offline instances of the problem. Computing a shortest (offline) transportation problem can be accomplished in polynomial time if the capacity of the elevator is one, that is, if the elevator can carry at most one person at a time. However, if the capacity is larger, then the problem becomes NP-hard. Fortunately, there exist approximation algorithms which can be used in a real-time environment (see Chapters 6 and 7 for complexity results and algorithms).

Roadmap and Overview of Results

Chapter 1 is intended mainly as a reference for the notation used in this thesis. It contains an introduction to online computation, competitive analysis and to approximation algorithms for offline problems. The most basic notation from graph theory and the theory of computation can be found in Appendix A.

In Chapters 2 to 5 we study *online dial-a-ride problems*. In an online dial-a-ride problem objects must be transported between points in a metric space by a server of limited capacity. Transportation requests arrive online, specifying the objects to be transported and the corresponding source and destination. Requests are presented to online algorithms according to the time stamp model. The elevator problem described in the introduction is a special case of an online dial-a-ride problem. In this case, the underlying metric space is induced by a path and the "objects" are the passengers which arrive online. Different objective functions lead to different dial-a-ride problems.

online dial-a-ride problem

In Chapter 2 we address the task of minimizing the *makespan* in online dial-a-ride problems. The makespan is the time the server has completed all transportation requests. We establish lower bounds on the competitive ratio of deterministic and randomized algorithms. We then present several competitive algorithms, among them a best possible deterministic online algorithm from a competitive analysis point of view.

makespan

The *online traveling salesman problem* is a special case of an online dial-a-ride problem. In Chapter 3 we study this problem on the metric space given

online traveling salesman problem

fair adversary

zealous algorithm

by $\mathbb{R}_0^+$, the non-negative part of the real line. The emphasis of the chapter is to investigate the effect of restricting the power of the adversary in the competitive analysis and of restricting the class of online algorithms allowed. We show that a very natural strategy achieves the best possible competitive ratio against the conventional adversary. We then go beyond standard competitive analysis and investigate algorithms against a restricted adversary: A *fair adversary* always keeps its server within the convex hull of the requested points released so far. We show that this adversary model indeed allows for lower competitive ratios. We also introduce and analyze a new class of online algorithms which we call *zealous algorithms*. Roughly speaking, the server of a zealous algorithm never sits idle while there is work to do. We prove that in general zealous algorithms are strictly weaker than algorithms that allow waiting time.

flow time

Chapter 4 deals with the tasks of minimizing the maximal or the average flow time in online dial-a-ride problems. The *flow time* of a request is the time span between its release time and its completion time. Intuitively, the average flow time measures the "throughput" of the system, while the maximal flow time can be identified with the maximal dissatisfaction of customers, for instance in an elevator system. It turns out that for the maximal and average flow time there can be no competitive algorithm, neither deterministic nor randomized. Thus, competitive analysis leads to a dead-end.

reasonable request sequence

reasonable load

We develop a new concept to analyze online algorithms on request sequences that fulfill a certain worst-case restriction: informally, a sequence of requests for the online dial-a-ride problem is *reasonable* if the requests that come up in a sufficiently large time period can be served in a time period of at most the same length. This new notion is a stability criterion implying that the system is not overloaded. Under *reasonable load* it is possible to obtain performance bounds for online algorithms and to distinguish the performance of some particular algorithms, which seems to be impossible by means of classical competitive analysis.

weighted sum of completion times

latency

We return to classical competitive analysis in Chapter 5. We consider online dial-a-ride problems with the objective of minimizing the *weighted sum of completion times* of the requests. This is also known as minimizing the *latency*. As in the other chapters, we give lower bounds on the competitive ratios of deterministic and randomized algorithms. We present a competitive deterministic algorithm and an improved randomized version.

offline dial-a-ride problem

Chapters 6 and 7 address *offline dial-a-ride problems* which arise as subproblems in the design of online-algorithms in previous chapters. In view of real-time applications it is highly desirable to solve the offline instances efficiently. We consider the task of minimizing the total distance traveled which in our context is equivalent to finding a shortest (offline) transportation for a given set of requests.

Chapter 6 deals with the case of a unit capacity server and precedence constraints between the transportation requests which emanate from the same source. This variant was motivated by applications where first-in-first-out waiting lines are present at the sources of the transportation jobs. In this case, jobs can be served only according to their order in the line. Examples with first-in-first-out lines are cargo elevator systems where at each floor conveyor belts deliver the goods to be transported. We show that in case of an elevator the resulting offline problem can be solved efficiently in polynomial time. As a byproduct we obtain a structural characterization of graphs that contain Eulerian cycles which respect certain precedence constraints on the arcs. In case of a general metric space, or even a tree, the offline dial-a-ride problem becomes NP-hard. We devise approximation algorithms with provable guarantees.

Eulerian cycles

Chapter 7 leads us back to elevators. As already remarked, elevators can be modeled by dial-a-ride problems on paths. We consider such an offline dial-a-ride problem and study its computational complexity. In contrast to Chapter 6 the server has capacity larger than one. The main result of the chapter is an approximation algorithm with provable performance.

In Chapter 8 we study the *online bin coloring problem*. The task in this problem is to pack colored items of unit size into bins of fixed size, such that the maximum number of different colors per bin is minimized. The packing process is subject to the constraint that at any moment in time at most a bounded number of bins are partially filled. In contrast to the online dial-a-ride problems, in the bin coloring problem requests are presented according to the sequence model. An online algorithm must pack each item irrevocably before the next item becomes known. The investigation of the bin coloring problem from a competitive analysis point of view reveals a number of oddities. A natural greedy-type strategy achieves a competitive ratio strictly worse than a trivial algorithm. Moreover, no algorithm can be substantially better than the trivial strategy. Even more surprising, neither randomization nor "resource augmentation" helps to overcome the general lower bound on the competitive ratio.

online bin coloring problem

Acknowledgments

At this point, I am greatly indebted to all my friends and colleagues who contributed in various ways to this thesis. First of all, I would like to thank Martin Grötschel for his constructive criticism of competitive analysis and for providing an excellent working environment at the *Konrad-Zuse Zentrum für Informationstechnik in Berlin*. I also want to thank explicitly Andreas Eisenblätter, Diana Poensgen, Jörg Rambau, and Hans-Christoph Wirth who did detailed reviews of this thesis and helped to improve its presentation.

Credits

Chapter 2 on minimizing the makespan in online dial-a-ride problems is based on joint work with Norbert Ascheuer and Jörg Rambau [AKR00]. The results reported in Chapter 3 on fair adversaries in the online traveling salesman problem were obtained together with Michiel Blom, Willem de Paepe and Leen Stougie [BK$^+$01]. Chapter 4 contains material from joint work with Dietrich Hauptmeier and Jörg Rambau [HKR00, HKR01]. The results in Chapter 5 were obtained jointly with Willem de Paepe, Diana Poensgen and Leen Stougie [KdP$^+$01a]. Chapter 6 is based on joint work with Dietrich Hauptmeier, Jörg Rambau, and Hans-Christoph Wirth [HK$^+$99]. Chapter 7 contains results from joint research with Jörg Rambau and Steffen Weider [KRW00]. The results reported in Chapter 8 were obtained jointly with Willem de Paepe, Jörg Rambau and Leen Stougie [KdP$^+$01b].

All the joint work mentioned above represents many hours of stimulating discussions with my colleagues. I am extremely grateful to all of them for their valuable time and for allowing me to include our joint work in this thesis.

1

Preliminaries

In this chapter we review the most important theoretical concepts that we are going to use. The first sections are dedicated to online computation, the last section deals with offline computation. The presentation is intended to make this thesis self-contained. Full details of the concepts given here can be found in the cited literature. Appendix A contains a compilation of the most basic notation used in this thesis.

In Sections 1.1 and 1.2 we introduce competitive analysis for deterministic and randomized algorithms. Section 1.3 discusses extensions of competitive analysis that have been suggested in the literature to overcome some of its weaknesses.

Section 1.4 contains a brief compilation of the most important notions from complexity theory. We also define approximation algorithms which, to some extent, are an analogue to competitive online algorithms with respect to offline computation.

1.1 What is an Online Problem?

In the introduction we defined an online problem informally by stating that it is a problem where the data is revealed step by step to an online algorithm according to either the *sequence model* or the *time stamp model*. This definition can be made precise with the help of *request-answer games* introduced in [BDB+94]. As we will see later, the general concept of a request-answer game comprises both, the sequence model and the time stamp model.

Definition 1.1 (Request-Answer Game)
A ***request-answer game*** $(R, \mathcal{A}, \mathcal{C})$ *consists of a request set* R*, a sequence of nonempty answer sets* $\mathcal{A} = A_1, A_2, \dots$ *and a sequence of cost functions* $\mathcal{C} = cost_1, cost_2, \dots$ *where* $cost_j\colon R^j \times A_1 \times \cdots \times A_j \to \mathbb{R}_+ \cup \{+\infty\}$.

request-answer game

We remark here that in the literature one frequently requires each answer set A_j to be finite. This assumption is made to avoid difficulties with the existence of expected values when studying randomization. However, the finiteness requirement is not of conceptual importance. As remarked in [BEY98, Chapter 7] an infinite or even continuous answer set can be "approximated" by a sufficiently large finite answer set.

Definition 1.2 (Deterministic Online Algorithm)

deterministic online algorithm

*A **deterministic online algorithm** ALG for the request-answer game* $(R, \mathcal{A}, \mathcal{C})$ *is a sequence of functions* $f_j \colon R^j \to A_j$, $j \in \mathbb{N}$. *The **output** of* ALG *on the input request sequence* $\sigma = r_1, \dots, r_m$ *is*

$$\mathsf{ALG}[\sigma] := (a_1, \dots, a_m) \in A_1 \times \dots \times A_m, \quad \textit{where } a_j := f_j(r_1, \dots, r_j).$$

cost

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

$$\mathsf{ALG}(\sigma) := \mathit{cost}_m(\sigma, \mathsf{ALG}[\sigma]).$$

We now define the notion of a randomized online algorithm.

Definition 1.3 (Randomized Online Algorithm)

randomized online algorithm

*A **randomized online algorithm*** RALG *is a probability distribution over deterministic online algorithms* ALG_x *(*x *may be thought of as the coin tosses of the algorithm* RALG*). The answer sequence* $\mathsf{RALG}[\sigma]$ *and the cost* $\mathsf{RALG}(\sigma)$ *on a given input* σ *are random variables.*

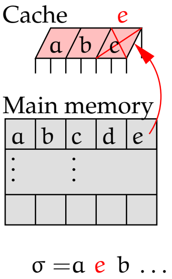
mixed strategy

pure strategy

In terms of game theory we have defined a randomized algorithm as a *mixed strategy* (where a deterministic algorithm is then considered to be a *pure strategy*). There is no harm in using this definition of a randomized algorithm since without memory restrictions all types of randomized strategies, *mixed strategies*, *behavioral strategies*, and *general strategies*, are equivalent (see e.g. [BEY98, Chapter 6]).

We illustrate request-answer games by two simple examples. The first example is the classical *paging problem*:

Example 1.4 (Paging Problem)

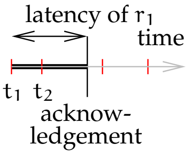


Paging problem with cache size $k = 3$.

Consider a two-level memory system (e.g., of a computer) that consists of a small fast memory (the cache) with k pages and a large slow memory consisting of a total of N pages. Each request specifies a page in the slow memory, that is, $r_j \in R := \{1, \dots, N\}$. In order to serve the request, the corresponding page must be brought into the cache. If a requested page is already in the cache, then the cost of serving the request is zero. Otherwise one page must be evicted from the cache and replaced by the requested page at a cost of 1. A paging algorithm specifies which page to evict. Its answer to request r_j is a number $a_j \in A_j := \{1, \dots, k\}$, where $a_j = p$ means to evict the page at position p in the cache.

The objective in the paging problem is to minimize the number of page faults. The cost function cost_j simply counts the number of page faults which can be deduced easily from the request sequence $r_1, \dots, r_j$ and the answers $a_1, \dots, a_j$. ◁

In the paging problem the answer to a request implies an irrevocable decision of how to serve the next request, that is, the paging problem is formulated within the sequence model. We will now provide a second example of a request-answer game which specifies a problem in the time stamp model, where decisions can be revoked as long as they have not been executed yet.



Online TCP acknowledgment problem.

Example 1.5 (Online TCP Acknowledgment)
In the online TCP acknowledgment problem introduced in [DGS98] a number of *packets* are received. Each packet j has a receipt time $t_j \geq 0$ and a weight $w_j \geq 0$. An online algorithm learns the existence of a packet only at its receipt time. All packets must be acknowledged at some time after their receipt. A single acknowledgment acknowledges all packets received since the last acknowledgment. There is a cost of 1 associated with each acknowledgment. Moreover, if packet j is acknowledged at time $t \geq t_j$, this induces a latency cost of $w_j(t - t_j)$. The goal in the online TCP acknowledgment problem is to minimize the sum of the acknowledgment cost and the latency cost.

At time t_j when packet $r_j = (t_j, w_j)$ is received, an online algorithm ALG must decide when to acknowledge all yet unconfirmed packets. Hence, the answer to request $r_j \in R := \mathbb{R}_+ \times \mathbb{R}_+$ is a real number $a_j \in A_j := \mathbb{R}_+$ with $a_j \geq t_j$. If an additional packet is received before time a_j, then ALG is not charged for the intended acknowledgment at time a_j, otherwise it incurs an acknowledgment cost of one. The cost function $cost_j$ counts the number of actual acknowledgments and adds for each packet r_j the latency cost resulting from the earliest realized acknowledgment in the answer sequence. The condition that packets can not be acknowledged before they are received can be enforced by defining the value $cost_j(r_1, \dots, r_j, a_1, \dots, a_j)$ to be $+\infty$ if $a_j < t_j$. $\triangleleft$

1.2 Competitive Analysis

We now define competitiveness of deterministic and randomized algorithms. While the deterministic case is straightforward, the randomized case is much more subtle.

1.2.1 Deterministic Algorithms

Suppose that we are given an online problem as a request-answer game. We define the *optimal offline cost* on a sequence $\sigma \in R^m$ as follows:

optimal offline cost

$$\text{OPT}(\sigma) := \min\{\, cost_m(\sigma, a) : a \in A_1 \times \cdots \times A_m \}.$$

The optimal offline cost is the yardstick against which the performance of a deterministic algorithm is measured in competitive analysis.

Definition 1.6 (Deterministic Competititive Algorithm)
Let $c \geq 1$ be a real number. A deterministic online algorithm ALG *is called* ***c-competitive*** *if*

c-competitive

$$\text{ALG}(\sigma) \leq c \cdot \text{OPT}(\sigma) \qquad (1.1)$$

competitive ratio

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

Observe that, in the above definition, there is no restriction on the computational resources of an online algorithm. The only scarce resource in competitive analysis is information.

We want to remark here that the definition of c-competitiveness varies in the literature. Often an online algorithm is called c-competitive if there exists a constant b such that

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

holds for any request sequence. Some authors even allow b to depend on some problem or instance specific parameters. In this thesis we will stick to the definition given above.

Since for a c-competitive algorithm ALG we require inequality (1.1) to hold for *any* request sequence, we may assume that the request sequence is generated by a malicious *adversary*. Competitive analysis can be thought of as a game between an *online player* (an online algorithm) and the adversary. The adversary knows the (deterministic) strategy of the online player, and can construct a request sequence which maximizes the ratio between the player's cost and the optimal offline cost.

adversary

online player

1.2.2 Randomized Algorithms

For randomized algorithms we have to be precise in defining what kind of information about the online player is available to the adversary. This leads to different adversary models which are explained below. For an in-depth treatment we refer to [BEY98, MR95].

If the online algorithm is randomized then according to intuition, the adversary has less power since the moves of the online player are no longer certain. The weakest kind of adversary for randomized algorithms is the *oblivious adversary*:

Definition 1.7 (Oblivious Adversary)
An ***oblivious adversary*** OBL *must construct the request sequence in advance based only on the description of the online algorithm but before any moves are made.*

oblivious adversary

We can now define competitiveness against this adversary:

Definition 1.8 (Competitive Algorithm against an Oblivious Adversary) *A randomized algorithm* RALG*, distributed over a set* $\{\mathsf{ALG}_y\}$ *of deterministic algorithms, is* c***-competitive against an oblivious adversary*** *for some* $c \geq 1$*, if*

c-competitive against an oblivious adversary

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

for all request sequences σ*. Here, the expression* $\mathbb{E}_Y[\mathsf{ALG}_y(\sigma)]$ *denotes the expectation with respect to the probability distribution* Y *over* $\{\mathsf{ALG}_y\}$ *which defines* RALG*.*

In case of a deterministic algorithm the above definition collapses to that given in Definition 1.6. In contrast to the oblivious adversary, an *adaptive adversary* can issue requests based on the online algorithm's answers to previous ones.

adaptive offline adversary

The *adaptive offline adversary* (ADOFF) defers serving the request sequence until he has generated the last request. He then uses an optimal offline algorithm. The *adaptive online adversary* (ADON) must serve the input sequence (generated by himself) online. Notice that in case of an adaptive adversary ADV, the adversary's cost $\mathsf{ADV}(\sigma)$ for serving σ is a random variable.

adaptive online adversary

Definition 1.9 (Competitive Algorithm against an Adaptive Adversary) *A randomized algorithm* RALG*, distributed over a set* $\{\mathsf{ALG}_y\}$ *of deterministic algorithms, is called* c***-competitive against an adaptive adversary*** ADV *for some* $c \geq 1$*, where* $\mathsf{ADV} \in \{\mathsf{ADON}, \mathsf{ADOFF}\}$*, if*

c-competitive against an adaptive adversary

$$\mathbb{E}_Y[\mathsf{ALG}_y(\sigma) - c \cdot \mathsf{ADV}(\sigma)] \leq 0.$$

for all request sequences σ*. Here,* $\mathsf{ADV}(\sigma)$ *denotes the adversary cost which is a random variable.*

The above adversaries differ in their power. Clearly, an oblivious adversary is the weakest of the three types and an adaptive online adversary is no stronger than the adaptive offline adversary. Moreover, as might be conjectured, the adaptive offline adversary is so strong that randomization adds no power against it. More specifically, the following result holds:

Theorem 1.10 ([BDB$^+$94]) *Let* $(R, \mathcal{A}, \mathcal{C})$ *be a request-answer game. If there exists a randomized algorithm for* $(R, \mathcal{A}, \mathcal{C})$ *which is* c*-competitive against any adaptive offline adversary, then there exists also a* c*-competitive deterministic algorithm for* $(R, \mathcal{A}, \mathcal{C})$*.* □

The strength of the adaptive online adversary can also be estimated:

Theorem 1.11 ([BDB$^+$94]) *Let* $(R, \mathcal{A}, \mathcal{C})$ *be a request-answer game. If there exists a randomized algorithm for* $(R, \mathcal{A}, \mathcal{C})$ *which is* c*-competitive against any adaptive online adversary, then there exists a* c^2*-competitive deterministic algorithm for* $(R, \mathcal{A}, \mathcal{C})$*.* □

In this thesis we are only going to consider the weakest adversary, the oblivious adversary, in the analysis of randomized algorithms.

1.2.3 Lower Bounds on the Competitive Ratio

A lower bound on the competitive ratio is usually derived by providing a set of specific instances on which no online algorithm can perform well compared to an optimal offline algorithm. Here, again, we have to distinguish between deterministic and randomized algorithms.

For deterministic algorithms finding a suitable set of request sequences is in most cases comparatively easy. For randomized algorithms, however, it is usually very difficult to bound the expected cost of an arbitrary randomized algorithm on a specific instance from below. In these cases, the following approach using *Yao's Principle* (see Theorem 1.12 below) is helpful.

Yao's Principle

Suppose that the set of possible request sequences for an online problem is indexed by the set $\mathcal{X}$, that is, the set of inputs is represented by $\{ \sigma_x : x \in \mathcal{X} \}$. Also, let $\{ \mathsf{ALG}_y : y \in \mathcal{Y} \}$ be the set of deterministic online algorithms for the problem.

To establish that no randomized algorithm can achieve a competitive ratio better than $\bar{c}$ against an oblivious adversary, we must show that for any probability distribution Y over the set of deterministic algorithms, there exists some $x \in \mathcal{X}$ such that the inequality

$$\mathbb{E}_Y[\mathsf{ALG}_y(\sigma_x)] \geq \bar{c} \cdot \mathsf{OPT}(\sigma_x)$$

holds. This is equivalent to $\inf_Y \sup_{x\in\mathcal{X}} \{\mathbb{E}_Y[\mathsf{ALG}_y(\sigma_x)] - \bar{c} \cdot \mathsf{OPT}(\sigma_x)\} \geq 0$. Let us denote by X a probability distribution over the set $\{ \sigma_x : x \in \mathcal{X} \}$ of sequences. Then we have

$$\begin{aligned}
&\inf_Y \sup_{x\in\mathcal{X}} \{\mathbb{E}_Y[\mathsf{ALG}_y(\sigma_x)] - \bar{c} \cdot \mathsf{OPT}(\sigma_x)\} \\
&= \inf_Y \sup_{x\in\mathcal{X}} \int_{\mathcal{Y}} (\mathsf{ALG}_y(\sigma_x) - \bar{c} \cdot \mathsf{OPT}(\sigma_x))\, dY \\
&= \inf_Y \sup_X \int_{\mathcal{X}} \int_{\mathcal{Y}} (\mathsf{ALG}_y(\sigma_x) - \bar{c} \cdot \mathsf{OPT}(\sigma_x))\, dY\, dX \\
&\geq \sup_X \inf_Y \int_{\mathcal{X}} \int_{\mathcal{Y}} (\mathsf{ALG}_y(\sigma_x) - \bar{c} \cdot \mathsf{OPT}(\sigma_x))\, dY\, dX \\
&= \sup_X \inf_Y \int_{\mathcal{Y}} \int_{\mathcal{X}} (\mathsf{ALG}_y(\sigma_x) - \bar{c} \cdot \mathsf{OPT}(\sigma_x))\, dX\, dY \\
&= \sup_X \inf_{y\in\mathcal{Y}} \int_{\mathcal{X}} (\mathsf{ALG}_y(\sigma_x) - \bar{c} \cdot \mathsf{OPT}(\sigma_x))\, dX.
\end{aligned}$$

Notice that the exchange of integration order above is justified by Fubini's Theorem (see e.g. [Rud76]). Define the *expected cost of the deterministic algorithm* ALG with respect to the distribution X over the set of sequences to be

expected cost $\mathbb{E}_X[\mathsf{ALG}(\sigma_x)]$ of a deterministic algorithm ALG

$$\mathbb{E}_X[\mathsf{ALG}(\sigma_x)] := \int_{\mathcal{X}} \mathsf{ALG}(\sigma_x)\, dX.$$

With this notation the above chain of inequalities can be rewritten as

$$\begin{aligned}&\inf_Y \sup_{x\in\mathcal{X}} \{\mathbb{E}_Y[\mathsf{ALG}_y(\sigma_x)] - \bar{c}\cdot \mathsf{OPT}(\sigma_x)\}\\ &\qquad\qquad \geq \sup_X \inf_{y\in\mathcal{Y}} \{\mathbb{E}_X[\mathsf{ALG}_y(\sigma_x)] - \bar{c}\,\mathbb{E}_X[\mathsf{OPT}(\sigma_x)]\},\end{aligned}$$

which is a special form of Yao's Principle from game theory.

Theorem 1.12 (Yao's Principle) *Let* $\{\mathsf{ALG}_y : y \in \mathcal{Y}\}$ *denote the set of deterministic online algorithms for an online minimization problem. 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*

$$\inf_{y\in\mathcal{Y}} \mathbb{E}_{\bar{X}}[\mathsf{ALG}_y(\sigma_x)] \geq \bar{c}\,\mathbb{E}_{\bar{X}}[\mathsf{OPT}(\sigma_x)], \tag{1.2}$$

then $\bar{c}$ *is a lower bound on the competitive ratio of any randomized algorithm against an oblivious adversary.* □

1.3 Extensions of Competitive Analysis

Competitive analysis is a type of worst-case analysis. It has (rightly) been criticized as being overly pessimistic. Often the adversary is simply too powerful and allows only trivial competitiveness results. This phenomenon is called "hitting the triviality barrier" (see [FW98]). To overcome this unsatisfactory situation various extensions and alternatives to pure competitive analysis have been investigated in the literature.

comparative analysis

In *comparative analysis* the class of algorithms where the offline algorithm is chosen from is restricted. This concept has been introduced in the context of the paging problem [KP94]. The authors compare the performance of an online algorithm for the paging problem with that of the best paging algorithm having limited lookahead. Let Π be a minimization (online) problem. The *comparative ratio* $c_{\mathsf{ALG},\mathcal{B}}$ of an algorithm ALG for Π relative to a class $\mathcal{B}$ of algorithms is defined as the worst case ratio between the solution cost produced by ALG and the best solution produced by an algorithm in $\mathcal{B}$:

comparative ratio

$$c_{\mathsf{ALG},\mathcal{B}} := \sup_{B\in\mathcal{B}} \sup_{\sigma} \frac{\mathsf{ALG}(\sigma)}{B(\sigma)}.$$

If $\mathcal{B}$ is the class of all offline algorithms for Π, then the comparative ratio reduces to the standard competitive ratio.

The comparative ratio has also been studied in the context of online financial problems. For most of these problems the standard adversary also appears to be too strong. To obtain meaningful (theoretical) results about the performance, e.g., of online portfolio selection algorithms, a comparison with a restricted class of offline algorithms is used. We refer to [BEY98, Chapter 14] for details.

We are going to use the concept of comparative analysis for investigating the online traveling salesman problem in Chapter 3 and show that a restricted adversary model (the "fair adversary") indeed allows for lower competitive ratios.

resource augmentation

Another approach to strengthen the position of an online algorithm is the concept of *resource augmentation* (see e.g. [PS+97, PK95, ABF96, ST85]). Here, the online algorithm is given more resources (e.g., more or faster machines in scheduling) to serve requests than the offline adversary.

diffuse adversary

The *diffuse adversary* model introduced in [KP94] deals with the situation where the input is chosen by an adversary according to some probability distribution. Although the online algorithm does not know the distribution itself, it is given the information that this distribution belongs to a specific class of distributions.

In Chapter 4 we are going to introduce the concept of reasonable load which enables us to to prove performance bounds in a case where competitive analysis fails to give decision support.

access graph model

statistical adversary

Other approaches to go beyond pure competitive analysis include the *access graph model* for paging [BI+91, BI+95, IKP92] and the *statistical adversary* [BK+96]. We refer to [FW98, Chapter 17] for a comprehensive survey.

All of the extensions and alternatives to competitive analysis have been proven to be useful for some *specific problem* and powerful enough to obtain significant results. However, none of these approaches has yet succeeded in replacing competitive analysis as the standard tool in the theoretical analysis of online algorithms.

1.4 Offline Computation and Approximation Algorithms

As mentioned before, many problems are not only online but also real-time. Unfortunately, it can be shown for many important problems that determining an optimal solution may be extremely time consuming due to their computational complexity. The class of NP-complete problems represents a large collection of such problems, which are all related in the sense that a polynomial-time solution of one of them implies the polynomial-time solvability of the whole class. Up to now, no polynomial-time algorithm for an NP-complete problem is known (see Appendix A.8 for a brief compilation of notation from the theory of computation).

If exact methods fail to produce answers in real-time the next step is to look for suboptimal solutions which have a guaranteed quality and which can be determined fast.

Definition 1.13 (Approximation algorithm)
A (deterministic)[1] *algorithm* ALG *for a minimization problem* Π *is called* ρ***-approximative****, if*

ρ-approximative

$$\mathsf{ALG}(I) \leq \rho\, \mathsf{OPT}(I) \tag{1.3}$$

holds for any problem instance I *of* Π*. The infimum of all values of* ρ *for which* ALG *is* ρ*-approximative is called* ***approximation ratio*** *or* ***performance ratio*** *of* ALG*.*

approximation ratio

performance ratio

In view of applications, in the design of approximation algorithms speed is of first priority since here computation time is the scarce resource. Thus, one usually restricts approximation algorithms to the class of polynomial-time algorithms.[2] In contrast, time complexity is not an issue in competitive analysis: there is (at least in theory) no bound on the computation time for an answer generated by an online algorithm.

We also use the term ρ-*approximation algorithm* to denote an algorithm which is ρ-approximative. Often it is difficult to determine the exact approximation ratio of an algorithm and one resorts to proving an upper bound on this ratio. Any such upper bound will be referred to as a *performance* of the algorithm.

ρ-approximation algorithm

performance

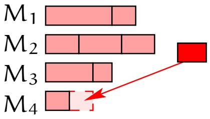
Approximation algorithms are closely related to competitive online algorithms. By the above definition, a c-competitive online algorithm is also c-approximative. Conversely, if a c-approximate algorithm is additionally online, it is also c-competitive. Many approximation algorithms have a simple structure and are in fact online.

Example 1.14 (Load Balancing on Identical Machines)
Consider the following *load balancing problem* arising in machine scheduling. One is given a sequence $j_1, \dots, j_m$ of jobs where job j_i has processing time p_i. The task is to distribute the jobs on n identical machines $M_1, \dots, M_n$ such that the maximum load of the machines is minimized. Here, the *load* of a

load balancing problem

[1] One can also define randomized approximation algorithms analogously to randomized online algorithms. In this thesis we are only going to use deterministic approximation algorithms.

[2] In the literature the notion of an approximation algorithm often includes the property of the algorithm being polynomial-time.



Graham's algorithm LIST: the next job is always assigned to the machine with the currently least load.

machine is defined to be the sum of job processing times assigned to the machine.

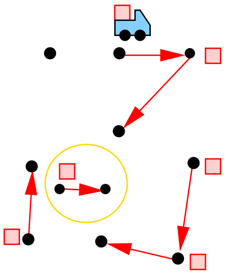
Graham [Gra66, Gra69] proposed the following greedy-type heuristic LIST: Consider the jobs in order of their occurrence in the input sequence $j_1, \ldots, j_m$. Always assign the next job to the machine which currently has the least load (breaking ties arbitrarily). $\triangleleft$

Clearly, LIST can be implemented to run in polynomial time. The algorithm LIST can be shown to be $(2 - 1/n)$-approximative [Gra66, Gra69]. Moreover, LIST is also an online algorithm for the online version of the scheduling problem where jobs are revealed to an online algorithm according to the sequence model. Hence, LIST is also $(2 - 1/n)$-competitive.

For NP-hard problems, polynomial-time approximation algorithms offer a way to trade solution quality for computation time. Polynomial-time approximation algorithms have been intensively considered within the last years. For comprehensive surveys on approximation algorithms we refer to [Mot92, Hoc97, AC$^+$99, Vaz01].

Minimizing the Makespan in Online-Dial-a-Ride Problems

In this chapter we introduce the basic setup for the generic online-dial-a-ride problem OLDARP, which can be outlined as follows: Transportation requests between points in a metric space arrive online, specifying the objects to be transported and the corresponding sources and destinations. These requests are to be handled by a server which picks up and drops objects at their sources and destinations.



Online dial-a-ride problem.

The makespan is arguably the simplest objective function and hence a good starting point for deriving competitiveness results for online-dial-a-ride problems. We analyze several competitive algorithms and establish tight competitiveness results. The first two algorithms, REPLAN and IGNORE are very simple and natural: REPLAN completely discards its (preliminary) schedule and recomputes a new one whenever a new request arrives. IGNORE always runs a (locally optimal) schedule for a set of known requests and ignores all new requests until this schedule is completed. We then present a somewhat less natural strategy SMARTSTART, which in contrast to the other two strategies may leave the server idle from time to time although unserved requests are known. The SMARTSTART-algorithm achieves the best-possible competitive ratio for deterministic algorithms.

This chapter is organized as follows: In Section 2.1 we introduce the basic framework for formulating single-server dial-a-ride problems. We will use this framework for stating the online and offline dial-a-ride problems in subsequent chapters.

The "closed makespan" $C^{\text{o}}_{\max}$ is the objective function on which we mainly focus in this chapter. It is introduced in Section 2.2. In Section 2.3 we show how certain machine scheduling problems can be modeled as special cases of dial-a-ride problems. In Section 2.4 we establish lower bounds for the competitive ratio of deterministic online algorithms. In Section 2.5 we analyze the strategies REPLAN and IGNORE. Section 2.6 contains our improved algorithm SMARTSTART. A simple randomized algorithm is presented in Section 2.7. In Section 2.8 we show how to extend our results to the non-closed makespan, that is, the case where the server is not required to return to its initial position at the end of its work.

Related Work

We do not claim originality for the two online-algorithms IGNORE and REPLAN; instead, we show how to analyze them for online dial-a-ride problems and how ideas from both strategies can be used to construct the new online strategy SMARTSTART with better competitive ratio.

The first, to the best of our knowledge, occurrence of the strategy IGNORE can be found in the paper by Shmoys, Wein, and Williamson [SWW95]: They show a general result about obtaining competitive algorithms for minimizing the total completion time (also called the makespan) in machine scheduling problems when the jobs arrive over time: If there exists a ρ-approximation algorithm for the offline version, then this implies the existence of a a 2ρ-competitive algorithm for the online-version, which is essentially the IGNORE strategy (we present an improvement of this result in Section 2.6).

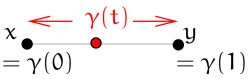
The results from [SWW95] show that IGNORE-type strategies are competitive for a number of online scheduling problems. The strategy REPLAN is probably folklore; it can be found also under different names like REOPT or OPTIMAL.

The difference of the OLDARP to the machine scheduling problems treated in [SWW95] are as follows: For OLDARP, the "execution time" of jobs depends on their execution order. OLDARP can be viewed as a generalized scheduling problem with order dependent setup costs and execution times (see also Section 2.3).

online traveling salesman problem

In [AF$^+$01, AF$^+$95, AF$^+$94] the authors studied the online traveling salesman problem (OLTSP) which is obtained as a special case of OLDARP, when for each request its source and destination coincide. It is shown in [AF$^+$01] that there is a metric space, namely the boundary of the unit square, where any deterministic algorithm for the OLTSP has a competitive ratio of at least 2. For the case that the metric space is the real line, a lower bound of $\frac{9+\sqrt{17}}{8} \approx 1.64$ is given in [AF$^+$95].

2.1 Single-Server Dial-a-Ride-Problems



Connected and smooth metric space.

Let $M = (X, d)$ be a metric space with distinguished origin $o \in X$. We assume that M is "connected and smooth´´ in the following sense: for all pairs (x, y) of points from M, there is a rectifiable path $\gamma\colon [0, 1] \to X$ in X with $\gamma(0) = x$ and $\gamma(1) = y$ of length $d(x, y)$ (see e.g. [AF$^+$01]). Examples of metric spaces that satisfy the above condition are the Euclidean space $\mathbb{R}^p$ and a metric space induced by an undirected connected edge-weighted graph.

An instance of the basic online dial-a-ride problem OLDARP in the metric space M consists of a sequence $\sigma = r_1, \ldots, r_m$ of *requests*. Each request is

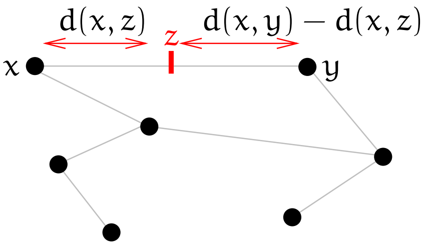
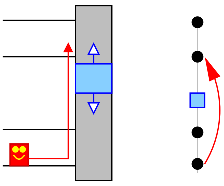


Figure 2.1
A weighted graph induces a connected and smooth metric space.

a triple $r_j = (t_j, \alpha_j, \omega_j) \in \mathbb{R}{\times}X{\times}X$ with the following meaning: $t_j = t(r_j) \geq 0$ is a real number, the time where request r_j is released (becomes known), and $\alpha_j = \alpha(r_j) \in X$ and $\omega_j = \omega(r_j) \in X$ are the source and destination, respectively, between which the object corresponding to request r_j is to be transported.

It is assumed that the sequence $\sigma = r_1, \dots, r_m$ of requests is given in order of non-decreasing release times, that is, $0 \leq t(r_1) \leq t(r_2) \leq \cdots \leq t(r_m)$. For a real number t we denote by $\sigma_{\leq t}$ the subsequence of requests in σ released up to and including time t. Similarly, $\sigma_{=t}$ and $\sigma_{<t}$ denote the subsequences of σ consisting of those requests with release time exactly t and strictly smaller than t, respectively.



Elevator and representation in OLDARP.

A server is located at the origin $o \in X$ at time 0 and can move at constant unit speed. The server has capacity C, i.e., it can carry at most C objects at a time. We do not allow *preemption*: once the server has picked up an object, it is not allowed to drop it at any other place than its destination.

An online algorithm for OLDARP does neither have information about the release time of the last request nor about the total number of requests. The online algorithm must determine the behavior of the server at a certain moment t of time as a function of all the requests released up to time t (and the current time t). In contrast, an offline algorithm has information about all requests in the whole sequence σ already at time 0.

transportation schedule

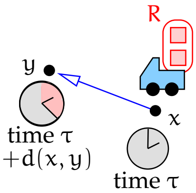
Informally, a feasible online/offline solution, called *transportation schedule*, for a sequence σ is a sequence of moves for the server such that the following conditions are satisfied: (i) the server starts in the origin at time 0, (ii) each request in σ is served, but picked up not earlier than the time it is released. If additionally (iii) the server ends its work at the origin, then the transportation schedule is called *closed*. Depending on the specific variant of OLDARP only closed schedules may be feasible.

closed transportation schedule

Given an objective function *cost*, the problem *cost*-OLDARP consists of finding a feasible schedule S^* minimizing $cost(S^*)$.

We now formally introduce the notions of a transportation schedule and an objective function for the OLDARP. The presentation is technical, but avoids ambiguities. Moreover, the definitions enable us to state the problem *cost*-OLDARP cleanly as a request-answer game: Essentially, an algorithm must answer each request with a transportation schedule which serves all yet unserved requests and which does not conflict with previous answers.

We will use the more informal definition of a transportation schedule given above in subsequent chapters as long as no confusion can occur.



Transportation move carrying objects R.

2.1.1 Transportation Schedules and Objective Functions

The building blocks of transportation schedules are transportation moves. A *transportation move* is a quadruple (τ, x, y, R), where $x \in X$ is the starting point, $y \in X$ the end point, and τ the starting time, while R is the (possibly empty) set of requests carried by the move. The arrival time of the move is the sum $\tau + d(x,y)$ of its starting time τ and the distance $d(x,y)$ covered.

transportation schedule

Definition 2.1 (Transportation Schedule)
A ***transportation schedule*** *for a sequence σ of requests is a sequence*

$$S = (\tau_1, x_1, y_1, R_1), (\tau_2, x_2, y_2, R_2), \dots, (\tau_\ell, x_\ell, y_\ell, R_\ell)$$

of transportation moves with the following properties:

(i) The $(i+1)$st move starts at the endpoint of the ith move and not earlier than the time that the ith move is completed, that is, $x_{i+1} = y_i$ and $\tau_{i+1} \geq \tau_i + d(x_i, y_i)$ for all i;

(ii) Each move carries at most C requests, that is, $|R_i| \leq C$ for all i;

(iii) For any request $r \in \sigma$, the subsequence of S consisting of those moves (τ_i, x_i, y_i, R_i) with $r \in R_i$ is a contiguous nonempty subsequence

$$S(r) = (\tau_l, x_l, y_l, R_l), \dots, (\tau_{l+p}, x_{l+p}, y_{l+p}, R_{l+p})$$

of S which forms a transportation from $\alpha(r)$ to $\omega(r)$, that is, $x_l = \alpha(r)$ and $y_{l+p} = \omega(r)$. The sub-transportation $S(r)$ does not start before r is released, that is, $\tau_l \geq t(r)$.

starting time/point

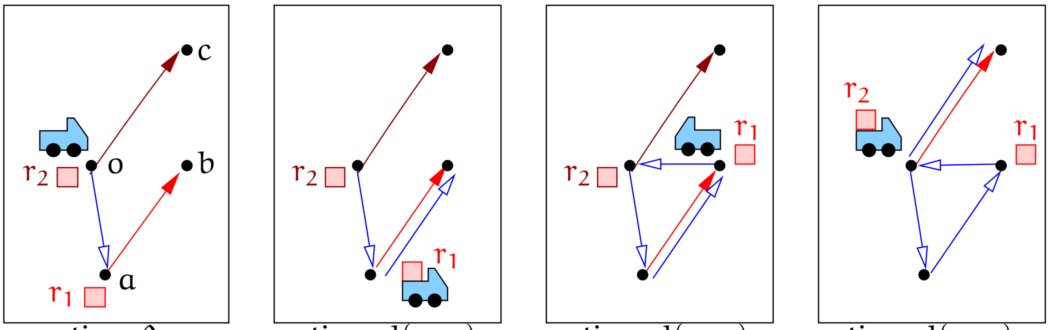
end time/point

length $l(S)$

The time τ_1 and the point $x_1 \in X$ are called the ***starting time*** *and the* ***starting point*** *of S. Similarly, the time $\tau_\ell + d(x_\ell, y_\ell)$ and the point y_ℓ are referred to as the* ***end time*** *, and the* ***end point*** *of S.*

A schedule which ends at the origin o is called a ***closed transportation schedule****. The* ***length*** *$l(S)$ of schedule S is defined as the difference between its end time and its start time, that is, $l(S) = \tau_\ell + d(x_\ell, y_\ell) - \tau_1$.*

Depending on the specific problem setup we may require that an algorithm produces a closed transportation schedule. For instance, in the *online traveling salesman problem* (OLTSP) (see Definition 2.6) only closed schedules have finite objective function value.



time 0 time $d(o,a)$ time $d(o,a)$ $+d(a,b)$ time $d(o,a)$ $+d(a,b)$ $+d(b,o)+1$

Figure 2.2
A transportation schedule for the sequence $\sigma = r_1, r_2$.

Example 2.2
Consider the request sequence $\sigma = r_1, r_2$, where $r_1 = (0, a, b)$ and $r_2 = (d(o,a) + d(a,b) + d(b,0) + 1, o, c)$. Figure 2.2 illustrates the (non-closed) transportation schedule

$$S = (0, o, a, \varnothing), (\tau_2, a, b, \{r_1\}), (\tau_3, b, o, \varnothing), (\tau_4, o, c, \{r_2\}),$$

for σ, where $\tau_2 = d(o,a)$, $\tau_3 = \tau_2 + d(a,b)$, $\tau_4 = \tau_3 + d(b,o) + 1$. Notice that the last move from o to c starts one time unit later than the previous move has ended. This is due to the fact that r_2 has not yet been released at time $\tau_3 + d(b,o)$. $\lhd$

Definition 2.3 (Objective Function for OLDARP)
*Let $\mathcal{R} = \mathbb{R}_+ \times X \times X$ be the set of all possible requests and $\mathcal{S}$ the set of all possible transportation schedules. Denote by $\mathcal{R}^*$ the set of all sequences which are composed of requests from $\mathcal{R}$ in order of non-decreasing release times. An **objective function for OLDARP** is a mapping $cost\colon \mathcal{R}^* \times \mathcal{S} \to \mathbb{R}_+ \cup \{+\infty\}$.*

objective function for OLDARP

Definition 2.4 (Online Dial-a-Ride Problem *cost*-OLDARP)
Given an instance of the basic online dial-a-ride problem OLDARP *and an objective function cost, the **online dial-a-ride problem** cost-*OLDARP *consists of finding a transportation schedule which starts at the origin and which minimizes cost.*

cost-OLDARP

2.1.2 Online Dial-a-Ride Problems as Request-Answer Games

In this section we show how to formulate the online dial-a-ride problem *cost*-OLDARP in the metric space (X, d) as a request-answer game. We set $R := \mathbb{R}_+ \times X \times X$ and for $n \in \mathbb{N}$ we define A_n to be the collection of all transportation schedules for sequences of requests from R. It remains to specify the sequence of cost functions $cost_i$, $i = 1, 2, \ldots$. Let

$$S = (\tau_1, x_1, y_1, R_1), (\tau_2, x_2, y_2, R_2), \ldots, (\tau_\ell, x_\ell, y_\ell, R_\ell)$$

be a transportation schedule.

prefix

The *prefix* of S at time t is the transportation schedule resulting from the movements of the server from time 0 until time t. We define the prefix more rigorously. To this end, let j with $1 \le j \le \ell$ be maximal with the property that the jth move starts no later than time t, that is, $\tau_j \le t$. Let $\gamma\colon [0,1] \to X$ be the rectifiable path in X between the starting point x_j and the end point y_j of the jth move. We set

$$z := \begin{cases} \gamma\left(\frac{t-\tau_j}{d(x_j,y_j)}\right)\Big) & \text{if } \tau_j \le t \le \tau_j + d(x_j, y_j) \\ y_j & \text{if } t > \tau_j + d(x_j, y_j). \end{cases}$$

In other words, $z \in X$ is the position in the connected and smooth metric space where according to the schedule S the server is located at time t. The first case in the above equation corresponds to the situation where the jth move is not yet completed at time t and the server is at a position "between x_j and y_j". The second case reflects the situation where the jth move is completed before time t. Observe that in this case the server is still at point y_j since by definition of j the $(j+1)$st move has not yet started. With these notation the prefix of S at time t is given by

$$(\tau_1, x_1, y_1, R_1), \ldots, (\tau_j, x_j, z, R_j).$$

Let $cost\colon \mathcal{R}^* \times \mathcal{S} \to \mathbb{R}_+ \cup \{+\infty\}$ be the objective function of *cost*-OLDARP (see Definition 2.3). We define the cost function $cost_i$, $i = 1, 2, \ldots$ in the request-answer game as follows:

$$cost_i(r_1, \ldots, r_i, a_1, \ldots, a_i) :=$$
$$\begin{cases} +\infty & \text{if } a_j \text{ is not a valid transportation schedule for } r_1, \ldots, r_j \\ & \text{for some } 1 \le j \le i \\ & \text{or} \\ & a_{j-1} \text{ and } a_j \text{ have different prefixes at time } t \\ & \text{for some } 1 \le j \le i \text{ and some } t \le t_j, \\ cost(a_i) & \text{otherwise.} \end{cases}$$

The case that a_{j-1} and a_j have different prefixes in the above definition corresponds to the fact that "history is irrevocable" for an online algorithm. At time t, all answers given so far (i.e., all transportation schedules produced so far) must have the same prefix.

2.2 Problem Definition

In the present chapter we consider the following objective functions for the OLDARP:

Closed Makespan $C^{o}_{\max}$: For a closed schedule S,

closed makespan

$$S = (\tau_1, x_1, y_1, R_1), \dots, (\tau_\ell, x_\ell, y_\ell, R_\ell),$$

the *closed makespan* $C^{o}_{\max}$ is defined to be the time when the schedule S ends, that is, $C^{o}_{\max}(S) = \tau_\ell + d(x_\ell, y_\ell)$. Moreover, $C^{o}_{\max}(S') := +\infty$ for all non-closed schedules S'.

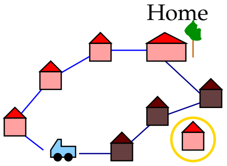
Makespan $C_{\max}$: The *makespan* $C_{\max}(S)$ of a schedule S equals the time when S ends, independent of whether S is closed or not.

makespan

Definition 2.5 (Online Dial-a-Ride Problem $C^{o}_{\max}$-/$C_{\max}$-OLDARP)
The problems $C^{o}_{\max}$-OLDARP and $C_{\max}$-OLDARP consist of finding a transportation schedule which starts at the origin and minimizes the closed makespan $C^{o}_{\max}$ and the (non-closed) makespan $C_{\max}$, respectively.

The objective function $C^{o}_{\max}$ may look somewhat artificial at first glance, since we force any competitive algorithm to make its server return to the origin at the end. However, other online problems, such as scheduling problems, can be formulated as a special case of $C^{o}_{\max}$-OLDARP (see Section 2.3). In particular, $C^{o}_{\max}$-OLDARP comprises the *online traveling salesman problem* (OLTSP) which was introduced in [AF^{+}01, AF^{+}95, AF^{+}94] as an online variant of the famous traveling salesman problem (TSP).

In the OLTSP cities (requests) arrive online over time while the salesman is traveling. The requests are to be handled by a salesman-server that starts and ends his work at a designated origin. The cost of such a route is the time when the server has served the last request and has returned to the origin (if the server does not return to the origin at all, then the cost of such a route is defined to be infinity). Notice that the OLTSP differs from its famous relative, the traveling salesman problem, in certain aspects: First, the cost of a feasible solution is not the length of the tour but the total travel-time needed by the server. The total travel time is obtained from the tour length plus the time where the server remains idle. Second, due to the online nature of the problem it may be unavoidable that a server reaches a certain point in the metric space more than once.



Online traveling salesman problem (OLTSP).

Definition 2.6 (Online Traveling Salesman Problem (OLTSP))
*The **online traveling salesman problem** (OLTSP) is the special case of the $C^{o}_{\max}$-OLDARP, when for each request r its source and destination coincide, that is, $\alpha(r) = \omega(r)$ for all r.*

If $\sigma = r_1, \ldots, r_m$ is a sequence of requests for the OLTSP we write briefly $r_j = (t_j, \alpha_j)$ instead of $r_j = (t_j, \alpha_j, \alpha_j)$. Observe that the capacity of the server is irrelevant in case of the OLTSP. We will investigate more aspects of the OLTSP in Chapter 3.

2.3 Application to Machine Scheduling

Suppose that there is a single machine which can process q different *types* $T_1, \ldots, T_q$ of jobs. A *setup cost* s_i is charged in order to process a job of type T_i, unless the preceeding job on the machine was of the same type. This setup cost models for instance the situation where special auxiliary devices must be installed at the machine to perform a certain job type.

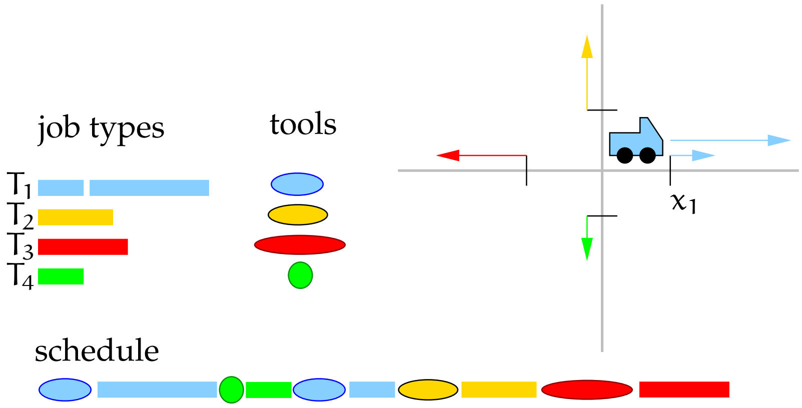
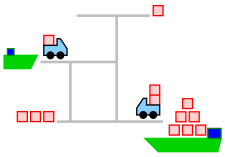


Figure 2.3
Modeling of a scheduling problem with setup costs (four job types) as C^o_{max}-OLDARP.

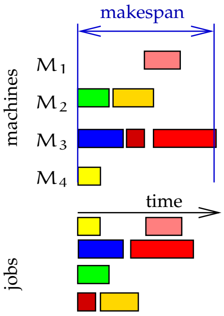
The problem of minimizing the makespan in this machine scheduling problem when jobs arrive over time can be modeled as a C^o_{max}-OLDARP with a unit-capacity server on a star-shaped metric space with center o (see Figure 2.3 for an illustration). For each of the q job types there is one ray in the metric space emanating from o. On ray i there is a special point x_i at distance $s_i/2$ from o. A job of type T_i with processing time p is modeled by a transportation request from the point x_i to the point $x_i + p/2$.

The above outlined transformation of the scheduling problem to the OLDARP can also be used for other objective functions such as the average/maximal flow time (see Chapter 4 for definitions). A c-competitive algorithm for *cost*-OLDARP with a unit capacity server then implies a c-competitive algorithm for the online scheduling problem of minimizing the objective function *cost* on a single machine when jobs arrive over time.



k-OLDARP: More than one server.

A natural generalization of the basic setting in OLDARP is to consider the situation where there are k servers with arbitrary capacities $C_1, \ldots, C_k \in \mathbb{N}$. We denote this generalized problem by k-OLDARP. The problem k-OLDARP with k unit-capacity servers can be used to model the scheduling problem with setup costs when there are k uniform machines instead of just a single machine. Using the above construction of the star-shaped metric space we obtain the following result:



Scheduling on parallel machines with jobs arriving over time.

Observation 2.7 *Suppose that there exists a* c*-competitive algorithm for the* k*-cost-*OLDARP *with unit capacity servers. Then there exists a* c*-competitive algorithm for the scheduling problem of minimizing the objective function cost on* k *uniform machines when jobs arrive over time.*

We will see later that the IGNORE and SMARTSTART-strategies (see Sections 2.5 and 2.6) can be applied to k-$C^{o}_{\max}$-OLDARP. Their competitive ratios of 5/2 and 2, respectively, hold as well in the generalized case. As a corollary, we obtain 2-competitive algorithms for a number of machine scheduling problems with setup-costs.

2.4 Lower Bounds

In this section we address the question how well online algorithms can perform compared to the optimal offline algorithm. To repeat the question posed in the introduction: How much does one lose by not having complete information?

Ausiello et al. established the following lower bounds for the OLTSP:

Theorem 2.8 ([AF$^+$01, AF$^+$95, AF$^+$94]) *Any deterministic online algorithm for the* OLTSP *in general metric spaces has a competitive ratio greater or equal to 2. Any deterministic online algorithm for the* OLTSP *on the real line has competitive ratio at least* $(9+\sqrt{17})/8$. □

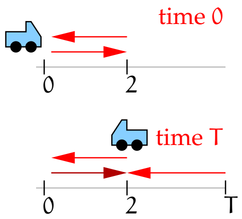
$\frac{9+\sqrt{17}}{8} \approx 1.640388203$

Since $C^{o}_{\max}$-OLDARP generalizes the OLTSP, the above theorem yields as a corollary a lower bound of 2 on the competitive ratio of any deterministic algorithm for $C^{o}_{\max}$-OLDARP in general metric spaces. As a consequence, the SMARTSTART-Algorithm presented in Section 2.6 is in fact best possible among all deterministic algorithms from a competitive analysis point of view.

Theorem 2.9 *Any deterministic online algorithm for the* $C^{o}_{\max}$*-*OLDARP *in general metric spaces has a competitive ratio greater or equal to 2. For the case of the real line, any determinstic algorithm has a competitive ratio at least* $c \geq 1+\sqrt{2}/2$.

$1+\sqrt{2}/2 \approx 1.707106781$

Proof: As noted above, the general lower bound is an immediate consequence of Theorem 2.8. We now address the case of the real line (with server capacity equal to one). Suppose that ALG is a deterministic online algorithm with competitive ratio $c \leq 1+\sqrt{2}/2$. We show that also $c \geq 1+\sqrt{2}/2$, which proves the claim of the theorem.



Lower bound construction in Theorem 2.9.

At time 0, the algorithm ALG is faced with two requests $r_1 = (0, o, 2)$ and $r_2 = (0, 2, o)$. The optimal offline cost to serve these two requests is 4.

The server operated by ALG must start serving request r_2 at some time $2 \le T \le 4c-2$, because otherwise ALG could not be c-competitive. At time T the adversary issues another request $r_3 = (T, T, 2)$. Then $\text{OPT}(r_1, r_2, r_3) = 2T$. On the other hand, $\text{ALG}(r_1, r_2, r_3) \ge 3T + 2$. Thus, the competitive ratio c of ALG satisfies

$$c \ge \frac{3T+2}{2T} = \frac{3}{2} + \frac{1}{T} \ge \frac{3}{2} + \frac{1}{4c-2}.$$

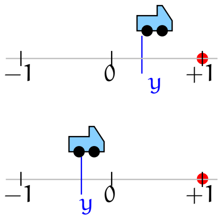
The smallest value $c \ge 1$ such that $c \ge 3/2 + 1/(4c-2)$ is $c = 1 + \sqrt{2}/2$. This completes the proof. □

We conclude the section with a lower bound for randomized algorithms. The lower bound will be shown for the OLTSP on the real line endowed with the usual Euclidean metric.

Theorem 2.10 *Any randomized algorithm for the* OLTSP *on the real line has competitive ratio greater or equal to 3/2 against an oblivious adversary.*

Proof: We use Yao's Principle (Theorem 1.12 on page 15) as a tool for deriving the lower bound. No request will be released before time 1. At time 1 with probability 1/2 there is a request at 1, and with probability 1/2 a request at -1. This yields a probability distribution X over the two request sequences $\sigma_1 = (1, 1)$ and $\sigma_2 = (1, -1)$.

Since $\text{OPT}(\sigma_1) = \text{OPT}(\sigma_2) = 2$ it follows that $\mathbb{E}_X[\text{OPT}(\sigma_x)] = 2$. We now calculate the expected cost of an arbitrary deterministic algorithm. Consider the deterministic online algorithm ALG_y which has its server at position $y \in \mathbb{R}$ at time 1 (clearly, any deterministic online algorithm is of this form). With probability 1/2, y is on the same side of the origin as the request which is released at time 1, with probability 1/2 the position y and the request are on opposite sides of the origin. In the first case, $\text{ALG}_y(\sigma_x) \ge 1 + (2 - y)$ (starting at time 1 the server has to move to 1 and back to the origin which needs time at least $2 - y$). In the other case, $\text{ALG}_y(\sigma_x) \ge 1 + (2 + y)$. This yields



Positions of the online server in the randomized lower bound construction in the proof of Theorem 2.10.

$$\mathbb{E}_X[\text{ALG}_y(\sigma_x)] = \frac{1}{2}(3 - y) + \frac{1}{2}(3 + y) = 3.$$

Hence $\mathbb{E}_X[\text{ALG}_y(\sigma_x)] \ge 3/2 \cdot \mathbb{E}_X[\text{OPT}(\sigma_x)]$ and the claimed lower bound follows by Yao's Principle. □

Corollary 2.11 *Any randomized algorithm for the* $C^o_{\max}$-OLDARP *on the real line has competitive ratio greater or equal to 3/2 against an oblivious adversary.* □

2.5 Two Simple Strategies

In this section we present and analyze two very natural online-strategies for $C^o_{\max}$-OLDARP. In Section 2.8 we show how to extend our results to the case of the $C_{\max}$-OLDARP where the server is not required to return to its initial position at the end of its work.

REPLAN

> **Algorithm REPLAN**
> As soon as a new request r_j arrives the server stops and replans: it computes a schedule with minimum length which starts at the current position of the server, takes care of all yet unserved requests (including those that are currently carried by the server), and ends at the origin. Then it continues using the new schedule.

IGNORE

> **Algorithm IGNORE**
> The server remains idle until the point in time t when the first requests become known. The algorithm then serves the requests released at time t immediately, following a shortest schedule S which starts and ends at the origin. All requests that arrive during the time when the algorithm follows S are temporarily ignored. After S has been completed and the server is back in the origin, the algorithm computes a shortest schedule for all unserved requests and follows this schedule. Again, all new requests that arrive during the time that the server is following the schedule are temporarily ignored. A schedule for the ignored requests is computed as soon as the server has completed its current schedule. The algorithm keeps on following schedules and temporarily ignoring requests in this way.

Both algorithms above repeatedly solve "offline instances" of the $C^o_{\max}$-OLDARP. These offline instances have the property that all release times are no less than the current time. Thus, the corresponding offline problem is the following: given a number of transportation requests (with release times all zero), find a shortest transportation for them. This offline dial-a-ride problem is investigated in Chapters 6 and 7.

offline dial-a-ride problem

For a sequence σ of requests and a point x in the metric space M, let $L^*(t, x, \sigma)$ denote the length of a shortest schedule (i.e., the time difference between its completion time and the start time t, see Definition 2.1) which starts in x at time t, serves all requests from σ (but not earlier than their release times) and ends in the origin.

$L^*(t, x, \sigma)$

Observation 2.12 *The function* L^* *has the following properties:*

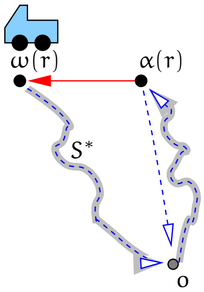
(i) $L^*(t', x, \sigma) \leq L^*(t, x, \sigma)$ *for all* $t' \geq t$;

(ii) $L^*(t, x, \sigma) \leq d(x, y) + L^*(t, y, \sigma)$ *for all* $t \geq 0$ *and all* $x, y \in X$*;*

(iii) $\text{OPT}(\sigma) = L^*(0, o, \sigma)$*;*

(iv) $\text{OPT}(\sigma) \geq L^*(t, o, \sigma)$ *for any time* $t \geq 0$.

We now derive another useful property of L^* for the special case that the server has unit-capacity. This result will be used in the proof of the competitiveness of the REPLAN-strategy.



Constructing a schedule starting at $\omega(r)$ (dashed lines) from S^* (thick solid lines).

Lemma 2.13 *Let* $\sigma = r_1, \ldots, r_m$ *be a sequence of requests for the* $C^o_{\max}$*-OLDARP with unit capacity, i.e., with* $C = 1$*. Then for any* $t \geq t_m$ *and any request* r *from* σ*,*

$$L^*(t, \omega(r), \sigma \backslash \{r\}) \leq L^*(t, o, \sigma) - d(\alpha(r), \omega(r)) + d(\alpha(r), o).$$

Here $\sigma \setminus \{r\}$ *denotes the sequence obtained from* σ *by deleting the request* r.

Proof: Consider a transportation schedule S^* which starts at the origin o at time t, serves all requests in σ and has length $L^*(t, o, \sigma)$. It suffices to construct another schedule S which starts in $\omega(r)$ no earlier than time t, serves all requests in $\sigma \backslash \{r\}$ and has length at most $L^*(t, o, \sigma) - d(\alpha(r), \omega(r)) + d(\alpha(r), o)$.

Let S^* serve the requests in the order $r_{j_1}, \ldots, r_{j_m}$ and let $r = r_{j_k}$. Notice that if we start in $\omega(r)$ at time t and serve the requests in the order

$$r_{j_k+1}, \ldots, r_{j_m}, r_{j_1}, \ldots, r_{j_{k-1}}$$

and move back to the origin, we obtain a schedule S with the desired properties. □

Let $\sigma = r_1, \ldots, r_m$ be any request sequence for $C^o_{\max}$-OLDARP. Since the optimal offline algorithm can not serve the last request $r_m = (t_m, \alpha_m, \omega_m)$ from σ before this request is released we get that

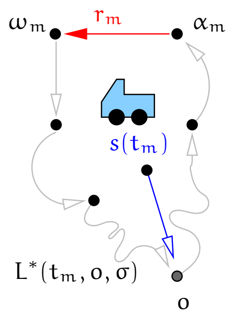
$$\text{OPT}(\sigma) \geq \max\{L^*(t, o, \sigma), t_m + d(\alpha_m, \omega_m) + d(\omega_m, o)\} \tag{2.1}$$

for any $t \geq 0$.

We are now ready to prove the first result about the performance of REPLAN:

Theorem 2.14 *Algorithm* REPLAN *is 7/2-competitive for the* $C^o_{\max}$*-OLDARP.*

Proof: Let $\sigma = r_1, \ldots, r_m$ be any sequence of requests. We first handle the general case with arbitrary capacity C of the server. We distinguish between two cases depending on the current load of the REPLAN-server at the time t_m, i.e., the time when the last request is released.

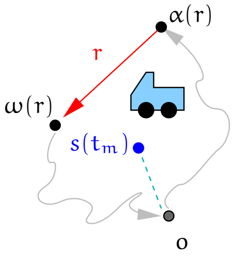


Theorem 2.14, Case 1: The server is empty at time t_m when the last request is released.

If the server is currently empty it recomputes an optimal schedule which starts at its current position, denoted by $s(t_m)$, serves all unserved requests, and returns to the origin. This schedule has length at most $L^*(t_m, s(t_m), \sigma) \leq d(o, s(t_m)) + L^*(t_m, o, \sigma)$. Thus,

$$\begin{aligned} \mathsf{REPLAN}(\sigma) &\leq t_m + d(o, s(t_m)) + L^*(t_m, o, \sigma) \\ &\leq t_m + d(o, s(t_m)) + \mathsf{OPT}(\sigma) \qquad \text{by (2.1)} \end{aligned} \tag{2.2}$$

Since the REPLAN server has traveled to position $s(t_m)$ at time t_m, there must be a request $r \in \sigma$ where either $d(o, \alpha(r)) \geq d(o, s(t_m))$ or $d(o, \omega(r)) \geq d(o, s(t_m))$. By the triangle inequality this implies that the optimal offline server will have to travel at least twice the distance $d(o, s(t_m))$ during its schedule. Thus, $d(o, s(t_m)) \leq \mathsf{OPT}(\sigma)/2$. Plugging this result into inequality (2.2) we get that the total time the REPLAN server needs is no more than $5/2\, \mathsf{OPT}(\sigma)$.

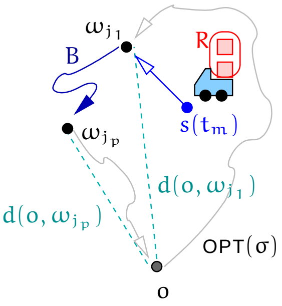


$\mathsf{OPT}(\sigma) \geq 2\, d(o, s(t_m))$

We now consider the second case, when the server is currently carrying a set of requests. Let $R = \{r_{j_1}, \dots, r_{j_p}\}$ with $p \leq C$ be the set of objects carried by the REPLAN-server at time t_m. Without loss of generality assume that the objects in R are dropped at their destination by OPT in the order $r_{j_1}, \dots, r_{j_p}$. Moreover, let B denote the length of a shortest Hamiltonian path in X which starts in ω_{j_1}, passes all points from $\{\omega_{j_1}, \dots, \omega_{j_{p-1}}\}$ and ends in ω_{j_p}.

By the triangle inequality we have

$$\mathsf{OPT}(\sigma) \geq d(o, \omega_{j_1}) + B + d(\omega_{j_p}, o). \tag{2.3}$$



Case 2 (general C): The server is carrying objects R.

On the other hand

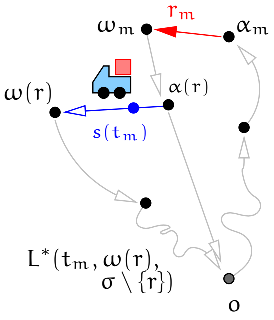
$$\begin{aligned} \mathsf{REPLAN}(\sigma) &\leq t_m + d(s(t_m), \omega_{j_1}) + B + d(o, \omega_{j_1}) + L^*(t_m, \omega_{j_p}, \sigma \setminus R) \\ &\leq t_m + d(s(t_m), o) + d(o, \omega_{j_1}) + B + d(\omega_{j_p}, o) + \mathsf{OPT}(\sigma) \\ &\leq t_m + d(s(t_m)) + 2\, \mathsf{OPT}(\sigma) \qquad \text{by (2.3)} \\ &\leq 7/2 \cdot \mathsf{OPT}(\sigma). \end{aligned}$$

Here, we have used (2.1) and $d(s(t_m), o) \leq \mathsf{OPT}(\sigma)/2$. This completes the proof of the fact that REPLAN is 7/2-competitive for general capacities. □

In case that the capacity of the server is one, we can prove a better result about the competitiveness of REPLAN:

Theorem 2.15 *In case of a server with unit-capacity* ($C = 1$), REPLAN *is* 5/2-*competitive for the* $C^o_{\max}$-OLDARP.

Proof: We have already shown in the proof of Theorem 2.14 above that the cost of the REPLAN-strategy is at most 5/2 times the optimum offline cost in case that the server is empty at the time t_m. Thus, it suffices to consider the case that the server is serving a request r at the time t_m when the last request r_m becomes known. The time needed to complete the current move is $d(s(t_m), \omega(r))$. A shortest schedule starting at $\omega(r)$ serving all unserved requests has length at most $L^*(t_m, \omega(r), \sigma \backslash \{r\})$. Thus, we have



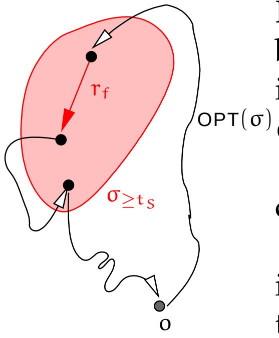
Case 2 (C=1): The server is serving request r.

$$\begin{aligned}
\text{REPLAN}(\sigma) &\le t_m + d(s(t_m), \omega(r)) + L^*(t_m, \omega(r), \sigma \backslash \{r\}) \\
&\le t_m + d(s(t_m), \omega(r)) + L^*(t_m, o, \sigma) \\
&\qquad - d(\alpha(r), \omega(r)) + d(\alpha(r), o) \quad \text{by Lemma 2.13} \\
&\le t_m + \text{OPT}(\sigma) - d(\alpha(r), \omega(r)) \\
&\quad + \underbrace{d(s(t_m), \omega(r)) + d(\alpha(r), s(t_m))}_{= d(\alpha(r), \omega(r))} + d(s(t_m), o) \\
&= t_m + d(o, s(t_m)) + \text{OPT}(\sigma).
\end{aligned}$$

Hence, inequality (2.2) also holds in case that the server is carrying an object at time t_m. As argued above, $t_m + d(o, s(t_m)) + \text{OPT}(\sigma) \le 5/2\, \text{OPT}(\sigma)$. This completes the proof. □

We are now going to analyze the competitiveness of the second simple strategy IGNORE.

Theorem 2.16 *Algorithm* IGNORE *is 5/2-competitive for the* $C^o_{\max}$-OLDARP.



Proof of Theorem 2.16: The inequality $\text{OPT}(\sigma) \ge t_S + L^*(t_S, \alpha_f, \sigma_{\ge t_S})$ holds.

Proof: We consider again the point in time t_m when the last request r_m becomes known. If the IGNORE-server is currently idle at the origin o, then it completes its last schedule no later than time $t_m + L^*(t_m, o, \sigma_{=t_m})$, where $\sigma_{=t_m}$ is the set of requests released at time t_m.

Since $L^*(t_m, o, \sigma_{=t_m}) \le \text{OPT}(\sigma)$ and $\text{OPT}(\sigma) \ge t_m$, it follows that in this case IGNORE completes no later than time $2\,\text{OPT}(\sigma)$.

It remains the case that at time t_m the IGNORE-server is currently working on a schedule S for a subset σ_S of the requests. Let t_S denote the starting time of this schedule. Thus, the IGNORE-server will complete S at time $t_S + L^*(t_S, o, \sigma_S)$. Denote by $\sigma_{\ge t_S}$ the set of requests presented after the IGNORE-server started with S at time t_S. Notice that $\sigma_{\ge t_S}$ is exactly the set of requests that are served by IGNORE in its last schedule. The IGNORE-server will complete its total service no later than time $t_S + L^*(t_S, o, \sigma_S) + L^*(t_m, o, \sigma_{\ge t_S})$.

Let $r_f \in \sigma_{\ge t_S}$ be the first request from $\sigma_{\ge t_S}$ served by OPT. Thus

$$\text{OPT}(\sigma) \ge t_f + L^*(t_f, \alpha_f, \sigma_{\ge t_S}) \ge t_S + L^*(t_m, \alpha_f, \sigma_{\ge t_S}). \tag{2.4}$$

Now, $L^*(t_m, o, \sigma_{\geq t_S}) \leq d(o, \alpha_f) + L^*(t_m, \alpha_f, \sigma_{\geq t_S})$ and $L^*(t_S, o, \sigma_S) \leq \text{OPT}(\sigma)$. Therefore,

$$\begin{aligned} \text{IGNORE}(\sigma) &\leq t_S + \text{OPT}(\sigma) + d(o, \alpha_f) + L^*(t_m, \alpha_f, \sigma_{\geq t_S}) \\ &\leq 2\,\text{OPT}(\sigma) + d(o, \alpha_f) && \text{by (2.4)} \\ &\leq \frac{5}{2}\,\text{OPT}(\sigma). \end{aligned}$$

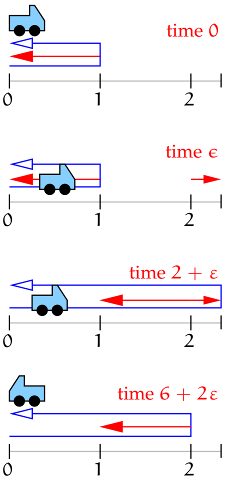
This completes the proof. □

The following set of instances shows that the competitive ratio of 5/2 proved for IGNORE is asymptotically tight even for the case when the metric space M is the real line.

At time 0 there is a request $r_1 = (0, 1, o)$. The next requests are $r_2 = (\varepsilon, 2, 2+\varepsilon)$ and $r_3 = (2+\varepsilon, 2, 1)$. It is easy to see that $\text{IGNORE}(r_1, r_2, r_3) = 10 + 2\varepsilon$, while $\text{OPT}(r_1, r_2, r_3) = 4 + 2\varepsilon$. Thus, the ratio

$$\frac{\text{IGNORE}(r_1, r_2, r_3)}{\text{OPT}(r_1, r_2, r_3)} = \frac{5+\varepsilon}{2+\varepsilon}$$

can be made arbitrarily close to 5/2 by choosing $\varepsilon > 0$ small enough.



Worst-case sequence for IGNORE.

Strategy IGNORE can be easily generalized to the case of k-$C^o_{\max}$-OLDARP (see Page 26). For k-$C^o_{\max}$-OLDARP the IGNORE strategy always plans schedules for its servers such that the length of the longest schedule is minimized. All schedules are constructed in such a way that they start and end in the origin. New requests are ignored until the last of the servers has returned to the origin. It is not too hard to see that the proof of Theorem 2.16 remains valid even for k-$C^o_{\max}$-OLDARP. This yields the following result:

Theorem 2.17 IGNORE *is* 5/2*-competitive even for the* k-$C^o_{\max}$-OLDARP. □

2.6 A Best-Possible Online-Algorithm

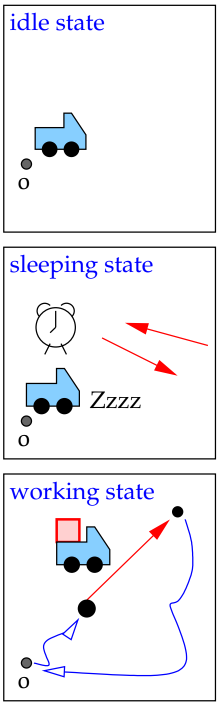
In this section we present and analyze our algorithm SMARTSTART which achieves a best-possible competitive ratio of 2 (cf. the lower bound given in Theorem 2.8). The idea of the algorithm is basically to emulate the IGNORE-strategy but to make sure that each sub-transportation schedule is completed "not too late": if a sub-schedule would take "too long" to complete then the algorithm waits for a specified amount of time. Intuitively this construction tries to avoid the worst-case situation for IGNORE where right after the algorithm starts a schedule a new request becomes known.

SMARTSTART has a fixed "waiting scaling" parameter $\theta > 1$. From time to time the algorithm consults its "work-or-sleep" routine: this subroutine

"work-or-sleep" routine

computes an (approximately) shortest schedule S for all unserved requests, starting and ending in the origin. If this schedule can be completed no later than time θt, i.e., if $t+l(S) \leq \theta t$, where t is the current time and $l(S)$ denotes the length of the schedule S, the subroutine returns (S, work), otherwise it returns (S, sleep).

In the sequel it will be convenient to assume that the "work-or-sleep" subroutine uses a ρ-approximation algorithm for computing a schedule: the approximation algorithm always finds a schedule of length at most ρ times the optimal one. While in online computation one is usually not interested in time complexity (and thus in view of competitive analysis we can assume that $\rho = 1$), employing a polynomial-time approximation algorithm will enable us to get a practical algorithm (and in particular to improve the result of [AF+01, AF+95] for the OLTSP).



States of the SMARTSTART-server.

The server of algorithm SMARTSTART can assume three states:

idle In this case the server has served all known requests, is sitting in the origin and waiting for new requests to occur.

sleeping In this case the server is sitting at the origin and knows of some unserved requests but also knows that they take too long to serve (what "too long" means will be formalized in the algorithm below).

working In this state the algorithm (or rather the server operated by it) is following a computed schedule.

We now formalize the behavior of the algorithm by specifying how it reacts in each of the three states.

SMARTSTART

Algorithm SMARTSTART

If the algorithm is idle at time T and new requests arrive, calls "work-or-sleep". If the result is (S, work), the algorithm enters the working state where it follows schedule S. Otherwise the algorithm enters the sleeping state with wakeup time t', where $t' \geq T$ is the earliest time such that $t'+l(S) \leq \theta t'$ and $l(S)$ denotes the length of the just computed schedule S, i.e., $t' = \min\{\, t \geq T : t+l(S) \leq \theta t \,\}$.

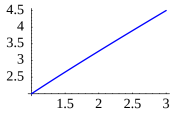
In the sleeping state the algorithm simply does nothing until its wakeup time t'. At this time the algorithm reconsults the "work-or-sleep" subroutine. If the result is (S, work), then the algorithm enters the working state and follows S. Otherwise the algorithm continues to sleep with new wakeup time $\min\{\, t \geq t' : t+l(S) \leq \theta t \,\}$.

In the working state, i.e, while the server is following a schedule, all new requests are (temporarily) ignored. As soon as the current schedule is completed the server either enters the idle-state (if there are no unserved requests) or it reconsults the "work-or-sleep" subroutine which determines the next state (sleeping or working).

Theorem 2.18 *For all real numbers* $\theta \geq \rho$ *with* $\theta > 1$, *Algorithm* SMARTSTART *is* c*-competitive for the* $C^{o}_{\max}$-OLDARP *with*

$$c = \max\left\{\theta, \rho\left(1 + \frac{1}{\theta - 1}\right), \frac{\theta}{2} + \rho\right\}.$$

Moreover, the best possible choice of θ *is* $\frac{1}{2}\left(1 + \sqrt{1 + 8\rho}\right)$ *and yields a competitive ratio of* $c(\rho) := \frac{1}{4}\left(4\rho + 1 + \sqrt{1 + 8\rho}\right)$.



Competitive ratio $c(\rho)$ of SMARTSTART for $\rho \geq 1$.

Proof: Let $\sigma_{=t_m}$ be the set of requests released at time t_m, where t_m denotes again the point in time when the last requests becomes known. We distinguish between different cases depending on the state of the SMARTSTART-server at time t_m:

Case 1: The server is idle.

In this case the algorithm consults its "work-or-sleep" routine which computes an approximately shortest schedule S for the requests in $\sigma_{=t_m}$. The SMARTSTART-server will start its work at time $t' = \min\{t \geq t_m : t + l(S) \leq \theta t\}$, where $l(S) \leq \rho L^*(t_m, o, \sigma_{=t_m})$ denotes the length of the schedule S.

If $t' = t_m$, then by construction the algorithm completes no later than time $\theta t_m \leq \theta\, \text{OPT}(\sigma)$. Otherwise $t' > t_m$ and it follows that $t' + l(S) = \theta t'$. By the performance guarantee ρ of the approximation algorithm employed in "work-or-sleep", we have that $\text{OPT}(\sigma) \geq l(S)/\rho = \frac{\theta-1}{\rho} t'$. Thus, it follows that

$$\begin{aligned}\text{SMARTSTART}(\sigma) &= t' + l(S)\\ &\leq \theta t' \leq \theta \cdot \frac{\rho\, \text{OPT}(\sigma)}{\theta - 1} = \rho\left(1 + \frac{1}{\theta - 1}\right) \text{OPT}(\sigma).\end{aligned}$$

Case 2: The server is sleeping.

Note that the wakeup time of the server is no later than $\min\{t \geq t_m : t + l(S) \leq \theta t\}$, where S is now a shortest schedule for all the requests in σ not yet served by SMARTSTART at time t_m, and we can proceed as in Case 1.

Case 3: The algorithm is working.

If after completion of the current schedule the server enters the sleeping state, then the arguments given above establish that the completion time of the SMARTSTART-server does not exceed $\rho\left(1 + \frac{1}{\theta-1}\right) \text{OPT}(\sigma)$.

The remaining case is that the SMARTSTART-server starts its final schedule S' immediately after having completed S. Let t_S be the time when the server started S and denote by $\sigma_{\geq t_S}$ the set of requests presented after the server started S at time t_S. Notice that $\sigma_{\geq t_S}$ is exactly the set of requests that are served by SMARTSTART in its last schedule S'.

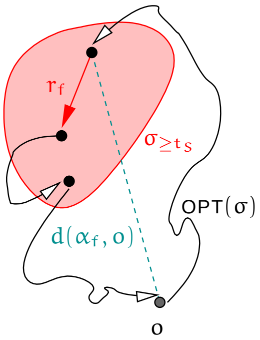
$$\text{SMARTSTART}(\sigma) = t_S + l(S) + l(S'). \tag{2.5}$$

Here, $l(S)$ and $l(S') \leq \rho L^*(t_m, o, \sigma_{\geq t_S})$ denotes the length of the schedule S and S', respectively. We have that

$$t_S + l(S) \leq \theta t_S, \tag{2.6}$$

since the SMARTSTART only starts a schedule at some time t if it can complete it not later than time θt. Let $r_f \in \sigma_{\geq t_S}$ be the first request from $\sigma_{\geq t_S}$ served by OPT.

Using the arguments given in the proof of Theorem 2.16 we conclude as in (2.4)that



The lower bound: $\text{OPT}(\sigma) \geq t_S + d(o, \alpha_f)$.

$$\text{OPT}(\sigma) \geq t_S + L^*(t_m, \alpha_f, \sigma_{\geq t_S}). \tag{2.7}$$

Moreover, since the tour of length $L^*(t_m, \alpha_f, \sigma_{\geq t_S})$ starts in α_f and returns to the origin, it follows from the triangle inequality that

$$L^*(t_m, \alpha_f, \sigma_{\geq t_S}) \geq d(o, \alpha_f).$$

Thus, from (2.7) we get

$$\text{OPT}(\sigma) \geq t_S + d(o, \alpha_f). \tag{2.8}$$

On the other hand

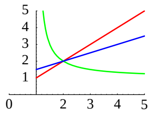
$$\begin{aligned} l(S') &\leq \rho\left(d(o, \alpha_f) + L^*(t_m, \alpha_f, \sigma_{\geq t_S})\right) \\ &\leq \rho\left(d(o, \alpha_f) + \text{OPT}(\sigma) - t_S\right) && \text{by (2.7).} \end{aligned} \tag{2.9}$$

Using (2.6) and (2.9) in (2.5) and the assumption that $\theta \geq \rho$, we obtain

$$\begin{aligned} \text{SMARTSTART}(\sigma) &\leq \theta t_S + l(S') && \text{by (2.6)} \\ &\leq (\theta - \rho) t_S + \rho\, d(o, \alpha_f) + \rho\, \text{OPT}(\sigma) && \text{by (2.9)} \\ &\leq \theta\, \text{OPT}(\sigma) + (2\rho - \theta) d(o, \alpha_f) && \text{by (2.8)} \\ &\leq \begin{cases} \theta\, \text{OPT}(\sigma) + (2\rho - \theta)\frac{\text{OPT}(\sigma)}{2} & \text{, if } \theta \leq 2\rho \\ \theta\, \text{OPT}(\sigma) & \text{, if } \theta > 2\rho \end{cases} \\ &\leq \max\left\{\frac{\theta}{2} + \rho, \theta\right\} \text{OPT}(\sigma) \end{aligned}$$

This completes the proof. □

For "pure" competitive analysis we may assume that each schedule S computed by "work-or-sleep" is in fact an optimal schedule, i.e., that $\rho = 1$. The best competitive ratio for SMARTSTART is then achieved for that value of θ where the three terms θ, $1 + \frac{1}{\theta - 1}$ and $\frac{\theta}{2} + 1$ are equal. This is the case for $\theta = 2$ and yields a competitive ratio of 2. We thus obtain the following corollary.



The three terms θ, $1 + \frac{1}{\theta-1}$ and $\frac{\theta}{2} + 1$ coincide for $\theta = 2$.

Corollary 2.19 *For $\rho = 1$ and $\theta = 2$, Algorithm* SMARTSTART *is 2-competitive for the $C^{o}_{\max}$-OLDARP.* □

OLTSP

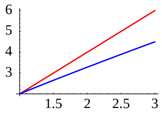
For the special case of the OLTSP Christofides' algorithm [Chr76] yields a polynomial-time approximation algorithm to solve the offline instances in the "work-or-sleep" subroutine with a performance of $\rho = 3/2$. For this value of ρ, the best competitive ratio of SMARTSTART is attained for $\theta = \frac{1+\sqrt{13}}{2}$ and equals $\frac{7+\sqrt{13}}{4}$.

$\frac{7+\sqrt{13}}{4} \approx 2.651387819$

Thus, our algorithm SMARTSTART can be used to obtain a polynomial-time competitive algorithm for the OLTSP with competitive ratio approximately 2.6514. This improves the result of [AF^{+}01, AF^{+}95] where a 3-competitive polynomial-time algorithm for the OLTSP was given.

Note that the SMARTSTART-strategy inherits some desirable properties from the IGNORE-strategy: The algorithm can also be used for the k-$C^{o}_{\max}$-OLDARP and provides the same competitive ratio.

Theorem 2.20 SMARTSTART *is 2-competitive for the* k-$C^{o}_{\max}$-OLDARP. □



Improved competitive ratio $c(\rho)$ of SMARTSTART-based polynomial-time algorithm vs. the competitive ratio of 2ρ from [SWW95] for $\rho \geq 1$.

The above theorem implies a 2-competitive algorithm for the machine scheduling problems with setup costs discussed in Section 2.3. Moreover, we can use SMARTSTART to obtain competitive polynomial-time algorithms for scheduling problems. Shmoys et al. showed the following result:

Theorem 2.21 ([SWW95]) *Suppose that there exists a polynomial-time ρ-approximation algorithm for the machine scheduling problem of minimizing the makespan in an environment where all jobs to be scheduled are available at time 0. For the analogous environment in which the existence of a job is unknown until its release date, there exists a polynomial-time algorithm which is 2ρ-competitive.* □

Using the transformation shown in Section 2.3 and applying the result from Theorem 2.18 about SMARTSTART, we obtain a polynomial-time scheduling algorithm which is $c(\rho)$-competitive with

$$c(\rho) = \frac{1}{4}\left(4\rho + 1 + \sqrt{1 + 8\rho}\right).$$

Since $c(\rho) < 2\rho$ for all $\rho > 1$, the Algorithm SMARTSTART improves the result from [SWW95] stated in the above theorem. In particular, we

$\frac{9+\sqrt{17}}{4} \approx$ 3.280776406

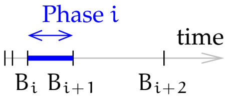
obtain a c(2)-competitive polynomial-time algorithm for scheduling on unrelated machines, where $c(2) = (9 + \sqrt{17})/4$. The previously best algorithm achieved a competitive ratio of 4 [SWW95].

2.7 A Simple Randomized Algorithm

In this section, we present a competitive randomized algorithm RANDSLEEP for $C^{o}_{\max}$-OLDARP. The competitive ratio of $1 + 1/\ln 2$, achieved by RANDSLEEP does not improve upon SMARTSTART. However, RANDSLEEP beats REPLAN and IGNORE. Moreover, the beauty of RANDSLEEP lies in its simplicity and the easy proof of its performance.

RANDSLEEP

Algorithm RANDSLEEP
Initialization: At the start, the algorithm chooses a random number $\delta \in (0, 1]$ according to the uniform distribution. After this random choice, the algorithm is completely deterministic.



Values B_i and phases in Algorithm RANDSLEEP.

Set L to be the earliest time when a request could 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 0. These requests are served at no cost). Until time L remain in 0. For $i = 0, 1, 2, \ldots,$ set $B_i := 2^{i-\delta}L$.

Phase i, for all $i = 1, 2, \ldots$: Phase i is started at time B_i. At this time, the algorithm considers all requests R_i that have been released up to time B_i but not been served yet. RANDSLEEP computes a shortest schedule S_i which starts and ends in the origin and which serves all requests in R_i.

If this schedule can be completed no later than time B_{i+1}, the server follows this schedule. Otherwise, if the schedule needs more than $B_{i+1} - B_i = B_i$ time units, then the server simply does nothing: it sleeps until time B_{i+1}.

In order to analyze the performance of RANDSLEEP we need an elementary lemma. This lemma will again be useful in Chapter 5, where we provide a randomized algorithm for the objective function of minimizing the sum of completion times.

Lemma 2.22 *Let* $z, L \in \mathbb{R}^+_0$ *with* $z \geq L$, *and let* $\delta \in (0, 1]$ *be a random variable uniformly distributed on* $(0, 1]$. *Define* B *by*

$$B := \max\{\, 2^{k-\delta}L : 2^{k-\delta}L < z \text{ and } k \in \mathbb{N} \,\}.$$

Then, the expected value of B *satisfies:* $\mathbb{E}\,[B] = \frac{z}{2\ln 2}$.

Proof: Suppose that $2^k L \leq z < 2^{k+1} L$ for some $k \geq 0$. Observe that

$$B = \begin{cases} 2^{k-\delta} L & \text{if } \delta \leq \log_2 \frac{2^{k+1} L}{z} \\ 2^{k+1-\delta} L & \text{otherwise} \end{cases}$$

Hence

$$\begin{aligned} \mathbb{E}\left[B\right] &= \int_0^{\log_2 \frac{2^{k+1}L}{z}} 2^{k-\delta} L \, d\delta + \int_{\log_2 \frac{2^{k+1}L}{z}}^{1} 2^{k+1-\delta} L \, d\delta \\ &= L 2^k \left[-\frac{1}{\ln 2} 2^{-\delta} \right]_0^1 + L 2^k \left[-\frac{1}{\ln 2} 2^{-\delta} \right]_{\log_2 \frac{2^{k+1}L}{z}}^{1} \\ &= \frac{z}{2 \ln 2}. \end{aligned}$$

This completes the proof. □

We are now ready to prove the competitiveness of RANDSLEEP.

Theorem 2.23 RANDSLEEP *is* c*-competitive for the* $C^o_{\max}$*-OLDARP, where* $c = 1 + 1/\ln 2$.

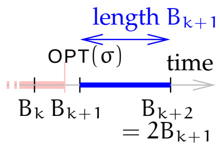
$1 + 1/\ln 2 \approx 2.442695041$

Proof: Let $k \in \mathbb{N}$ be such that $2^{k-x} L < \text{OPT}(\sigma) \leq 2^{k+1-x} L$ (such a k exists, since L is a lower bound on $\text{OPT}(\sigma)$ and $\delta > 0$). From Lemma 2.22 we obtain that $\mathbb{E}\left[2^{k-\delta} L\right] = \text{OPT}(\sigma)/(2 \ln 2)$.

From the fact that the optimal schedule can be completed before time $2^{k+1-\delta} L$, we can conclude two facts: first, all requests have been released by time $2^{k+1-\delta} L$, and second, in the $(k+1)$st phase RANDSLEEP scheduled all requests, since in this phase it allows $2^{k+1-\delta} L$ time units for a schedule which serves all remaining requests. Clearly, the length of the schedule computed in the $(k+1)$st phase can not exceed $\text{OPT}(\sigma)$. Hence,

$$\mathbb{E}\left[\text{RANDSLEEP}(\sigma)\right] \leq 4 \mathbb{E}\left[2^{k-1-\delta} L\right] + \text{OPT}(\sigma) = \left(1 + \frac{1}{\ln 2}\right) \text{OPT}(\sigma).$$

This is what we wanted to show. □

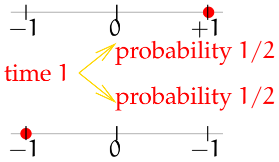


Phase $k+1$ in RANDSLEEP allows time $B_{k+1} \geq \text{OPT}(\sigma)$.

2.8 Extension to the Non-Closed Makespan

We briefly show how to extend our results to the case where the objective is the (non-closed) makespan $C_{\max}$ and, hence, the server is not required to return to the origin at the end of its service.

2.8.1 Lower Bounds



Randomized lower bound construction in Theorem 2.24.

Theorem 2.24 *Any randomized algorithm for the* OLTSP *on the real line has competitive ratio greater or equal to 2 against an oblivious adversary.*

Proof: The construction is essentially the same as in Theorem 2.10: No request will be released before time 1. At time 1, with probability $1/2$ there is a request at 1 and with probability $1/2$ a request at -1. It is straightforward to verify that for any deterministic algorithm this yields an expected cost of at least 2 while the expected offline cost equals 1. The claim of the theorem now follows by Yao's Principle. □

Corollary 2.25 *Any randomized algorithm (and thus also any deterministic algorithm) for the* $C_{\max}$-OLDARP *on the real line has competitive ratio greater or equal to 2 against an oblivious adversary.* □

2.8.2 Competitive Ratios of the Algorithms

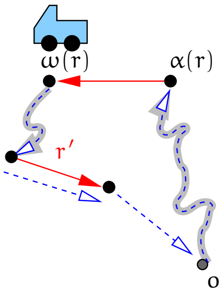
The algorithms REPLAN, IGNORE, and SMARTSTART are modified in such a way that each schedule computed is a shortest schedule which starts at the current position of the server, but not necessarily ends at the origin. In addition, we build into REPLAN the constraint that its server only moves on shortest paths between points of known requests, that is, at any time t, the REPLAN-server only moves on shortest paths between points in the set $\{\alpha(r), \omega(r) : r \in \sigma_{\leq t}\}$.

For a sequence σ of requests and a point x in the metric space M let $\tilde{L}^*(t, x, \sigma)$ denote the length of a shortest schedule which starts in x at time t and serves all requests from σ. The difference to $L^*(t, x, \sigma)$ defined in Section 2.5 is that we do not require the schedule to end at the origin.

For $t' \geq t$ we have that $\tilde{L}^*(t', x, \sigma) \leq \tilde{L}^*(t, x, \sigma)$. Moreover, $\text{OPT}(\sigma) = \tilde{L}^*(0, o, \sigma)$ and thus $\text{OPT}(\sigma) \geq \tilde{L}^*(t, o, \sigma)$ for any time $t \geq 0$. Since the optimal offline server OPT can not serve the last request r_m from σ before this request is released at time $t_m = t(r_m)$ we get that

$$\text{OPT}(\sigma) \geq \max\{L^*(t, o, \sigma), t_m + d(\alpha_m, \omega_m)\} \quad \text{for any } t \geq 0. \tag{2.10}$$

The following lemma can be proved similarly to Lemma 2.13:



Schedule starting at $\omega(r)$ (dashed) from $\tilde{L}(t, o, \sigma)$ (solid).

Lemma 2.26 *Let* $\sigma = r_1, \ldots, r_m$ *be a sequence of requests for the* $C_{\max}$-OLDARP *with unit capacity, i.e., with* $C = 1$*. Then for any* $t \geq t_m$ *and any request* $r \in \sigma$ *there exists a* $r' \in \sigma$ *such that*

$$\tilde{L}^*(t, \omega(r), \sigma \setminus \{r\}) \leq \tilde{L}^*(t, o, \sigma) - d(\alpha(r), \omega(r)) + d(\omega(r'), o),$$

in particular

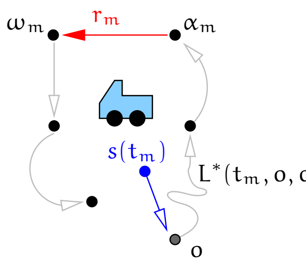
$$\tilde{L}^*(t, \omega(r), \sigma \setminus \{r\}) \leq 2\tilde{L}^*(t, o, \sigma) - d(\alpha(r), \omega(r)).$$

□

We are now ready to establish the analogue of Theorem 2.14 for $C_{\max}$.

Theorem 2.27 *Algorithm* REPLAN *is 9/2-competitive for the* $C_{\max}$*-*OLDARP.

Proof: If at the time t_m when the last requests from σ are released, the REPLAN-server is empty, then the total time needed by REPLAN is no more than



Case 1: The REPLAN-server is empty at time t_m.

$$\begin{aligned} t_m + \tilde{L}^*(t_m, s(t_m), \sigma) &\leq t_m + d(s(t_m), o) + \tilde{L}^*(t_m, o, \sigma) \\ &\leq d(o, s(t_m)) + 2\,\text{OPT}(\sigma). \end{aligned}$$

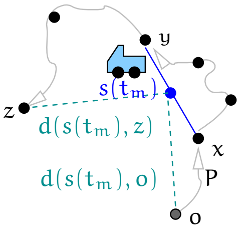
Since $d(o, s(t_m)) \leq t_m$ by the unit-speed of the server, it follows from the above inequality that $\text{REPLAN}(\sigma) \leq 3\,\text{OPT}(\sigma)$.

Now consider the situation where at time t_m the REPLAN-server is carrying a set $R = \{r_{j_1}, \ldots, r_{j_p}\}$ of requests. By construction, $s(t_m)$ is on a shortest path between two points x and y, where

$$x, y \in \{\, \alpha(r) : r \in \sigma_{\leq t_m} \,\} \cup \{\, \omega(r) : r \in \sigma_{\leq t_m} \,\}.$$

Let $P = (v_0 = o, v_1, \ldots, v_{p+2})$ be a shortest Hamiltonian path in X through all the points $\{x, y, \omega_{j_1}, \ldots, \omega_{j_p}\}$ subject to the constraint that P starts at the origin o.

We denote by $d(P) = \sum_{i=0}^{p+1} d(v_i, v_{i+1})$ the length of P and by z the endpoint of P. Without loss of generality we assume that, starting at o, the point x is visited before y on P. Then with the help of the triangle inequality we obtain

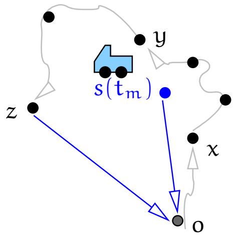


Case 2: At time t_m, the REPLAN-server is working. Path P is a shortest Hamiltonian path.

$$\begin{aligned} \text{OPT}(\sigma) &\geq d(P) \\ &\geq d(o, x) + d(x, y) + d(y, z) \\ &= d(o, x) + d(x, s(t_m)) + d(s(t_m), y) + d(y, z) \\ &\geq d(s(t_m), o) + d(s(t_m), z) \qquad (2.11) \end{aligned}$$

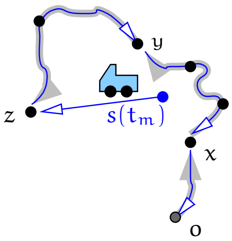
Here we have used the fact that $s(t_m)$ is on a shortest path between x and y. From (2.11) we see that $d(o, s(t_m)) \leq d(P)/2$ or $d(s(t_m), z) \leq d(P)/2$.

If $d(o, s(t_m)) \leq d(P)/2$, then, starting at time t_m, the REPLAN-server could move from it current position at $s(t_m)$ to the origin (which needs $d(s(t_m), o) \leq d(P)/2$ units of time), then follow P until its endpoint z and then move back to the origin empty. After that, all unserved requests can be served at cost at most $\tilde{L}(t_m, o, \sigma) \leq \text{OPT}(\sigma)$. Hence



Case (a): $d(o, s(t_m)) \leq d(P)/2$.

$$\begin{aligned}\text{REPLAN}(\sigma) &\leq t_m + \frac{3}{2}d(P) + d(z,o) + \text{OPT}(\sigma) \\ &\leq \frac{5}{2}\,\text{OPT}(\sigma) + t_m + d(z,o) && \text{by (2.11)} \\ &\leq \frac{9}{2}\,\text{OPT}(\sigma).\end{aligned}$$



Case (b): $d(s(t_m), z) \leq d(P)/2$.

Here, in order to obtain the last inequality we have used the fact that $\text{OPT}(\sigma) \geq t_m$ and $\text{OPT}(\sigma) \geq d(o,z)$, where the latter estimate stems from the fact that z is either a source or a target of a request from σ.

If $d(s(t_m), z) \leq d(P)/2$, then at time t_m one option for REPLAN is as follows: Move from $s(t_m)$ to z (at cost $d(s(t_m), z) \leq d(P)/2$), then follow P in reverse direction from z to the origin o. At this point the server is empty and can serve the remaining requests at cost at most $\text{OPT}(\sigma)$. The total time needed by the REPLAN-server is

$$\text{REPLAN}(\sigma) \leq t_m + \frac{3}{2}d(P) + \text{OPT}(\sigma) \leq \frac{7}{2}\,\text{OPT}(\sigma).$$

This proves the claim of the theorem. □

As in the case of the closed makespan $C^o_{\max}$, we can establish a better competitive ratio for a server with unit-capacity.

Theorem 2.28 *In case of a unit capacity server,* REPLAN *is* 3*-competitive for the* $C_{\max}$-OLDARP.

Proof: In view of the proof of Theorem 2.27 given above, we only need to consider the situation where the REPLAN-server is serving a request r at time t_m. We have

$$\begin{aligned}\text{REPLAN}(\sigma) &\leq t_m + d(s(t_m), \omega(r)) + \tilde{L}^*(t_m, \omega(r), \sigma) \\ &\leq t_m + d(s(t_m), \omega(r)) + 2\,\tilde{L}^*(t_m, o, \sigma) - d(\alpha(r), \omega(r)) \\ &\qquad\qquad\qquad\qquad\qquad\qquad \text{by Lemma 2.26} \\ &\leq t_m + 2\,\tilde{L}^*(t_m, o, \sigma) \\ &\leq 3\,\text{OPT}(\sigma).\end{aligned}$$

This completes the proof. □

We now turn to Algorithm IGNORE.

Theorem 2.29 *Algorithm* IGNORE *is* 4*-competitive for the* $C_{\max}$-OLDARP.

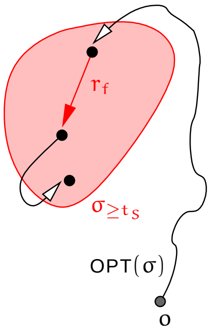
Proof: The only interesting case is that at the time t_m, when the last requests becomes known, the IGNORE server is currently working on a schedule S. Suppose that S was started at time t_S and has starting point x and endpoint y. Then the schedule will be completed no later than time $t_S + \tilde{L}^*(t_S, x, \sigma_S)$, where σ_S denotes the subset of requests served in the current schedule S. Since by construction of IGNORE all requests in σ_S have release times at least t_S, it follows that

$$\tilde{L}^*(t_S, x, \sigma_S) = \tilde{L}^*(t_m, x, \sigma_S) \leq d(o, x) + \text{OPT}(\sigma).$$

Hence, the IGNORE server will complete its total work no later than time

$$t_S + d(o, x) + \text{OPT}(\sigma) + \tilde{L}^*(t_m, y, \sigma_{\geq t_S}),$$

where $\sigma_{\geq t_S}$ denotes the set of requests released no earlier than t_S. Let r_f be the first request from the set $\sigma_{\geq t_S}$ of ignored requests served by OPT. Then



Proof of Theorem 2.29: $\text{OPT}(\sigma) \geq t_f + \tilde{L}^*(t_S, \alpha_f, \sigma_{\geq t_S})$

$$\begin{aligned}\text{OPT}(\sigma) \geq t_f + \tilde{L}^*(t_S, \alpha_f, \sigma_{\geq t_S}) &\geq t_S + \tilde{L}^*(t_S, \alpha_f, \sigma_{\geq t_S}) \\ &\geq t_S + \tilde{L}^*(t_m, \alpha_f, \sigma_{\geq t_S}) \\ &\geq t_S + \tilde{L}^*(t_m, y, \sigma_{\geq t_S}) - d(y, \alpha_f).\end{aligned}$$

Thus, we have that

$$\text{IGNORE}(\sigma) \leq d(o, x) + d(y, \alpha_f) + 2\,\text{OPT}(\sigma)$$

It is easy to see that both values $d(x, o)$ and $d(y, \alpha_f)$ are bounded from above by $\text{OPT}(\sigma)$, and so the theorem follows. □

We finally address the SMARTSTART-strategy.

Theorem 2.30 *For all $\theta \geq 2$, Algorithm* SMARTSTART *is c-competitive for the* $C_{\max}$-OLDARP *with*

$$c = \max\left\{\theta + 1, 2\left(1 + \frac{1}{\theta - 1}\right)\right\}.$$

The best choice for θ is $\theta = 1 + \sqrt{2}$ and yields a competitive ratio of $2 + \sqrt{2}$.

$2 + \sqrt{2} \approx 3.414213562$

Proof: SMARTSTART always computes shortest schedules which start with an empty server either at the origin or at the destination $\omega(r)$ of a request $r \in \sigma$. Since $\tilde{L}(t, \omega(r), \sigma') \leq 2\,\tilde{L}(t, o, \sigma')$ for any subset σ' of the requests and all $r \in \sigma'$, we are essentially in the situation when SMARTSTART uses a 2-approximation algorithm in its "work-or-sleep" routine.

Herlitz distribution center in Falkensee near Berlin.

Suppose that at time t_m the SMARTSTART-server is idle or sleeping. Following the lines of the proof of Theorem 2.18 we can conclude that in either case

$$\text{SMARTSTART}(\sigma) \leq \max\left\{\theta, 2\left(1+\frac{1}{\theta-1}\right)\right\} \text{OPT}(\sigma).$$

It remains the case that at time t_m, SMARTSTART is working on a schedule S. As in the proof of Theorem 2.18 it suffices to consider the situation that the SMARTSTART-server starts its final schedule S' immediately after having completed S. Let t_S be the time when the server started S and denote by $\sigma_{\geq t_S}$ the set of requests presented after the server started S at time t_S. Let $r_f \in \sigma_{\geq t_S}$ be the first request in $\sigma_{\geq t_S}$ served by the optimal offline algorithm. It follows that $\text{OPT}(\sigma) \geq t_S + \tilde{L}(t_m, \alpha_f, \sigma_{\geq t_S})$ which yields

$$\begin{aligned} l(S') &\leq d(o,\alpha_f) + \tilde{L}(t_m, \alpha_f, \sigma_{\geq t_S}) \\ &\leq \text{OPT}(\sigma) - t_S + d(o,\alpha_f) \\ &\leq 2\,\text{OPT}(\sigma) - t_S. \end{aligned} \tag{2.12}$$

Hence

$$\begin{aligned} \text{SMARTSTART}(\sigma) &= t_S + l(S) + l(S') \\ &\leq (\theta-1)t_S + 2\,\text{OPT}(\sigma) && \text{by (2.12)} \\ &\leq (\theta+1)\text{OPT}(\sigma). \end{aligned}$$

This completes the proof. □

2.9 Remarks

Automatic pallet transportation system.

Our investigations of the $C^o_{\max}$-OLDARP were originally motivated by the performance analysis of a large distribution center of Herlitz AG, Berlin [AG+98]. Its automatic pallet transportation system employs several vertical transportation systems (elevators) in order to move pallets between the floors of the building. The pallets that have to be transported during one day of production are not known in advance. If the objective is chosen as minimizing the makespan then this can be modeled by the $C^o_{\max}$-OLDARP where the metric space is induced by a graph which is a path.

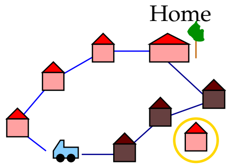
Table 2.1 provides an overview over the results presented in this chapter.

Problem	Competitive Ratios	Lower Bounds
$C^{o}_{\max}$-OLDARP	REPLAN: $5/2$ $(C = 1)$	deterministic algorithms
	(Theorem 2.15)	in general metric spaces: 2
	$7/2$ (general C)	*[AF^{+}01, AF^{+}95, AF^{+}94]*
	(Theorem 2.14)	on the real line: $\frac{9+\sqrt{17}}{8}$
		(Theorem 2.9)
	IGNORE: $5/2$	
	(Theorem 2.16)	
	SMARTSTART: 2	
	(Corollary 2.19)	
	RANDSLEEP: $1 + 1/\ln 2$	randomized algorithms
	(Theorem 2.23)	on the real line: $3/2$
		(Theorem 2.10)
$C_{\max}$-OLDARP	REPLAN: 3 $(C = 1)$	deterministic algorithms
	(Theorem 2.27)	in general metric spaces: 2
	$9/2$ (general C)	on the real line: 2
	(Theorem 2.28)	*(Corollary 2.25)*
	IGNORE: 4	
	(Theorem 2.29)	
	SMARTSTART: $2 + \sqrt{2}$	
	(Theorem 2.30)	
		randomized algorithms: 2
		in general metric spaces: 2
		on the real line: 2
		(Corollary 2.25)

Table 2.1: Results for the minimization of the makespan in online dial-a-ride problems

3

Competing with a Fair Adversary in the Online-TSP



Online traveling salesman problem (OLTSP).

In the online traveling salesman problem (OLTSP) requests for visits to cities (points in a metric space) arrive online while the salesman is traveling. The salesman moves at unit speed and starts and ends his work at a designated origin. The objective is to find a routing for the salesman which finishes as early as possible. The OLTSP is a special case of the $C^{o}_{\max}$-OLDARP introduced in Chapter 2.

In this chapter we mainly consider the OLTSP on the metric space given by $\mathbb{R}_0^+$, the non-negative part of the real line endowed with the Euclidean metric. We show that a very natural strategy is 3/2-competitive against the conventional adversary. This matches the lower bound on competitive ratios achievable for algorithms for this problem.

The emphasis of this chapter is to study the effect of restricting the power of the adversary in the competitive analysis and of restricting the class of online algorithms allowed. We deal with an objection frequently encountered against competitive analysis concerning the unrealistic power of the adversary against which performance is measured. For the OLTSP on the real line all known lower bound constructions work with an adversary which uses its clairvoyant abilities to move to some point x, far away from all previous requests, arriving there just when a request at point x is released. The online algorithm which has no information about the request at x before this request is released, must be close to previous requests or the origin to be competitive at all.

fair adversary

We introduce an adversary, called the *fair adversary*, who is in a natural way restricted in the context of the OLTSP. A fair adversary always keeps its server within the convex hull of the requests released so far. This adversary can be seen as a more reasonable adversary model.

We show that the fair adversary model indeed allows improved competitive ratios. For instance, the above mentioned 3/2-competitive MRIN-strategy against the conventional adversary is 4/3-competitive against the fair adversary. Moreover, there exists an algorithm with competitive ratio $(1+\sqrt{17})/4 \approx 1.28$ against the fair adversary.

We also introduce and analyze a new class of online algorithms which zealous algorithms

we call *zealous algorithms*. Roughly speaking, the server of a zealous algorithm never sits idle while there is work to do. A similar concept was used for scheduling problems in [LL74]. A precise definition of zealousness is presented in Section 3.2, where we also show that in general zealous algorithms are strictly weaker than algorithms that allow waiting time. Our result is the first one that shows that waiting is advantageous in the OLTSP.

This chapter is organized as follows. Section 3.1 contains notation and a restatement of the OLTSP. Zealous Algorithms are introduced in Section 3.2. In Section 3.3 we investigate the competitive ratio of algorithms for the OLTSP in $\mathbb{R}_0^+$ against the conventional adversary. The influence of a fair adversary is studied in Section 3.4.

Related Work

Ausiello et al. [AF+95, AF+01] present a 2-competitive algorithm for OLTSP which works in general metric spaces. Recall that the SMARTSTART-strategy from Section 2.6 is also 2-competitive even for the more general case of the C^o_{max}-OLDARP. 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 [AF+01]. Recently, Lipmann [Lip99] designed an algorithm for the problem on the real line with a competitive ratio that matches the just mentioned lower bound $(9 + \sqrt{17})/8 \approx 1.64$.

3.1 Problem Definition

The online traveling salesman problem (OLTSP) has already been introduced as a special case of the C^o_{max}-OLDARP (see Definition 2.6). We repeat the definition of the OLTSP for convenience:

Definition 3.1 (Online Traveling Salesman Problem (OLTSP))
The ***Online Traveling Salesman Problem*** *(*OLTSP*) is the special case of the* C^o_{max}-OLDARP, *where for each request* r *its source and destination coincide, that is,* $\alpha(r) = \omega(r)$ *for all* r.

C^o_{max}-OLDARP: see Definition 2.4

The objective in the OLTSP *is to minimize the* ***closed makespan****, that is, the time when the server has served all requests and returned to the origin.*

A delicate issue arises in the design of an online algorithm for the OLTSP: Suppose that at some moment in time all known requests have been served. If the algorithm wants to produce a solution with finite cost, then its server must return to the origin after a finite amount of waiting time. But how long should this waiting time be? If the server returns immediately, then a new request might become known and all the traveling to the origin has been in vain. However, a too large waiting time before returning to the origin increases the cost of the solution unnecessarily.

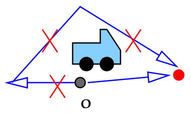
As in Chapter 2 we use the shorter notation $r_j = (t_j, \alpha_j)$ for a request in a sequence $\sigma = r_1, \ldots, r_m$ for the OLTSP. In this chapter we are mainly concerned with the special case that the metric space (M, d) is $\mathbb{R}_0^+$, the non-negative part of the real line endowed with the Euclidean metric, i.e., $X = \mathbb{R}_0^+ = \{ x \in \mathbb{R} : x \geq 0 \}$, and $d(x, y) = |x - y|$; the origin o coincides with the point 0.

Recall that for a real number t we denote by $\sigma_{\leq t}$, $\sigma_{=t}$ and $\sigma_{<t}$ the subsequence of requests in σ released up to time t, exactly at time t and strictly before time t, respectively.

3.2 Zealous Algorithms

In this section we introduce a particular class of algorithms for OLTSP which we call *zealous algorithms*. Intuitively, a zealous algorithm should never sit idle when it could serve yet unserved requests. A zealous server should also move towards work directly without any detours. To translate this intuition into a rigorous definition requires some care.

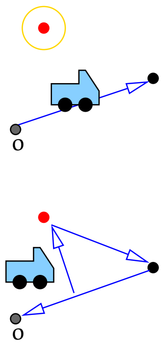
zealous algorithms



A zealous algorithm must move directly towards known requests.

Definition 3.2 (Zealous Algorithm)
An algorithm ALG *for the* OLTSP *is called* ***zealous****, if it satisfies the following conditions:*

1. *If there are unserved requests, then the direction of the server operated by* ALG *changes only if*
   - *a new request becomes known, or*
   - *the server is either at the origin or at a request which has just been served.*
2. *At any time when there are unserved requests, the server operated by* ALG *moves either 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.)*

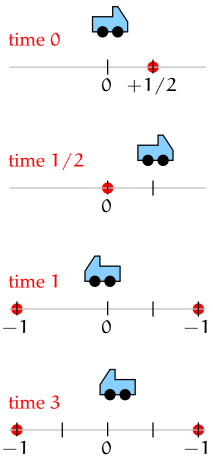


Upon arrival of a new request (red) a zealous algorithm may change its direction.

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

As noted in Theorem 2.8 there is a lower bound of $(9 + \sqrt{17})/8 \approx 1.64$ for the competitive ratio of deterministic online algorithms for the OLTSP in $\mathbb{R}$. We now establish a corresponding lower bound for zealous algorithms. This lower bound shows that the 7/4-competitive algorithm presented in [AF$^+$01], which is, in fact, a zealous algorithm, is best possible within the class of zealous algorithms for the OLTSP on the real line.

Lemma 3.3 *Let* ALG *be a zealous online algorithm for* OLTSP *on the real line* $\mathbb{R}$. *Then the competitive ratio of* ALG *is at least* $7/4$.



Lower bound construction of Lemma 3.3.

Proof: Suppose that ALG is a zealous algorithm for OLTSP on the real line. Consider the following adversarial input sequence. At time 0 and 1/2 two requests $r_1 = (0, 1/2)$ and $r_2 = (1/2, o)$ are released. There are no further requests before time 1. Thus, by the zealousness of the algorithm the server will be at the origin at time 1.

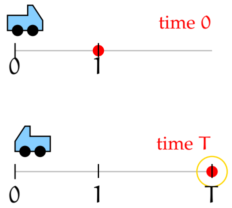
At time 1 two new requests at points 1 and -1, respectively, are released. Since the algorithm is zealous, starting at time 1 it must move its server to one of these requests at maximum, i.e., unit, speed. Without loss of generality assume that this is the request at 1. ALG's server will reach this point 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 units of time to serve -1 and 1 and to return at the origin. Therefore, it is not 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. This yields the desired result. □

3.3 The Online-TSP on the Non-Negative Part of the Real Line

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

Theorem 3.4 *Let* ALG *be any deterministic algorithm for* OLTSP *on* $\mathbb{R}_0^+$. *Then the competitive ratio of* ALG *is at least* $3/2$.



Lower bound construction of Theorem 3.4.

Proof: At time 0 the request $r_1 = (0, 1)$ is released. Let $T \geq 1$ be the time that the server operated by ALG has served the request r_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 $\text{OPT}(r_1) = 2$. Thus, assume that $T < 3$.

In this case the adversary releases a new request $r_2 = (T, T)$. Clearly, $\mathrm{OPT}(r_1, r_2) = 2T$. On the other hand $\mathrm{ALG}(r_1, r_2) \geq 3T$, 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):

MRIN

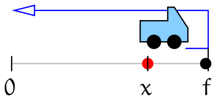
Algorithm MRIN ("Move-Right-If-Necessary")
If a new request is released and the request is to the right of the current position of the server operated by MRIN, then the MRIN-server starts to move right at unit speed. The server continues to move right as long as there are yet unserved requests to the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

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

Theorem 3.5 MRIN *is a zealous* $3/2$*-competitive algorithm for the* OLTSP *on* $\mathbb{R}_0^+$.

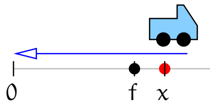
Proof: We establish the theorem by induction on the number of requests in the sequence σ. It clearly holds if σ contains at most one request. The induction hypothesis states that the claim of the theorem holds for any sequence of $m-1$ requests.

Suppose that request $r = (t_m, x)$ is the request from $\sigma_{=t_m}$ which is furthest away from the origin. If $t_m = 0$, then MRIN is obviously $3/2$-competitive, so we will assume that $t_m > 0$. Let f be the position of the request unserved by the MRIN-server at time t_m (excluding r_m), which is furthest away from the origin (if all requests in $\{r_1, \ldots, r_{m-1}\}$ have already been served by MRIN at time t_m then we set $f = 0$).



If $x \leq f$, MRIN's cost does not increase.

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.

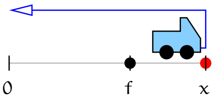


If $x > f$, but the MRIN-server is to the right of x, the cost does not increase either.

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

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

□



If $x > f$, and the MRIN-server is to the left of x, then $\mathrm{MRIN}(\sigma) \leq t_m + 2x$.

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 after all requests have been served.

Lemma 3.6 *There is an optimal offline strategy for the* k-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^+) be the leftmost and rightmost request in σ. We can assume that $x_0^- < 0$ and $x_0^+ > 0$, since otherwise the claim is trivial. For any time t we denote by $\sigma_{>t}$ the subsequence of requests released strictly after time t.

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^+, the second moves to the left until it reaches x_0^-. We now describe the strategy for the first server after it has arrived at x_0^+. 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 $T' := \max\{T + x_0^+ - x_T^+, t_T^+\}$. The time parameter T is updated to $T := T'$, 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. □

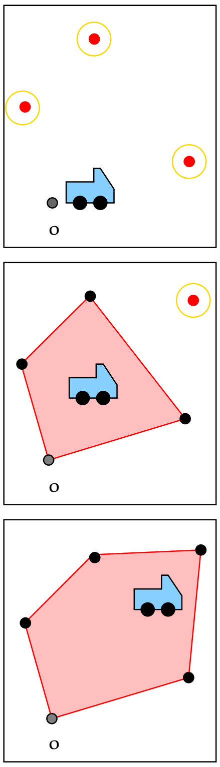
Corollary 3.7 *There is a 3/2-competitive algorithm for the* k-OLTSP *with* $k \geq 2$ *servers on the real line.*

Proof: The online algorithm uses two servers. One server handles all requests on $\mathbb{R}_0^+$, the other one serves the requests on $\mathbb{R}_0^-$. The server in $\mathbb{R}_0^+$ uses the MRIN-strategy, the other server the analogous "mirrored version" for*$\mathbb{R}_0^-$. It follows from Theorem 3.5 and Lemma 3.6 that this strategy is 3/2-competitive. □

3.4 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 baptized *fair adversary*. We show that this model allows improved competitive ratios for the OLTSP on $\mathbb{R}_0^+$. Under this adversary model there also exists a distinction in competitiveness between zealous and non-zealous algorithms.



The fair adversary moves in the convex hull of points released.

Definition 3.8 (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 $o = 0$ among all requests released so far.

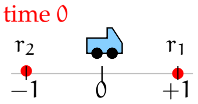
The following lower bound result shows that the OLTSP on the real line against a fair adversary is still a non-trivial problem.

Theorem 3.9 *Let* ALG *be any deterministic algorithm for* OLTSP *on* $\mathbb{R}$. *Then the competitive ratio of* ALG *against a fair adversary is at least* $(5 + \sqrt{57})/8$.

$\frac{5+\sqrt{57}}{8} \approx$ 1.568729304

Proof: Suppose there exists a c-competitive online algorithm ALG, with $c \leq (5 + \sqrt{57})/8$. The adversarial sequence starts with two requests at time 0, namely $r_1 = (0, 1)$ and $r_2 = (0, -1)$. Without loss of generality, we assume that r_1 is the first request which is served by ALG. At time 2 the online server can not have served both requests. We distinguish two main cases divided in some subcases.

Case 1: None of the requests has been served at time 2.

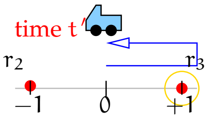


Beginning of the lower bound construction in Theorem 3.9.

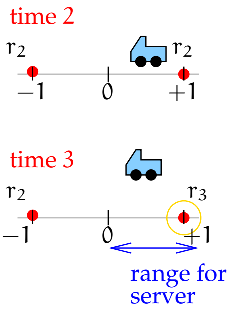
- If at time 3 request r_1 is still unserved, let t' be the first time the server crosses the origin after serving the request r_1. Clearly, $t' \geq 4$. At time t' the online server still has to visit the request in -1.

  If $t' > 4c - 2$ the server can not be c-competitive because the fair adversary can serve the sequence and be back in the origin at time 4.

  Thus, suppose that $4 \leq t' \leq 4c-2$. At time t', a new request $r_3 = (t', 1)$ is released. The online server can not finish the complete sequence before $t' + 4$, whereas the adversary needs time $t' + 1$. Therefore, $c \geq \frac{t'+4}{t'+1}$. For $4 \leq t' \leq 4c - 2$ we have that the expression $\frac{t'+4}{t'+1}$ is decreasing in t'. Thus
$$c \geq \frac{(4c-2)+4}{(4c-2)+1} = \frac{4c+2}{4c-1},$$
  implying $c \geq (5 + \sqrt{57})/8$.



At time $4 \leq t' \leq 4c - 2$ when the server crosses the origin after having served r_1, a new request r_3 is released in 1.



At time 2 no request has been served. If at time 3 the request r_1 has been served, the server can not be left of the origin at this time.

- If at time 3 the request r_1 has already been served, the online server can not 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 issues a new request $r_3 = (3, 1)$. There are two possibilities: either r_2, the request in -1, is served before r_3, or vice versa.

  If the server decides to serve r_2 before r_3, then it can not 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 r_3 first, then, again, let t' be the time that the server crosses the origin after serving r_3. As before, we must have $4 \leq t' \leq 4c - 2$. At time t', the fourth request $r_4 = (t', 1)$ is released. The same arguments as above apply to show that the algorithm's competitive ratio is at least $(5 + \sqrt{57})/8$.

Case 2: r_1 has been served at time 2 by the online server.
At time 2, the third request $r_3 = (2, 1)$ 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. □

For comparison, the best possible algorithm for the OLTSP in $\mathbb{R}$ against an adversary which is not restricted to be fair is $(9 + \sqrt{17})/8$-competitive (see [Lip99]). So far, we have not been able to design an algorithm that has competitive ratio $(5 + \sqrt{57})/8$ against a fair adversary in $\mathbb{R}$. In that sense the picture is complete for the non-negative part of the real line (see Theorems 3.11 and 3.12 for zealous algorithms and Theorems 3.10 and 3.13 for non-zealous algorithms below).

$\frac{1+\sqrt{17}}{4} \approx 1.280776406$

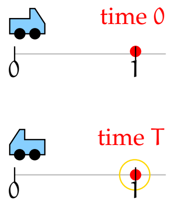
Theorem 3.10 *Let* ALG *be any deterministic algorithm for* OLTSP *in* $\mathbb{R}_0^+$. *Then the competitive ratio of* ALG *against a fair adversary is at least* $(1 + \sqrt{17})/4$.

Proof: Suppose that ALG is c-competitive for some $c \geq 1$. At time 0 the adversary releases the request $r_1 = (0, 1)$. Let T denote the time by which the server operated by ALG has served this request and is back at the origin. For ALG to be c-competitive, we must have $T \leq c \cdot \text{OPT}(r_1) = 2c$. At time T the adversary releases a second request $r_2 = (T, 1)$. 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, yielding a completion time of $T + 1$. Therefore,

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

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



Lower bound construction in Theorem 3.10.

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

Theorem 3.11 *Suppose that* ALG *is 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 $r_1 = (0,1)$, $r_2 = (1,0)$, and $r_3 = (2,1)$.

By its zealousness the online algorithm will start traveling to 1 at time 0, back to the origin o 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. □

Just recently, de Paepe [dP01] was able to establish a lower bound of 8/5 for the competitive ratio of zealous algorithms against a fair adversary in case of the OLTSP in $\mathbb{R}$.

We show now that the algorithm MRIN presented before has a better competitive ratio against the fair adversary than the 3/2 against a conventional adversary. In fact we show the ratio matches the lower bound for zealous algorithms proved in Theorem 3.11.

Theorem 3.12 *Algorithm* MRIN *is* 4/3*-competitive for the* OLTSP *on* $\mathbb{R}_0^+$ *against a fair adversary.*

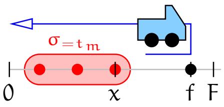
Proof: Again we use induction on the number of requests in the sequence σ to establish the claim. The claim clearly holds if σ contains at most one request. The induction hypothesis states that the claim of the theorem holds for any sequence of $m-1$ requests.

Let $\sigma = r_1, \ldots, r_m$ be any sequence of requests. We consider the time $t := t_m$ when the last set of requests $\sigma_{=t_m}$ is released. If $t = 0$, then the claim obviously holds, so we will assume for the remainder of the proof that $t > 0$. Let $r = (t, x)$ be that request of $\sigma_{=t_m}$ which 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 $r_f = (t_f, f)$ 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 $r_F = (t_F, F)$ be the furthest request in $\sigma_{<t}$. Notice that by definition $f \leq F$.

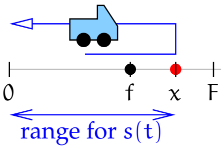
We distinguish three different cases depending on the position x of the request r relative to f and F.



Case 1: If $x \leq f$, MRIN's cost does not increase.

Case 1: $x \leq f$

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.



Case 2: $f \leq x < F$ and $s(t) \leq x$.

Case 2: $f \leq x < F$

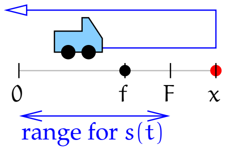
If $s(t) \geq x$, again MRIN's completion time does not increase compared to the situation before the requests in $\sigma_{=t}$ were released, so we may assume that $s(t) \leq x$. The MRIN-server will now travel to point x which needs time $d(s(t), x)$, and then return to the origin. Thus, $\text{MRIN}(\sigma)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

$$\frac{\text{MRIN}(\sigma)}{\text{OPT}(\sigma)} \leq 1 + \frac{d(s(t), x)}{\text{OPT}(\sigma)} \tag{3.1}$$

We now show that $\text{OPT}(\sigma)$ 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

$$\text{OPT}(\sigma) \geq t + x \geq F + d(F, s(t)) + x. \tag{3.2}$$

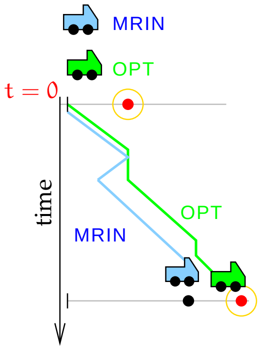
Clearly, each of the terms on the right hand side of inequality (3.2) is at least $d(s(t), x)$.



Case 3: $f \leq F \leq x$.

Case 3: $f \leq F \leq x$

First recall that the MRIN-server always moves to the right if there are yet unserved requests to the right of his present position. Since the last request (t, x) is at least as far away from the origin as F, the optimal offline server will only move left after it has served the furthest request in σ, in this case at x. In fact, the optimal fair offline strategy is as follows: as long as there are unserved requests to the right of the server, move right, otherwise wait at the current position. As soon as the last request (t, x) has been released and the offline server has reached x, it moves to the origin and completes its work (see also the description of the optimal offline strategy which has been described in the proof of Lemma 3.6).



The fair adversary is always to the right of the MRIN-server.

Hence, 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.

Because the offline server does not move left 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

$$d(s(t), s^*(t)) \leq W^*(t). \tag{3.3}$$

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

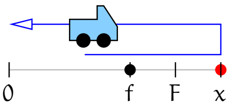
$$t = d(o, s^*(t)) + W^*(t). \tag{3.4}$$

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

$$\begin{align}
\text{OPT}(\sigma) &\geq t + d(s^*(t), F) + d(F, x) + x \tag{3.5}\\
&= d(o, s^*(t)) + W^*(t) + d(s^*(t), F) + d(F, x) + x \qquad \text{by (3.4)}\\
&= W^*(t) + F + d(F, x) + x\\
&= W^*(t) + 2x\\
&\geq W^*(t) + 2d(s(t), s^*(t))\\
&\geq 3d(s(t), s^*(t)) \qquad \text{by (3.3)} \tag{3.6}
\end{align}$$

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

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



Track of the MRIN-server starting at time t.

Hence,

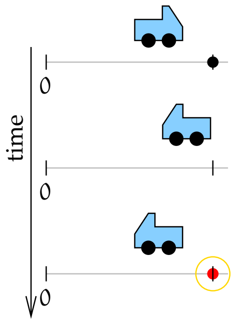
$$\begin{aligned}
\frac{\text{MRIN}(\sigma)}{\text{OPT}(\sigma)} &= \frac{t + d(s^*(t), F) + d(F, x) + x}{\text{OPT}(\sigma)} + \frac{d(s(t), s^*(t))}{\text{OPT}(\sigma)}\\
&\leq 1 + \frac{d(s(t), s^*(t))}{\text{OPT}(\sigma)} \qquad \text{by (3.5)}\\
&\leq \frac{4}{3} \qquad \text{by (3.6).}
\end{aligned}$$

This proves the claim. □

Given the lower bound for general non-zealous algorithms in Theorem 3.10, we conclude that online algorithms which 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 is released. In this case the MRIN-server has to return to a position it just left. Algorithm WS presented below avoids this pitfall successfully.

WS



Bad case for the Algorithm MRIN.

Algorithm WS (“Wait Smartly”)
The WS-server moves right if there are yet unserved requests to the right of its present position. Otherwise, it takes the following actions. Suppose it arrives at its present position $s(t)$ at time t.

1. Compute the optimal offline solution value $\text{OPT}(\sigma_{\leq t})$ for all requests released up to time t.
2. Determine a waiting time $W := \alpha \, \text{OPT}(\sigma_{\leq t}) - s(t) - t$, 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.

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 at time $t + W + s(t) = \alpha \cdot \text{OPT}(\sigma_{\leq t})$ 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.

$\frac{1+\sqrt{17}}{4} \approx$ 1.280776406

Theorem 3.13 *WS is α-competitive for the* OLTSP *on $\mathbb{R}_0^+$ against a fair adversary with $\alpha = (1 + \sqrt{17})/4 < 1.2808$.*

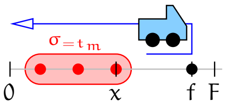
Proof: From the design of Algorithm WS the claim follows if we can prove that at any point where a waiting time is computed this waiting time is non-negative. In that case the server will always be back at the origin before time $\alpha \, \text{OPT}(\sigma)$. The claim is clearly true if the sequence 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 $\sigma = r_1, \ldots, r_m$ be any sequence of requests. As in the proof of Theorem 3.12 we consider the time $t := t_m$ when the last set of requests $\sigma_{=t}$ is released and let $r = (t, x)$ be that request of $\sigma_{=t}$ which 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- and the fair adversary's server at time t, respectively. As before, we let $r_f = (t_f, f)$ be the furthest (i.e. most remote from the origin) yet unserved request by WS at time t of $\sigma_{<t}$. Finally, let $r_F = (t_F, F)$ be the furthest released request in $\sigma_{<t}$.

Again, we distinguish three different cases depending on the position of x relative to f and F. Recall that $f \leq F$.
Case 1: $x \leq f$
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 $s(t)$ can not decrease since the optimal offline completion time can not decrease by an additional request.



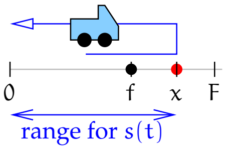
Case 1: If $x \le f$, the cost of WS does not increase.

Case 2: $f \le x < F$

If $s(t) \ge x$, then again by the induction hypothesis and the fact that the route length of WS's server does not increase, the possible waiting time at $s(t)$ is non-negative.

Thus, we can assume that $s(t) < x$. 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

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

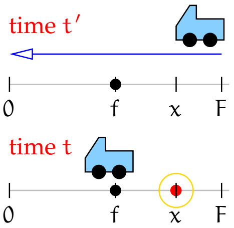


Case 2: $f \le x < F$ and $s(t) < x$.

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

$$W \ge (\alpha - 1)\text{OPT}(\sigma) - d(s(t), x). \tag{3.7}$$

To bound $\text{OPT}(\sigma)$ 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 hold that $t' < t$ since otherwise $s(t)$ could not be smaller than x as assumed. Thus, the subsequence $\sigma_{\le t'}$ of σ does not contain (t, x). By the induction hypothesis, WS is α-competitive for the sequence $\sigma_{\le t'}$. At time t', when WS left F, it would have been able to arrive at the origin at α times the optimal offline cost $\text{OPT}(\sigma_{\le t'})$ on that subsequence:



Case 2: WS was at position F at time $t' < t$.

$$t' + F = \alpha \cdot \text{OPT}(\sigma_{\le t'}). \tag{3.8}$$

Notice that $t \ge t' + d(F, s(t))$. Since $\text{OPT}(\sigma_{\le t'}) \ge 2F$ we obtain from (3.8) that

$$t \ge \alpha 2F - F + d(F, s(t)) = (2\alpha - 1)F + d(s(t), F). \tag{3.9}$$

Since by assumption we have $s(t) < x < F$ we get that $d(s(t), x) \le d(s(t), F)$ and $d(s(t), x) \le F$, which inserted in (3.9) yields

$$t \ge (2\alpha - 1)d(s(t), x) + d(s(t), x) = 2\alpha d(s(t), x). \tag{3.10}$$

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

$$\text{OPT}(\sigma) \ge 2\alpha d(s(t), x) + x \ge (2\alpha + 1)d(s(t), x). \tag{3.11}$$

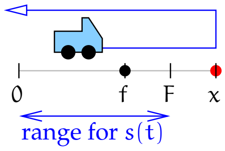
Using inequality (3.11) in (3.7) gives

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

This completes the proof for the second case.

Case 3: $f \leq F \leq x$

Starting at time t, the WS-server moves to the right until it reaches x, and after waiting there an amount W, the server returns to the origin, with



Case 3: $f \leq F \leq x$.

$$W = \alpha \text{OPT}(\sigma) - (t + d(s(t), x) + x). \tag{3.12}$$

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

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

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

$$\text{OPT}(\sigma) \geq t + d(F, x) + x. \tag{3.13}$$

Insertion into (3.12) yields

$$\begin{aligned} W &\geq (\alpha - 1)\text{OPT}(\sigma) + \text{OPT}(\sigma) - (t + d(s(t), x) + x) \\ &\geq (\alpha - 1)\text{OPT}(\sigma) + d(F, x) - d(s(t), x) \\ &= (\alpha - 1)\text{OPT}(\sigma) - d(s(t), F) \qquad (3.14) \end{aligned}$$

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

The rest of the argumentation is similar to 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 maximal). By the induction hypothesis, the WS-server would have returned to the origin at time $\alpha \, \text{OPT}(\sigma_{<t'})$ (if the requests in $\sigma_{=t}$ had not been released). Hence,

$$t' + F = \alpha \, \text{OPT}(\sigma_{<t'}). \tag{3.15}$$

Notice that $t \geq t' + d(s(t), F)$ and $\text{OPT}(\sigma_{<t'}) \geq 2F$ (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 (3.15) 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 (3.13) and the fact that $x \geq F \geq d(s(t), F)$ to achieve

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

Using this inequality in (3.14) 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. □

3.5 Remarks

We introduced an alternative, more fair, performance measure for online algorithms for the online traveling salesman problem. This model is fairer than the conventional one and the first results are encouraging. The fair model allows a strictly lower competitive ratio than the conventional model with an omnipotent adversary on the non-negative part of the real line.

We also considered a restricted class of algorithms for the online traveling salesman problems, called zealous algorithms. We showed that, in general, zealous algorithms have strictly higher competitive ratios than algorithms which sometimes leave the server idle, in order to wait for possible additional information.

In online routing companies, like courier services or transportation companies, waiting instead of immediately starting as soon as requests are presented is common practice. Our results support this strategy.

For the problem on the real line our results together with the recent result of Lipmann [Lip99] show that non-zealous algorithms can do strictly better than zealous algorithms against a conventional adversary. Our results suggest that this is the same against a fair adversary. However, it remains open to find a non-zealous algorithm that beats the best possible zealous ones against a fair adversary in $\mathbb{R}$.

We notice here that for general metric spaces the lower bound of 2 on the competitive ratio of algorithms in [AF$^+$01] is established with a fair adversary as opponent (however, the metric space itself, the boundary of the unit-square, is not convex). Moreover, a zealous algorithm is known which has a competitive ratio that meets the lower bound.

Table 3.1 summarizes the results of this chapter and known results from the literature for the OLTSP in $\mathbb{R}_0^+$ and $\mathbb{R}$.

Problem	Competitive Ratio	Lower Bound
OLTSP in $\mathbb{R}$ with general adversary	zealous algorithms: 7/4 *[AF$^+$01]* general algorithms: $\frac{9+\sqrt{17}}{8}$ *[Lip99]*	zealous algorithms: 7/4 *(Lemma 3.3)* general algorithms: $\frac{9+\sqrt{17}}{8}$ *[AF$^+$01]*
OLTSP in $\mathbb{R}$ with fair adversary	zealous algorithms: 7/4 *[AF$^+$01]* general algorithms: $\frac{9+\sqrt{17}}{8}$ *[Lip99]*	zealous algorithms: 8/5 *[dP01]* general algorithms: $\frac{5+\sqrt{57}}{8}$ *Theorem 3.9*
OLTSP in $\mathbb{R}_0^+$ with general adversary	zealous algorithms: 3/2 *(Theorem 3.5)* general algorithms: 3/2 *(Theorem 3.5)*	zealous algorithms: 3/2 *(Theorem 3.4)* general algorithms: 3/2 *(Theorem 3.4)*
OLTSP in $\mathbb{R}_0^+$ with fair adversary	zealous algorithms: 4/3 *(Theorem 3.12)* general algorithms: $\frac{1+\sqrt{17}}{4}$ *(Theorem 3.13)*	zealous algorithms: 4/3 *(Theorem 3.11)* general algorithms: $\frac{1+\sqrt{17}}{4}$ *(Theorem 3.10)*

Table 3.1: Results for the OLTSP.

4

Minimizing Flow Times and Waiting Times

In this chapter, we consider the task of minimizing the maximal respective average flow time in online dial-a-ride problems. The *flow time* of a request is defined as the difference between its completion time and its release time, that is, the time the request is “unserved in the system”.

flow time

Consider the elevator application from the introduction (Chapter 0). In this context, the *maximal flow time* can be considered as a measure for the maximal “dissatisfaction of the passengers”. On the other hand, the *average flow time* measures intuitively the “throughput of the system”.

maximal flow time

average flow time

Unfortunately, there can be no competitive algorithm, neither deterministic nor randomized, for the minimization of flow times. Hence, from a competitive analysis point of view, *all* algorithms are equally bad and can not be distinguished from each other by their performance. This unsatisfactory situation motivates the search for alternative analysis methods.

Our idea is to introduce the notion of Δ-*reasonable* request sets. A set of requests is Δ-reasonable if, roughly speaking, requests released during any period of time $\delta \geq \Delta$ can be served in time at most δ by an optimal offline algorithm. A sequence of requests σ is *reasonable* if there exists a $\Delta < \infty$ such that σ is Δ-reasonable. This means that for non-reasonable request sequences we find arbitrarily large periods of time where requests are released faster than they can be served, even if the server has the optimal offline schedule. When a system has only to cope with reasonable request sets, we call this situation *reasonable load*.

Δ-reasonable

reasonable load

Under reasonable load it is possible to obtain performance bounds for online algorithms and to distinguish the performance of particular algorithms, which seems to be impossible by means of classical competitive analysis.

This chapter is organized as follows. In Section 4.1 we formally define the objective functions F_{avg} (“average flow time”) and $F_{\max}$ (“maximal flow time”) and introduce the corresponding online dial-a-ride problems which are the subject of our studies. Section 4.2 is essentially a collection of bad news. We show that there can not exist competitive algorithms for the

F_{avg}

$F_{\max}$

problems under investigation. In Section 4.3 we introduce the new concept of reasonable load. We use this concept in Section 4.4 to prove worst-case bounds on the flow times of the IGNORE- and SMARTSTART-Strategies from Chapter 2. Section 4.5 illustrates that similar bounds for the REPLAN-strategy do not exist.

Related Work

In queuing theory, continuously operating systems are usually modeled by a stability assumption: the rate of incoming requests is at most the rate of requests served. To the best of our knowledge, there has been nothing similar so far in the existing theory of discrete online algorithms. Since for most instances we have no exploitable information about the distributions of requests, we want to develop a worst-case model rather than a stochastic model for stability of a continuously operating system. The notion of reasonable load can be viewed as a discrete and deterministic analogue of the stability assumption from queuing theory.

Offline dial-a-ride problems with release times (where all requests are known at the start of the algorithm) are known to be NP-hard to solve for the objective functions of minimizing the average or maximal flow time. They are even very hard to approximate. More specifically, there can be no approximation algorithm with constant performance ratio [KTW96].

4.1 Problem Definition

Let $S = (\tau_1, x_1, y_1, R_1), \dots, (\tau_\ell, x_\ell, y_\ell, R_\ell)$ be a feasible transportation schedule for a given sequence σ of requests. Let $r_j \in \sigma$ be any request and let $S(r_j) = (\tau_l, x_l, y_l, R_l), \dots, (\tau_{l+k}, x_{l+k}, y_{l+k}, R_{l+k})$ be the subsequence of S consisting of those moves which transport r_j (see Definition 2.1 (iv) on page 22).

completion time

The *completion time* of request r_j in S, denoted by $C_j(S)$, is defined to be the time when $S(r_j)$ is completed, that is $C_j(S) = \tau_{l+k} + d(x_{l+k}, y_{l+k})$. We omit the reference to the specific transportation schedule S if it is clear from the context. The *flow time* of request r_j in schedule S, denoted by $F_j(S)$ is defined as the difference between the completion time $C_j(S)$ of r_j and its release time $t(r_j)$, i.e., the time the request is in the system: $F_j(S) := C_j(S) - t(r_j)$. In this chapter we consider the following objective functions for OLDARP:

flow time

Maximal Flow Time $F_{\max}$: By $F_{\max}(S) := \max_{1 \le j \le m} F_j(S)$ we denote the *maximal flow time* of a request in the transportation schedule S.

maximal flow time

average flow time

Average Flow Time $F_{\mathbf{avg}}$: The *average flow time* of a transportation schedule is defined as $F_{\text{avg}}(S) := \frac{1}{m} \sum_{j=1}^{m} F_j(S)$.

Observe that minimizing the average flow time is equivalent to minimizing the sum $\sum_{j=1}^{m} F_j(S)$ of the flow times.

Definition 4.1 (Online Dial-a-Ride Problem F_{max}-OLDARP)
The problem F_{max}-OLDARP consists of finding a transportation schedule which starts at the origin and which minimizes the maximal flow time.

F_{max}-OLDARP

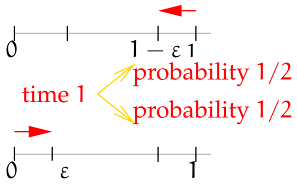
Definition 4.2 (Online Dial-a-Ride Problem F_{avg}-OLDARP)
The problem F_{avg}-OLDARP consists of finding a transportation schedule which starts at the origin and which minimizes the average flow time.

F_{avg}-OLDARP

4.2 Lower Bounds

How well can an online algorithm for the F_{max}-OLDARP or the F_{avg}-OLDARP perform from a competitive analysis point of view? We start with a discouraging result for the F_{max}-OLDARP which shows that even randomization can not help to achieve a constant competitive ratio.

Theorem 4.3 *No randomized algorithm for the F_{max}-OLDARP on $\mathbb{R}_0^+$ can achieve a constant competitive ratio against an oblivious adversary.*



Lower bound construction in Theorem 4.3.

Proof: Let $0 < \varepsilon < 1$ be a small constant. With probability $1/2$ there will be one request at time 1 from 1 to $1-\varepsilon$. With probability $1/2$ there will be one request at time 1 from the origin o to ε. This yields a probability distribution X over the two request sequences $\sigma_1 = (1, 1, 1-\varepsilon)$ and $\sigma_2 = (1, o, \varepsilon)$. Each sequence can be served by an optimal offline algorithm at cost ε, whence $\mathbb{E}_X[\text{OPT}(\sigma_x)] = \varepsilon$.

Consider a generic deterministic online algorithm ALG_y which has its server at position $y \in \mathbb{R}_0^+$ at time 1. Since either $y < 1/2$ or $y \geq 1/2$ it follows that on average ALG_y incurs a cost of at least $1/2$. Hence the expected cost of the algorithm satisfies $\mathbb{E}_X[\text{ALG}_y(\sigma_x)] \geq 1/2$.

This yields

$$\frac{\mathbb{E}_X[\text{ALG}_y(\sigma_x)]}{\mathbb{E}_X[\text{OPT}(\sigma_x)]} \geq \frac{1}{2\varepsilon}.$$

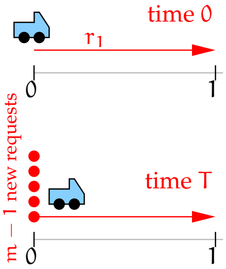
By choosing $\varepsilon > 0$ small enough the above ratio becomes unbounded. The theorem now follows by applying Yao's Principle. □

Corollary 4.4 *No deterministic algorithm for the F_{max}-OLDARP on the non-negative part of the real line $\mathbb{R}_0^+$ can achieve a constant competitive ratio.* □

The situation is even worse in case of the F_{avg}-OLDARP. The competitive ratio of any algorithm is doomed to grow with the number of requests in the request sequence as the following lower bound results show.

Theorem 4.5 *Any deterministic algorithm for the* F_{avg}-OLDARP *on the non-negative part of the real line* $\mathbb{R}_0^+$ *has competitive ratio greater or equal to* m, *where* $m := |\sigma|$ *denotes the number of requests in the input sequence.*

Any randomized algorithm for the F_{avg}-OLDARP *on* $\mathbb{R}_0^+$ *has competitive ratio in* $\Omega(\sqrt{m})$ *against an oblivious adversary.*



Lower bound construction in Theorem 4.5.

Proof: We only prove the first part of the theorem. The second statement is an immediate consequence of the corresponding lower bound result for online machine scheduling in [Ves94] (cf. Section 2.3 for the relation between dial-a-ride problems and machine scheduling).

Suppose that ALG is an arbitrary deterministic online algorithm for the F_{avg}-OLDARP. We release a request $r_1 = (0, o, 1)$ at time 0. Let $T \geq 0$ be the time when ALG starts to serve request r_1. At this time the adversary releases $m - 1$ new requests $r_j = (T, o, o)$ $(2 \leq j \leq m)$.

After having served r_1 (incurring a flow time of $T + 1$), ALG must serve the requests $r_2, \ldots, r_m$. If ALG starts serving the small requests immediately after having completed r_1 and as quickly as possible, then the flow time of each of the $m - 1$ requests $r_2, \ldots, r_m$ is $T + 2$. Hence the average flow time of ALG satisfies

$$\text{ALG}(\sigma) \geq \frac{1}{m}(T + 1 + (m - 1)(T + 2)) = T + 2 - \frac{1}{m}.$$

The adversary can process the sequence σ by first serving the requests $r_2, \ldots, r_m$ and then taking care of r_1. Each of the requests $r_2, \ldots, r_m$ incurs a zero flow time. Hence, the average flow time of the optimal offline algorithm is bounded from above by $\frac{1}{m}(T + 1)$. Thus,

$$\frac{\text{ALG}(\sigma)}{\text{OPT}(\sigma)} \geq m + \frac{m - 1}{T + 1} > m,$$

and, consequently, the algorithm ALG can not achieve a competitive ratio smaller than m. □

4.3 Reasonable Load

Considering the negative results from the previous section we can not hope for performance guarantees that may be relevant in practice. In particular, the algorithms REPLAN, IGNORE and SMARTSTART (see Chapter 2) can not be distinguished by classical competitive analysis.

In a continuously operating system we wish to guarantee that work can be accomplished at least as fast as it is presented. In the following we propose a mathematical set-up which models this idea in a worst-case fashion.

Since we are always working on finite subsets of the whole request set, the request set itself may be infinite, modeling a continuously operating system.

In the sequel we slightly abuse notation and treat a (possibly infinite) sequence $\sigma = r_1, r_2, \ldots$ of requests also as a *set* $\sigma = \{r_1, r_2, \ldots\}$ of requests. This allows us to use the usual notation for mappings.

We start by relating the release spans of finite subsets of a request set to the time we need to fulfill the requests.

Definition 4.6 (Offline Version of Requests and Sequences)
*The **offline version** of request $r = (t, a, b)$ is the request*

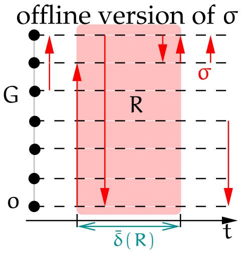
offline version

$$r^{\mathit{offline}} := (0, a, b).$$

*Let $\sigma = r_1, r_2, \ldots$ be any (possibly infinite) sequence of requests. The **offline version of** σ is the request sequence*

offline version of σ

$$\sigma^{\mathit{offline}} := \{\, r^{\mathit{offline}} : r \in \sigma \,\}.$$



Release span $\bar{\delta}(R)$ of a finite request set R. The metric space is induced by a path (drawn vertically). The time axis is shown horizontally.

An important characteristic of a request set with respect to system load considerations is the time period in which it is released. Recall that $t(r)$ denotes the release time of request r.

Definition 4.7 (Release Span)
*Let σ be any sequence of requests and $R \subseteq \sigma$ be a finite subset. The **release span** $\bar{\delta}(R)$ of R is defined as*

$$\bar{\delta}(\sigma) := \max_{r \in R} t(r) - \min_{r \in R} t(r).$$

In the case of dial-a-ride problems, provably good polynomial time offline-algorithms exist for the (closed) makespan (see e.g. Chapters 6 and 7) and the weighted sum of completion times [GK96, BC+94, AK99]. How can we make use of these algorithms in order to get performance guarantees for minimizing the maximal (average) waiting (flow) times? We suggest a way of characterizing request sets which we want to consider "reasonable".

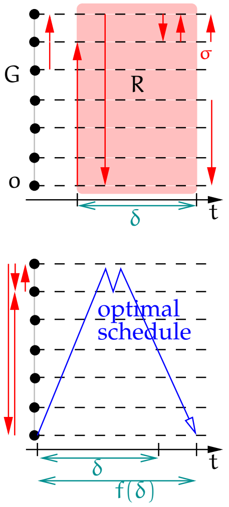
Define $\mathsf{OPT}_{C^o_{\max}}(R)$ to be the *length of the optimal closed makespan* for a sequence R of requests.

$\mathsf{OPT}_{C^o_{\max}}(R)$

Definition 4.8 (Load Bound)
*Let σ be a request sequence. A weakly monotone function $f\colon \mathbb{R}_0^+ \to \mathbb{R}_0^+$ is a **load bound** on σ if for any $\delta \in \mathbb{R}$ and any finite $R \subseteq \sigma$ with $\bar{\delta}(R) \leq \delta$ the closed makespan $\mathsf{OPT}_{C^o_{\max}}(R^{\mathit{offline}})$ of the optimal schedule for the offline version R^{offline} of R is at most $f(\delta)$, that is, if*

load bound

$$\mathsf{OPT}_{C^o_{\max}}(R^{\mathit{offline}}) \leq f(\delta).$$

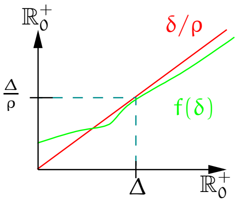


Load bound f for a request set. If $\bar{\delta}(R) \leq \delta$, then $\text{OPT}_{C^o_{\max}}(R^{\textit{offline}}) \leq f(\delta)$. The set $R^{\textit{offline}}$ is derived from R by setting all release times of the requests in R to zero. The track of the optimal offline solution $\text{OPT}_{C^o_{\max}}(R^{\textit{offline}})$ is shown in blue.

Remark 4.9 If the whole request sequence σ is finite then there is always the trivial load bound given by the total completion time $\text{OPT}_{C^o_{\max}}(\sigma)$ of σ. For every load bound f we may set $f(0)$ to be the maximal completion time needed for a single request, and nothing better can be achieved.

Intuitively, a "stable situation" would be obtained by a load bound equal to the identity id on $\mathbb{R}_0^+$, $\text{id}(x) = x$. (By "stable" we mean that the number of unserved requests in the system does not become arbitrarily large.) In that case we would never get more work to do than we can accomplish. If a request sequence σ has a load bound equal to a function id/ρ, where id is the identity and where $\rho \geq 1$, then ρ measures the "tolerance" of the request set in the following sense: assume that we have an offline ρ-approximation algorithm at our disposal that is by (at most) a factor ρ worse than the optimal offline algorithm. Then we can still ensure stability by using the IGNORE-strategy: compute a ρ-approximate schedule (with respect to the closed makespan) for the set R of all released but unserved requests. The load bound and the performance guarantee ensure that the schedule takes no longer than $\rho \cdot \bar{\delta}(R)/\rho = \bar{\delta}(R)$ to complete. Thus, the set of requests that are released in the meantime has a release span no larger than $\bar{\delta}(R)$, and we can proceed by computing a ρ-approximate schedule for that set.

However, we cannot expect that the identity (or any linear function) is a load bound because of the following observation: a request set consisting of one single request has a release span of 0 whereas in general, it takes nonzero time to serve this request. In the following definition we introduce a parameter describing how far a request set is from being load-bounded by the identity.



Graph of a (Δ, ρ)-reasonable load bound f.

Definition 4.10 (Reasonable Request Sequence)
*Let $\Delta, \rho \in \mathbb{R}_0^+$ be fixed. A load bound f is **(Δ,ρ)-reasonable**, if*

$$f(\delta) \leq \delta/\rho \quad \textit{for all } \delta \geq \Delta.$$

*A request sequence σ is **(Δ,ρ)-reasonable** if it has a (Δ,ρ)-reasonable load bound. For $\rho = 1$, we say that the request sequence is **Δ-reasonable**.*

In other words, a load bound is *(Δ,ρ)-reasonable*, if it is bounded from above by $\text{id}(\delta)/\rho$ for all $\delta \geq \Delta$ and by the constant function with value Δ/ρ otherwise.

Remark 4.11 If Δ is sufficiently small, such that all request sets consisting of two or more requests have a release span larger than Δ, then the first-come-first-serve strategy is good enough to ensure that there are never more than two unserved requests in the system. Hence, the request set does not require scheduling the requests in order to provide for a stable system.

In a sense, Δ is a measure for the combinatorial difficulty of the request sequence σ. Thus, it is natural to ask for performance guarantees for algorithms in terms of this parameter. This is done in the next section.

4.4 Bounds for the Flow Times

We are now in a position to prove bounds for the maximal flow time for the algorithms IGNORE and SMARTSTART. As in previous chapters we use the term "(approximately) shortest schedule" for a transportation schedule which (approximately) minimizes the closed makespan.

4.4.1 Analysis of IGNORE

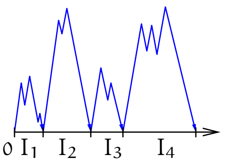
In our initial version of the IGNORE-strategy we required each schedule computed to be a shortest one. It will be useful to relax these requirements and extend the IGNORE-strategy in the way we did for the SMARTSTART-algorithm. We allow each schedule computed to be approximately optimal with respect to the closed makespan. For convenience we restate the algorithm IGNORE from Section 2.5 with this minor modification:

> **Algorithm IGNORE**
> The server remains idle until the point in time t when the first requests become known. It serves the requests released at time t immediately, following an approximately shortest schedule S which starts and ends at the origin. All requests that arrive during the time when the algorithm follows S are temporarily ignored. As soon as S has been completed and the server is back in the origin, the algorithm computes an approximately shortest schedule for all unserved requests and follows this schedule. Again, all new requests that arrive during the time that the server is following the schedule are temporarily ignored. A schedule for the ignored requests is computed as soon as the server has completed its current schedule.

IGNORE

Let us consider the intervals in which IGNORE organizes its work in more detail. The algorithm IGNORE induces a dissection of the time axis $\mathbb{R}_0^+$ in the following way: We can assume, without loss of generality, that the first set of requests is released at time 0. Let $\delta_0 = 0$, i.e., the point in time where the first set of requests is released (these are processed by IGNORE in its first transportation schedule). For $i = 1, 2, \dots$ let δ_i be the duration of the time period the server is working on the requests that have been ignored during the last δ_{i-1} time units. Then the time axis is split into the intervals

$$[\delta_0 = 0, \delta_0], (\delta_0, \delta_1], (\delta_1, \delta_1 + \delta_2], (\delta_1 + \delta_2, \delta_1 + \delta_2 + \delta_3], \dots$$



Track of the IGNORE-server on a graph which is a path over time.

Let us denote these intervals by $I_0, I_1, I_2, I_3, \ldots$. Moreover, let R_i be the set of those requests that are released in I_i. Clearly, the complete set of requests R is the disjoint union of all the R_i.

At the end of each interval I_i we solve an offline problem: all requests to be scheduled are already available. The work on the computed schedule starts immediately (at the end of interval I_i) and is completed δ_{i+1} time units later (at the end of interval I_{i+1}). On the other hand, the time we need to serve the schedule is not more than ρ times the optimal completion time of R_i^{offline}. In other words:

Lemma 4.12 *For all $i \geq 0$ the estimate*

$$\delta_{i+1} \leq \rho \cdot \mathrm{OPT}_{C^o_{\max}}(R_i^{\mathit{offline}}).$$

holds. □

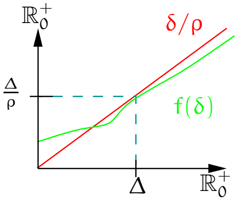
Let us now prove the first positive result for the online minimization of the maximal flow time.

Theorem 4.13 *Let $\Delta > 0$ and $\rho \geq 1$. For all instances of the $F_{\max}$-OLDARP with (Δ, ρ)-reasonable request sets,* IGNORE *employing a ρ-approximation algorithm for solving offline instances of the $C^o_{\max}$-OLDARP yields a maximal flow time of no more than 2Δ.*

IGNORE$[\sigma]$: Solution produced by IGNORE on input σ, see Definition 1.2

Proof: Let r be an arbitrary request in R_i for some $i \geq 0$, i.e., r is released in I_i. By construction, the transportation schedule containing r is completed at the end of interval I_{i+1}, i.e., at most $\delta_i + \delta_{i+1}$ time units later than r was released. This results in the following upper bound:

$$F_{\max}(\text{IGNORE}[\sigma]) \leq \sup_{i \geq 0} \left(\delta_i + \delta_{i+1}\right).$$



Graph of a (Δ, ρ)-reasonable load bound f.

If we can show that $\delta_i \leq \Delta$ for all $i > 0$ then we are done. To this end, let f be a (Δ, ρ)-reasonable load bound for R. Since $\bar{\delta}(R_i) \leq \delta_i$ we have $\mathrm{OPT}_{C^o_{\max}}(R_i^{\mathit{offline}}) \leq f(\delta_i)$ by definition of a load bound. By Lemma 4.12, we get for all $i > 0$

$$\delta_{i+1} \leq \rho \mathrm{OPT}_{C^o_{\max}}(R_i^{\mathit{offline}}) \leq \rho f(\delta_i) \leq \max\{\delta_i, \Delta\}.$$

Using $\delta_0 = 0$ the claim now follows by induction on i. □

From the fact that the average flow time can never be larger than the maximal flow time, we obtain as a trivial consequence the following corollary:

Corollary 4.14 *For (Δ, ρ)-reasonable request sets,* IGNORE *employing a ρ-approximation algorithm for solving offline instances of the $C^o_{\max}$-OLDARP yields an average flow time of no more than 2Δ.* □

4.4.2 Analysis of SMARTSTART

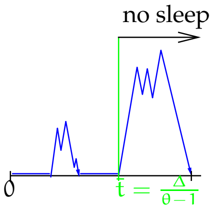
We continue to analyze the SMARTSTART-strategy from Section 2.6 with respect to its behavior under reasonable load. For an easier reference, we repeat the definition of the SMARTSTART-algorithm:

SMARTSTART

> **Algorithm SMARTSTART**
> If the algorithm is idle at time T and new requests arrive, SMARTSTART calls "work-or-sleep". If the result is (S, work), the algorithm enters the working state where it follows schedule S. Otherwise, the algorithm enters the sleeping state with wakeup time t', where $t' \geq T$ is the earliest time such that $t' + l(S) \leq \theta t'$ and $l(S)$ denotes the length of the just computed schedule S, i.e., $t' = \min\{\, t \geq T : t + l(S) \leq \theta t \,\}$.
>
> In the sleeping state the algorithm does nothing until its wakeup time t'. At this time the algorithm reconsults the "work-or-sleep" subroutine. If the result is (S, work), then the algorithm enters the working state and follows S. Otherwise, the algorithm continues to sleep with new wakeup time $\min\{\, t \geq t' : t + l(S) \leq \theta t \,\}$.
>
> In the working state, i.e, while the server is following a schedule, all new requests are (temporarily) ignored. As soon as the current schedule is completed the server either enters the idle state (if there are no unserved requests) or it reconsults the "work-or-sleep" subroutine which determines the next state (sleeping or working).

Recall that the "work-or-sleep" routine of SMARTSTART computes an approximately shortest schedule S for all unserved requests which starts and ends in the origin. Let $\theta > 1$. If the schedule S can be completed no later than time θt, i.e., if $t + l(S) \leq \theta t$, where t is the current time, the subroutine returns (S, work), otherwise it returns (S, sleep).

The analysis of SMARTSTART essentially parallels that of IGNORE, so we only highlight the differences. The crucial observation needed is formulated in the following lemma:



Track of the SMARTSTART-server on a path graph over time. The server does never sleep after time $\bar{t} = \frac{\Delta}{\theta-1}$.

Lemma 4.15 *For (Δ, ρ)-reasonable request sequences the server of* SMARTSTART *never sleeps after time $\bar{t} := \frac{\Delta}{\theta-1}$.*

Proof: Consider a call to the "work-or-sleep" routine at an arbitrary time $t \geq \bar{t}$. Let R be the set of requests not served by SMARTSTART at time t and let S be a ρ-approximate shortest schedule for R. By the (Δ, ρ)-reasonability of the input sequence, the length of schedule S for R can be bounded from above by

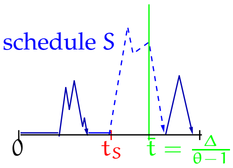
$$l(S) \leq \rho \cdot \max\left\{\frac{\Delta}{\rho}, \frac{\bar{\delta}(R)}{\rho}\right\} = \max\{\Delta, \bar{\delta}(R)\}.$$

Trivially, we have $\bar{\delta}(R) \leq t$, since all requests in R have been released at time t. Hence, it follows that

$$\begin{aligned} t + l(S) &\leq t + \max\{\Delta, \bar{\delta}(R)\} \\ &\leq t + \max\{\Delta, t\} && \text{(since } \bar{\delta}(R) \leq t) \\ &= t + \max\{(\theta - 1)\bar{t}, t\} && \text{(since } \bar{t} = \Delta/(\theta - 1)) \\ &\leq \theta t && \text{(since } t \geq \bar{t}). \end{aligned}$$

Consequently, the "work-or-sleep" routine does not return the invitation to sleep.

The same arguments as given above show that, if SMARTSTART goes to sleep before some time $t < \bar{t}$, the wakeup time is no later than time $\bar{t}$. Hence, the lemma follows. □



After completing schedule S which was started at time t_S, SMARTSTART behaves exactly like IGNORE.

Let S be the last schedule started by SMARTSTART no later than time $\bar{t}$ and denote by $t_S \leq \bar{t}$ its start time. From Lemma 4.15 we conclude that from time $\bar{t}$ on, SMARTSTART behaves like IGNORE, provided the input sequence is (Δ, ρ)-reasonable. Using the arguments given in the proof of Theorem 4.13 we can conclude that the flow time of any request released after time t_S is bounded from above by 2Δ.

It remains to treat the requests released before time $\bar{t}$. Using again the arguments of Theorem 4.13 we derive that all requests released after time t_S have flow time at most 2Δ and we finally need to consider those requests released until time t_S. Each of these requests is either served by S or by an even earlier schedule. Since by definition of SMARTSTART, the transportation schedule S is completed no later than time $\theta t_S < \theta \bar{t} = \frac{\theta}{\theta-1}\Delta$, we obtain the following result:

Theorem 4.16 *Let $\Delta > 0$ and $\rho \geq 1$. For all instances of the F_{max}-OLDARP with (Δ, ρ)-reasonable request sets, Algorithm SMARTSTART employing a ρ-approximation algorithm in its "work-or-sleep" routine yields a maximal flow time of no more than $\max\left\{\frac{\theta}{\theta-1}\Delta, 2\Delta\right\}$. In particular, if $\theta \geq 2$, then the maximal flow time provided by SMARTSTART is bounded from above by 2Δ.* □

As in the case of IGNORE a we can derive a trivial upper bound of 2Δ for the average flow time of SMARTSTART under reasonable load.

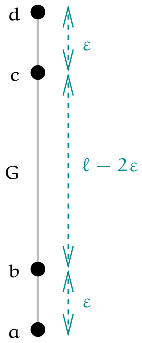
4.5 A Disastrous Example for REPLAN

We first recall the strategy of algorithm REPLAN from Section 2.5.

> **Algorithm REPLAN**
> As soon as a new request r_j arrives the server stops and replans: it computes a schedule with minimum length which starts at the current position of the server, takes care of all yet unserved requests (including those that are currently carried by the server), and ends at the origin. Then it continues using the new schedule.

REPLAN

In the sequel, we provide an instance of the F_{max}-OLDARP and a Δ-reasonable request sequence σ such that the maximal and the average flow time provided by REPLAN is unbounded, thereby proving that a positive result as in Theorems 4.13 and 4.16 is not possible for REPLAN.



Path-graph G used in the proof of Theorem 4.17.

Theorem 4.17 *There is an instance of the* F_{max}-OLDARP *under reasonable load such that the maximal and the average flow time of* REPLAN *are unbounded.*

Proof: Figure 4.1 contains a sketch of an instance for the F_{max}-OLDARP. The metric space is a path on four nodes a, b, c, d with origin a; the length of the path is ℓ, the distances are $d(a, b) = d(c, d) = \varepsilon$, and hence $d(b, c) = \ell - 2\varepsilon$. At time 0 a request from a to d is issued; starting at time $\frac{3}{2}\ell - \varepsilon$, the remaining requests periodically come in pairs from b to a and from c to d, respectively. The time distance between them is $\ell - 2\varepsilon$.

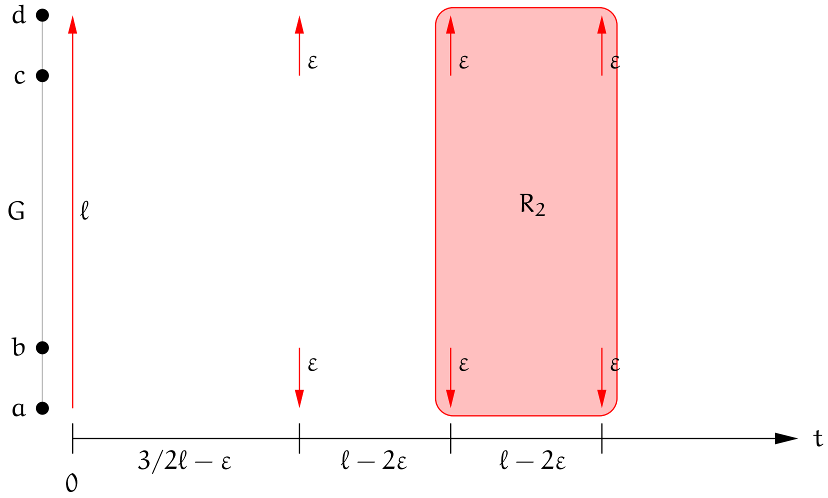
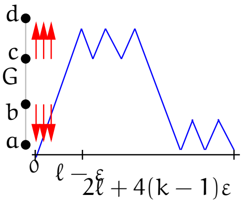


Figure 4.1 Sketch of a $\frac{8}{3}\ell$-reasonable instance of the F_{max}-OLDARP. The horizontal axis holds the time, the vertical axis depicts the metric space in which the server moves. A request is denoted by an arrow horizontally positioned at its release time.

We show that for $\ell = 18\varepsilon$, the request sequence σ indicated in the picture is $\frac{8}{3}\ell$-reasonable. Indeed, it is easy to see that the first request from a to d does not influence reasonability. Consider an arbitrary set R_k of k adjacent pairs of requests from b to a and from c to d. Then the release span $\bar{\delta}(R_k)$ of R_k is

$$\bar{\delta}(R_k) = (k-1)(\ell - 2\varepsilon).$$



Track of the optimal closed makespan for $R_k{}^{\mathit{offline}}$ (here $k = 3$).

The offline version $R_k{}^{\mathit{offline}}$ of R_k can be served as follows: first, move the server to c, the starting point of the upper requests: this contributes cost $\ell - \varepsilon$. Next, serve all the upper requests and go back to c: this contributes cost $2k\varepsilon$. Then, go down to b, the starting point of the lower requests: this contributes another $\ell - 2\varepsilon$ to the cost. Now, serve the first lower requests: the additional cost for this is ε. Finally, serve the remaining lower requests at an additional cost of $(k-1) \cdot 2\varepsilon$. In total, we have the following:

$$\mathsf{OPT}_{C^o_{\max}}(R_k{}^{\mathit{offline}}) = 2\ell + 4(k-1)\varepsilon.$$

In order to find the smallest parameter Δ for which the request set R_k is Δ-reasonable, we solve

$$f(\bar{\delta}(R_k)) = \mathsf{OPT}_{C^o_{\max}}(R_k{}^{\mathit{offline}})$$

for the integer $k-1$ and get

$$k - 1 = \left\lceil \frac{2\ell}{\ell - 6\varepsilon} \right\rceil = 3 \quad \text{for } \ell = 18\varepsilon.$$

Hence, we can set Δ to

$$\Delta := \mathsf{OPT}_{C^o_{\max}}(R_4{}^{\mathit{offline}}) = \tfrac{8}{3}\ell.$$

Now we define the function $f\colon \mathbb{R}_0^+ \to \mathbb{R}_0^+$ by

$$f(\delta) := \begin{cases} \Delta & \text{for } \delta < \Delta, \\ \delta & \text{otherwise.} \end{cases}$$

By construction, f is a load bound for R_4. Because the time gap after which a new pair of requests occurs is certainly larger than the time it increases the offline makespan, f is also a load bound for the whole sequence σ. Thus, the request sequence σ is Δ-reasonable, as desired.

How does REPLAN perform on this instance? Figure 4.2 shows the track of the server following the schedules produced by REPLAN on the request sequence σ. Since a new pair of requests is issued exactly when the server is still closer to the requests at the top all the requests at the bottom will be postponed. Thus, the server always returns to the top when a new pair of requests arrives.

The maximal flow time of REPLAN on this instance is realized by the flow time of the request $(\frac{3}{2}\ell - \varepsilon, b, a)$, which is unbounded.

Moreover, since all requests from b to a are postponed after serving all the requests from c to d we get that REPLAN produces an unbounded average flow time as well. □

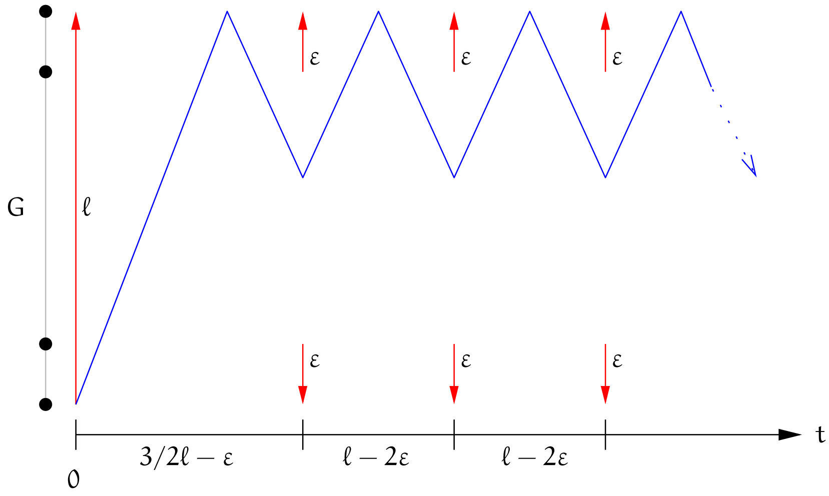


Figure 4.2
The track of the REPLAN-server on the instance described in Theorem 4.17.

In Figure 4.3 we show the track of the server of the IGNORE-algorithm. After an initial inefficient phase the server ends up in a stable operating mode. This example also shows that the analysis of IGNORE in Section 4.4 is sharp.

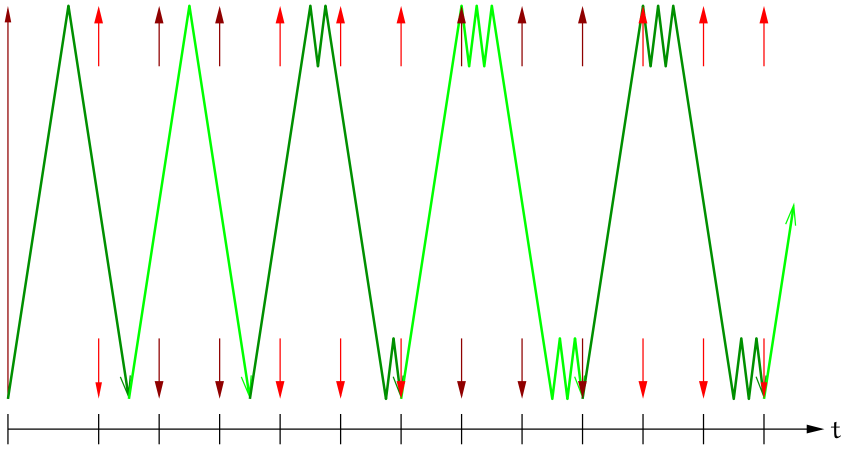


Figure 4.3
The track of the IGNORE-server.

4.6 Replanning with a Different Objective

It is quite natural to ask whether modified replan-strategies that repeatedly minimize the maximal respective average flow times on the known request sets (instead of the closed makespan) would give provable bounds on the maximal and average flow times under reasonable load. Call these modified replan-strategies REPLAN$_{F_{\max}}$ and REPLAN$_{F_{\text{avg}}}$.

REPLAN$_{F_{\max}}$

REPLAN$_{F_{\text{avg}}}$

We mentioned already that the offline problem of minimizing the average flow time is computationally very hard in general (see e.g. [KTW96]). In the offline problem that $\mathsf{REPLAN}_{F_{avg}}$ has to solve, however, all requests have release times in the past. It is easy to see that the problem is equivalent to the minimization of the average completion time counted from the point in time where the planning takes place. Moreover, since the average flow time is larger by the "average age" of the requests, the performance guarantees of approximation algorithms minimizing the average completion time carry over. The following result shows that even under reasonable load we cannot expect a worst case stable behavior of $\mathsf{REPLAN}_{F_{avg}}$.

Theorem 4.18 *There is an instance of the* F_{max}-OLDARP *under reasonable load such that the maximal and average flow times of* $\mathsf{REPLAN}_{F_{avg}}$ *are unbounded.*

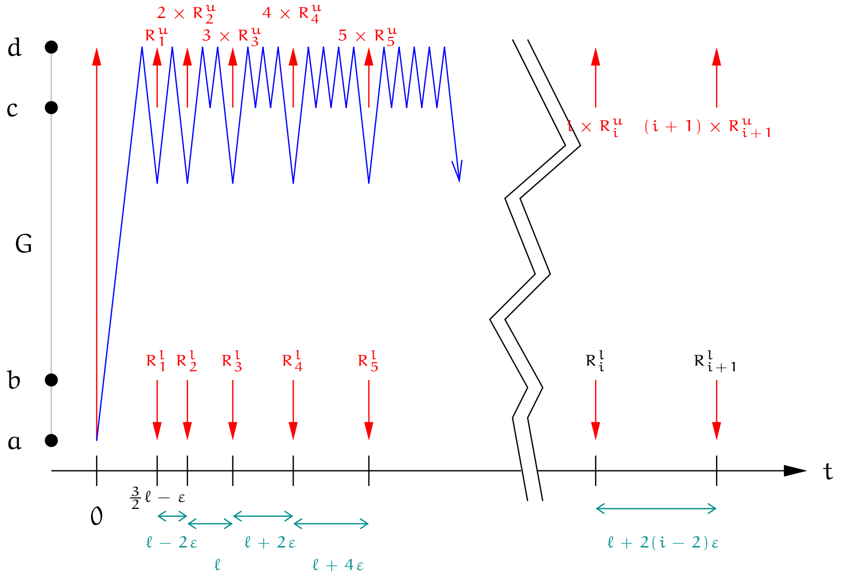


Figure 4.4
The track of the $\mathsf{REPLAN}_{F_{avg}}$-server on the example from Theorem 4.18.

Proof: We construct a set of requests in the same metric space as in the previous Section 4.5 as follows:

- At time 0 we issue again one request from a to d.
- At time $T_0 := \frac{3}{2}\ell - \varepsilon$ we issue a pair consisting of one "upper" request R^u_1 from c to d and one "lower" request R^l_1 from b to a.
- At time $T_{i+1} := T_i + \ell + 2(i-2)\varepsilon$ we issue
  - a set of i "upper" requests R^u_{i+1} from c to d and
  - one "lower" request R^l_{i+1} from b to a.

Figure 4.4 sketches the construction. For $\ell = 18\varepsilon$ this request set is again $\frac{8}{3}\ell$-reasonable since we have increased the time intervals between the release times of the requests by the additional amount that is needed to serve the additional copies of upper requests.

At time T_i, for all $i > 0$, $\text{REPLAN}_{F_{avg}}$ has still to serve as many upper requests as there are lower requests. Thus, at T_i the schedule with minimum average flow time for the currently available requests serves the upper requests first. Hence, the requests at the bottom have to wait for an arbitrarily long period of time.

In order to prove the assertion concerning the average flow time we consider the result that $\text{REPLAN}_{F_{avg}}$ produces on the input set R_N which contains all requests up to time T_N.

In order to estimate the average flow time of a request in the solution produced by $\text{REPLAN}_{F_{avg}}$ we first consider the sum of all flow times $\sum F_j(\text{REPLAN}_{F_{avg}}[R_N])$. Clearly, the sum of the waiting times of the lower requests is a lower bound on the sum of the flow times:

$$\begin{aligned}\sum F_j(\text{REPLAN}_{F_{avg}}[R_N]) &\geq \sum_{k=1}^{N}\sum_{i=k}^{N}(\ell + 2(i-2)\varepsilon)\\ &\geq \sum_{k=1}^{N}\sum_{i=k}^{N}(i-2)\varepsilon.\end{aligned}$$

The number of requests $|R_N|$ in R_N is

$$|R_N| = 1 + \sum_{k=1}^{N}(k+1),$$

so that

$$F_{avg}(\text{REPLAN}_{F_{avg}}[R_N]) = \frac{\sum F_j(\text{REPLAN}_{F_{avg}}[R_N])}{|R_N|} \overset{N\to\infty}{\longrightarrow} \infty,$$

which completes the proof. □

A strategy that minimizes just the maximal flow time does not make a lot of sense since sometimes this only determines which request is to be served first; the order in which all the other requests are scheduled is unspecified. Thus, the most sensible strategy in this respect seems to be the following: consider an offline instance of the dial-a-ride problem. A vector consisting of all flow times of requests in a feasible solution ordered decreasingly is called a *flow vector*. All flow vectors are ordered lexicographically. The online strategy $\text{REPLAN}_{F_{max}}$ for the online dial-a-ride problem does the fol- $\text{REPLAN}_{F_{max}}$

lowing: whenever a new request becomes available REPLAN$_{F_{\max}}$ computes a new schedule of all yet unserved requests minimizing the flow vector.

It is an open problem what the performance of this strategy is under Δ-reasonable load. In practice, however, it is probably too difficult to solve the offline problem with this objective function.

4.7 Remarks

To overcome the weakness of competitive analysis, we have introduced the mathematical notion "Δ-reasonable´´ describing the combinatorial difficulty of a possibly infinite request set for $F_{\max}$-OLDARP. For reasonable request sets we were able to prove bounds on the maximal respective average flow time of the algorithms IGNORE and SMARTSTART for $F_{\max}$-OLDARP; in contrast to this, there are instances of $F_{\max}$-OLDARP where the algorithm REPLAN yields an unbounded maximal and average flow time.

One key property of our results is that they can be applied in continuously working systems. Computer simulations have meanwhile supported the theoretical results in the sense that algorithm IGNORE does not delay individual requests for an arbitrarily long period of time, whereas REPLAN has a tendency to do so [GH$^+$99, GH$^+$00].

While the notion of Δ-reasonability is applicable to minimizing maximal flow time, it would be of interest to investigate an average analogue in order to prove non-trivial bounds for the average flow time.

Table 4.1 summarizes the discouraging results on lower bound for the competitive ratios of algorithms for the $F_{\max}$-OLDARP and the F_{avg}-OLDARP. Table 4.2 gives an overview over the performance bounds of algorithms under reasonable load obtained in this chapter.

Problem	Competitive Ratios	Lower Bounds
$F_{\max}$-OLDARP	—	deterministic algorithms: no constant ratio possible *(Corollary 4.4)*
	—	randomized algorithms: no constant ratio possible *(Theorem 4.3)*
F_{avg}-OLDARP	—	deterministic algorithms: $m = \lvert\sigma\rvert$ *(Theorem 4.5)*
	—	randomized algorithms: $\Omega(m)$ *[Ves94]*

Table 4.1: Results about competitive algorithms for the $F_{\max}$-OLDARP and the F_{avg}-OLDARP.

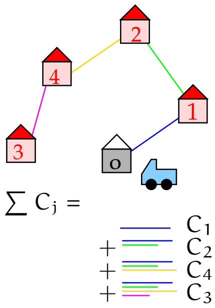
Algorithm	Bound on $F_{\max}$ under (Δ, ρ)-reasonable load
IGNORE	2Δ *(Theorem 4.13)*
SMARTSTART	$\max\left\{\frac{\theta}{\theta-1}\Delta, 2\Delta\right\}$ *(Theorem 4.16)*
REPLAN	no upper bound possible *(Theorem 4.17)*
REPLAN$_{F_{\text{avg}}}$	no upper bound possible *(Theorem 4.18)*

Table 4.2: Performance bounds under reasonable load.

5

Minimizing the Sum of Completion Times

In the *traveling repairman problem* (TRP) [AC+86] a server must visit a set of m points $x_1, \dots, x_m$ in a metric space. Given a tour through the m points, the completion time C_j of point x_j is defined as the time traveled by the server on the tour until it reaches x_j ($j = 1, \dots, m$). Each point x_j has a weight w_j, and the objective of the TRP is to find a 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*.



Traveling repairman problem (TRP)

In this chapter we consider an 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 r_j at point x_j with release time t_j is the first time at which the repairman visits x_j after the release time t_j. The OLTRP is a special case of the dial-a-ride problem $\sum w_j C_j$-OLDARP (similar as the OLTSP is a special case of the dial-a-ride problem $C^{o}_{\max}$-OLDARP).

We present competitive algorithms and randomized lower bounds for the OLTRP and the more general $\sum w_j C_j$-OLDARP. Our algorithms improve the competitive ratios given in previous works. For the case of the $\sum w_j C_j$-OLDARP our algorithms are the first competitive algorithms. The randomized lower bounds are the first ones for the OLTRP and the $\sum w_j C_j$-OLDARP.

This chapter is organized as follows. In Section 5.1 we define the OLTRP and the $\sum w_j C_j$-OLDARP formally. In Section 5.2 we prove lower bounds on the competitive ratio of randomized algorithms against an oblivious adversary. Section 5.3 contains the deterministic algorithm INTERVAL and the proof of its competitive ratio. In Section 5.4 we present the randomized algorithm RANDINTERVAL which achieves a better competitive ratio.

Related Work

In [FS01] Feuerstein and Stougie presented a 9-competitive algorithm for the OLTRP on the real line and a 15-competitive algorithm for the $\sum w_j C_j$-

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 OLTRP and $\sum w_j C_j$-OLDARP, respectively, are proved.

The offline traveling repairman problem TRP is known to be NP-hard even on trees [AC+86]. Approximation algorithms for the TRP have been studied in [GK96, BC+94, AK99].

5.1 Problem Definition

In addition to the framework for online dial-a-ride problems described in Section 2.1, in the $\sum w_j C_j$-OLDARP each request $r_j \in \sigma = r_1, \ldots, r_m$ additionally specifies a *weight* $w_j := w(r_j)$, that is, each request r_j is a quadruple $(t_j, \alpha_j, \omega_j, w_j) \in \mathbb{R}_0^+ \times X \times X \times \mathbb{R}_0^+$.

completion time

Recall that the *completion time* of request r_j in a transportation schedule S, denoted by $C_j(S)$, is defined to be the time that $S(r_j)$, the subschedule in which r_j is served, is completed (see Section 4.1). In this chapter we are concerned with the following objective function:

latency

Weighted Sum of Completion Times $\sum w_j C_j$: This function is defined as $\sum_{j=1}^{m} w_j C_j(S)$ and is also referred to as the *latency* of a transportation schedule.

We are now ready to state the problems $\sum w_j C_j$-OLDARP and OLTRP formally:

Definition 5.1 (Online Dial-a-Ride Problem $\sum w_j C_j$-OLDARP)
The $\sum w_j C_j$-OLDARP consists of finding a transportation schedule S, which starts at the origin and which minimizes the weighted sum of completion times $\sum w_j C_j(S)$.

Definition 5.2 (Online Traveling Repairman Problem OLTRP)
*The **online traveling repairman problem** (OLTRP) is the special case of the $\sum w_j C_j$-OLDARP, when for each request r its source and destination coincide, that is, $\alpha(r) = \omega(r)$ for all r.*

5.2 Lower Bounds

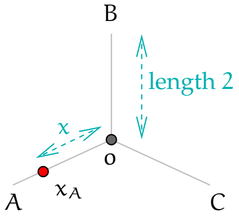
Feuerstein and Stougie proved the following lower bounds for deterministic algorithms:

Theorem 5.3 ([FS01]) *For the $\sum w_j C_j$-OLDARP on the real line, any deterministic online algorithm has a competitive ratio greater or equal to 3. Any deterministic online algorithm for the* OLTRP *on the real line has competitive ratio at least* $1+\sqrt{2}$. □

$1+\sqrt{2} \approx$ 2.414213562

In this section we establish lower bounds for the competitive ratio of any randomized algorithm against an oblivious adversary for the problem $\sum w_j C_j$-OLDARP. The basic method for deriving such a lower bound is once more Yao's Principle (see Theorem 1.12).

Recall that the heart of Yao's Principle is to specify a probability distribution $\bar{X}$ over input sequences such that on average, every deterministic algorithm performs badly, that is, such that $\mathbb{E}_{\bar{X}}[\mathsf{ALG}_y(\sigma_x)]$ is high compared to the expected optimal cost $\mathbb{E}_{\bar{X}}[\mathsf{OPT}(\sigma_x)]$. In this chapter we use a method proposed in [Sei00] to compute a suitable distribution once our ground set of request sequences has been fixed.



The star with three rays used in the lower bound construction for $\sum w_j C_j$-OLDARP.

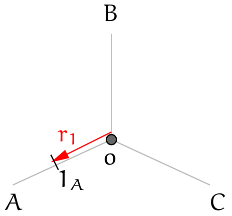
5.2.1 A General Lower Bound for $\sum w_j C_j$-OLDARP

We provide a general lower bound where the metric space is a star, which consists of three rays of length 2 each. The center of the star is the origin, denoted by o. The server capacity equals one.

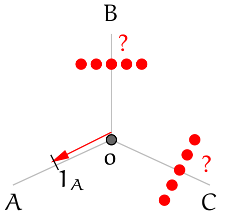
Let the rays of the star be named A, B and C. Let x_A be the point on A with distance x to o, $0 < x \leq 2$. Points on B and C will be denoted in the same manner. Let $k \in \mathbb{N}$, $k \geq 2$ be arbitrary.

At time 0 there is one request from o to 1_A with weight 1, denoted by $r_1 = (0, o, 1_A, 1)$. With probability θ there are no further 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 $\sigma_{0,R} = r_1$ for $R \in \{B,C\}$, and

$$\sigma_{x,R} = r_1, \underbrace{(2x, 2x_R, 2x_R, 1), \ldots, (2x, 2x_R, 2x_R, 1)}_{k \text{ times}} \qquad \text{for } 0 < x \leq 1 \text{ and } R \in \{B,C\}.$$

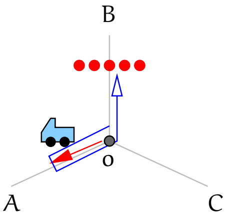


Situation at time 0.

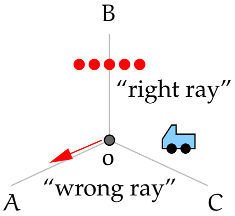


With probability $1-\theta$ a set of k new requests arrives either on ray B or on ray C.

We first calculate the expected cost $\mathbb{E}_X[\mathsf{OPT}(\sigma_{x,R})]$ of an optimal offline algorithm with respect to the distribution X on Σ. With probability θ there is only request r_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



Case 1: The set of k new requests arrives after the online algorithm has started to serve r_1.



Case 2: The set of k new requests arrives before the online algorithm has started to serve r_1.

the requests arrive. First serving request r_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 r_1 results in a total cost of $2kx+4x+1$. Hence, for $k \geq 2$, we have

$$\mathbb{E}_X[\mathsf{OPT}(\sigma_{x,R})] = \theta + \int_0^1 (2kx + 4x + 1)p(x)\,dx. \tag{5.1}$$

The strategy of a generic deterministic online algorithm ALG_y can be cast into the following framework: ALG_y starts serving request r_1 at time $2y$ where $y \geq 0$, unless further requests are released before time $2y$. If the sequence ends after r_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 r_1, the algorithm must first finish r_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 r_1 and can serve the k requests before r_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 $0 \leq l \leq y$ since otherwise, the server cannot start r_1 at time $2y$. We may assume that ALG_y is either on ray B or C, since moving onto ray A without serving r_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 $y \leq 1$

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

This results in

$$\begin{aligned}\mathbb{E}_X[\mathsf{ALG}_y(\sigma_{x,R})] \geq &\theta(2y+1) + \int_y^1 (2y + 1 + k(2y + 2 + 2x))\,p(x)\,dx \\ &+ \int_0^y (4kx + 6x + 1)p(x)\,dx. \\ =:&\ F(y).\end{aligned} \tag{5.2}$$

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

$$\mathbb{E}_X[\mathsf{ALG}_y(\sigma_x)] \geq \theta(2y+1) + \int_0^1 (2y + 1 + k(2y + 2 + 2x))\,p(x)\,dx \geq F(1).$$

Hence in what follows it suffices to consider the case $y \leq 1$. 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. [Sei00]): 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

$$p(x) = \beta(k-2x)^{\frac{1}{2}(k-1)}. \tag{5.3}$$

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

$$\beta = \frac{1-\theta}{\int_0^1 (k-2x)^{\frac{1}{2}(k-1)}\,dx} = \frac{(1+k)\ (\theta-1)}{(k-2)^{\frac{1+k}{2}} - k^{\frac{1+k}{2}}}. \tag{5.4}$$

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

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

and substituting p and β from (5.3) and (5.4), 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 (5.2) and (5.1). This results in

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

and

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

Hence we conclude that

$$\frac{\mathbb{E}_X\left[\mathsf{ALG}_y(\sigma_{x,R})\right]}{\mathbb{E}_X\left[\mathsf{OPT}(\sigma_{x,R})\right]} \geq \frac{-5(k-2)^{1+\frac{k}{2}}\,k+\sqrt{k-2}\,k^{\frac{1+k}{2}}\,(3+4k)}{\sqrt{k-2}\,k^{\frac{1+k}{2}}(3+2k)-(k-2)^{\frac{2+k}{2}}\,(4+3k)}. \tag{5.5}$$

For $k \to \infty$, the right hand side of (5.5) converges to $\frac{4e-5}{2e-3}$. Hence by Yao's Principle we obtain the following result:

$\frac{5-4e}{3-2e} \approx$ 2.410414067

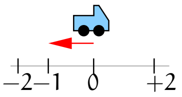
Theorem 5.4 *Any randomized algorithm for the $\sum w_j C_j$-OLDARP has a competitive ratio greater or equal to $\frac{4e-5}{2e-3}$ against an oblivious adversary.* □

5.2.2 A Lower Bound on the Real Line

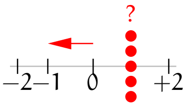
The lower bound construction on the star uses the fact that the online server does not know on which ray to move if it wants to anticipate on the arrival of the k requests at time $2x$. If the metric space is the real line, there are only two rays, and this argument is no longer valid. The server can 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.

$\frac{\ln 16+1}{\ln 16-1} \approx$ 2.128293312

Theorem 5.5 *Any randomized algorithm for $\sum w_j C_j$-OLDARP on the real line has competitive ratio greater or equal to $\frac{\ln 16+1}{\ln 16-1}$ against an oblivious adversary.*



Situation at time 0 in the proof of Theorem 5.5.



With probability $1-\theta$ a set of k new requests arrives.

Proof: We use the following request set: At time 0 there is one request $r_1 = (0,0,-1,1)$. With probability θ there are no further requests. With probability $1-\theta$ there are k requests 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

$$\sigma_x = \begin{cases} r_1 & \text{for } x = 0 \\ r_1, \underbrace{(2x,2x,2x,1),\ldots,(2x,2x,2x,1)}_{k \text{ times}} & \text{for } 0 < x \leq 1. \end{cases}$$

By a similar argumentation as in the previous section we obtain

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

and

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

By choosing

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

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

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

we obtain

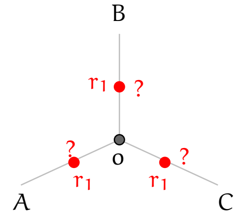
$$\frac{\mathbb{E}_X\left[\mathsf{ALG}_y(\sigma_x)\right]}{\mathbb{E}_X\left[\mathsf{OPT}(\sigma_x)\right]} \geq 1 + \frac{2}{(1+k)\;\left(-2+k^{\frac{1+k}{1-k}}\,(-1+2k)^{\frac{1+k}{-1+k}}\right)}. \tag{5.6}$$

For $k \longrightarrow \infty$, the right hand side of (5.6) converges to $\frac{\ln 16+1}{\ln 16-1}$. The theorem now follows from Yao's Principle. □

5.2.3 Lower Bounds for the OLTRP

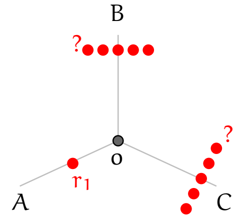
For the OLTRP we provide a general lower bound, again using a star with three rays of length 2 as a metric space.

Theorem 5.6 *Any randomized algorithm for* OLTRP *has competitive ratio greater or equal to 7/3 against an oblivious adversary.* □



Situation at time 0. The ray for request r_1 is chosen uniformly at random among the three rays.

Proof: Let $k \in \mathbb{N}$ be arbitrary. With probability $\theta = \frac{k+1}{k+2}$, the input sequence consists of only one request r_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 r_1.



With probability $1-\theta = \frac{1}{k+2}$ a set of k new requests arrives on one of the two remaining rays.

The cost for the adversary is given by

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

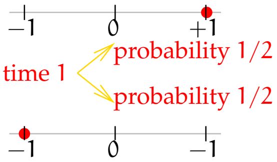
It is easy to show that no online algorithm can do better than one whose server is in the origin at time 1 and, at time 2, at distance $0 \leq y \leq 1$ from the origin on the ray where r_1 is located. Using this fact, we get

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

This leads to

$$\frac{\mathbb{E}_X[\mathrm{ALG}_y(\sigma_x)]}{\mathbb{E}_X[\mathrm{OPT}(\sigma_x)]} \geq \frac{7k+10}{3k+6}. \tag{5.7}$$

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



Construction for the proof of Theorem 5.7.

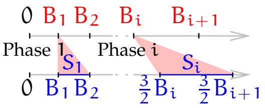
On the real line, a lower bound is obtained by the following very simple randomized request sequence. With probability 1/2 we give a request at time 1 in -1, and with probability 1/2 we give a request at time 1 in $+1$. This leads to the following theorem.

Theorem 5.7 *Any randomized algorithm for* OLTRP *on the real line has competitive ratio greater or equal to 2 against an oblivious adversary.* □

5.3 A Deterministic Algorithm

Our deterministic strategy is an adaption of the GREEDY-INTERVAL algorithm presented in [HSW96, HS⁺97] for online scheduling. The proof of performance borrows concepts of the proofs in [HSW96, HS⁺97].

INTERVAL



Computation and execution of schedules in INTERVAL.

Algorithm INTERVAL
Phase 0: In this phase the algorithm is initialized.

Set L to be the earliest time when a request could 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. These requests are served at no cost. Until time L remain in o. For $i = 0, 1, 2, \ldots$, define $B_i := 2^{i-1}L$.

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 endpoint x_{i-1} of schedule S_{i-1} (we set $x_0 := 0$).

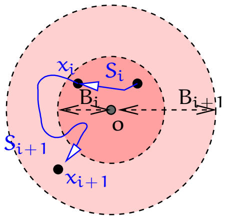
(ii) Schedule S_i ends at a point x_i with an empty server such that $d(o, x_i) \leq B_i$.

(iii) The length of schedule S_i, denoted by $l(S_i)$, satisfies

$$l(S_i) \le \begin{cases} B_1 & \text{if } i = 1 \\ \frac{3}{2}B_i & \text{if } i \ge 2. \end{cases}$$

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

If $i = 1$, then follow S_1 starting at time L until time B_2. If $i \ge 2$, follow S_i starting at time $\frac{3}{2}B_i$ until $\frac{3}{2}B_{i+1}$.

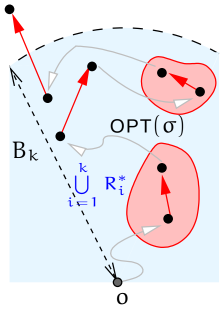


Schedules in INTERVAL.

To justify the correctness of the algorithm notice that by definition transportation schedule S_i computed at time B_i can actually be finished before time $\frac{3}{2}B_{i+1}$, the time when transportation schedule S_{i+1}, computed at time B_{i+1}, needs to be started.

Lemma 5.8 *Let R_i be the set of requests served by schedule S_i computed at time B_i, $i = 1, 2, \dots$, 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*

$$\sum_{i=0}^{k} w(R_i) \ge \sum_{i=0}^{k} w(R_i^*) \quad \textit{for } k = 1, 2, \dots .$$



The first B_k units of time of the optimal schedule yield a schedule S serving all requests in $\bigcup_{i=1}^{k} R_i^*$.

Proof: We first argue that for any $k \ge 1$ we can obtain from the optimal offline solution S^* a schedule S 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 with the modification that, if a request is picked up in S^* before time B_k but not delivered before time B_k, omit this action. Observe that this implies that the server is empty at the end of this schedule. We thereby obtain a schedule S 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 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 $\sum_{i=0}^{k} w(R_i^*)$ have been scheduled by INTERVAL. If $k = 1$, the transportation schedule S obtained as outlined above satisfies already all conditions (i)–(iii) required by INTERVAL. If $k \ge 2$, then condition (i) might be violated, since S starts in the origin. However, we can obtain a new schedule S' from S starting at the endpoint x_{k-1} of the schedule from the previous phase by starting in x_{k-1}, moving the empty server from x_{k-1} to the origin and then following S. Since $d(x_{k-1}, o) \le B_{k-1} = B_k/2$, the new schedule S' has length at most $B_k/2 + l(S) \leq B_k/2 + B_k = 3/2 \cdot B_k$ which means that it satisfies all the properties (i)–(iii) required by INTERVAL.

Recall that schedule S and thus also S' 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 in previous phases. As omitting requests can never increase the length of a transportation schedule, in phase k, INTERVAL 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 is at least $\bigcup_{i=1}^{k} R_k^*$ as claimed. □

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

Corollary 5.9 *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 *is at most* p*. Schedule* S_p *computed at time* B_p *by* INTERVAL *is completed no later than time* $\frac{3}{2}B_{p+1}$*.*

Proof: By Lemma 5.8 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, schedule S_p computed in phase p completes by time $\frac{3}{2}B_{p+1}$, the claim follows. □

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

Lemma 5.10 *Let* $a_i, b_i \in \mathbb{R}_0^+$ *for* $i = 1, \ldots, p$*, for which*

(i) $\sum_{i=1}^{p} a_i = \sum_{i=1}^{p} b_i$*;*

(ii) $\sum_{i=1}^{p'} a_i \geq \sum_{i=1}^{p'} b_i$ *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 5.11 INTERVAL *is 6-competitive for the* $\sum w_j C_j$*-*OLDARP*.*

Proof: Let $\sigma = r_1, \ldots, r_m$ be any sequence of requests. By definition of INTERVAL, each request served in schedule S_i completes no later than time $\frac{3}{2}B_{i+1}$. Summing over all phases $1, \ldots, p$ yields

$$\text{INTERVAL}(\sigma) \leq \frac{3}{2}\sum_{i=1}^{p} B_{i+1} w(R_i) = 6\sum_{i=1}^{p} B_{i-1} w(R_i). \tag{5.8}$$

From Lemma 5.8 we know that

$$\sum_{i=0}^{k} w(R_i) \geq \sum_{i=0}^{k} w(R_i^*) \quad \text{for } k = 1, 2, \ldots,$$

and from Corollary 5.9 we know that

$$\sum_{i=0}^{p} w(R_i) = \sum_{i=0}^{p} w(R_i^*).$$

Therefore, application of Lemma 5.10 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

Statement of Lemma 5.10: $\sum_{i=1}^{p} \tau_i a_i \leq \sum_{i=1}^{p} \tau_i b_i$

$$6 \sum_{i=1}^{p} B_{i-1} w(R_i) \leq 6 \sum_{i=1}^{p} B_{i-1} w(R_i^*) \tag{5.9}$$

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

$$6 \sum_{i=1}^{p} B_{i-1} w(R_i^*) = 6 \sum_{j=1}^{m} B_{\phi_j} w_j \leq 6 \sum_{j=1}^{m} w_j C_j^*. \tag{5.10}$$

(5.8), (5.9), and (5.10) together complete the proof. □

Corollary 5.12 INTERVAL *is 6-competitive for the* OLTRP. □

5.4 An Improved Randomized Algorithm

In this section we use techniques from Section 2.7 to devise a randomized algorithm RANDINTERVAL. At the beginning, RANDINTERVAL 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 presented in the previous section. For $i \geq 0$ define $B_i' := 2^{i-1-\delta} L$, where again L is the earliest time that a request could be completed by OPT. As stated before in the case of INTERVAL we can assume that $L > 0$.

The difference to INTERVAL is that all phases are defined using $B_i' := 2^{i-1-\delta} L$ instead of B_i, $i \geq 1$. Phase 1 is still started at time L.

Lemma 5.13 *Let* R_i *be the set of requests scheduled in phase* $i \geq 1$ *of Algorithm* RANDINTERVAL *and denote by* R_i^* *the set of requests that are completed by* OPT *in the time interval* $(B'_{i-1}, B'_i]$. *Then*

$$\sum_{i=1}^{k} w(R_i) \geq \sum_{i=0}^{k} w(R_i^*) \quad \textit{for } k = 1, 2, \dots .$$

Proof: We only have to ensure that schedule S_1 is finished before time $\frac{3}{2}B'_2$. The rest of the proof is the same as that for Lemma 5.8. The proof for the first phase follows from the fact that $\frac{3}{2}B'_2 - L = (3\cdot 2^{-\delta} - 1)L > 2^{-\delta}L = B'_1$. □

We can now use the proof of Theorem 5.11 with Lemma 5.8 replaced by Lemma 5.13. This enables us to conclude that for a sequence $\sigma = r_1, \dots, r_m$ of requests the expected objective function value of RANDINTERVAL satisfies

$$\mathbb{E}\left[\text{RANDINTERVAL}(\sigma)\right] \leq \mathbb{E}\left[6\sum_{i=1}^{m} B'_{\phi_j} w_j\right] = 6\sum_{i=1}^{m} w_j \mathbb{E}\left[B'_{\phi_j}\right], \tag{5.11}$$

where $(B'_{\phi_j}, B'_{\phi_{j+1}}]$ is the interval containing the completion time C_j^* of request r_j in the optimal solution OPT(σ). From Lemma 2.22 we can conclude that $\mathbb{E}\left[B'_{\phi_j}\right] = \frac{1}{2\ln 2} C_j^*$. Using this result in inequality (5.11) yields the following theorem:

$\frac{3}{\ln 2} \approx 4.328085123$

Theorem 5.14 RANDINTERVAL *is* c*-competitive for the* $\sum w_j C_j$-OLDARP *with* $c = \frac{3}{\ln 2}$ *against an oblivious adversary.* □

Corollary 5.15 RANDINTERVAL *is* c*-competitive for the* OLTRP *with* $c = \frac{3}{\ln 2}$ *against an oblivious adversary.* □

5.5 Remarks

In addition to the first randomized lower bounds, we we have presented the first competitive algorithms for the $\sum w_j C_j$-OLDARP. The application of these algorithms for the special case of the OLTRP yielded an improvement over the previously best known competitive ratios.

Table 5.1 gives an overview over the results for the OLTRP and the $\sum w_j C_j$-OLDARP.

Problem	Competitive Ratios	Lower Bounds
OLTRP	INTERVAL: 6 *(Corollary 5.12)*	deterministic algorithms in general metric spaces: $1+\sqrt{2}$ *[FS01]* on the real line: $1+\sqrt{2}$ *[FS01]*
	RANDINTERVAL: $3/\ln 2$ *(Corollary 5.15)*	randomized algorithms in general metric spaces: $7/3$ *(Theorem 5.6)* on the real line: 2 *(Theorem 5.7)*
$\sum w_j C_j$-OLDARP	INTERVAL: 6 *(Theorem 5.11)*	deterministic algorithms in general metric spaces: 3 *[FS01]* on the real line: 3 *[FS01]*
	RANDINTERVAL: $3/\ln 2$ *(Theorem 5.14)*	randomized algorithms in general metric spaces: $\frac{4e-5}{2e-3}$ *(Theorem 5.4)* on the real line: $\frac{\ln 16+1}{\ln 16-1}$ *(Theorem 5.5)*

Table 5.1: Results for the OLTRP and $\sum w_j C_j$-OLDARP.

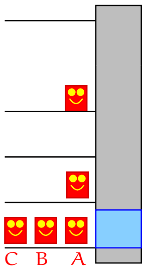
6

Offline Dial-a-Ride Problems with Precedence Constraints

In an offline dial-a-ride problem all requests are known in advance to an algorithm. We consider the case where additionally all release times of the requests are equal to zero. In the offline dial-a-ride problem CDARP (short for “capacitated dial-a-ride problem”) we are given (finitely many) transportation requests between points in a metric space and the goal is to find a shortest transportation schedule which serves all the requests. The problem CDARP arises, for instance, as a subproblem in the algorithms REPLAN, IGNORE and SMARTSTART presented in Chapter 2. The problem DARP is the restriction of the CDARP to a (single) server with unit capacity.

It turns out that it is helpful to view the sources and destinations of the requests as the vertices of a weighted graph G which induces the underlying metric space. This enables us to exploit the structural properties of the graph G in the design of our algorithms.

A natural extension of the DARP is the addition of precedence constraints between the requests that start at the same vertex. This variant which we call SOURCE-DARP is motivated by applications in which first-in-first-out (FIFO) waiting lines are present at the sources of the transportation requests. In this case, requests can be served only according to their order in the line. Examples with first-in-first-out lines are cargo elevator systems where at each floor conveyor belts deliver the goods to be transported. Elevators also motivate the restriction of DARP to paths, i.e., to the case where the underlying graph forms a path.



FIFO waiting line: Before serving C, requests A and B must be served.

In this chapter we consider the SOURCE-DARP for a (single) server with unit capacity. Section 6.1 contains a formal statement of the problem. We also show that the SOURCE-DARP can be equivalently formulated as a graph augmentation problem. This key observation will be used to design our algorithms. In Section 6.2 we prove structural facts about Eulerian cycles in a graph that respect a given “source-order” on the arcs. (A formal definition of source-orders appears in Section 6.1.2.) The class of source-orders contains as a special case partial orders which can be used to model first-in-first-out waiting lines. Section 6.3 contains a polynomial algorithm for SOURCE-DARP when restricted to paths. In Section 6.4 we present an approximation algorithm for general graphs with performance of $(\rho_{\text{TSP}}+3)/2$, where ρ_{TSP} is the performance of the best approximation for the traveling salesman problem (TSP) with triangle inequality. An improved algorithm for trees with performance 3/2 is given in Section 6.5. Section 6.6 is dedicated to hardness results. Section 6.7 briefly discusses extensions of our results.

Related Work

stacker-crane-problem

The problem DARP is also known as the *stacker-crane-problem*. In [FG93] it is shown that the stacker-crane-problem is NP-hard even on trees. In [FHK78] the authors present a 9/5-approximation algorithm for the stacker-crane-problem on general graphs. An improved algorithm for trees with performance 5/4 is given in [FG93]. On paths the stacker-crane-problem can be solved in polynomial time [AK88].

The problem CDARP with server capacity $C > 1$ has been addressed in [Gua98, CR98]. For capacity $C > 1$ the problem becomes NP-hard even on paths. In [CR98] an approximation algorithm with performance $\mathcal{O}(\sqrt{C}\log n \log\log n)$ was given, where C denotes the capacity of the server and n denotes the number of vertices in the graph. We will investigate CDARP with server capacity $C > 1$ on paths in Chapter 7.

Chinese postman problem

Precedence constraints have been studied in the case of Chinese postman tours in [DST87] (recall that the Chinese postman problem consists of finding a shortest walk in a graph that traverses *all* edges and arcs). The authors show that for general precedence relations it is NP-hard to determine a Chinese postman tour of minimum length. Under strong restrictions on the precedence relation the problem can be solved in time $\mathcal{O}(n^5)$, where n denotes the number of vertices in the input graph.

6.1 Problem Definition

We assume that the reader is familiar with the basics from graph theory [Har72, AMO93]. A brief compilation of the necessary definitions with respect to graphs can be found in Appendix A.7.

A *multiset* Y over a ground set U, denoted by $Y \sqsubset U$, is a mapping $Y\colon U \to \mathbb{N}$, where for $u \in U$ the number $Y(u)$ denotes the multiplicity of u in Y. For a weight function $d\colon U \to \mathbb{R}$ the weight of a multiset $Y \sqsubset U$ is defined by $d(Y) := \sum_{u\in U} d(u)Y(u)$. By $Y + Z$, $Y - Z$ and $Y \cap Z$ we denote the union, difference and intersection of the multisets Y and Z (see Appendix A.1).

Let $G = (V, E, R)$ be a mixed graph, consisting of a set V of vertices, a set E of undirected edges without parallels, and a multiset R of directed arcs (parallel arcs allowed). An undirected edge between vertices u and v will be denoted by $[u,v]$ and a directed arc from u to v will be denoted by (u,v). In this chapter we let $n := |V|$, $m_E := |E|$ and $m_R := |R|$ be the number of vertices, edges and arcs, respectively.

For $v \in V$ we let R_v be the set of arcs in R emanating from v. For edge set E, denote by

$$E^{\rightleftarrows} := \{\, (u,v), (v,u) : [u,v] \in E \,\} \tag{6.1}$$

the set of arcs which contains for each undirected edge $e \in E$ a pair of anti-parallel arcs between the endpoints of e.

For $v \in V$ we let R_v be the set of arcs in R emanating from v. If $X \sqsubset R$, we briefly write $\deg^+_X(v)$ and $\deg^-_X(v)$ instead of $\deg^+_{G[X]}(v)$ and $\deg^-_{G[X]}(v)$ for the outegree and indegree of vertex v in the subgraph $G[X]$ induced by X. A graph G is called *degree-balanced* if

degree-balanced

$$\deg^+_G(v) = \deg^-_G(v)$$

for all vertices $v \in V$.

Let $G = (V, E, R)$ be a mixed graph and suppose that $W = (x_1, \dots, x_p)$, $x_i \in E + R$ for $i = 1, \dots, p$, is an oriented walk in G. For a cost function $d\colon E + R \to \mathbb{R}^+_0$ the *cost* of the walk W is given by

cost

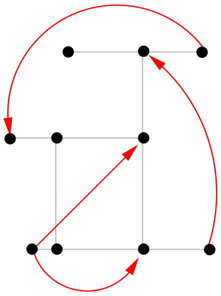
$$d(W) := \sum_{i=1}^{p} d(x_i).$$

6.1.1 Basic Problem

In the capacitated dial-a-ride problem CDARP we are given a finite transportation network and a finite set of transportation requests. Each request specifies the source and target location which are both part of the network. The goal of the CDARP is to find a shortest closed transportation schedule for the requests which starts and ends at a designated origin. The problem DARP is the restriction of the CDARP to those instances where the server has capacity one. As mentioned before, in this chapter we consider the case of a server with unit capacity.

Since the release times of all requests are zero, in all what follows we can restrict ourselves to transportation schedules which do not use waiting time. That is, we can assume without loss of generality that a feasible transportation schedule $S = (\tau_1, x_1, y_1, R_1), , \dots, (\tau_\ell, x_\ell, y_\ell, R_\ell)$ satisfies: $\tau_1 = 0$ and $\tau_i = \sum_{j<i} d(x_j, y_j)$. Hence, the length of a transportation schedule is exactly the distance traveled by the server in the metric space.

We model the transportation network by an edge weighted undirected connected graph. Each request is modeled by a directed arc from its source



Offline dial-a-ride problem DARP: The requests (red) have sources and destinations in the transportation network (black) modeled by an edge weighted undirected graph.

node to its target node. The length of the arc is adjusted to reflect the length of a shortest path in the network connecting its endpoints. Then a closed oriented walk in the resulting mixed graph which traverses each arc corresponds to a transportation for all the requests in the transportation network. More formally, we define the DARP as follows:

Definition 6.1 (Offline Dial-a-Ride Problem DARP)
The input for the DARP consists of a finite mixed graph $G = (V, E, R)$*, an origin vertex* $o \in V$ *and a nonnegative weight function* $d \colon E \to \mathbb{R}_0^+$.

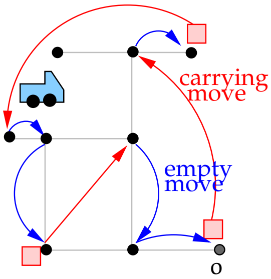
The weight function d *is extended to* R *by defining for each arc* $r \in R$, $r = (v, w)$*, its cost* $d(r)$ *to be the length of a shortest path from* v *to* w *in* $G[E]$.

The goal of the DARP is to find a closed oriented walk in G *of minimum length which starts (and ends) in* o *and traverses each arc in* R.

It turns out that for the purpose of stating algorithms in a more convenient way, it is helpful to use an equivalent formulation of the DARP as a *graph augmentation* problem (cf. [AK88]). To this end consider the arc set $E^{\rightleftarrows}$ defined in (6.1). Let the cost of each arc in $E^{\rightleftarrows}$ equal the cost of the corresponding edge in E.

graph augmentation

Definition 6.2 (Graph Augmentation Version of DARP)
Given a mixed graph $G = (V, E, R)$ *with origin* $o \in V$*, and a cost function* $d \colon E \to \mathbb{R}_0^+$*, extend* d *to* $R + E^{\rightleftarrows}$ *as described in Definition 6.1 and the previous paragraph. The goal is to find a multiset* S, $S \sqsubset E^{\rightleftarrows}$*, of minimum cost such that the graph* $G[R + S]$ *is Eulerian and contains* o.



A feasible solution for DARP corresponds to an augmenting set of arcs. Each empty move of the server yields a new blue arc.

We argue that Definitions 6.1 and 6.2 are in fact equivalent. Let W be a feasible solution for the DARP as stated in Definition 6.1, that is, a closed oriented walk that starts in o and traverses each arc in R. Construct a multiset $S \sqsubset E^{\rightleftarrows}$ of arcs in the following way: Traverse edges and arcs along W. For each time an undirected edge $e \in E$, $e = [u, v]$, is traversed from u to v, add a copy of the directed arc (u, v) to the multiset S. Then the graph $G[R + S]$ contains o, and since W defines a cycle, graph $G[R + S]$ must be Eulerian.

Conversely, let $S \sqsubset E^{\rightleftarrows}$ be a multiset of arcs such that $G[R + S]$ is Eulerian and includes the origin o. Construct an oriented walk W as follows: Traverse an Eulerian cycle C in $G[R + S]$ starting in o. If the current arc a from C is in R then add a to walk W, otherwise add the undirected edge corresponding to a. By this construction, W is a closed oriented walk in G traversing each arc from R and including o.

In both cases we have $d(W) = d(R + S)$, i.e., the cost of walk W equals the cost of the multiset S plus cost of the arc set R.

6.1.2 Precedence Constraints

In real applications of the DARP there are often additional constraints on the order of the execution of transportation requests. This can be modeled by introducing a partial order $\prec$ on the set of arcs. A feasible oriented walk is then required to satisfy the condition that, whenever $a \prec b$, then a must be traversed before b by the walk.

In some transportation networks there is a "waiting pool" at each node where transportation requests originate. Each of the pools constrains the order of execution of requests starting from this node while requests starting from other nodes are not affected. For instance there might be waiting pools with first-in first-out queues or waiting stacks (last-in first-out). This motivates the definition of a *source-order*:

Definition 6.3 (Source-Order, Total-Source Order)
A ***source-order*** *on the arcs of a mixed graph* $G = (V, E, R)$ *is a partial order* $\prec$ *on* R *with the following property:* $a \prec b$ *implies that* a *and* b *emanate from the same source node.*

If a source-order $\prec$ *has the property that*

$$a \text{ and } b \text{ share the same source node} \Rightarrow a \prec b \vee a \prec b,$$

then $\prec$ *is called a* ***total source-order****.*

This leads to problem SOURCE-DARP (shorthand for "DARP with source-orders") which is the main focus of this chapter:

Definition 6.4 (Offline Dial-a-Ride Problem SOURCE-DARP)
An instance of the SOURCE-DARP *consists of an instance of the* DARP*, together with a source-order* $\prec$ *on the arc set* R*. The goal is to find a closed oriented walk satisfying the requirements specified in Definition 6.1 and the precedence constraints given by* $\prec$*.*

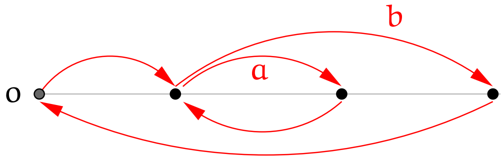
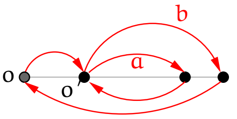


Figure 6.1
The precedence constraint $b \prec a$ increases the cost of an optimal solution.

Figure 6.1 shows an example where an optimal source-order respecting transportation is strictly longer than the optimal transportation neglecting the precedences. If no constraints have to be obeyed, then the requests can be served traversing only along the arcs. If the constraint $b \prec a$ must be obeyed then two additional empty moves along undirected edges are necessary.

It will be useful to restate the SOURCE-DARP as a graph augmentation problem. We need some additional notation:



Given the precedence constraint $b \prec a$, the directed graph indicated by the red arcs is $\prec$-Eulerian with start o' but not with start o.

Definition 6.5 ($\prec$-respecting Eulerian Cycle, $\prec$-Eulerian)
*Let $H = (V, A)$ be a directed graph, $\prec$ be a source-order on the arcs A, and $o \in V$. A **$\prec$-respecting Eulerian cycle** in H **with start** o is a Eulerian cycle C in H such that $a \prec a'$ implies that in the oriented walk from o along C the arc a appears before a'. The graph H is then called **$\prec$-Eulerian with start** o.*

Notice that in contrast to the case of classical Eulerian cycles, for $\prec$-respecting Eulerian cycles it is meaningful to specify a start node explicitly.

Definition 6.6 (Graph Augmentation Version of Source-Darp)
An instance of the problem Source-Darp *consists of the same input as for* Darp *and additionally a source-order $\prec$ on the arc set R. The goal is to find a multiset S of arcs from $E^{\rightleftarrows}$ minimizing the weight $d(R+S)$ such that $G[R+S]$ is $\prec$-Eulerian with start o, and to determine a $\prec$-respecting Eulerian cycle in $G[R+S]$.*

S^*, OPT

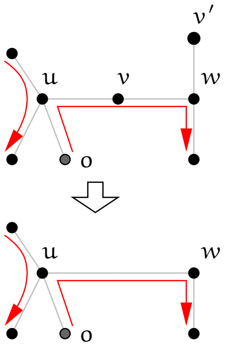
Throughout this chapter we use $S^* \sqsubset E^{\rightleftarrows}$ to denote an optimal solution (augmentation set) for the Source-Darp and $\text{OPT} := d(R + S^*)$ to denote its cost.

6.1.3 Basic Observations

We first start with some technical assumptions about input instances $(G = (V, E, R), d, o, \prec)$ of the Source-Darp depending on the structure of the undirected graph $G[E]$ specified in the instance. While all these assumptions are without loss of generality they simplify the presentation of our algorithms.

essential vertex

Definition 6.7 (Essential Vertex)
*A vertex $v \in V$ is called **essential**, if $\deg_R(v)^+ \deg^-_R(v) \geq 1$, that is, if v is incident with at least one arc from R.*



A non-essential vertex $v \neq o$ of degree 2 in $G[E]$ can be removed by merging the two adjacent edges. A non-essential vertex $v' \neq o$ of degree 1 can be removed.

Suppose that $G[E]$ is a tree. We show that we can assume that most of the nodes in V are essential. To this end, assume that $v \neq o$ is not essential. If v is of degree 2 in $G[E]$, we can replace the two adjacent edges, say $[u, v]$ and $[v, w]$, by the single edge $[u, w]$ of cost equal to the sum of the two edges. If v is a leaf, it can be removed without affecting the optimal solution (cf. [FG93] for the Darp on trees).

Assumption 6.8 (Technical assumption for the Source-Darp on trees)
Each vertex of degree 1 or 2 in $G[E]$, unless the origin o, is essential.

If $G[E]$ is a path, then there are no longer vertices in $G[E]$ which are of degree 3 or greater. Thus, it is easy to see that we can make an even stronger assumption without loss of generality (cf. [AK88] for the Darp on paths):

Assumption 6.9 (Technical assumption for the Source-Darp on paths)
Each vertex $v \in V$ is essential.

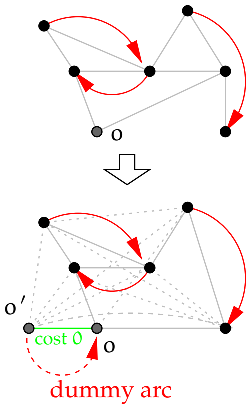
We now turn to the Source-Darp on general graphs.

Assumption 6.10 (Tech. ass. for the Source-Darp on general graphs)

(i) *Each vertex $v \in V$ is essential.*

(ii) $G[E]$ *is complete.*

(iii) *The cost function* d *obeys the triangle inequality, that is, for any edge* $e \in E$, $e = [u, v]$, *the cost* $d(e)$ *equals* $\mathrm{dist}_E(u, v)$, *the length of a shortest path in* $G[E]$ *between* u *and* v.



On general graphs the start vertex o can be enforced to be essential. All other non-essential vertices can be removed by completing the graph along shortest paths (complete graph indicated by dotted edges).

Assumption 6.10 can be enforced without increasing the value of an optimal solution. If the start vertex o is not essential, insert a new vertex o' joined by arc (o', o) and edge $[o, o']$ to the start vertex, each of cost zero. To satisfy the triangle inequality, for every pair u, v of vertices add a new edge $[u, v]$ of cost equal to the shortest path in $G[E]$ between u and v. Afterwards each bundle of parallel edges can be replaced by retaining the cheapest edge of the bundle, and vertices which are not incident to an arc can be removed safely (cf. [FHK78] for the Darp).

However, Assumption 6.10 can not be made without loss of generality for Source-Darp on trees, since the suggested modification of the graph destroys the “tree-property”.

6.1.4 Balancing

The formulation of the Source-Darp as a graph augmentation problem suggests to investigate the structure of Eulerian graphs. A necessary condition for a graph to be Eulerian is that for each node its indegree equals its outdegree. It turns out to be helpful for solving Source-Darp to search for augmenting sets which guarantee the resulting graph to be *balanced* in that way.

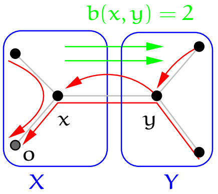
balanced

Definition 6.11 (Balancing Set)
Let $G = (V, E, R)$ *be a mixed graph. A multiset* $B \sqsubset E^{\rightleftarrows}$ *of arcs is called a* ***balancing set*** *if* $\deg^+_{R+B}(v) = \deg^-_{R+B}(v)$ *for all vertices* $v \in V$.

balancing set

Suppose that $G[E]$ is a tree and that Assumption 6.8 is satisfied. We now show how to determine a multiset of arcs which is necessarily contained in any feasible solution.

For a partition $V = X \cup Y$ of the vertex set define the cut $(X : Y)$ to consist of all edges and arcs from $E + R$ with one endpoint in X and the other one in Y. Any edge $[x, y] \in E$ defines a partition $V = X \cup Y$ of the node set, where X and Y are defined by the connected components after removing the edge $[x, y]$ from the tree. Obviously, the cut $(X : Y)$ must be traversed by any closed oriented walk W the same number of times in each direction. Denote by $\phi(X, Y) := |\{(x, y) \in R : x \in X \wedge y \in Y\}|$ the number of arcs emanating from X. Hence, if W traverses all arcs from R, it must traverse edge $[x, y]$ from x to y at least $b(x, y)$ times, where



Any feasible closed oriented walk must traverse edge $[x, y]$ in direction from x to y at least $b(x, y) = 2$ times.

$$b(x, y) := \begin{cases} 1 & \text{if } \phi(X, Y) = \phi(Y, X) = 0 \\ \phi(Y, X) - \phi(X, Y) & \text{if } \phi(Y, X) > \phi(X, Y) \\ 0 & \text{otherwise.} \end{cases}$$

This observation has the following consequence for the graph augmentation version: If $B \sqsubset E^{\rightleftarrows}$ is a multiset of arcs such that $B((x, y)) = b(x, y)$ for each (x, y), that is, B contains exactly $b(x, y)$ copies of the directed arc (x, y), then there is at least one optimal solution S^* such that $B \subseteq S^*$. This yields the following lemma which is proved in [AK88, FG93].

Lemma 6.12 *Let* (G, d, o) *be an instance of the* DARP *such that* $G[E]$ *is a tree. Then in time* $\mathcal{O}(nm_R)$ *one can find a balancing set* $B \sqsubset E^{\rightleftarrows}$ *such that* $B \subseteq S^*$ *for some optimal solution* S^*. □

It is straightforward to verify that Lemma 6.12 remains valid even in the presence of source-orders. As is also shown in [AK88, FG93] the time bound of $\mathcal{O}(nm_R)$ can be improved to $\mathcal{O}(n + m_R)$ by allowing balancing arcs to be from $V \times V$ instead of just $E^{\rightleftarrows}$. This does not change the problem: the cost function d can basically be extended from $E^{\rightleftarrows}$ to $V \times V$ by using the length of shortest paths.

6.2 Euler Cycles Respecting Source-Orders

Assume for the moment that $\prec$ is a total source order on the arcs of a given directed graph $H = (V, A)$. It is easy to decide whether H is $\prec$-Eulerian with start o: The $\prec$-respecting cycle (if it exists) is uniquely determined and can be found by an oriented walk through the graph where at each vertex v we always choose among the yet unused arcs from A_v the minimal (with respect to $\prec$).

In the sequel we prove a necessary and sufficient condition for a graph to be $\prec$-Eulerian with start at a given vertex.

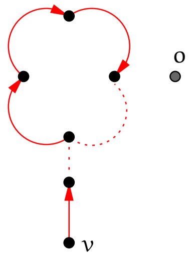
Consider an Eulerian cycle with start o in a connected directed graph. The cycle visits each vertex, and by this defines for each vertex v an arc emanating from v which is traversed last.

Definition 6.13 (Set of Last Arcs)
Let C be an Eulerian cycle with start o. We define the ***set of last arcs*** *of C, denoted by L_C, to contain for each vertex $v \in V$ the unique arc emanating from v which is traversed last by C.*

set of last arcs

Recall that a *directed in-tree rooted at* $o \in V$ is a subgraph of a directed graph $H = (V, A)$ which is a tree and which has the property that for each $v \in V$ it contains a directed path from v to o.

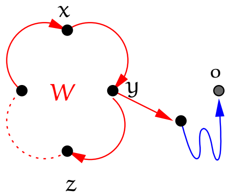
Observation 6.14 *The set L_C of last arcs consists of a directed in-tree rooted at o plus one single arc emanating from o.*



Consequence of the contradictory assumption: Following the arcs in D yields a cycle which does not contain o.

Proof: Let D denote the set L_C without the arc emanating from o. Since $|D| = |V| - 1$ it suffices to show that for each vertex $v \in V$ there exists a directed path from v to the origin o using only arcs from D. Suppose for the sake of a contradiction that there is no such path from a certain vertex v to o. Since by definition for each vertex from $V \setminus \{o\}$ there is exactly one arc from D emanating from this vertex and the graph D is finite, it follows that D contains a cycle W. By construction D does not contain any arc emanating from o. Hence, W does not touch o.

Let (x, y) be the arc from W which is traversed last among all arcs in W by the Eulerian cycle C with start o. Let (y, z) be the successor of (x, y) in W. Since C ends in o, there must be an arc from D emanating from y with endpoint not in W and which is traversed by C after (y, z). This, however, contradicts the fact that (y, z) was a last arc. □



The Eulerian cycle C can not leave the cycle *W* composed of last arcs.

We now investigate the case of Eulerian cycles which respect to source-orders. Let $\prec$ be a source-order. We denote the set of *maximal elements* with respect to $\prec$ by $M_\prec$, that is,

maximal elements

$$M_\prec := \{\, a \in A : \text{there is no arc } a' \text{ such that } a \prec a' \,\}.$$

Definition 6.15 (Possible Set of Last Arcs)
Let $H = (V, A)$ be a directed graph and $o \in V$ be a distinguished vertex. A set $L \subseteq A$ is called a ***possible set of last arcs****, if $H[L]$ is a directed in-tree rooted at o plus an additional arc emanating from o.*

possible set of last arcs

Using the proof of Observation 6.14 it is easy to derive the following equivalent characterization, which will be useful in the sequel.

Observation 6.16 *A set $L \subseteq A$ is a possible set of last arcs in $H = (V, A)$ if and only if it satisfies the following conditions:*

(i) $\deg^+_L(v) = 1$ *for all $v \in V$, and*

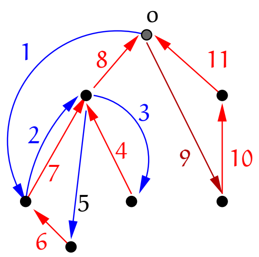
(ii) for each $v \in V$ there is a path from o to o in $H[L]$.

□

The following theorem justifies the nomenclature “possible set of last arcs”.

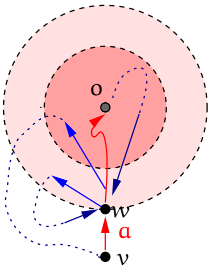
Theorem 6.17 *Let $H = (V, A)$ be a directed Eulerian graph with distinguished vertex $o \in V$ and let $\prec$ be a source-order with maximal elements $M_\prec$. Suppose that a possible set L of last arcs satisfies $L \subseteq M_\prec$.*

Then there exists an $\prec$-respecting Eulerian cycle C with start o in H such that $L_C = L$, i.e., such that L is the set of last arcs of C. This cycle can be found in time $\mathcal{O}(|V| + |A|)$.



Proof of Theorem 6.17. The numbers indicate in which order the arcs are traversed by the constructed Eulerian cycle C.

Proof: Color the arcs from L red and the arcs in $A \setminus L$ blue. We claim that by the following procedure we construct an Eulerian cycle C in H with the desired properties. Start with current vertex o. As long as there is a blue untraversed arc emanating from the current vertex, choose one which is not $\prec$-preceeded by any other untraversed arc, otherwise choose the red arc. Traverse the chosen arc, let its target be the new current vertex, and repeat the iteration. Stop, if there is no untraversed arc emanating from the current vertex. Call the resulting path of traversed arcs C. Since H is Eulerian by assumption, for each vertex its indegree equals its outdegree. Therefore, C must end in the origin o and forms in fact a cycle. Moreover, C is $\prec$-respecting by construction since $L \subseteq M_\prec$. Hence, if we can show that C traverses all arcs from A then this implies $L = L_C$ and the proof is complete.



All arcs emanating from w have been traversed. Since $\deg^+_H(w) = \deg^-_H(w)$, also all incoming arcs including a must be contained in the cycle C.

To this end, define for each node $v \in V$ the value $\text{dist}(v, o)$ to be the distance to o (i.e., the number of arcs on the shortest path from v to o) in the subgraph $H[L]$. We show by induction on $\text{dist}(v, o)$ that all arcs emanating from v are contained in C.

If $\text{dist}(v, o) = 0$ then $v = o$. Since our procedure stopped, all arcs emanating from o are contained in C. This proves the induction basis. Assume that the claim holds true for all vertices with distance $t \geq 0$ and let $v \in V$ with $\text{dist}(v, o) = t + 1$. Let $a = (v, w)$ be the unique red arc emanating from v. Then $\text{dist}(w, o) = t$ and by the induction hypothesis all arcs emanating from w are contained in C. Since $\deg^+_H(w) = \deg^-_H(w)$, it follows that all arcs entering w, in particular arc a, are also contained in C. Since red arc a is chosen last by our procedure, all other arcs emanating from v must be contained in C. This completes the induction. Hence, C is actually an Eulerian cycle with the claimed properties. □

Corollary 6.18 (Characterization of $\prec$-Eulerian Graphs) *Let $H = (V, A)$ be a directed graph, $o \in V$ and $\prec$ a source-order. Then the following two statements are equivalent:*

1. H *is* $\prec$*-Eulerian with start* o.

2. H *is Eulerian and the set* $M_{\prec}$ *of maximal elements with respect to* $\prec$ *contains a possible set of last arcs.*

Proof: Suppose that the directed graph H is $\prec$-Eulerian with start o, and let C be an $\prec$-respecting Eulerian cycle with start o in H. Then $L_C \subseteq M_{\prec}$. Thus, Statement 1 implies 2. The other direction is an immediate consequence of Theorem 6.17. □

The above corollary implies a polynomial time algorithm for deciding whether a given graph H is $\prec$-Eulerian with start o. Provided H is Eulerian it suffices to check whether the subgraph formed by the arcs from $M_{\prec}$ contains a possible set of last arcs. By the remark above this check can essentially be performed by testing whether $M_{\prec}$ contains a directed in-tree D rooted at o (which can be done in linear time).

6.3 A Polynomial Time Algorithm on Paths

We now present Algorithm ALG-PATH which solves the SOURCE-DARP on paths. Let $G = (V, E, R)$ be a mixed graph such that $G[E]$ is a path. We assume throughout this section that the technical Assumption 6.9 holds. An illustration of the execution of ALG-PATH is shown in Figure 6.2.

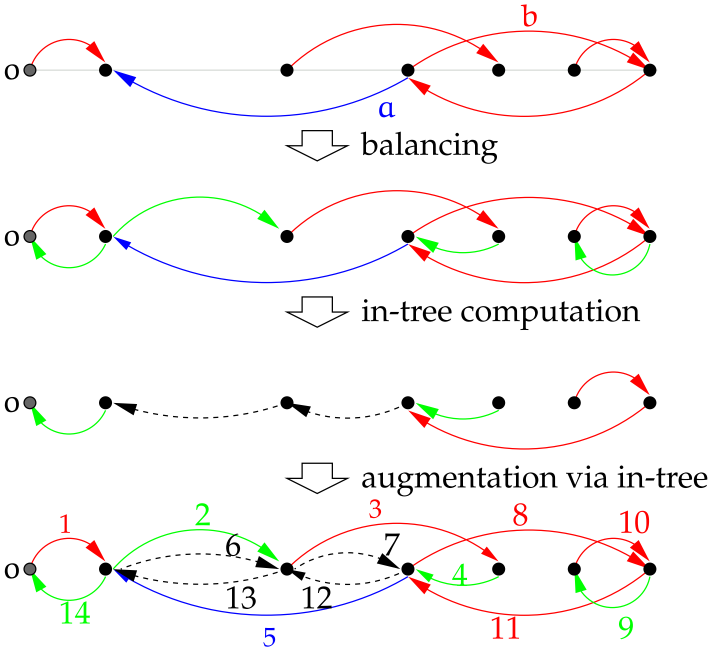


Figure 6.2 Steps of Algorithm ALG-PATH. Initial instance (top), graph $G[R + B]$ after balancing, minimum weight in-tree D (the arcs in D with non-zero cost are shown as dashed arcs), final solution (bottom, the numbers indicate the order of arcs in the $\prec$-respecting Eulerian cycle).

```
Input: A mixed graph $G = (V, E, R)$, such that $G[E]$ is a path, a cost
       function $d$ on $E$, an initial vertex $o \in V$, and a source-order $\prec$.
1 Let $M_\prec$ be the set of maximal elements with respect to $\prec$.
2 Compute a balancing set $B \sqsubset E^{\rightleftarrows}$ such that $B \subseteq S^*$ for some optimal
  solution $S^*$.
3 Compute a directed in-tree $D$ rooted at $o$ in $G[B + M_\prec + E^{\rightleftarrows}]$ of minimum
  weight $d'(D)$, where weight function $d'$ is defined as follows:
      $d'(r) = \begin{cases} 0 & \text{if } r \in B + M_\prec \ , \\ d(r) & \text{if } r \in E^{\rightleftarrows} \setminus (B + M_\prec) \, . \end{cases}$
4 Define a possible set $L$ of last arcs by $L := D \cup \{r\}$, where $r$ is an arbitrary
  arc from $R_o \cap (M_\prec + B)$.
      { Such an arc must exist since $o$ is essential
        and $G[R + B]$ is degree-balanced. }
5 Let $D_+ := D - (B + M_\prec)$ and $N := E^{\rightleftarrows} \cap (D_+ \cup D_+^{-1})$ .
      { The set $N$ contains the set $D_+$ of "new arcs"
        from the tree $D$ and their inverses $D_+^{-1}$. }
6 Use the method from Theorem 6.17 to find a $\prec$-respecting Eulerian cy-
  cle $C$ with start $o$ in $G[R + B + N]$ such that $L$ is the set of last arcs of $C$.
7 return the multiset $B + N$ and the cycle $C$.
```

Algorithm 6.1: Algorithm ALG-PATH for the SOURCE-DARP on paths.

The Algorithm ALG-PATH starts in Step 2 by determining a suitable balancing set $B \sqsubset E^{\rightleftarrows}$ which is guaranteed to be contained in some optimal solution (following the results of Lemma 6.12). At this point, the graph $G[R + B]$ is degree-balanced but may consist of several connected components. In order to turn the graph $\prec$-Eulerian with start o, the idea is to connect the components by pairs of antiparallel arcs from set $E^{\rightleftarrows}$, and in the same moment ensuring the existence of a possible set of last arcs.

This task is performed in two parts by the algorithm: In Step 3, a directed in-tree rooted at o of minimum cost is computed with respect to an auxiliary cost function. This cost function is adjusted to measure only the additional arcs that are not yet contained in $R + B$. In Step 5 an auxiliary arc set N is defined which contains those additional arcs together with their inverse arcs. This guarantees that $G[R+B+N]$ is in fact $\prec$-Eulerian which is proved formally in the following lemma:

Lemma 6.19 *The set* $B+N$ *returned by Algorithm* ALG-PATH *is a feasible solution for the* SOURCE-DARP *i.e.,* $G[R + B + N]$ *is* $\prec$*-Eulerian with start* o.

Proof: Since $G[R + B]$ is degree-balanced and N consists of pairs of anti-parallel arcs, also $G[R + B + N]$ is degree-balanced. By construction, $G[R + B + N]$ contains a directed in-tree rooted at o, namely the tree D computed in Step 3. Hence, it is strongly connected and Eulerian.

The set L of arcs determined in Step 4 is clearly a possible set of last arcs. The claim now follows from Theorem 6.17. □

It remains to show that the solution produced by ALG-PATH is not only feasible but also of minimum cost.

Theorem 6.20 ALG-PATH *finds an optimal solution for the* SOURCE-DARP *on paths.*

Proof: Let S^* be an optimal solution such that $B \subseteq S^*$ (by Lemma 6.12 such a multiset S^* exists). By feasibility of S^* the graph $G[R + S^*]$ is $\prec$-Eulerian with start o. Consider the multiset

$$Z := (R + S^*) - (R + B) = S^* - B$$

which contains the arcs from the optimal solution that are not contained in the current intermediate balanced graph $G[R + B]$. We show that the cost of the arcs added by ALG-PATH in the remaining steps of the algorithm is bounded from above by $d(Z)$. This shows that the solution returned is indeed optimal.

Since $G[R + B]$ and $G[R + S^*] = G[R + B + Z]$ are both degree-balanced and $Z \cap (R + B) = \emptyset$, we can decompose the set Z into arc disjoint cycles. Since Z consists of (multiple copies of) arcs from $E^{\rightleftarrows}$ and $G[E]$ is a tree, $r \in Z$ implies that $r^{-1} \in Z$.

Let C be a $\prec$-respecting Eulerian cycle in $G[R + S^*]$ and let L be its set of last arcs. Notice that $L \subseteq B + M_\prec + Z$, where $M_\prec$ is the set of maximal elements with respect to $\prec$ as defined in Step 1 of the algorithm.

The set L must contain a directed in-tree D' rooted at o. This tree spans at least all those components which contain essential vertices. Since by Assumption 6.9 all nodes are essential, we can conclude that tree D' is a spanning tree. Now we can compare its weight to the weight of tree D which is computed in Step 3 of the algorithm. Since the tree D is of minimum weight with respect to the weight function d' we have

$$d'(D') \geq d'(D). \tag{6.2}$$

We partition D' into the sets $D'_{B+M_\prec} := D' \cap (B + M_\prec)$ and $D'_Z := D' \cap Z$. Thus, $d'(D'_{B+M_\prec}) = 0$ and $d'(D'_Z) = d(D'_Z)$. Since we have seen that for each arc $r \in Z$ also its anti-parallel version $r^{-1} \in Z$ (and D'_Z does not contain a pair of anti-parallel arcs) we get that

$$d(Z) \geq 2d(D'_Z) = 2d'(D'_Z) + 2d'(D'_{B+M_\prec}) = 2d'(D') \overset{(6.2)}{\geq} 2d'(D). \tag{6.3}$$

The set N computed in Step 5 has cost

$$d(N) = 2d(D - (B + M_{\prec})) = 2d'(D - (B + M_{\prec})) = 2d'(D) \overset{(6.3)}{\leq} d(Z). \tag{6.4}$$

Using this result yields that

$$\begin{aligned} d(R + B + N) &= d(R + B) + d(N) \\ &= d(R + (S^* - Z)) + d(N) \\ &\leq d(R + (S^* - Z)) + d(Z) && \text{by (6.4)} \\ &= d(R + S^*). \end{aligned}$$

Thus, $B + N$ is an optimal solution as claimed. □

We briefly comment on the running time of Algorithm ALG-PATH. A balancing set B can be found in time $\mathcal{O}(nm_R)$ by techniques as shown in [AK88]. As noted before this time bound can be improved to $\mathcal{O}(n + m_R)$ by allowing balancing arcs to be from $V \times V$ instead of just $E^{\rightleftarrows}$. A directed in-tree of minimum weight in a graph with n vertices and m arcs can be computed in time $\mathcal{O}(\min\{m \log n, n^2\})$ by the algorithm from [Tar77].

Thus Algorithm ALG-PATH can be implemented to run in time $\mathcal{O}(n+m_R+\min\{(m_R + n) \log n, n^2\})$.

6.4 An Approximation Algorithm for General Graphs

In this section we present an approximation algorithm for the SOURCE-DARP on general graphs. The algorithm uses ideas similar to the ones in [FHK78]. In this section we will assume tacitly that Assumption 6.10 is satisfied.

Our final algorithm actually consists of *two* different sub-algorithms, ALG-TSP and ALG-LA, which are run both and the best solution is picked. The first sub-algorithm, ALG-TSP, uses the fact that a transportation schedule must visit at least every vertex v with $\deg_R^+(v) > 0$ and, hence, a transportation schedule can be related to a TSP-tour on the vertices with positive outdegree. Our second algorithm, ALG-LA, is based on similar ideas as the algorithm from Section 6.3 for paths.

6.4.1 TSP-based Algorithm

Suppose that S is a arbitrary feasible solution for the SOURCE-DARP. Since the corresponding $\prec$-respecting Eulerian cycle in $G[R+S]$ traverses each arc from R, this cycle must also visit each vertex $v \in V$ from the set V_{source}, where

$$V_{\text{source}} := \{ v \in V : \deg^+_R(v) > 0 \}.$$

Thus, it follows that any feasible solution S has cost at least the length of an optimal TSP-tour on the vertices V_{source}. ALG-TSP computes a shortest TSP-tour that visits each vertex in V_{source}, i.e., a shortest Hamiltonian cycle in the complete subgraph induced by V_{source}. Then, it uses this TSP-tour to obtain a feasible solution for the SOURCE-DARP. The solution traverses along the TSP-tour, and whenever a vertex v is reached where arcs are emanating from, for each arc in R_v, that is, for each arc emanating from v, the TSP-tour is augmented by a loop traversing that arc and a shortest path back to vertex v. ALG-TSP is displayed in Algorithm 6.2. An example of the execution is given in Figure 6.3.

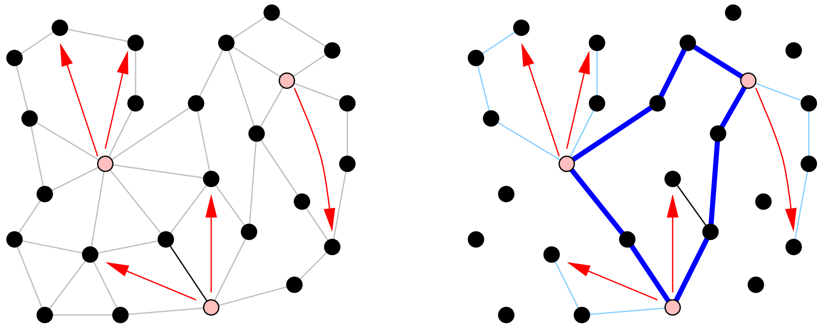


Figure 6.3 Illustration of Algorithm ALG-TSP. The vertices in V_{source} are emphasized (left). The solution on the right hand side consists of the TSP-tour (thick lines) and loops (arcs from R and thin lines).

We now prove a bound on the quality of the solution found by the TSP-based algorithm ALG-TSP.

Lemma 6.21 *If in Step 3 of* ALG-TSP *a* ρ_{TSP}*-approximation algorithm for computing a* TSP*-tour is employed, then the algorithm finds a solution of cost at most* $\rho_{\text{TSP}} \cdot \text{OPT} + 2d(R)$.

Proof: Let S^* be an optimal augmenting set and C^* be a $\prec$-respecting Eulerian cycle in $G[R + S^*]$ starting at o. By definition $d(C^*) = \text{OPT}$. As already remarked, the length of C^* (which equals OPT) is at least that of a shortest TSP-tour on V_{source}. Thus, the tour computed in Step 3 will have length at most $\rho_{\text{TSP}} \cdot \text{OPT}$. The additional cost incurred in Step 9 is not greater than $2d(R)$, since each path added has the weight of the corresponding arc from R. □

Since the cost of the optimal tour serving all requests is at least $d(R)$, Lemma 6.21 implies that ALG-TSP is a $(\rho_{\text{TSP}} + 2)$-approximation algorithm for SOURCE-DARP. Using Christofides' algorithm [Chr76] we get $\rho_{\text{TSP}} = 3/2$ and thus ALG-TSP implies a 7/2-approximation for the SOURCE-DARP. In the sequel we will improve this bound by providing a second algorithm and combining this algorithm with ALG-TSP.

```
Input: A mixed graph $G = (V, E, R)$, a cost function $d$ on $E$, an initial
       vertex $o \in V$, and a source-order $\prec$
Let $V_{\text{source}}$ be the set of vertices which are sources of arcs from $R$.
Compute a complete undirected auxiliary graph $U$ with vertex set
  $V_{\text{source}}$. The weight $d(v, w)$ of edge $[v, w]$ is set to be the length of a shortest
  path in $G[E]$ from $v$ to $w$.
Find an approximately shortest TSP-tour $p$ in $U$. Assume that $p$ visits
  the nodes of $V_{\text{source}}$ in order $(o = v_0, v_1, \dots, v_s, v_{s+1} = o)$.
Construct a feasible cycle $C$ for SOURCE-DARP as follows:
Start with the empty cycle $C$.
for $i := 0, \dots, s$ do
  Assume that $R_{v_i} = \{r_1, \dots, r_k\}$ with $r_i \prec r_j \Rightarrow i < j$.
  For $j = 1, \dots, k$, let $p_j$ be a directed shortest path in $G[E^{\rightleftarrows}]$ from the
    endpoint of $r_j$ to $v_i$.
  Add the $k$ loops $r_1 p_1, \dots, r_k p_k$ to $C$.
  Append to $C$ the directed shortest path in $G[E^{\rightleftarrows}]$ from $v_i$ to $v_{i+1}$.
end for
Let $S \leftarrow C - A$.
return the set $S$ and the cycle $C$.
```

Algorithm 6.2: The TSP-based Approximation Algorithm ALG-TSP for the SOURCE-DARP.

6.4.2 Algorithm Based on a Set of Last Arcs

Our second algorithm, ALG-LA, is based on similar ideas as the algorithm from Section 6.3 for paths. We first compute a set of balancing arcs B which makes $G[R + B]$ degree balanced. Again, we then compute a directed in-tree rooted at the origin o of minimum cost, double the new arcs which are not yet in $R + B$ and add the resulting set N to the solution. ALG-LA is shown in Algorithm 6.3.

By a proof similar to Lemma 6.19 it follows that the set $B + N$ found by Algorithm ALG-LA is indeed a feasible solution. We proceed to bound the cost of the solution produced by ALG-LA. This is accomplished in a number of steps.

Lemma 6.22 *The balancing set B found in Step 1 of algorithm* ALG-LA *has cost at most* OPT $- d(R)$. *Step 1 can be accomplished in the time needed for one minimum cost flow computation on a directed graph with n vertices and $2m_E$ arcs.*

Proof: Let $S^* \sqsubset E^{\rightleftarrows}$ be an optimal solution, i.e., an augmenting multiset of arcs from $E^{\rightleftarrows}$ with minimum cost. Recall that OPT $= d(R) + d(S^*)$. Since $G[R + S^*]$ contains a $\prec$-respecting Eulerian cycle with start o, this graph is

```
Input: A mixed graph $G = (V, E, R)$, a cost function $d$ on $E$, an initial
       vertex $o \in V$, and a source-order $\prec$.
1  Compute a balancing multiset $B \sqsubset E^{\rightleftarrows}$ of minimum cost.
                     { How this step can be accomplished with the help of a
                                  minimum cost flow computation is described
                                                  in detail in Lemma 6.22. }
2  Follow steps 1 and 3 to 6 of Algorithm ALG-PATH to compute a set $N$ of
   arcs and a $\prec$-respecting Eulerian cycle $C$ with start $o$.
3  return the set $B + N$ and the cycle $C$.
```

Algorithm 6.3: Algorithm ALG-LA "mimicking" the algorithm for paths.

in particular degree-balanced. Thus, the cost $d(S^*) = \text{OPT} - d(R)$ is at least that of a minimum cost set $B \sqsubset E^{\rightleftarrows}$ which achieves the degree-balance.

Step 1 can be carried out by performing a minimum cost flow computation in the auxiliary graph $F = (V, E^{\rightleftarrows})$. A vertex v has charge $\deg^-_G(v) - \deg^+_G(v)$, and the cost of sending one unit of flow over arc $a \in E^{\rightleftarrows}$ equals its cost $d(a)$. We then compute an integral minimum cost flow in F. If the flow on an arc a is $t \in \mathbb{N}$, we add t copies of arc a to the multiset B. □

We continue to prove an upper bound on the cost of the set N of new arcs and their inverses computed in Step 2 of ALG-LA.

Lemma 6.23 *The cost of the arc set N computed by* ALG-LA *is at most* $2\text{OPT} - 2d(R)$.

Proof: The proof of the lemma is similar to the one for Theorem 6.20. The major difference is that, in general, we can not assure that the balancing set B computed in Step 1 is a subset of an optimal solution.

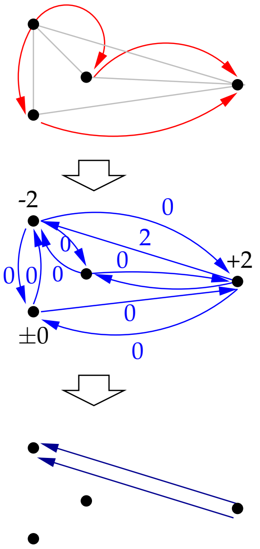
Let S^* be again an optimal augmenting set and L be the set of last arcs of a $\prec$-respecting Eulerian cycle in $G[R + S^*]$. Then $L \subseteq R + S^*$ and thus

$$\text{OPT} - d(R) = d(S^*) \geq d(L - (R + B)). \tag{6.5}$$

The only arcs from R that L can contain are those from the set $M_\prec$. Thus $L - (R + B) = L - (M_\prec + B)$ and

$$d(L - (R + B)) = d(L - (M_\prec + B)) = d'(L). \tag{6.6}$$

The last equality follows from the definition of the weight function d' (see step 3 in Algorithm ALG-PATH). Since L contains an in-tree rooted at o and the tree D computed is weight-minimal with respect to d' we have $d'(L) \geq d'(D)$.



Balancing step in general graphs. From the given instance (top), a flow network is computed (middle) with node-charges as shown by the numbers at the nodes. The minimum cost flow (indicated by numbers at the arcs in the center figure) is used to compute a balancing set of minimum cost (bottom).

Using the same arguments as in inequality (6.4) of Theorem 6.20 we conclude that $d(N) = 2d'(D)$. Using this equality together with (6.5) and (6.6) shows the claim of the lemma. □

Lemma 6.22 and Lemma 6.23 imply the following bound on the performance of Algorithm ALG-LA:

Corollary 6.24 ALG-LA *finds a solution of cost at most* $3 \cdot \text{OPT} - 2d(R)$.

Proof: By Lemma 6.22, $d(R+B) \leq \text{OPT}$. Lemma 6.23 establishes that $d(N) \leq 2 \cdot \text{OPT} - 2d(R)$. Thus $d(R+B+N) \leq 3 \cdot \text{OPT} - 2d(R)$ as claimed. □

6.4.3 Combining Both Algorithms

We are now ready to combine our algorithms ALG-TSP and ALG-LA into one with an improved performance guarantee. The combined algorithm ALG-COMBINE simply runs both algorithms and picks the better solution.

Theorem 6.25 ALG-COMBINE *is* ρ*-approximative with* $\rho = 1/2 \cdot (\rho_{\text{TSP}} + 3)$.

Proof: Let $\beta := 4/(3 - \rho_{\text{TSP}})$. If $\text{OPT} \geq \beta d(R)$, then the solution returned by ALG-TSP has cost at most

$$\left(\rho_{\text{TSP}} + \frac{2}{\beta}\right) \cdot \text{OPT} = \left(\rho_{\text{TSP}} + 2\,\frac{3 - \rho_{\text{TSP}}}{4}\right) \cdot \text{OPT} = \frac{\rho_{\text{TSP}} + 3}{2} \cdot \text{OPT}.$$

If $\text{OPT} < \beta d(R)$, then the cost of the solution found by ALG-LA is bounded from above by

$$\left(3 - \frac{2}{\beta}\right) \cdot \text{OPT} = \left(3 - 2\,\frac{3 - \rho_{\text{TSP}}}{4}\right) \cdot \text{OPT} = \frac{\rho_{\text{TSP}} + 3}{2} \cdot \text{OPT}.$$

This shows the claim of the theorem. □

alg.	upper bound
ALG-TSP	$\rho_{\text{TSP}} \cdot \text{OPT}$ $+2d(R)$
ALG-LA	$3\,\text{OPT}$ $-2d(R)$

Using Christofides' algorithm [Chr76] with $\rho_{\text{TSP}} = 3/2$ results in a performance guarantee of $3/4 + 3/2 = 9/4$ for algorithm ALG-COMBINE.

Theorem 6.26 *There exists an approximation algorithm for* SOURCE-DARP *with performance 9/4. This algorithm can be implemented to run in time* $\mathcal{O}(\max\{n^3 + m_R m_E + m_R n \log n, m_E^2 \log n + m_E n \log^2 n\})$.

Proof: The performance has already been proved. The running time of Algorithm ALG-TSP is dominated by that of Christofides' algorithm, which can be implemented to run in time $\mathcal{O}(n^3)$ [CLR90, JRR95], and the time needed for the addition of the paths in Step 7 which can be done in total time $\mathcal{O}(m_R m_E + m_R n \log n)$. The running time of ALG-LA is dominated by the minimum cost flow computation which can be accomplished in time $\mathcal{O}(m_E^2 \log n + m_E n \log^2 n)$ by using Orlin's enhanced capacity scaling algorithm [AMO93]. □

6.5 Improved Approximation Algorithm on Trees

For graph classes where the TSP can be approximated within a factor better than 3/2, the performance of ALG-COMBINE automatically improves over the one stated in Theorem 6.26. In particular, for trees where the TSP can be solved in polynomial time Theorem 6.25 already implies a 2-approximation algorithm. However, we can still improve this performance guarantee.

Our algorithm for trees, called ALG-COMBINE-T, uses the same outline as ALG-COMBINE. It runs two algorithms and chooses the best solution returned by its sub-algorithms. The first sub-algorithm is ALG-TSP which, as noted above, on trees is able to find a solution of cost at most $\text{OPT} + 2d(R)$ in polynomial time due to the polynomial time solvability of the TSP on trees ($\rho_{\text{TSP}} = 1$).

Lemma 6.27 *For the* SOURCE-DARP *on trees* ALG-TSP *finds a solution of cost at most* $\text{OPT} + 2d(R)$. □

The second sub-algorithm is the modified version of ALG-LA shown in Algorithm 6.4 and denoted by ALG-LA-T. The modifications are as follows: We defer the removal of the non-essential vertices in V and the completion of G via shortest paths until after the (modified) balancing step. The balancing step (Step 1 of ALG-LA-T) uses the fact that on trees we can determine a balancing subset which is a subset of some optimal solution.

```
Input: A mixed graph $G = (V, E, R)$, a cost function $d$ on $E$, an initial
       vertex $o \in V$, and a source-order $\prec$.
1 Compute a balancing multiset $B \sqsubset E^{\rightleftarrows}$ such that $B \subseteq S^*$ for some optimal solution $S^*$.
2 Complete the graph and remove all nodes not incident with arcs from $R + B$.
3 Follow steps 1 and 3 to 6 of Algorithm ALG-PATH to compute a set $N$ of
  arcs and a $\prec$-respecting Eulerian cycle $C$ with start $o$.
4 return the set $B + N$ and the cycle $C$.
```

Algorithm 6.4: Modified algorithm ALG-LA-T based on a set of last arcs for SOURCE-DARP on trees.

Lemma 6.28 ALG-LA-T *finds a solution of cost at most* $2\,\text{OPT} - 2d(R)$.

Proof: Let $I = (G = (V, E, R), d, o, \prec)$ be the original instance of the SOURCE-DARP such that $G[E]$ is a tree. We can consider the instance $I' = (G' = (V, E, R + B), d, o, \prec)$ (still on a tree) which results from adding the balancing arcs B as new transportation requests. Since any feasible solution to I will have to use the arcs from B anyway (cf. Lemma 6.12), we get that $\text{OPT}(I) = \text{OPT}(I')$.

Now look at the instance I'' of the SOURCE-DARP which is obtained by removing vertices and completing G along shortest paths as in our algorithm. It is easy to see that $\text{OPT}(I'') = \text{OPT}(I')$. Notice also that we can transform any feasible solution of I'' to a feasible solution of I' of the same cost: simply replace arcs not in $E^{\rightleftarrows}$ by shortest paths.

Let S^* and S'' be optimal solutions for I and I'', respectively. Define the multisets $Z := S^* - B$. Since

$$d(R+B) + d(Z) = \text{OPT}(I) = \text{OPT}(I'') = d(R+B) + d(S''), \tag{6.7}$$

we have that

$$d(Z) = d(S'') \tag{6.8}$$

and

$$d(Z) = \text{OPT}(I) - d(R+B). \tag{6.9}$$

Let $R+B+N$ be the solution of instance I found by ALG-LA-T. We have

$$d(R+B+N) = d(R+B) + d(N) = d(S^*) - d(Z) + d(N). \tag{6.10}$$

We can now use the arguments of Lemma 6.23 (taking into account that in the current situation of instance I'', set S'' is the optimal augmenting set). This yields

$$d(N) \leq 2d(S'') \overset{(6.8)}{=} 2d(Z) \tag{6.11}$$

in the current context. Plugging this inequality into (6.10) results in

$$\begin{aligned} d(R+B+N) &\leq d(S^*) + d(Z) && \text{by (6.11)} \\ &= \text{OPT}(I) - d(R) + \text{OPT}(I) - d(R+B) && \text{by (6.7) and (6.9)} \\ &= 2\,\text{OPT}(I) - 2d(R). \end{aligned}$$

This completes the proof. □

We are now ready to establish a bound on the performance of the combined algorithm ALG-COMBINE-T.

Theorem 6.29 ALG-COMBINE-T *is an algorithm for the* SOURCE-DARP *on trees with performance* $3/2$*. It can be implemented to run in time* $\mathcal{O}(nm_R + n^2 \log n)$.

Proof: We can estimate the cost of the best of the two solutions returned by ALG-TSP and ALG-LA-T by the techniques from the proof of Theorem 6.25: if $d(R) \leq \frac{1}{4}\,\text{OPT}$, then the cost of the solution produced by ALG-TSP algorithm (see Lemma 6.27) is bounded from above by

$$\text{OPT} + 2d(R) \leq \text{OPT} + \frac{1}{2}\,\text{OPT} = \frac{3}{2}\,\text{OPT}\,.$$

On the other hand, if $d(R) > \frac{1}{4}\,\text{OPT}$, then we know from Lemma 6.28 that the cost of the solution returned by ALG-LA-T is no greater than

$$2 \cdot \text{OPT} - 2d(R) \leq 2 \cdot \text{OPT} - \frac{1}{2}\,\text{OPT} = \frac{3}{2}\,\text{OPT}.$$

This yields an overall performance of 3/2 as claimed.

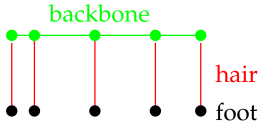
alg.	upper bound
ALG-TSP	OPT $+2d(R)$
ALG-LA-T	$2\,\text{OPT}$ $-2d(R)$

The time bound for the algorithm is derived as follows: We can solve the TSP on the metric space induced by $G[E]$ in time $\mathcal{O}(n)$. We then root the tree $G[E]$ at an arbitrary vertex. With $\mathcal{O}(n)$ preprocessing time, the least common ancestor of any pair of vertices can be found in constant time (see [HT84, SV88]). Thus, we can implement ALG-TSP in such a way that the invocations of Step 7 take total time $\mathcal{O}(nm_R)$. This means that ALG-TSP can be implemented to run in time $\mathcal{O}(nm_R)$.

The balancing in ALG-LA-T can be accomplished in time $\mathcal{O}(n+m_R)$. Completion of the graph by computing all-pairs shortest paths can be done in time $\mathcal{O}(nm_E + n^2 \log n) = \mathcal{O}(n^2 \log n)$ [CLR90, AMO93]. All other steps can be carried out in time $\mathcal{O}(n^2)$ where again the algorithm from [Tar77] is employed for computing a minimum weight in-tree. □

6.6 Hardness Results

Since the SOURCE-DARP generalizes the DARP, it follows from the hardness result in [FG93] that SOURCE-DARP is NP-hard to solve even on trees. We show that this hardness continues to hold even if the source-order $\prec$ is a total source-order. We can also strengthen the hardness result of [FG93] and show that the DARP is hard on caterpillar graphs. This is in contrast to an application of DARP in the next section where caterpillar graphs naturally arise.



A caterpillar graph.

A *caterpillar graph* is a special case of a tree, consisting of a path, called the *backbone* of the caterpillar, and additional vertices of degree one, called the *feet* of the caterpillar. The edges between vertices on the path and feet are called *hairs*. We restrict the class of caterpillars further to those graphs where no two hairs are incident, i.e., the nodes on the backbone are of maximum degree 3.

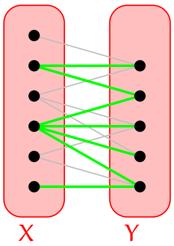
caterpillar graph

backbone

feet

hairs

Theorem 6.30 *The* DARP *and the* SOURCE-DARP *are* NP*-hard to solve even on caterpillar graphs. This result continues to hold, if the transportation requests are restricted to have sources and targets only in the feet of the caterpillar. Furthermore, all hardness results for* SOURCE-DARP *remain true if the source-order is restricted to be total.*



Steiner tree (thick edges) in a bipartite graph spanning all vertices in Y.

Proof: We first address the hardness of the DARP. The hardness is shown by a reduction from the Steiner tree problem on bipartite graphs, BIPARTITE-STP. An instance of BIPARTITE-STP consists of a bipartite graph $H = (X \cup Y, F)$ and a nonnegative number $k \leq |F|$. It is NP-complete to decide whether there exists a subtree of H that spans all the vertices in Y and has at most k edges, see [GJ79, Problem ND12].

One can make two assumptions on H without loss of generality: First, each vertex in Y has degree at least two. Otherwise, for a vertex $y \in Y$ with degree one, there is no other choice than including the unique edge incident to y in the Steiner tree. Second, H is connected. Otherwise, there is either a connected component containing Y, or H can not contain a Steiner tree for the set Y.

Let $H = (X \cup Y, F)$ be an instance of BIPARTITE-STP. We create an instance $I = (G = (V, E, R), d, o)$ of DARP. The construction of the graph G is illustrated in Figure 6.4. Start with a graph consisting of $|F|$ pairwise non-incident edges and their $2|F|$ endpoints. For each edge $[x, y] \in F$ where $x \in X$ and $y \in Y$, choose a yet unlabeled edge in G and label its endpoints by x and y, respectively. Now create the backbone of the caterpillar graph by inserting a path of $|F| - 1$ additional edges. These backbone edges are inserted between nodes with labels from X in such a way that for each $x \in X$ the graph induced by the nodes labeled x in G is a connected path, denoted by $P(x)$.

Set the weight of a backbone edge with two endpoints sharing the same label to 0, and the weight of backbone edges with two different labels to some large number $M := 2|F| + 1$. Set the weight of a hair to 1.

The arc set R is constructed as follows: For $y \in Y$ denote by $S(y)$ the set of foot vertices in G labeled with y. Then, for each $y \in Y$, choose a directed simple cycle connecting $S(y)$ and add its arc set to R. Finally, the origin o of the server is chosen to be the source of an arbitrary arc in R. Observe that by construction the graph $G[R]$ is degree-balanced. It consists of the set of connected components $\{ S(y) : y \in Y \}$. Each of the components is strongly connected and Eulerian.

Let $D = \sum_{r \in A} d(r)$. We claim that H contains a Steiner tree with at most k edges if and only if there is a feasible solution to the instance I of the DARP with cost at most $D + 2k$.

Suppose that T is a Steiner tree in H with at most k edges connecting the vertices in Y. We construct a multiset S of arcs, $S \sqsubset E^{\rightleftarrows}$, such that $G[R + S]$

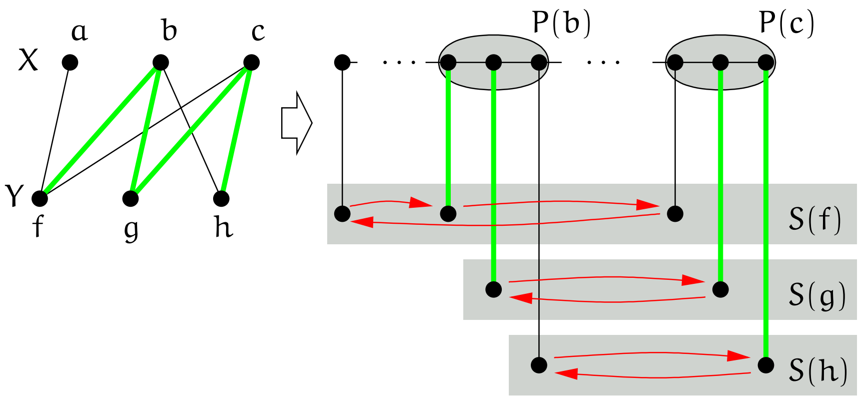


Figure 6.4 Transformation of the BIPARTITE-STP into the DARP on a caterpillar. The thick lines represent a feasible solution and its transformation.

is Eulerian and contains o and $d(R+S) \leq C+2k$: For each $x \in X$ spanned by T, add all arcs from the set $P(x)^{\rightleftarrows}$ to S (these arcs are of cost 0). For each edge $[x,y] \in T$, add a pair of antiparallel arcs between the endpoints of the unique hair labeled $[x,y]$ to the multiset S (these arcs are of cost 1 each). By this construction, graph $G[R+S]$ is degree-balanced. Since T was spanning Y and connected, $G[R+S]$ is strongly connected and hence Eulerian. Further, $d(R+S) \leq D+2k$. This shows the first direction.

Assume conversely that S, $S \sqsubset E^{\rightleftarrows}$, is a feasible solution for instance I, and its cost satisfies $d(R+S) \leq D+2k$. Then $G[R+S]$ is Eulerian and contains o. The set S can not contain any arc of cost M, since otherwise $d(R+S) = d(R)+d(S) \geq D+M = D+2|F|+1 > D+2k$.

We now define a subgraph T of H as follows: For each $x \in X$, $y \in Y$, if S contains (at least one copy) of an arc between a vertex in $P(x)$ and $S(y)$ in arbitrary direction, then the edge $[x,y]$ is included in T. Since $G[R]$ and $G[R+S]$ are degree-balanced and $S \cap R = \varnothing$, we can decompose S into arc disjoint cycles. Since $S \sqsubset E^{\rightleftarrows}$ and $G[E]$ is a tree it follows that $r \in S$ implies that the inverse arc r^{-1} must also be contained in S. Hence, T consists of at most $d(S)/2 = k$ edges.

It remains to show that T is connected and spans the vertices in Y. To this end, let y_1 and y_2 be two arbitrary vertices from Y. Since y_1, y_2 are incident with arcs from R, and $G[R+S]$ is strongly connected, there is a directed path $(r_1, \ldots, r_t)$ from a node labeled y_1 to one labeled y_2 in $G[R+S]$. The node labels along this path change only when r_i is of type $r_i = (x,y)$ or $r_i = (y,x)$ for suitable $x \in X$ and $y \in Y$. By construction, tree T contains edge $[x', y']$ in this case. Hence, T connects y_1 and y_2. This completes the proof of the hardness results for the DARP.

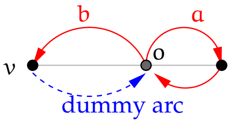
Since $|R_v| \leq 1$ for all nodes v in the construction used above, by choosing $\prec$ to be the empty relation the hardness for the SOURCE-DARP immediately follows. □

6.7 Extensions

6.7.1 Non-Closed Schedules

All our results in this chapter have been derived for the case where the server has to return to the origin o at the end (this corresponds to the closed makespan in the online situation). In this section we describe how to extend our algorithmic results to the problem SOURCE-DARP$^{\text{open}}$ where the server needs not return to o (which corresponds to the non-closed makespan). We denote by DARP$^{\text{open}}$ the problem SOURCE-DARP$^{\text{open}}$ without precedence constraints.

Suppose that ALG is a ρ-approximation algorithm for the SOURCE-DARP. We show how to construct a new approximation algorithm for the SOURCE-DARP$^{\text{open}}$. The essence of the construction is that we try to "guess" the endpoint of an optimal (or approximately optimal) non-closed schedule.



Given the precedence constraint $a \prec b$ the optimal solution $S_v = \emptyset$ for the instance I_v resulting from adding the dummy arc (v, o) (dashed) is not feasible for the original instance I of the SOURCE-DARP$^{\text{open}}$.

Let $I = (G = (V, E, R), d, o, \prec)$ be an arbitrary instance of the SOURCE-DARP$^{\text{open}}$. For $v \in V \setminus \{o\}$ denote by I_v the instance of the SOURCE-DARP resulting from I by adding an additional dummy arc $a_v = (v, o)$ to R and making a_v a maximal element with respect to the source-order, that is, $r \prec a_v$ for all $r \neq a_v$ emanating from v. Also, denote by $I_o := I$ the instance I of the SOURCE-DARP$^{\text{open}}$ interpreted as an instance of the SOURCE-DARP.

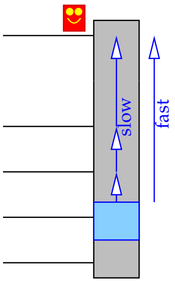
We run ALG on the $n = |V|$ instances I_v $(v \in V)$ of the SOURCE-DARP. Let S_v be the corresponding solution produced by ALG on input I_v. It is easy to see that for each v the multiset $S_v + \{a_v\}$ is a feasible solution for the original instance I of the SOURCE-DARP$^{\text{open}}$. In case that the original instance contained no precedence constraints (that is, it was actually an instance of the DARP$^{\text{open}}$), even each set S_v is already feasible for the original instance (unfortunately this property can not be guaranteed in general in the case of the SOURCE-DARP$^{\text{open}}$).

An easy calculation shows that if we choose the multiset $S_v + \{a_v\}$ with the least cost, this yields a solution for the original instance of cost no more than $2\rho\,\text{OPT}(I)$. In case of the DARP$^{\text{open}}$, that is, when there are no precedence constraints at all, the multiset S_v with least cost is $(2\rho - 1)$-approximative. This yields the following theorem:

Theorem 6.31 *If there is a ρ-approximation algorithm for the* SOURCE-DARP, *then there exists a 2ρ-approximation algorithm for the* SOURCE-DARP^{open}. *If there is a ρ-approximation algorithm for the* DARP, *then there exists a $(2\rho - 1)$-approximation algorithm for the* DARP^{open}. □

6.7.2 Start- and Stop-Penalties

In this section we show how to extend our results to the case when there are start- and stop-penalties for the server, which makes the problem more realistic in view of applications. In elevator systems the time that the elevator needs to accelerate or decelerate in order to pick up or deliver its load can usually not be neglected. Thus, it is natural to penalize each stop and start of the server on its route.



Stopping at each empty floor and accelerating again is slower than going to the top floor directly.

Start- and stop-penalties do not introduce a completely new situation: the problem with penalties can be modeled as a DARP on a slightly larger graph. However, the penalties change the complexity of the problem when restricted to the simplest class of graphs. We prove that the problem with penalties becomes NP-hard even on paths without any precedence constraints, which contrasts with the polynomial solvability of the problem without penalties.

The dial-a-ride problem with penalties is denoted by PENALTY-SOURCE-DARP. An instance of PENALTY-SOURCE-DARP consists of the same input as an instance of SOURCE-DARP, but additionally specifies penalty functions p^+ and p^- on the set of vertices, where $p^+(v)$ is the time penalty for starting from a vertex and $p^-(v)$ is the penalty for stopping at a vertex. The goal is to find a closed oriented walk serving all requests, such that the cost of the walk plus the cost of starting and stopping is minimized.

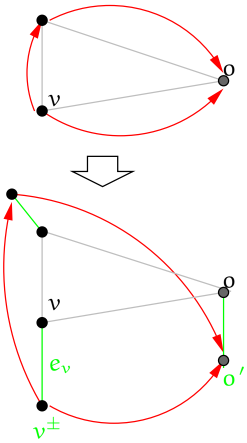
As in the preceding sections, we formulate PENALTY-SOURCE-DARP as a graph augmentation problem. To do this in in a meaningful way, we have to allow augmenting arcs from $V \times V$ and not just from $E^{\rightleftarrows}$, since each arc corresponds to a move and incurs a start and a stop penalty. The cost function $d\colon E \to \mathbb{R}_0^+$ is extended by defining the cost of arc (v, w) to be the length of a shortest path from v to w in $G[E]$.

Definition 6.32 (Graph augm. version of PENALTY-SOURCE-DARP) *An instance of the* PENALTY-SOURCE-DARP *consists of the same input as for the* SOURCE-DARP *together with additional penalty functions* $p^+, p^-\colon V \to \mathbb{R}_0^+$ *on the set of vertices* V*. The objective is to find a multiset* S *of arcs,* $S \sqsubset V \times V$*, minimizing the weight*

$$d(R+S) + \sum_{u \in U^+} \deg^+_{R+S}(u) p^+(u) + \sum_{u \in U^-} \deg^-_{R+S}(u) p^-(u)$$

such that $G[R+S]$ *is* $\prec$*-Eulerian with start* o*. Here,* U^+ *is the set of sources of arcs in* $R+S$ *and* U^- *is the set of endpoints of arcs in* $R+S$*.*

In the sequel we show that PENALTY-SOURCE-DARP can be reduced to SOURCE-DARP. We emphasize that the reduction does not preserve all structural properties of the graph. This prevents us from applying all results from the preceding sections directly to PENALTY-SOURCE-DARP.

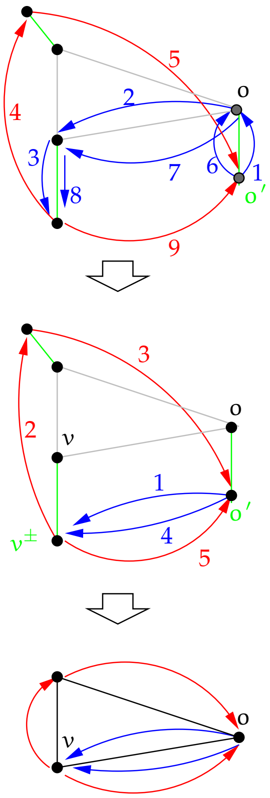


Transformation of an instance of the Penalty-Source-Darp (top) to an instance of the Source-Darp (bottom).

Let $I = (G = (V, E, R), d, o, \prec, p^-, p^+)$ be an arbitrary instance of the Penalty-Source-Darp. We construct an an equivalent instance $I' = (G' = (V', E', R'), d', o', \prec')$ of the Source-Darp on a slightly larger graph (the term "equivalent instance" will be formalized below).

The transformation is accomplished as follows: For each vertex $v \in V$ we add both v and a new vertex $v^{(\pm)}$ to V'. Vertex $v^{(\pm)}$ is used to model starting or stopping at vertex v. The set E' consists of the edges in E and an additional edge e_v between v and $v^{(\pm)}$ for each vertex $v \in V$. The cost of the new edges is $d'(e_v) = (p^+(v) + p^-(v))/2$. The cost function d' coincides with d on the set E. For each arc $r = (u, v) \in R$ we add an arc $a' = (u^{(\pm)}, v^{(\pm)})$ to R' (the arcs in R are not contained in R'). The partial order on the set R' is induced naturally by that on R. Finally, the start vertex o' equals $o^{(\pm)}$.

Lemma 6.33 *Let I be an instance of the Penalty-Source-Darp and I' the instance of the Source-Darp constructed by the above method. Then, I and I' are equivalent in the following sense: Any feasible solution for I' can be transformed into a feasible solution for I of the same cost and vice versa. This transformation can be accomplished in polynomial time.*



A solution for the Source-Darp is transformed into a solution for the original instance of the Penalty-Source-Darp. The numbers on the arcs indicate the order in which they are traversed.

Proof: Let S' be a valid solution for the instance I' of the Source-Darp where S' is an augmenting set of arcs. Let C' be a $\prec$-respecting Eulerian cycle in $G'[R' + S']$ with start o'.

We first construct an auxiliary set M of arcs by traversing C' and replacing all chains of arcs from S' with a single arc from the start vertex of the chain to its end vertex. Notice that all endpoints of arcs in M are contained in $V' \setminus V$. We now construct a solution S by replacing each arc $(u^{(\pm)}, v^{(\pm)})$ by (u, v). It is easy to see that S is in fact a valid solution for I of cost equal to that of S'.

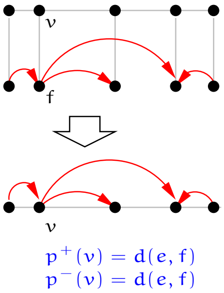
Conversely, let S be a feasible solution for I. We can construct a solution S' for I' with equal cost by adding for each arc (u, v) in S the arc $(u^{(\pm)}, v^{(\pm)})$ to S'.

The time bound is obvious from the construction. □

It follows from the construction that if $G[E]$ is a tree then $G'[E']$ is also a tree. Thus, the previous lemma implies that approximation results for the Source-Darp on trees can be applied directly to the Penalty-Source-Darp on trees. Similarly, approximation results for general graphs carry over immediately. Hence, we obtain the following result:

Corollary 6.34 *The Penalty-Source-Darp can be approximated on trees with performance guarantee $3/2$, and on general graphs with performance guarantee $9/4$.* □

However, if we transform an instance of the PENALTY-SOURCE-DARP where $G[E]$ is a path, this yields an instance of the SOURCE-DARP where $G'[E']$ is a caterpillar graph. This seems unfortunate, since we know from Theorem 6.30 that the SOURCE-DARP is NP-hard to solve on caterpillars. Is there a better transformation? More specific, is PENALTY-SOURCE-DARP on paths still polynomial time solvable?



Any instance of the SOURCE-DARP on a caterpillar can be transformed into an equivalent instance of the PENALTY-SOURCE-DARP on a path.

The caterpillar constructed in the proof of Theorem 6.30 has the property that requests have sources and targets only in the feet of the caterpillar. Actually every instance of the SOURCE-DARP on caterpillars with these properties can be transformed into an equivalent instance of the PENALTY-SOURCE-DARP on a path: Let f be a foot and v be its unique adjacent vertex on the backbone. We replace all arcs from R which are incident with f by corresponding arcs with source or target v. We then remove foot f. The start- and stop-penalties on v are set to the length $d(f, v)$ of the hair between v and the foot f.

It follows by arguments similar to those given in Lemma 6.33 that the constructed instance of the PENALTY-SOURCE-DARP on the path (which corresponds to the former backbone) is in fact an equivalent instance to the instance of the SOURCE-DARP on the caterpillar. Thus, we obtain the following result which contrasts with the polynomial solvability of the SOURCE-DARP on paths:

Theorem 6.35 *The PENALTY-SOURCE-DARP on paths is NP-hard to solve.* □

6.8 Remarks

This chapter presented a natural extension of an offline dial-a-ride problem. We have shown that even in the presence of source-order constraints for the transportation requests the problem can be solved in polynomial time on paths, which generalizes the result of [AK88]. On trees, however, the problem is NP-hard. The application to more realistic elevator systems with non-negligible acceleration of the moving server motivated the formulation of a problem variant with start- and stop-penalties.

Table 6.1 gives an overview over the results obtained in this chapter and the known results from literature for the DARP.

Graph class	DARP	SOURCE-DARP	PENALTY-SOURCE-DARP
Paths	Polynomial time solvable *[AK88]*	Polynomial time solvable *(Theorem 6.20)*	NP-hard *(Theorem 6.35)* Approximable within 3/2 *(Corollary 6.34)*
Trees	NP-hard, even on caterpillars *(Theorem 6.30)* Approximable within 5/4 *[FG93]*	NP-hard, even on caterpillars *(Theorem 6.30)* Approximable within 3/2 *(Theorem 6.29)*	NP-hard Approximable within 3/2 *(Corollary 6.34)*
General Graphs	NP-hard *[FHK78]* Approximable within 9/5 *[FHK78]*	NP-hard Approximable within 9/4 *(Theorem 6.26)*	NP-hard Approximable within 9/4 *(Corollary 6.34)*

Table 6.1: Complexity and approximation results for the SOURCE-DARP and related problems.

7

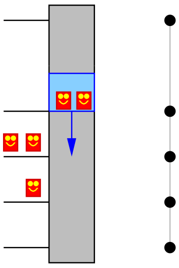
A Capacitated Offline Dial-a-Ride Problem on Paths

In this chapter we continue our study of offline dial-a-ride problems. While in the previous chapter we considered the case of a unit-capacity server (DARP, SOURCE-DARP), in this chapter we are concerned with the capacitated (offline) dial-a-ride problem CDARP where the capacity of the server is arbitrary.

We focus on the CDARP for the special case when the underlying metric space is induced by a simple path. As already mentioned, this setting arises when modelling an elevator system as a dial-a-ride problem.

Unfortunately, the CDARP is NP-hard to solve even on paths, which are arguably one of the simplest types of metric spaces. This is in contrast to the unit-capacity case which we showed to be polynomial time solvable in the previous chapter even under the presence of precedence constraints (see Section 6.3).

This chapter is organized as follows. In Section 7.1 we formally define the problem CDARP. The main result of this chapter is a polynomial time approximation algorithm for the CDARP on paths with performance 3. We present this algorithm in Section 7.2. The proof of the correctness and the performance of the algorithm is divided into several parts. It is contained in Sections 7.3 to 7.5.



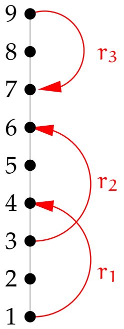
Elevator with capacity 2 and path $G[E]$ representing the transportation network.

Related Work

Guan showed that the CDARP is NP-hard even on paths if the capacity of the server is two [Gua98]. A version of the proof can be found in [Wei00]. The preemptive version is polynomial time solvable on paths [Gua98], but NP-hard even on trees and even if the capacity of the server is one [FG93]. In [CR98] an approximation algorithm for the CDARP on general graphs with performance $\mathcal{O}(\sqrt{C} \log n \log\log n)$ was given, where C denotes the capacity of the server. In the same paper the authors claimed an approximation algorithm with performance 2 for the CDARP on paths but neither the algorithm nor a proof of its performance was presented.

7.1 Problem Definition

Definition 7.1 (Offline Dial-a-Ride Problem CDARP)
An instance of the CDARP *is given by a finite mixed graph* $G = (V, E, R)$ *with edge weights given by* $d\colon E \to \mathbb{R}_0^+$, *a capacity* $C \in \mathbb{N}$ *and start position* $o \in V$ *for the server. The goal of the* CDARP *is to find a closed transportation schedule starting at* o *with minimum length.*



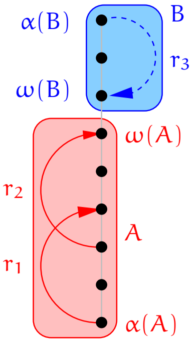
"Elevator-visualization" of a mixed graph given in an instance of the CDARP.

We denote the optimum cost of a feasible transportation for instance I of CDARP by $\text{OPT}(I)$. We drop the reference to I provided that no confusion can occur.

In this chapter we are concerned with the situation that $G[E]$ is a simple path and o is one of its end points. For notational convencience we assume that $V = \{1, \ldots, n\}$, $E = \{[i, i+1], i = 1, \ldots, n-1\}$ and that the initial position o of the server is 1.

In our presentation we imagine the simple path $G[E]$ as a vertical line with vertex 1 being the lowest and vertex n the highest vertex. This is motivated by the background of the elevator application.

In all what follows we assume without loss of generality that vertex n is essential, that is, at least one arc of R is incident with vertex n. Notice that this implies $\text{OPT} \geq 2 \cdot d(1, n)$, since any feasible transportation must visit vertex n and return to its starting position at $o = 1$. We also assume that the number of requests $|R|$ is larger than C. If this is not the case, then a single upward and downward motion of the server from 1 to n and backwards can be used to obtain a transportation of cost $2 \cdot d(1, n)$ which by the previous comment must then be optimal.



Upward requests $R^{\uparrow} = \{r_1, r_2\}$ (solid arcs) and downward requests $R^{\downarrow} = \{r_3\}$ (dashed arcs).

7.2 An Approximation-Algorithm

Let $G = (V, E, R)$ be a mixed graph given in an instance of CDARP, where $G[E]$ is a path. We define the set of *upward requests* and *downward requests* as follows:

$$R^{\uparrow} = \{ r \in R : \alpha(r) < \omega(r) \}$$
$$R^{\downarrow} = \{ r \in R : \alpha(r) > \omega(r) \}.$$

Let A be a subset of R which is either completely contained in $R^{\uparrow}$ or $R^{\downarrow}$. In this case we let

$$\alpha(A) = \min\{ \alpha(r) : r \in A \} \quad \text{and} \quad \omega(A) = \max\{ \omega(r) : r \in A \}, \text{if } A \subseteq R^{\uparrow}$$
$$\alpha(A) = \max\{ \alpha(r) : r \in A \} \quad \text{and} \quad \omega(A) = \min\{ \omega(r) : r \in A \}, \text{if } A \subseteq R^{\downarrow}.$$

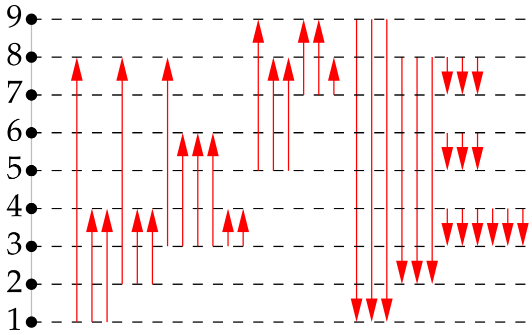
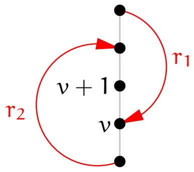


Figure 7.1
An instance of the CDarp on a path G with 9 vertices, and a set of requests R. For a less cluttered display the directed arcs corresponding to R are drawn parallel to the path G[E].

In the sequel we have to refer to those request arcs r which have to be transported over a specific edge $[v, v+1]$. To facilitate the presentation we define the notions of covers and segments.

cover



Both requests, r_1 and r_2, cover edge $[v, v+1]$.

Definition 7.2 (Cover)
Let $e = [v, v+1]$ *be an edge in the undirected graph* $G[E]$*. A request arc* $r = (\alpha(r), \omega(r)) \in R$ *is said to* ***cover*** e *if* $\alpha(r) \leq v$ *and* $\omega(r) \geq v+1$ *or* $\alpha(r) \geq v+1$ *and* $\omega(r) \leq v$*. A set of arcs* R' ***covers edge*** e *if at least one arc from* R' *covers* e.

Definition 7.3 (Segment)
Let $R' \subseteq R$*. A* ***segment*** *of* R' *is an inclusionwise maximal subset* $S \subseteq R'$ *with the property that the set of edges from* G *covered by* S *forms a connected subpath of* G.

Observe that the segments of a set $R' \subseteq R$ form a partition of R'.

We are now ready to state our approximation algorithm. The main algorithm FIND-AND-PASTE is shown in Algorithm 7.1. This algorithm uses the subroutines FIND-UP-ARCS presented in Algorithm 7.2 and FIND-DOWN-ARCS displayed in Algorithm 7.3.

Before we prove the correctness and the performance of the algorithm, we illustrate the execution of FIND-AND-PASTE on a small example. Consider the instance of CDarp shown in Figure 7.1. In this instance $G[E]$ is a path with 9 vertices and there are $|R| = 36$ requests to be transported by a server of capacity $C = 3$.

First the requests in $R^{\uparrow}$ are processed. At the beginning, the set $R^{\uparrow}$ consists only of one segment $S = R^{\uparrow}$. Now, FIND-UP-ARCS is called $C = 3$ times producing a total of six sets M_1^j, M_2^j, $j = 1, 2, 3$. Two sequences of upward moves U_1 and U_2 are constructed for the sets $X_1^{\uparrow} = M_1^1 + M_1^2 + M_1^3$ and $X_2^{\uparrow} = M_2^1 + M_2^2 + M_2^3$. Corresponding arcs are added to the auxiliary instance of the Darp which will be used to chain all moves in a final paste step (see Figure 7.2 for an illustration).

All requests transported in the upward moves in U_1 and U_2 are removed from $R^{\uparrow}$. Since $R^{\uparrow} \neq \emptyset$, the subroutine FIND-UP-ARCS is called again $C = 3$

```
**Input:** An instance of CDARP where $G[E]$ is a path.
1  Compute the set of upward requests $R^{\uparrow}$ and downward requests $R^{\downarrow}$.
2  $A := \emptyset$        {Each arc in the multiset $A$ corresponds to a transportation.
                           In a final "paste"-step these transportations will be pasted
                           together by considering the DARP instance $(V, E, A)$.}
3  while $R^{\uparrow} \neq \emptyset$ do
4     Let $S$ be a segment of $R^{\uparrow}$
5     Let $L := \alpha(S)$ and $U := \omega(S)$.
6     Call Algorithm FIND-UP-ARCS$(S)$ $C$ times with $v_L := L$ and $v_U := U$ to
      obtain $2C$ sets of arcs $M_1^j$ and $M_2^j$, $j = 1, \ldots, C$.
               {Notice that $S$ shrinks with each call to FIND-UP-ARCS provided
                it is not yet empty, but the values of $v_L$ and $v_U$ remain fixed.}
7     Set $X_i^{\uparrow} := \biguplus_{j=1}^{C} M_i^j$ for $i = 1, 2$.
8     Construct two sequences of upward moves, $U_1$ and $U_2$, where $U_i$
      $(i = 1, 2)$ transports all objects from $X_i^{\uparrow}$. $U_i$ starts at $\alpha(X_i^{\uparrow})$ and ends
      at $\omega(X_i^{\uparrow})$.
9     $A := A + (\alpha(X_1^{\uparrow}), \omega(X_1^{\uparrow}))$        {Add elements to the multiset $A$.}
10    if $X_2^{\uparrow} \neq \emptyset$ then
11       $A := A + (\alpha(X_2^{\uparrow}), \omega(X_2^{\uparrow}))$
12    end if
13    $R^{\uparrow} := R^{\uparrow} \setminus (X_1^{\uparrow} \cup X_2^{\uparrow})$
14 end while
15 while $R^{\downarrow} \neq \emptyset$ do
16    In the same way as above, call Algorithm FIND-DOWN-ARCS $C$ times
      and construct two sequences $D_1$ and $D_2$ of downward moves from
      the $2C$ sets of downward arcs. Add directed arcs $(\alpha(D_i), \omega(D_i))$ of $D_i$
      $(i = 1, 2)$ to the multiset $A$. Remove the downward arcs form the $2C$
      sets found by FIND-DOWN-ARCS from $R^{\downarrow}$.
17 end while
18 Consider the instance $\Pi$ of the SOURCE-DARP with underlying undi-
   rected graph $G[E]$, request set $A$, empty source-order $\prec$, and start ver-
   tex $o = 1$. Each arc in $A$ corresponds to one sequence of (upward or
   downward) moves constructed above in steps 8 and 16.
               {$\Pi$ contains no precedence constraints and could also be considered
                as an instance of the simpler problem DARP.}
19 Find an optimal solution $T_{\Pi}$ for $\Pi$ in polynomial time with the help of
   Algorithm ALG-PATH from Section 6.3.
20 Chain the sequences of upwards and downwards moves found in
   steps 8 and 16 to a transportation by taking them in the order as the
   corresponding arcs appear in $T_{\Pi}$.
```

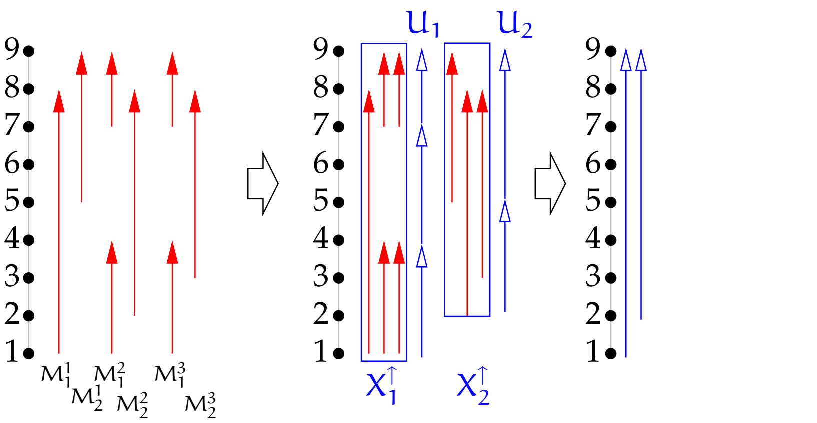
Algorithm 7.1: Algorithm FIND-AND-PASTE

```
Input: A multiset of requests $S \sqsubset R^{\uparrow}$, two vertices $v_L \le v_U$ from $V$
                              { The multiset $S$ is modified by FIND-UP-ARCS
                                        and the modified set is returned. }
Let $H$ be the subpath of $G$ formed by the vertices $v_L, v_L + 1, \dots, v_U$.
if there exists edges $[v, v+1]$ in $H$ with $v_L \le v < v+1 \le v_U$ which are
not covered by any arc from $S$ then
    For each of these edges $[v, v+1]$ add a dummy arc $(v, v+1)$ to $S$.
end if
$M_l := \varnothing$, $M_u := \varnothing$        { Here, $u$ stands for "upper" and $l$ for "lower".
                              We maintain the invariant that $\omega(M_l) \le \omega(M_u)$. }
while $M_u = \varnothing$ or $\omega(M_u) < \omega(S)$ do
    Find a path $P$ in the directed (acyclic) graph $(V, S)$ with

    (7.1)        $\omega(M_l) \le \alpha(P) \le \omega(M_u) < \omega(P)$

    such that $\omega(P)$ is maximum among all those paths. (Here we set
    $\omega(\varnothing) := \alpha(S)$).
    Set $M_l := M_l \cup P$          { Add the arcs from $P$ to the "lower" set $M_l$. }
    $S := S - P$                     { Remove the arcs from $P$ from the multiset $S$. }
    Interchange the sets $M_l$ and $M_u$.
end while
return $M_l$, $M_u$ and the modified multiset $S$.
```

Algorithm 7.2: Algorithm FIND-UP-ARCS



3 calls to FIND-UP-ARCS

instance of DARP

Figure 7.2 The sets $M_1^j, M_2^j, j = 1, 2, 3$ obtained by the first three calls to FIND-UP-ARCS (left), the resulting sequences of upward moves U_1 and U_2 (middle), and the arcs added to the auxiliary instance of the DARP (right).

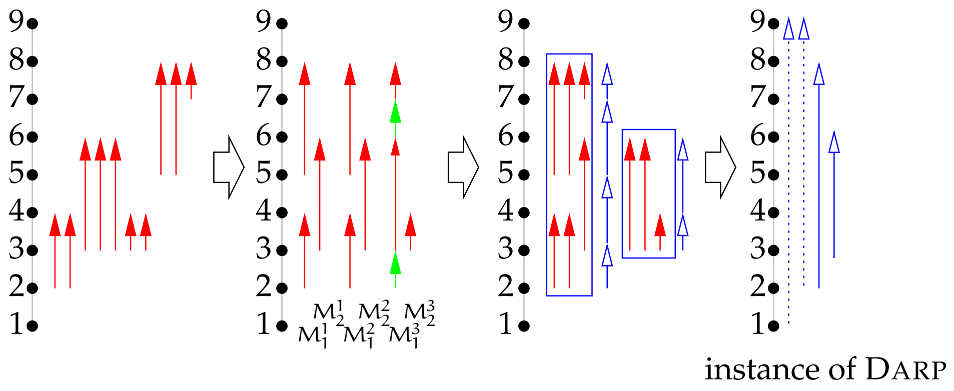


Figure 7.3 Remaining arcs in $R^{\uparrow}$ (left). FIND-UP-ARCS is called again to construct sets of arcs, upward moves and new arcs in the auxiliary instance of the DARP (right). The dotted arcs are the result of the previous iteration.

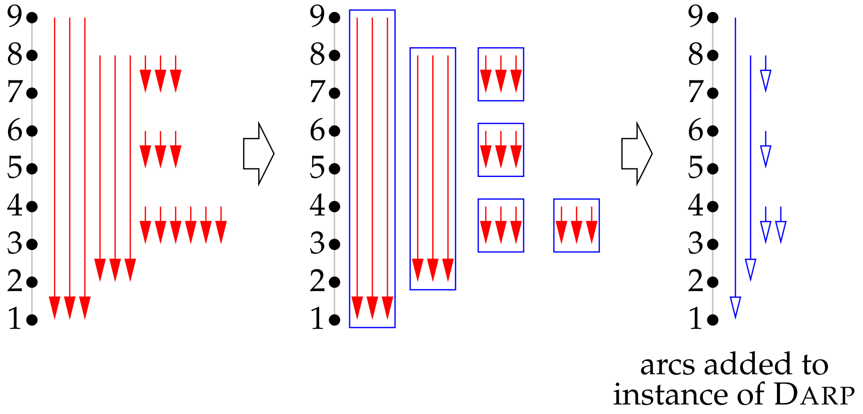


Figure 7.4 The set $R^{\downarrow}$ and moves transporting all objects in $R^{\downarrow}$. The picture on the right shows the arcs added to the instance Π of the DARP.

times. Notice that the modified set $R^{\uparrow}$ still consists of one segment, but now covering only the edges $[v, v+1]$, $v = 2, \ldots, 7$. The result of the second round of calls to FIND-UP-ARCS is illustrated in Figure 7.3. Notice that in this round a new situation arises. After the second call to FIND-UP-ARCS in this round, the residual set $R^{\uparrow}$ does not cover a connected path anymore. Hence, dummy arcs $(2,3)$ and $(6,7)$ are added (green arcs in Figure 7.3). The dummy arcs are not included in the sequences constructed in Step 8.

After the second round of calls to FIND-UP-ARCS there remains only one more request $(3,4)$. This arc will be transported by a single move. Now, the downward requests $R^{\downarrow}$ are processed in a similar way with the help of FIND-DOWN-ARCS. This results in the sequences of moves shown in Figure 7.4.

Now FIND-AND-PASTE constructs an instance Π of the SOURCE-DARP and uses Algorithm ALG-PATH from Section 6.3 to find an optimal solution T_{Π} for Π. The instance Π and its solution are shown in Figure 7.5. The dotted arcs in the solution correspond to balancing arcs added by the algorithm.

The final solution found by FIND-AND-PASTE is displayed in Figure 7.6.

We proceed to establish the performance of FIND-AND-PASTE in two steps. In a first step we derive some useful properties of the subroutines FIND-UP-ARCS and FIND-DOWN-ARCS. In the second step we analyze the

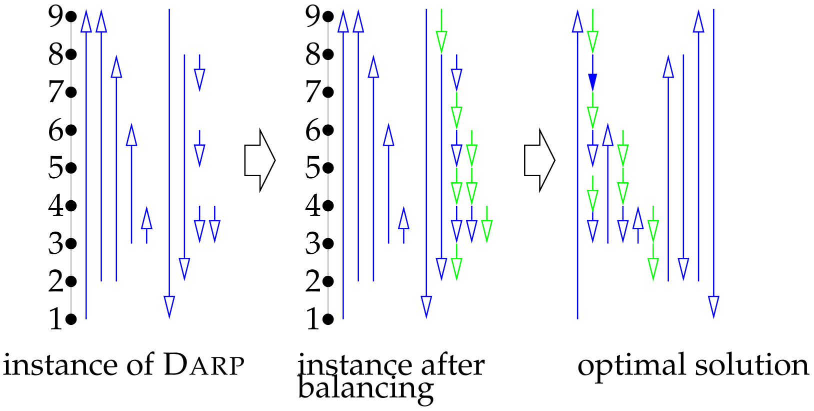


Figure 7.5 The instance Π of the DARP constructed by FIND-AND-PASTE (left), the balanced instance (middle) and the optimal solution (right).

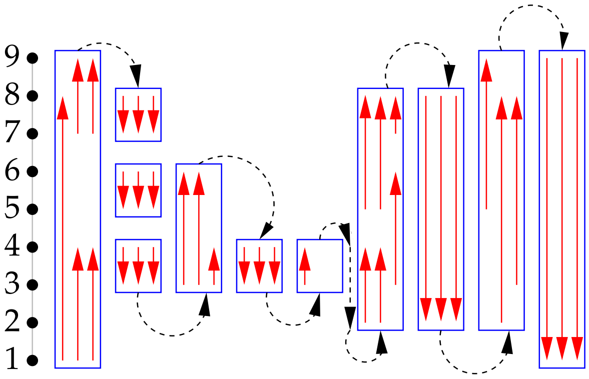


Figure 7.6 The final solution found by FIND-AND-PASTE. The upward and downward movements are shown as rectangles. They transport the requests displayed within the rectangles and are executed in the order indicated by the dashed arcs.

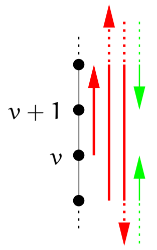
overall interaction between the algorithms.

7.3 Properties of the Subroutines

We start by analyzing properties of the subroutine FIND-UP-ARCS. At the beginning, both sets, the "lower set" M_l and the "upper set" M_u are empty. In each iteration of FIND-UP-ARCS a directed path is added to the "lower set" M_l satisfying condition (7.1) which means in particular that $\omega(M_l)$ increases to a value strictly larger than $\omega(M_u)$. This ends the current iteration after which the roles of M_u and M_l are exchanged. Thus, we have the following observation:

Observation 7.4 *At the beginning of each iteration of* FIND-UP-ARCS *the relation* $\omega(M_l) \leq \omega(M_u)$ *holds true.* □

However, it is not trivial that in each iteration of FIND-UP-ARCS a path with the desired property (7.1) exists. Our goal is to show that in fact such a path can be determined in each iteration. To this end we investigate the structural properties which such a potential path must satisfy and then use them to derive the existence of the path.



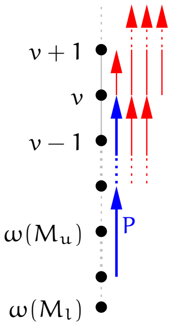
Requests covering edge $[v, v+1]$ are indicated via thick lines. In the example $\mu_v(D) = 3$.

Definition 7.5 (Number μ_v of Covering Requests)
For a vertex $v \in V \setminus \{n\}$ and a subset $D \subseteq R$ we define $\mu_v(D)$ to be the number of request arcs in D that cover the edge $[v, v+1]$. We also set $\mu_n(D) := 0$. We omit the set D if it is clear from the context.

Lemma 7.6 *Suppose that Algorithm* FIND-UP-ARCS *is called with a nonempty set $S \subseteq R^{\uparrow}$ and that P is a path which is found in Step 7 of* FIND-UP-ARCS. *Then this path P satisfies:*

$$\mu_{\omega(P)-1}(S) > \mu_{\omega(P)}(S)$$

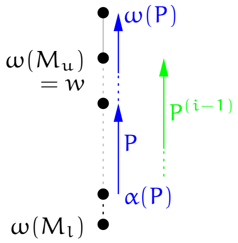
for the current set S when P is added to M_l in Step 8.



If $\mu_{v-1}(S) \leq \mu_v(S)$, then the path P (thick lines) could be extended.

Proof: Suppose that the claim were not true for some path P. Let $v = \omega(P)$. Since $\mu_{v-1}(S) \leq \mu_v(S)$, it follows that for any arc ending in v there must be at least one arc from S emanating from v. However, P ends in v and thus we could extend P by at least one arc from S which starts in v. This contradicts the property that P was chosen in such a way that $\omega(P) = v$ is maximum. □

Lemma 7.7 *Suppose that Algorithm* FIND-UP-ARCS *is called with a nonempty set $S \subseteq R^{\uparrow}$. Then in any iteration of Step 7 there exists a path P with the required properties.*



Properties claimed for the path $P = P^{(i)}$ to be found in the ith iteration.

Proof: We show the claim by induction on the number of iterations. In the first iteration, we are in the situation that $M_u = M_l = \varnothing$. Hence, $\omega(M_u) = \omega(M_l) = \alpha(S)$ and condition (7.1) reduces to

$$\alpha(S) = \alpha(P) < \omega(P). \tag{7.2}$$

Clearly, there must be a path starting at $\alpha(S)$, since S is nonempty. Any such path satisfies (7.2).

Assume now that we have reached the ith iteration ($i \geq 2$) and for all previous iterations it was possible to find a path. Let $P^{(i-1)}$ be the path found in the previous iteration $i-1$. To shorten notation we set $w := \omega(P^{(i-1)})$. Let $S^{(i-1)}$ and $S^{(i)}$ denote the set S at the beginning of iteration $(i-1)$ and i, respectively. Notice that for the set M_u at the beginning of iteration i we have $\omega(M_u) = w$. Thus (7.2) can be restated as

$$\omega(M_l) \leq \alpha(P) \leq w < \omega(P). \tag{7.3}$$

At the beginning of the first iteration we have $\mu_v(S^{(1)}) \geq 1$ for all $\alpha(S) \leq v < \omega(S)$ by Step 3. Since in all previous iterations we have removed only such arcs from S that end in vertices $u \leq w$, it follows that

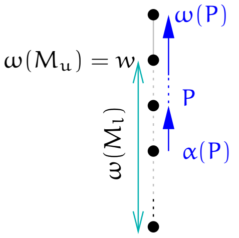
$$\mu_u(S^{(i-1)}) = \mu_w(S^{(i)}) \quad \text{for all } u \geq w. \tag{7.4}$$

By Lemma 7.6 we have $\mu_{w-1}(S^{(i-1)}) > \mu_w(S^{(i-1)})$. Together with (7.4) we obtain

$$\mu_{w-1}(S^{(i)}) \geq \mu_w(S^{(i)}) \geq 1. \tag{7.5}$$

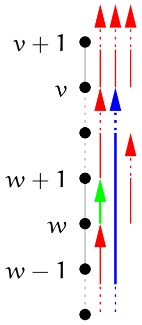
Our first step is to construct a path P formed by arcs from $S^{(i)}$ such that

$$\alpha(P) \leq \omega(M_u) = w < \omega(P). \tag{7.6}$$



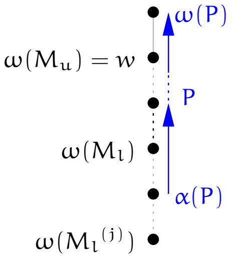
Path P constructed as a first step. The position of $\omega(M_l)$ is neglected.

To this end, we distinguish two cases. If there exists an arc in $S^{(i)}$ which starts in w, then the path consisting of this single arc already satisfies (7.6). In the second case, no arc from $S^{(i)}$ emanates from w. However, since $\mu_w(S^{(i)}) \geq 1$ by (7.5) in this case there must be an arc $r \in S^{(i)}$ with $\alpha(r) < w < \omega(r)$. Again, the path formed by this arc satisfies condition (7.6).



Since $\mu_w(S^{(i)}) \geq 1$, there must exist already a path formed by a single arc (cases indicated by thick highlighted lines) which satisfies the desired properties.

We now show that for any path P satisfying (7.6), we have in fact that $\omega(M_l) \leq \alpha(P)$ which proves the claim of the lemma. If $M_l = \emptyset$ then there is nothing to show. Hence assume that $M_l \neq \emptyset$. Suppose that $\omega(M_l) > \alpha(P)$. It follows that in one of the previous iterations $j < i$ the path P satisfied $\alpha(P) \geq \omega({M_l}^{(j)})$ for the then current version ${M_l}^{(j)}$ of M_l. But this means that P met all the conditions required in Step 7. Thus, the path chosen in iteration j was not maximum with respect to its end vertex. This is a contradiction. □



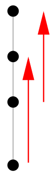
If $\alpha(P) < \omega(M_l)$, then P would have been eligible in a previous iteration $j < i$.

Corollary 7.8 *Suppose that* FIND-UP-ARCS *is called with a nonempty set* $S \subseteq R^{\uparrow}$. FIND-UP-ARCS *terminates after at most* $n-1$ *iterations with arc sets* M_1 *and* M_2 *such that for each edge* e *covered by* S *there exists at least one and at most two arcs in* $M_1 \cup M_2$ *covering* e. *Moreover, all dummy arcs added in Step 3 are contained in* $M_1 \cup M_2$.

Proof: The property that FIND-UP-ARCS terminates after no more than $n-1$ iterations follows from Lemma 7.7 and the fact that in each iteration $\omega(M_u)$ increases strictly.

We now consider the covering property. By Lemma 7.7 the path found in the first iteration starts at $\alpha(S)$. It is easy to show by induction that after each iteration all edges $[v, v+1]$ with $v < \omega(M_u)$ are covered. Hence, at termination all edges are covered at least once.

The arcs in M_1 do not overlap (where we call r and r′ overlapping if $\alpha(r) < \alpha(r') < \min\{\omega(r), \omega(r')\}$ or vice versa) and neither do those in M_2. Hence we can conclude that any edge is covered by at most two arcs, that is at most one from M_1 and at most one from M_2. If a dummy arc $(v, v+1)$ was added in Step 3 then by construction it is the only arc covering edge $[v, v+1]$. Since we have shown that each edge is covered by the arcs in $M_1 \cup M_2$ it follows that the dummy arc must be contained in $M_1 \cup M_2$. □



Two overlapping arcs.

We close this section by commenting on Algorithm FIND-DOWN-ARCS which is needed in Step 16 of the main Algorithm FIND-AND-PASTE. FIND-DOWN-ARCS works on $R^{\downarrow}$ in the analogous way as FIND-UP-ARCS processes $R^{\uparrow}$. Basically the only difference is that the sets M_1 and M_2 "grow downwards" from $\alpha(S)$ to $\omega(S)$ instead of "growing upwards" as in FIND-UP-ARCS.

```
Input:  A multiset of requests $S \sqsubset R^{\downarrow}$, two vertices $v_L \leq v_U$ from $V$
 1 Let $H$ be the subpath of $G$ formed by the vertices $v_L, v_L + 1, \dots, v_U$.
 2 if there exists edges $[v, v+1]$ in $H$ with $v_L \leq v < v+1 \leq v_U$ which are
   not covered by any arc from $S$ then
 3     For each of these edges $[v, v+1]$ add a dummy arc $(v+1, v)$ to $S$.
 4 end if
 5 $M_l := \varnothing$, $M_u := \varnothing$
 6         { We maintain the invariant that $\omega(M_l) \leq \omega(M_u)$. Here, $u$ stands for
   "upper" and $l$ for "lower". }
 7 while $M_l = \varnothing$ or $\omega(M_l) > \omega(S)$ do
 8     Find a path $P$ in the directed (acyclic) graph $(V, S)$ with

       (7.7)          $\omega(M_u) \geq \alpha(P) \geq \omega(M_l) > \omega(P)$

       such that $\omega(P)$ is minimum among all those paths. (Here we set
       $\omega(\varnothing) := \alpha(S)$).
 9     Set $M_u := M_u \cup P$          { Add the arcs from $P$ to the "upper" set $M_u$. }
10     $S := S - P$                     { Remove the arcs from $P$ from the multiset $S$. }
11     Interchange the sets $M_l$ and $M_u$.
12 end while
13 return $M_1$, $M_2$ and the modified multiset $S$.
```

Algorithm 7.3: Algorithm FIND-DOWN-ARCS

The following property of FIND-DOWN-ARCS can be proven analogously to the corresponding results about FIND-UP-ARCS:

Lemma 7.9 *Suppose that* FIND-DOWN-ARCS *is called with a nonempty set* $S \subseteq R^{\downarrow}$. FIND-DOWN-ARCS *terminates after at most* $n - 1$ *iterations with arc sets* M_1 *and* M_2 *such that for each edge* e *covered by* S *there exists at least one and at most two arcs in* $M_1 \cup M_2$ *covering* e. *Moreover, all dummy arcs added in Step 3 are contained in* $M_1 \cup M_2$. □

7.4 Correctness and Running Time of the Algorithm

It is straightforward to see that FIND-AND-PASTE delivers a feasible solution provided it terminates. We argue that FIND-AND-PASTE in fact terminates after a finite number of steps (and hence outputs a feasible solution). Denote by $n := |V|$ the number of vertices in the input graph and by $m_R := |R|$ the number of requests specified in the instance.

For any vertex v which is contained in a segment S in Step 6, the value μ_v decreases strictly by a single call to FIND-UP-ARCS provided it is greater than zero since by Corollary 7.8 at least one arc from $M_1 \cup M_2$ covers $[v, v+1]$. Since in Step 6 FIND-UP-ARCS is called C times in iteration k we conclude that

$$\mu_v(R^{\uparrow(k+1)}) \leq \max\{\mu_v(R^{\uparrow(k)}) - C, 0\}. \tag{7.8}$$

Here $R^{\uparrow(k)}$ denotes the set $R^{\uparrow}$ at the beginning of the kth iteration of the **while**-loop in FIND-AND-PASTE enclosing Step 6. As an immediate consequence of (7.8) we get that the **while**-loop is executed $\mathcal{O}(m_R/C)$ times. After this $R^{\uparrow}$ must be empty and hence all upward requests are contained in upward moves. Analogous arguments apply to the downward requests and downward moves constructed in the second **while**-loop. It now follows that FIND-AND-PASTE must in fact terminate after a finite number of steps.

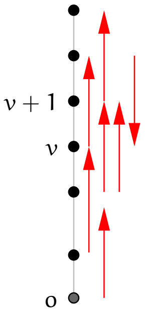
We now address the running time of FIND-AND-PASTE. By the arguments given above, FIND-UP-ARCS and FIND-DOWN-ARCS are called $\mathcal{O}(m_R)$ times. Since ALG-PATH needs time $\mathcal{O}(\min\{(m_R+n)\log n, n^2\})$ (see Section 6.3), this yields a total running time of $\mathcal{O}(m_R \cdot \text{Time}_{\text{find}} + \min\{(m_R+n)\log n, n^2\})$, where $\text{Time}_{\text{find}}$ denotes the effort for a single call to FIND-UP-ARCS or FIND-DOWN-ARCS. Since the path-computation in FIND-UP-ARCS/FIND-DOWN-ARCS can be accomplished in $\mathcal{O}(n+m_R)$-time by depth-first-search (see for example [CLR90]) and the **while**-loop in FIND-UP-ARCS/FIND-DOWN-ARCS is executed $\mathcal{O}(n)$ times, we obtain the following result:

Theorem 7.10 FIND-AND-PASTE *computes a feasible solution for an instance of* CDARP *in time* $\mathcal{O}(nm_R(n+m_R)$. □

7.5 Proof of Performance

We are finally going to establish the performance of our algorithm, that is, the property that given any instance of the CDARP Algorithm FIND-AND-PASTE finds a solution of cost at most 3 OPT.

In fact, we are going to show a stronger result, namely that the cost of the solution found by FIND-AND-PASTE is at most thrice the value of a *lower bound* on OPT, called the *flow bound*.



For capacity $C = 2$, $\lambda[v, v+1] = \max\{\lceil \frac{3}{2} \rceil, \lceil \frac{1}{2} \rceil\} = 2$.

Notice that $\max\{\lceil \mu_v(R^{\uparrow})/C \rceil, 1\}$ is a lower bound on the number of times any feasible transportation must traverse edge $[v, v+1]$ in the direction from vertex v to $v+1$. Similarly, $\max\{\lceil \mu_v(R^{\downarrow})/C \rceil, 1\}$ is a lower bound any transportation must traverse $[v, v+1]$ in direction from $v+1$ to v.

Definition 7.11 (Flow Bound)
Let I *be any instance of the* CDARP *such that* $G[E]$ *is a path and* o *is one of the end vertices of* $G[E]$. *For edge* $e = [v, v+1]$ *of* G *we define*

$$\lambda[v, v+1] := \max\left\{\lceil \mu_v(R^{\uparrow})/C \rceil, \lceil \mu_v(R^{\downarrow})/C \rceil, 1\right\}$$

The value

$$LB_{flow}(I) := 2 \sum_{1 \le v \le n-1} \lambda[v, v+1] \cdot d(v, v+1)$$

is called the ***flow bound*** *for instance* I.

flow bound

Since any feasible transportation always returns to the start point, it follows that in fact the flow bound LB_{flow} is a lower bound on the optimal solution cost:

Observation 7.12 *For any instance* I *of* CDARP *the inequality*

$$\text{OPT}(I) \ge LB_{flow}(I)$$

holds true. □

One ingredient for bounding the cost of the solution found by FIND-AND-PASTE lies in a closer look at Algorithm ALG-PATH from Section 6.3 for solving the SOURCE-DARP on paths. We have to consider the situation occurring in the call issued by FIND-AND-PASTE when there are no precedence constraints. In this case ALG-PATH essentially reduces to the algorithm for DARP on paths presented in [AK88].

For convenience we briefly recall the actions taken by ALG-PATH. Let $G = (V, E, A)$ be the mixed graph given in the instance Π of SOURCE-DARP. ALG-PATH first "balances" the graph (V, A) by adding additional "balancing" arcs B such that for any edge $[v, v+1]$ the number of upward arcs from $A \cup B$ covering $[v, v+1]$ equals the number of downward arcs from $A \cup B$ covering $[v, v+1]$. This implies that in the graph $(V, A \cup B)$ each vertex $v \in V$ satisfies $\deg^{+}_{A \cup B}(v) = \deg^{-}_{A \cup B}(v)$. In a second step the algorithm adds a set N of "connecting arcs" of minimum weight such that $(V, A \cup B \cup N)$ is strongly connected and the degree-balance is maintained. The graph $(V, A \cup B \cup N)$ is Eulerian and a Eulerian cycle in $(V, A \cup B \cup N)$ yields an optimum transportation (see Section 6.3 for details).

We are ready to prove the main result about FIND-AND-PASTE:

Theorem 7.13 *Given any instance* I *of* CDARP, FIND-AND-PASTE *finds a solution of cost at most* $3LB_{flow}(I)$.

Proof: We show that for any edge $[v, v+1]$ the number of upward moves constructed in Step 8 of FIND-AND-PASTE which traverse $[v, v+1]$ in direction from v to $v+1$ is bounded from above by $2\lambda[v, v+1]$. The bound for the number of downward moves constructed in Step 16 is established analogously.

By construction of FIND-AND-PASTE, upward moves which traverse the edge $[v, v+1]$ can only be added to the final solution in one of the following two situations:

(i) Edge $[v, v+1]$ is contained in a segment S used in Step 4 in an iteration of the **while**-loop.

(ii) Empty upward moves are added to connect the sequences of moves according to the solution of the instance Π of SOURCE-DARP in Step 20.

We first address situation (i). Let us denote the set $R^{\uparrow}$ at the beginning of the kth iteration of the **while**-loop in FIND-AND-PASTE enclosing Step 8 by $R^{\uparrow(k)}$. Then $R^{\uparrow(1)} = R^{\uparrow}$ and $\mu_v(R^{\uparrow(1)}) \leq \lambda[v, v+1]$ for any $v \in V$. Suppose that edge $[v, v+1]$ is contained in a segment S used in Step 4 in the kth iteration of the **while**-loop. In this case $\mu_v(R^{\uparrow(k)}) > 0$. From inequality (7.8) we can conclude that $[v, v+1]$ can be used at most $\lceil \mu_v(R^{\uparrow})/C \rceil$ times in a segment in Step 4 and thus by at most $2\lceil \mu_v(R^{\uparrow})/C \rceil \leq 2\lambda[v, v+1]$ upward moves constructed in Step 8. As noted above, the analogous bound for the number of downward moves traversing $[v, v+1]$ is established similarly.

So far we know that for each edge $[v, v+1]$ at most $2\lambda[v, v+1]$ upward moves from Step 8 and at most $2\lambda[v, v+1]$ downward moves from Step 16 traverse $[v, v+1]$. We now consider the chaining of the sequences of moves in Step 20 with the help of ALG-PATH from Section 6.3.

By adding balancing arcs (and corresponding empty moves) the maximum number of moves that traverse $[v, v+1]$ does not increase. Hence, it suffices to consider the empty moves corresponding to connecting arcs N. Since the set of arcs $\{ (v, v+1), (v+1, v) : 1 \leq v \leq n-1 \}$ is a feasible set of connecting arcs (which also preserves balance), it follows that the cost of the connecting arcs C chosen by ALG-PATH can not be more than $2d(1, n)$. Thus, the total weight of the solution found by FIND-AND-PASTE is not greater than

$$\left(\sum_{1 \leq v \leq n-1} 2\lambda[v, v+1] d(v, v+1) \right) + 2d(1, n) = 2\,\mathrm{LB}_{\mathrm{flow}} + 2d(1, n) \leq 3\,\mathrm{LB}_{\mathrm{flow}}.$$

This completes the proof. □

Corollary 7.14 FIND-AND-PASTE *is 3-approximative for* SOURCE-DARP. □

7.6 Remarks

In this chapter we have investigated the CDARP, an offline dial-a-ride problem with server capacity larger than one. Table 7.1 gives an overview over the results obtained in this chapter and the known results from literature. The table also includes the complexity and approximation results for the preemptive version of the CDARP which from a complexity point of view is somewhat ″easier″ to solve.

Graph class	CDARP without preemption	CDARP with preemption
Paths	Polynomial time solvable for capacity $C = 1$ *[AK88]* NP-hard for capacity $C \geq 2$ *[Gua98, Wei00]* Approximable within 3 *(Theorem 7.13)*	Polynomial time solvable for any capacity *[Gua98]*
Trees	NP-hard for all capacities *[Gua98]*	NP-hard for all capacities *[Gua98]* Approximable within 2 *[CR98]*
General Graphs	NP-hard Approximable within $\mathcal{O}(\sqrt{C} \log n \log \log n)$ *[CR98]*	NP-hard Approximable within $\mathcal{O}(\log n \log \log n)$ *[CR98]*

Table 7.1: Complexity and approximation results for the CDARP.

8

Online Bin Coloring

One of the commissioning departments in the distribution center of Herlitz PBS AG, Falkensee, a main distributor of office supply in Europe, is devoted to greeting cards. The cards are stored in parallel shelving systems. Order pickers on automated guided vehicles collect the orders from the storage systems, following a circular course through the shelves. At the loading zone, which can hold q vehicles, each vehicle is logically "loaded" with B orders which arrive online. The goal is to avoid congestion among the vehicles (see [AG+98] for details). Since the vehicles are unable to pass each other and the "speed" of a vehicle is correlated to the number of different stops it must make, this motivates to assign the orders to vehicles in such a way that the vehicles stop as few times as possible.

Commissioning of greeting cards.

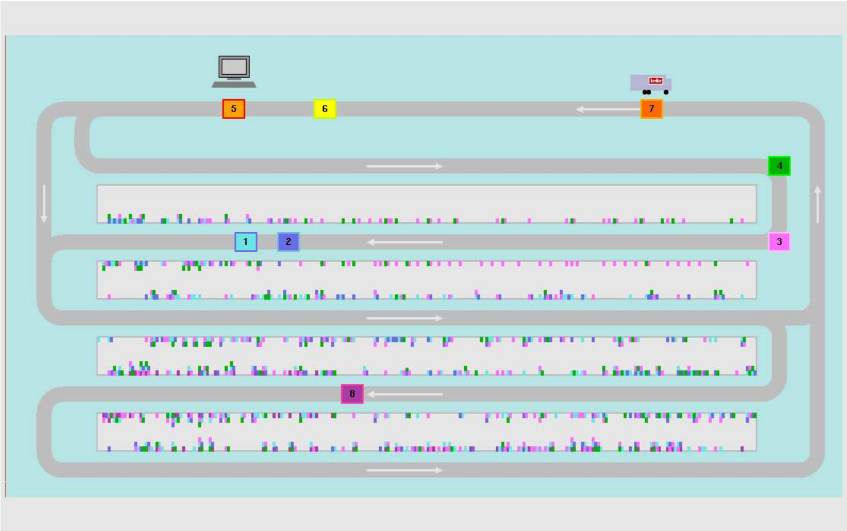
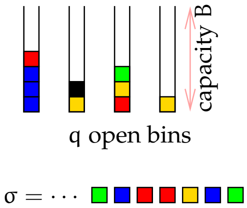


Figure 8.1 Course for the automated guided vehicles through the shelves.

The above situation motivates the following *bin coloring problem*:[1] 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

[1] Due to the nature of the bin coloring problem this chapter contains a number of colored figures. The best way to read it is either on a colored printout or in Postscript or PDF form.

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.)



Online bin coloring.

In the *online version*, 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.

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.

GREEDYFIT

We first analyze a natural "greedy-type" strategy, called GREEDYFIT, 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, that only uses one open bin at any time, has a strictly better competitive ratio of $2q - 1$. Then we show that surprisingly no deterministic algorithm can be substantially better than the trivial strategy. More specifically, we prove that no deterministic algorithm can, in general, have a competitive ratio less than q. Even more surprising, the general lower bound of q for the competitive ratio continues to hold for randomized algorithms against an oblivious adversary. 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.

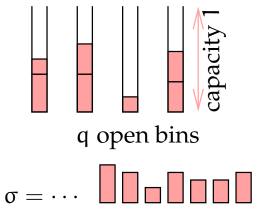
ONEBIN

The chapter is organized as follows. In Section 8.1 we formally define the OLBCP and introduce notation. In Section 8.2 we describe and analyze the obvious algorithm GREEDYFIT. In Section 8.3 we introduce and analyze the trivial algorithm ONEBIN which surprisingly obtains a better competitive ratio than GREEDYFIT. Sections 8.4 and 8.5 contain general lower bounds for deterministic and randomized algorithms.

Related Work

bin packing problem

The OLBCP can be viewed as a variant of the bounded space *bin packing problem*, a well studied problem in mathematics and computer science (see [CGJ97, CW98] for recent surveys). In the bin packing problem one receives a sequence of items $\sigma = r_1, \ldots, r_m$ where each item has a size from $(0, 1]$. The goal is to pack the items into bins of unit size so as to minimize the number of bins used. In the bounded space variant only a fixed number of partially-filled bins may be open to further items at any time in the packing process. In the *offline* version of the Bin Packing Problem the request sequence is a priori known to the algorithm. In the *online* version each item must be packed without knowledge of any future items.

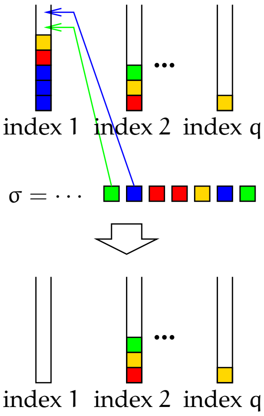


Online bounded space bin packing.

The online bin packing problem was first studied by Johnson [Joh74]. He showed that the so-called NEXTFIT-algorithm achieves a competitive ratio of 2. Competitiveness results and lower bounds for the bin packing problem have been improved in "horserace-papers" [Joh01] over the last years (we refer again to the surveys [CGJ97, CW98] for details). The currently best lower bound on the competitive ratio of deterministic algorithms is 1.53635 [Bro79, Lia80]. The best competitive ratio had been for a long time 1.58872 claimed by Richey for the HARMONIC+1-algorithm [Ric91]. Just recently Seiden found a bug in the analysis of HARMONIC+1 and, moreover, showed that HARMONIC+1 can not achieve a competitive ratio smaller than 1.59217 [Sei01]. In the same paper Seiden presents a new algorithm HARMONIC++ and proves a competitive ratio of 1.58889 for this algorithm.

8.1 Problem Definition

We start by defining the problem under study.



The green item must be packed before the blue item. Packing both items into the first open bin (the bin with index 1) closes this bin and replaces it by a new empty bin (which has again index 1).

Definition 8.1 (Online Bin Coloring Problem)
In the online bin coloring problem (OLBCP$_{B,q}$) *with parameters* $B, q \in \mathbb{N}$ *(*$B, q \geq 2$*), one is given a sequence* $\sigma = r_1, \dots, r_m$ *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* $k > i$*.*

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

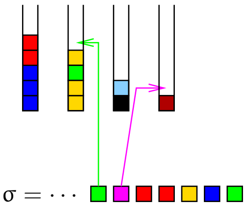
index

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.

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 less than q bins are non-empty). Ties are broken arbitrarily.



Since a green item is already present in the bin with index 2, GREEDYFIT places the new green item there. The magenta item is packed into the bin with the least number of colors, which at the moment is the last open bin.

The analysis of the competitive ratio of GREEDYFIT is essentially via a pigeon-hole principle argument. We first show a lower bound on the number of bins that *any* algorithm can use to distribute the items in a contiguous subsequence and then relate this number to the number of colors in the input sequence.

Lemma 8.2 *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 moment in time. To close an open bin b_j, ALG needs $f(b_j)$ 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. Thus, it follows that the maximum number of bins ALG can use is bounded from above as claimed in the lemma. □

Theorem 8.3 *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$(r_1, \dots, r_{s-1}) = w - 1$ and GREEDYFIT$(r_1, \dots, r_s) = w$. By the construction of GREEDYFIT, after processing $r_1, \dots, r_{s-1}$ each of the currently open bins must contain exactly $w-1$ different colors. Moreover, since $w \geq 2$, after processing 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 $b_1, \dots, b_q$.

Let bin b_j be the bin among $b_1, \dots, b_q$ that has been opened last by GREEDYFIT. Let $r_{s'}$ be the first item that was assigned to b_j. Then, the subsequence $\sigma' = r_{s'}, \dots, r_s$ 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 8.2 OPT distributes the items of σ' into at most $2q + \lfloor (qB - 3q + 1)/B \rfloor$ bins. Consequently,

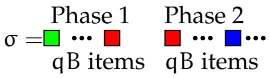
$$\text{OPT}(\sigma) \geq \frac{w}{2q + \lfloor (qB - 3q + 1)/B \rfloor},$$

which proves the claim. □

Corollary 8.4 *Algorithm* GREEDYFIT *is* c*-competitive for the* OLBCP$_{B,q}$ *with* $c = \min\{3q - 1, B\}$. □

We continue to prove a lower bound on the competitive ratio of algorithm GREEDYFIT. This lower bound shows that the analysis of the previous theorem is essentially tight.

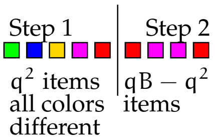
Theorem 8.5 GREEDYFIT *has a competitive ratio greater or equal to* $2q$ *for the* OLBCP$_{B,q}$ *if* $B \geq 2q^3 - q^2 - q + 1$.



Phases in the lower bound construction of Theorem 8.5.

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 which 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:



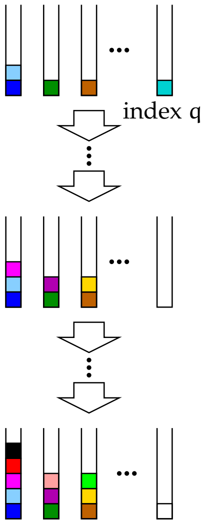
Structure of a single phase.

Property 1 The bins with indices $1, \dots, q-1$ are never closed.

Property 2 The bins with indices $1, \dots, q-1$ 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.



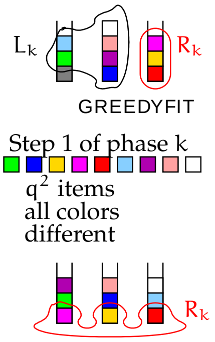
General development of the bins managed by GREEDYFIT.

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 $1, \dots, q-1$ 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 8.8 (iv)). We now partition the at most q^2 colors in $|R_k|$ into q disjoint non-empty sets $S_1, \dots, S_q$. 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 8.8 (iii), GREEDYFIT will pack all items given in Step 2 into bin q. Hence bins $1, \dots, q-1$ 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).



Packing of the items in Step 1 of a phase by GREEDYFIT and by the adversary in Theorem 8.5.

Lemma 8.6 *At the end of Phase k where $k = 1, \dots, M-1$, bin q of* GREEDYFIT *contains exactly $B - \sum_{j \leq k} |L_j|$ 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 $\sum_{j \leq k} |L_j|$ 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

$$kqB - \sum_{j \leq k} |L_j| \mod B \equiv B - \underbrace{\sum_{j \leq k} |L_j|}_{<B} \mod B.$$

We show that $B - \sum_{j \leq k} |L_j| \geq q^2$. This implies that $B - \sum_{j \leq k} |L_j| \mod B = B - \sum_{j \leq k} |L_j|$.

Since $k < M$ we know that each of the bins 1 through $q-1$ contains at most $2q^2 - 1$ colors. Thus, $\sum_{j \leq k} |L_j| \leq (2q^2-1)(q-1) = 2q^3 - 2q^2 - q + 1$. It follows from the assumption on B that $B - \sum_{j \leq k} |L_j| \geq q^2$. □

Corollary 8.7 *For any Phase $k < M$, bin q is never closed by* GREEDYFIT *before the end of Step 1 of Phase k.*

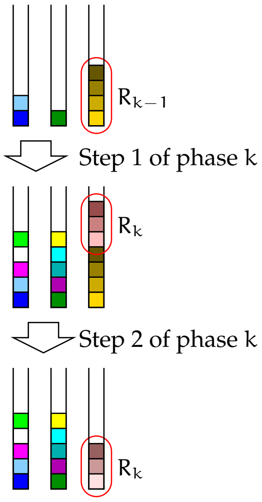
Proof: The claim clearly holds for the first phase. Hence for the remainder we consider the case $k > 1$.

Since there are exactly q^2 items presented in Step 1 of any phase, the claim is true by Lemma 8.6 as soon as $|\sum_{j<k} L_j| \geq q^2$ 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 $|L_1| + |L_2| \geq q^2$ which implies that $|\sum_{j<k} L_j| \geq q^2$ for $k \geq 2$.

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 8.6, 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.



Statement of Lemma 8.8.

Lemma 8.8 *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 8.6) 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 8.7 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 $1, \dots, q - 1$ 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 $1, \dots, q - 1$ 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 $1, \dots, q - 1$ 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 $1, \dots, q - 1$ 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:

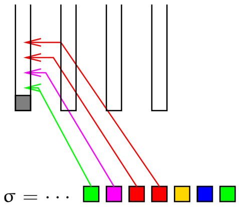
Lemma 8.9 *In every two subsequent Phases k and $k + 1$, either $|L_k \cup L_{k+1}| > 0$ or bin q contains items with $2q^2$ different colors during one of the two phases.*

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 8.8 (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 8.7). 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 8.9 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 8.5. □

8.3 The Trivial Algorithm ONEBIN

This section is devoted to arguably the simplest (and most trivial) algorithm for the OLBCP, which surprisingly has a better competitive ratio than GREEDYFIT. Moreover, as we will see later this algorithm achieves essentially the best competitive ratio for the problem.



The trivial strategy ONEBIN only uses one open bin to pack all items.

Algorithm ONEBIN
The algorithm uses only at most 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 8.10 *Let $\sigma = r_1, \dots, r_m$ be any request sequence. Then for $i \geq 0$ any algorithm packs the items $r_{iB+1}, \dots, r_{(i+1)B}$ into at most $\min\{2q-1, B\}$ bins.*

Proof: It is trivial that the B items $r_{iB+1}, \dots, r_{(i+1)B}$ 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 $\sigma' = r_{iB+1}, \dots, r_{(i+1)B}$ 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 $b_1, \dots, b_t$. Let $f(b_j) \in \{1, \dots, B-1\}$ be the number of empty places in bin b_j, $j = 1, \dots, t$. Notice that

$$\sum_{j=1}^{t} f(b_j) \equiv 0 \mod B. \tag{8.1}$$

Suppose that ALG uses at least $2q$ bins to distribute the items of σ'. By arguments similar to those given in Lemma 8.2, 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

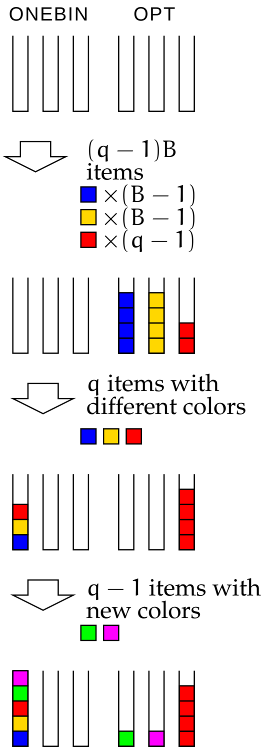
$$\sum_{j=1}^{t} f(b_j) + q \leq B. \tag{8.2}$$

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 $j = 1, \dots, t$. □

As a consequence of the previous lemma we obtain the following bound on the competitive ratio of ONEBIN.

Theorem 8.11 *Algorithm* ONEBIN *is* c*-competitive for the* $\text{OLBCP}_{B,q}$ *where* $c = \min\{2q-1, B\}$.

Proof: Let $\sigma = r_1, \dots, r_m$ be any request sequence for the $\text{OLBCP}_{B,q}$ and suppose that $\text{ONEBIN}(\sigma) = w$. Let $\sigma' = r_{iB+1}, \dots, r_{(i+1)B}$ of σ be the subsequence on which ONEBIN gets w different colors. Clearly, σ' contains items with exactly w colors. By Lemma 8.10 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 of $2q-1$ is tight for ONEBIN.

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

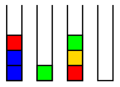
In this section we prove a general lower bound on the competitive ratio of any deterministic online algorithm for the OLBCP. We establish a lemma which immediately leads to the desired lower bound but which is even more powerful. In particular, this lemma will allow us to derive essentially the same 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 8.12 ($\mathcal{C}$-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 $K\colon \{1,\dots,q\} \to \mathcal{S}_B$, where*

$$\mathcal{S}_B = \{\, S : S \textit{ is a multiset over } \mathcal{C} \textit{ containing at most } B \textit{ elements from } \mathcal{C}\,\}$$

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.



$\mathcal{C} = \{$ 🔴🔵🟡🟢 $\}$

$\mathcal{C}$-configuration for a four color set $\mathcal{C}$.

We are now ready to prove the key lemma which will be used in our lower bound constructions.

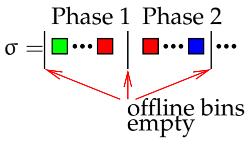
Lemma 8.13 *Let $B, q, s \in \mathbb{N}$ be numbers such that $s \geq 1$ and the inequality $B/q \geq s-1$ holds. There exists a finite set $\mathcal{C}$ of colors and a constant $L \in \mathbb{N}$ with the following property: For any deterministic algorithm ALG and any $\mathcal{C}$-configuration K there exists an input sequence $\sigma_{\text{ALG},K}$ of $\text{OLBCP}_{B,q}$ such that*

(i) The sequence $\sigma_{\text{ALG},K}$ uses only colors from $\mathcal{C}$ and $|\sigma_{\text{ALG},K}| \leq L$, that is, $\sigma_{\text{ALG},K}$ consists of at most L requests.

(ii) If ALG starts with initial $\mathcal{C}$-configuration K then $\text{ALG}(\sigma_{\text{ALG},K}) \geq (s-1)q$.

(iii) If OPT starts with the empty configuration (i.e., all bins are empty), then $\text{OPT}(\sigma_{\text{ALG},K}) \leq s$. Additionally, OPT can process the sequence in such a way that at the end again the empty configuration is attained.

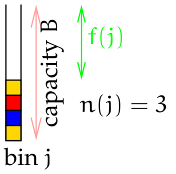
Moreover, all of the above statements remain true even in the case that the online algorithm is allowed to use $q' \geq q$ bins instead of q (while the offline adversary still only uses q bins). In this case, the constants $|\mathcal{C}|$ and K depend only on q' but not on the particular algorithm ALG.



Phases in the general lower bound construction in Lemma 8.13.

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 $\mathcal{C}$-configuration K. The construction of the request sequence $\sigma_{\text{ALG},K}$ 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, $P_1 = \varnothing$ 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.



Number of different colors $n(j)$ and free space $f(j)$ in bin j.

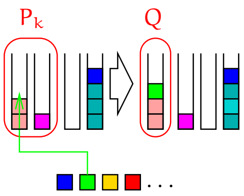
For bin j of ALG we denote by $n(j)$ the number of different colors currently in bin j and by $f(j)$ the space left in bin j. Then every bin $j \in P_k$ satisfies $n(j) + f(j) \geq (s-1)q$. By $\min P_k := \min_{j \in P_k} n(j)$ we denote the minimum number of colors in a bin from P_k.

The idea of the construction is the following (cf. Claim 8.15 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: The online algorithm puts an item into a bin $p \in P_k$.

Case 1 ALG puts an item into a bin $p \in P_k$.

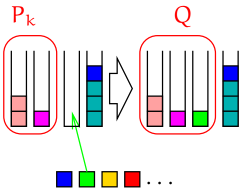
In this case we let $Q := P_k \setminus \{j \in P_k : n(j) < n(p)\}$, that is, we remove all bins from P_k which have less than $n(p)$ colors.

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).

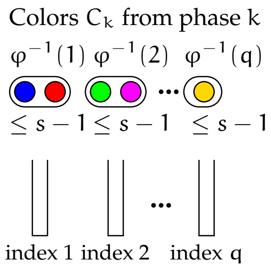


Case 2: The online algorithm puts an item into a bin $j \in P_k$ with $n(j) + f(j) \geq (s-1)q$.

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).

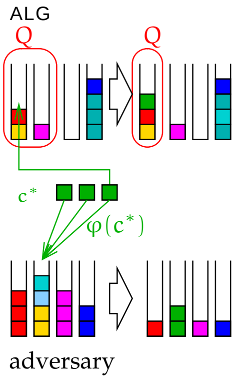
Since the adversary started the phase with all bins empty and during the current phase we have given no more than $(s-1)q$ colors, the adversary can assign the items to bins such that no bin contains more than $s-1$ different colors (we will describe below how this is done precisely). Notice that due 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.



Organization of the adversary bins.

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:



Case (a): ALG puts an item into a bin $j \in Q$.

1. (Start of clearing loop iteration) Choose a color $c^* \in C_k$ which is not contained in any bin from Q. If there is no such color, goto the "good end" of the clearing loop (Step 4).

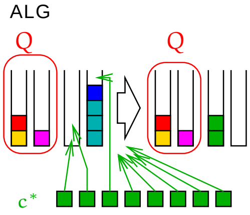
2. Let $F \leq qB$ denote the current total empty space in the open offline bins. Present items of color c^* until one of the following things happens:

   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

$$Q' := Q \setminus \{b \in Q : n(b) < n(j)\},$$

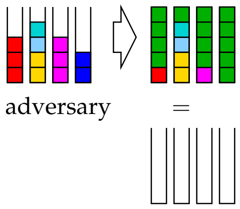
in other words, we remove all bins b from Q which currently have less than $n(j)$ colors. This guarantees that

$$\min_{b \in Q'} n(b) > \min_{b \in Q} n(b) \geq \min P_k. \tag{8.5}$$



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): ALG puts none of the F items into a bin from Q.

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 (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 8.14 *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 8.15 *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 8.15, 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 $\sigma_{\text{ALG},K}$. Clearly, the length of $\sigma_{\text{ALG},K}$ can be bounded by a constant L independent of ALG and K.

Proof of Claim 8.15 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 8.13. □

As an immediate consequence of Lemma 8.13 we obtain the following lower bound result for the competitive ratio of any deterministic algorithm:

Theorem 8.16 *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* $\frac{s-1}{s} \cdot q$. *Consequently, the competitive ratio of any deterministic algorithm for fixed* B *and* q *is at least* $\left(1 - \frac{q}{B+q}\right) q$. *In particular, 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.

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.* □

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 $\mathrm{OLBCP}_{B,q}$.

Theorem 8.17 *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* $\mathrm{OLBCP}_{B,q}$ *can achieve a competitive ratio less than* $\frac{s-1}{s} \cdot q$ *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 $\mathcal{A} := \{\, \mathsf{ALG}_y : y \in \mathcal{Y} \,\}$ be the set of deterministic algorithms for the $\mathrm{OLBCP}_{B,q}$. 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 $\mathsf{ALG}_y \in \mathcal{A}$

$$\mathbb{E}_X [\mathsf{ALG}_y(\sigma_x)] \geq (s-1)q,$$

and, moreover,

$$\mathbb{E}_X [\mathsf{OPT}(\sigma_x)] \leq s.$$

The claim of the theorem then follows by Yao's Principle.

Let us recall the essence of Lemma 8.13. 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 $\Sigma = \{\sigma_1, \dots, \sigma_N\}$ 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

$$\{\, \sigma_{i_1}, \sigma_{i_2}, \dots, \sigma_{i_k} : k \in \mathbb{N}, 1 \leq i_j \leq N \,\}$$

such that σ_{i_j} is chosen from Σ uniformly and independently of all previous subsequences $\sigma_{i_1}, \dots, \sigma_{i_{j-1}}$. We call subsequence σ_{i_k} the k*th phase*.

Let $\mathsf{ALG}_y \in \mathcal{A}$ be arbitrary. Define ϵ_k by

$$\epsilon_k := \Pr_X \left[\begin{array}{c} \mathsf{ALG}_y \text{ has at least one bin with} \\ \text{at least } (s-1)q \text{ colors during Phase } k \end{array} \right].$$

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

$$p_k := \Pr_X \left[\mathsf{ALG}_y(\sigma_{i_1} \dots \sigma_{i_{k-1}} \sigma_{i_k}) \geq (s-1)q \right].$$

Then the probabilities p_k satisfy the following recursion:

$$p_0 = 0 \tag{8.6}$$

$$p_k = p_{k-1} + (1 - p_{k-1})\epsilon_k \tag{8.7}$$

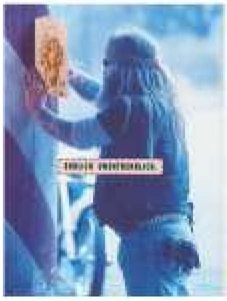
The first term in (8.7) corresponds to the probability that ALG_y has already cost at least $(s-1)q$ after Phase $k-1$, the second term accounts for the probability that this is not the case but cost at least $(s-1)q$ is achieved in Phase k. By construction of X, these events are independent. Since $\epsilon_k \geq 1/N$ 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 $\mathbb{E}_X[\text{ALG}_y(\sigma_x)]$ 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 8.13. □

8.6 Remarks

We have studied the online bin coloring problem OLBCP, which was motivated by applications in a robotized assembly environment. The investigation of the problem from a competitive analysis point of view revealed a number of oddities. A natural greedy-type strategy (GREEDYFIT) achieves a competitive ratio strictly worse than arguably the most stupid algorithm (ONEBIN). Moreover, no algorithm can be substantially better than the trivial strategy (ONEBIN). Even more surprising, neither randomization nor "resource augmentation" helps to overcome the $\Omega(q)$ lower bound on the competitive ratio. This is in contrast to [PK95, PS$^+$97] where the concept of resource augmentation was applied successfully to scheduling problems. Intuitively, the strategy GREEDYFIT should perform well "on average" (which was confirmed in preliminary experiments with random data, see [Höf01]).



Advertisement for greeting cards by Herlitz.

An open problem remains the existence of a deterministic (or randomized) algorithm which achieves a competitive ratio of q (matching the lower bound of Theorems 8.16 and 8.17). However, the most challenging issue raised by our work seems to be an investigation of OLBCP from an average-case analysis point of view.

Table 8.1 provides an overview over the results obtained in this chapter.

Problem	Competitive Ratios	Lower Bounds
OLBCP	ONEBIN: $\min\{2q-1, B\}$ *(Theorem 8.11)* GREEDYFIT: $\min\{3q, B\}$ *(Corollary 8.4)*	deterministic algorithms: $\left(1-\frac{q}{B+q}\right)q$ *(Theorem 8.16)* randomized algorithms: $\left(1-\frac{q}{B+q}\right)q$ *(Theorem 8.17)*

Table 8.1: Results for the OLBCP.

9

Conclusions

We have investigated a number of online problems from a theoretical point of view. Competitive analysis has been one of the main tools for deriving worst-case bounds on the performance of algorithms. Even for seemingly easy online optimization problems such as the online traveling salesman problem on the non-negative part of the real line, whose offline version is trivial to solve, competitive analysis has lead to mathematical problems that are surprisingly difficult.

To vanquish the pessimism of competitive analysis we have investigated modifications and alternatives for analyzing online algorithms. We showed that in case of the online traveling salesman problem a natural restriction of the power of the offline adversary leads to algorithms with improved performance in comparison to a "fair adversary".

For the minimization of the flow times in online dial-a-ride problems competitive analysis was not able to deliver any decision support. All online algorithms are equally bad from a competitive analysis point of view. We introduced the new concept of reasonable load which enables us to distinguish certain algorithms and to prove performance bounds on the maximal respective average flow time.

A further shortcoming of competitive analysis is that it does not take into account the real-time requirements that are present in many real-world systems. Our study of online dial-a-ride problems naturally lead to offline dial-a-ride problems. Instances of these offline problems have to be solved repeatedly by all our competitive algorithms. The investigation of offline dial-a-ride problems resulted in efficient (polynomial time) algorithms and approximation algorithms with provable performance guarantees. These algorithms are fast enough to be used under real-time requirements.

It would be bold to claim that the methods and results presented in this thesis have immediate impact to real-world applications. Real-world systems are far more complex than the elementary problems presented here. A rigorous (competitive) analysis seems to be out of reach in most cases.

Nevertheless the analysis of elementary problems should lead the right way: it is possible to get an idea about what kind of strategies are promising for real-world systems and why. For instance, the concept of reasonable load was motivated by simulation experiments [GH$^+$99, GH$^+$00], where a "stable behavior" of the IGNORE-strategy could be observed. In turn, further experimental work [Glü00] indicates that algorithms which perform well under reasonable load in theory also exhibit a good performance in practice.

So, is there hope for the persons standing in front of the elevator from the introduction? The answer is: it depends! If you have one cleanly defined objective, such as the makespan or the maximum flow time, then this thesis contains algorithms which are theoretically "bullet proof" and also perform well in experiments (see [GH$^+$99, GH$^+$00, Glü00, Hau99] for simulation studies).

But there are a few hitches. If you do not have a single objective but aim at minimizing a number of antagonistic objective function simultaneously, these algorithms are not able to provide you with an answer (yet).

Finally, no algorithm can keep the maximal dissatisfaction of the passengers (the maximal flow time) bounded if zillions of people keep on requesting transportation. If the load is unreasonable, the only reasonable way out seems to be to install a second, third and fourth elevator—or just a single mirror. Experiments show that people estimate their waiting times to be much shorter if they can watch themselves in a mirror while waiting!

A

Notation

This chapter is intended mainly as a reference for the notation used in this thesis and the foundations this work relies on. We assume that the reader is familiar with elementary graph theory, graph algorithmic concepts, and combinatorial optimization as well as with basic results from complexity theory. For detailed reviews we refer the reader to monographs and textbooks which are listed at the end of this chapter.

A.1 Basics

By $\mathbb{R}$ ($\mathbb{Q}$, $\mathbb{Z}$, $\mathbb{N}$) we denote the set of real (rational, integral, natural) numbers. The set $\mathbb{N}$ of natural numbers does not contain zero. $\mathbb{R}_0^+$ ($\mathbb{Q}_0^+$, $\mathbb{Z}_0^+$) denotes the nonnegative real (rational, integral) numbers.

The rounding of real numbers $x \in \mathbb{R}_0^+$ is denoted by the notation $\lfloor x \rfloor := \max\{\, n \in \mathbb{N} \cup \{0\} : n \leq x \,\}$ and $\lceil x \rceil := \min\{\, n \in \mathbb{N} : n \geq x \,\}$.

By 2^S we denote the *power set* of a set S, which is the set of all subsets of set S (including the empty set $\varnothing$ and S itself).

A.2 Sets and Multisets

multiset

A *multiset* Y over a ground set U, denoted by $Y \sqsubset U$, can be defined as a mapping $Y\colon U \to \mathbb{N}$, where for $u \in U$ the number $Y(u)$ denotes the multiplicity of u in Y. We write $u \in Y$ if $Y(u) \geq 1$. If $Y \sqsubset U$ then $X \sqsubset Y$ denotes a multiset over the ground set $\{\, u \in U : Y(u) > 0 \,\}$.

multiset union

multiset difference

multiset intersection

If $Y \sqsubset U$ and $Z \sqsubset U$ are multisets over the same ground set U, then we denote by $Y + Z$ their *multiset union*, by $Y - Z$ their *multiset difference* and by $Y \cap Z$ their *multiset intersection*, defined for $u \in U$ by

$$\begin{aligned}
(Y + Z)(u) &= Y(u) + Z(u) \\
(Y - Z)(u) &= \max\{Y(u) - Z(u), 0\} \\
(Y \cap Z)(u) &= \min\{Y(u), Z(u)\}.
\end{aligned}$$

The multiset $Y \sqsubset U$ is a subset of the multiset $Z \sqsubset U$, denoted by $Y \subseteq Z$, if $Y(u) \leq Z(u)$ for all $u \in U$. For a weight function $c\colon U \to \mathbb{R}$ the weight of a multiset $Y \sqsubset U$ is defined by $c(Y) := \sum_{u \in U} c(u)Y(u)$. We denote the cardinality of a multiset $Y \sqsubset U$ by $|Y| := \sum_{u \in U} Y(u)$.

Any (standard) set can be viewed as a multiset with elements of multiplicity 0 and 1. If X and Y are two standard sets with $X \subseteq Y$ and $X \neq Y$, then X is a *proper subset* of Y, denoted by $X \subset Y$. Two subsets $X_1 \subseteq Y$, $X_2 \subseteq Y$ of a standard set Y form a *partition* of Y, if $Y = X_1 \cup X_2$ and $X_1 \cap X_2 = \emptyset$.

proper subset

partition

A.3 Analysis and Linear Algebra

References: [Rud76]

metric space

distance function

metric

A *metric space* (X, d) consists of a nonempty set X and a *distance function* or *metric* $d\colon X \times X \to \mathbb{R}_0^+$ which satisfies the following three conditions:

(i) $d(x, y) > 0$ if $x \neq y$; $d(x, x) = 0$;

(ii) $d(x, y) = d(y, x)$;

(iii) $d(x, y) \leq d(x, z) + d(z, y)$ for all $z \in X$.

triangle inequality

Inequality (iii) is called the *triangle inequality*. An example of a metric space is the set $\mathbb{R}^p$ endowed with the Euclidean metric which for vectors $x = (x_1, \ldots, x_p) \in \mathbb{R}^p$ and $y = (y_1, \ldots, y_p) \in \mathbb{R}^p$ is defined by

$$d(x, y) := \left(\sum_{i=1}^{p} (x_i - y_i)^2 \right)^{1/2} .$$

Euclidean space

path

rectifiable

This metric space is usually referred to as the *Euclidean space*.

A *path* in a metric space (X, d) is a continuous function $\gamma\colon [0, 1] \to X$. The path γ is called *rectifiable*, if for all dissections $0 = t_0 < t_1 < \cdots < t_k = 1$ of the interval $[0, 1]$ the sum

$$\sum_{i=1}^{k} d(\gamma(t_i), \gamma(t_{i-1}))$$

length

is bounded from above. The supremum of the sums, taken over all dissections, is then referred to as the *length* of the path γ.

A.4 Growth of Functions

References: [CLR90, AHU74]

$\mathcal{O}(g(n))$

Let g be a function from $\mathbb{N}$ to $\mathbb{N}$. The set $\mathcal{O}(g(n))$ contains all those functions $f\colon \mathbb{N} \to \mathbb{N}$ with the property that there exist constants $c > 0$ and $n_0 \in \mathbb{N}$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$. A function f belongs to the set $\Omega(g(n))$, if and only if $g(n) \in \mathcal{O}(f(n))$. The notation $f(n) \in \Theta(g(n))$ means that $f(n) \in \mathcal{O}(g(n))$ and $f(n) \in \Omega(g(n))$. Finally, we write $f(n) \in o(g(n))$, if $\lim\limits_{n\to\infty} f(n)/g(n) = 0$. $o(g(n))$

A.5 Particular Functions

We use $\log_a$ to denote the *logarithm* function to the basis of a. We omit the basis in the case of $a = 2$ for the sake of convenience. By $\ln n$ we denote the *natural logarithm* of a positive number n, that is, $\ln n := \log_e n$.

logarithm

natural logarithm

A.6 Probability Theory

References: [Fel68, Fel71, MR95]

A *probability space* $(\Omega, \mathbb{F}, \Pr)$ consists of a σ-field $(\Omega, \mathbb{F})$ with a probability measure Pr defined on it. When specifying a probability space, $\mathbb{F}$ may be omitted which means that the σ-field referred to is $(\Omega, 2^{\Omega})$.

probability space

In this thesis we are mainly concerned with the case that Ω is either the the set of real numbers $\mathbb{R}$ or an interval contained in $\mathbb{R}$. In this context a *density function* is a non-negative function $p\colon \mathbb{R} \to \mathbb{R}_0^+$ whose integral, extended over the real numbers, is unity, that is $\int_{-\infty}^{+\infty} p(x)\,dx = 1$. The density corresponds to the probability measure Pr, which satisfies

density function

$$\Pr\left[x \in (-\infty, t]\right] = \int_{-\infty}^{t} p(x)\,dx.$$

A.7 Graph Theory

References: [Har72, AMO93]

A *mixed graph* $G = (V, E, R)$ consists of a set V of *vertices* (or *nodes*), a set E of *undirected edges*, and a multiset R of *directed arcs*. We usually denote by $n := |V|$, $m_E := |E|$ and $m_R := |R|$ the number of vertices, edges and arcs, in G respectively. Throughout the thesis we assume that V, E, and R are all finite. If $R = \varnothing$, we briefly write $G = (V, E)$ and call G an *undirected graph* (or simply *graph*) with vertex set V and edge set E. If $E = \varnothing$, we refer to $G = (V, R)$ as a *directed graph* with vertex set V and arc (multi-) set R.

mixed graph

vertices/nodes

undirected edges

directed arcs

undirected/directed graph

incident

source/target

incoming/outgoing arc

Each undirected edge is an unordered pair $[u, v]$ of distinct vertices $u \neq v$. The edge $[u, v]$ is said to be *incident* to the vertices u and v. Each arc is an ordered pair (u, v) of vertices which is incident to both u and v. We refer to vertex u as the *source* of arc (u, v) and to vertex v as its *target*. The arc (u, v) *emanates* from vertex u and *terminates* at vertex v. An arc (u, v) is *incident* to both vertices u and v. The arc (u, v) is an *outgoing arc* of node u and an *incoming arc* of vertex v. We call two vertices *adjacent*, if there is an edge or an arc which is incident with both of them.

adjacent

parallel

Two arcs are called *parallel arcs* if they refer to copies of the same element (u, v) in the multiset R. Arcs (u, v) and (v, u) are termed *anti-parallel* or *inverse*. We write $(u, v)^{-1} := (v, u)$ to denote an inverse arc to (u, v). For a set R of arcs we denote by R^{-1} the set $R^{-1} := \{ r^{-1} : r \in R \}$.

anti-parallel

inverse

subgraph

subgraph of G induced by X

Let $G = (V, E, R)$ be a mixed graph. A graph $H = (V_H, E_H, R_H)$ is a *subgraph* of G if $V_H \subseteq V$, $E_H \subseteq E$ and $R_H \subseteq R$. For a multiset $X \sqsubset E + R$ we denote by $G[X]$ the *subgraph of* G *induced by* X, that is, the subgraph of G consisting of the arcs and edges in X together with their incident vertices. A subgraph of G *induced by vertex set* $X \subseteq V$ is a subgraph with node set X and containing all those edges and arcs from G which have both endpoints in X.

outdegree

indegree

For $v \in V$ we let R_v be the set of arcs in R emanating from v. The *outdegree* of a vertex v in G, denoted by $\deg^+_G(v)$, equals the number of arcs in G leaving v. Similarly, the *indegree* $\deg^-_G(v)$ is defined to be the number of arcs entering v. If $X \sqsubset R$, we briefly write $\deg^+_X(v)$ and $\deg^-_X(v)$ instead of $\deg^+_{G[X]}(v)$ and $\deg^-_{G[X]}(v)$. The *degree* of a vertex v in an undirected graph $G = (V, E)$ is defined to be the number of edges incident with v.

clique

complete graph

A subset C of the vertices of an undirected graph $G = (V, E)$ such that every pair of vertices is adjacent is called a *clique* of size $|C|$ in the graph G. A graph G whose vertex set forms a clique is said to be a *complete graph*.

path

A *path* P in an undirected graph $G = (V, E)$ is defined to be an alternating sequence $p = (v_1, e_1, v_2, \dots, e_k, v_{k+1})$ of nodes $v_i \in V$ and edges $e_i \in E$, where for each triple (v_i, e_i, v_{i+1}) we have $e_i = (v_i, v_{i+1})$. We use equivalently the alternative notation $P = (v_1, v_2, \dots, v_{k+1})$ and $P = (e_1, e_2, \dots, e_k)$ when the meaning is clear. For directed graphs $G = (V, R)$, edges are replaced by arcs, and we require $r_i = (v_i, v_{i+1})$ and $r_i \in R \cup R^{-1}$ for each triple. If the stronger condition $r_i \in R$ holds, the path is called *directed*.

walk

oriented walk

For mixed graphs, we define a *walk* which traverses arbitrarily edges and directed arcs. An *oriented walk* is a "directed version" of a walk in the sense that for any two consecutive vertices v_i and v_{i+1} we require that either x_i is an undirected edge $[v_i, v_{i+1}]$ between v_i and v_{i+1} or a directed arc (v_i, v_{i+1}) from v_i to v_{i+1}.

If all nodes of the path or walk are pairwise different (without considering the pair v_1, v_{k+1}), the path or walk is called *simple*. A path or walk with coincident start and endpoint is *closed*. A closed and simple path or walk is a *cycle*. An *Eulerian cycle* in a directed graph $G = (V, R)$ is a directed cycle which contains (traverses) every arc from R exactly once. The directed graph G is called *Eulerian* if it contains an Eulerian cycle. A *Hamiltonian path Hamiltonian cycle*) is a simple path (cycle) which touches every vertex in a directed (or undirected) graph.

cycle

Eulerian cycle

Hamiltonian path/cycle

A mixed graph $G = (V, E, R)$ is *connected* (*strongly connected*), if for every pair of vertices $u, v \in V$ with $u \neq v$ there is an walk (oriented walk) from u to v in G. A (strongly) connected subgraph of G which is maximal with respect to set inclusion is called *(strongly) connected component* of G.

connected

strongly connected

A *tree* is a connected graph that contains no cycle. A node in a tree is called a *leaf* if its degree equals 1, and an *inner node* otherwise. A *spanning tree* of a graph G is a tree which has the same vertex set as G.

tree

leaf

inner node

A *Steiner tree* with respect to a subset K of the vertices of an undirected graph G, is a tree which is a subgraph of G and whose vertex set includes K. The vertices in K are called *terminals*.

A *directed in-tree rooted at* $o \in V$ is a subgraph of a directed graph $H = (V, A)$ which is a tree and which has the property that for each $v \in V$ it contains a directed path from v to o.

Additional definitions to the basic ones presented above will be given in the respective contexts.

A.8 Theory of Computation

References: [GJ79, Pap94, GLS88, CLR90]

Model of Computation

The *Turing machine* [GJ79] is the classical model of computation that was used to define the *computational complexity* of algorithms. However, for practical purposes it is fairly more convenient to use a different model. In the *random access machine* or *RAM model* [Pap94, MR95] we have a machine which consists of an infinite array of registers, each capable of containing an arbitrarily large integer, possibly negative. The machine is capable of performing the following types of operations involving registers and main memory: input-output operations, memory-register transfers, indirect addressing, arithmetic operations and branching. The arithmetic operations permitted are addition, subtraction, multiplication and division of numbers. Moreover, the RAM can compare two numbers and evaluate the square root of a positive number.

Turing machine

RAM model

There are two types of RAM models used in literature. In the *log-cost RAM* the execution time of each instruction takes time proportional to the encoding length, i.e. proportional to the logarithm of the size of its

log-cost RAM

unit-cost RAM

operands, whereas in the *unit-cost RAM* each instruction can be accomplished in one time step. A log-cost RAM is equivalent to the Turing machine under a polynomial time simulation [Pap94]. In contrast, in general there is no polynomial simulation for a unit-cost RAM, since in this model we can compute large integers too quickly by using multiplication. However, if the encoding lengths of the operands occurring during the run of an algorithm on a unit-cost RAM are bounded by a polynomial in the encoding length of the input, a polynomial time algorithm on the unit-cost RAM will transform into a polynomial time algorithm on a Turing machine [GLS88, Pap94]. This argument remains valid in the case of nondeterministic programs.

For convenience, we will use the general unit-cost RAM to analyze the running time of our algorithms. This does not change the essence of our results, because the algorithms in which we are interested involve only operations on numbers that are not significantly larger than those in the input.

Computational Complexity

Classical complexity theory expresses the running time of an algorithm in terms of the "size" of the input, which is intended to measure the amount of data necessary to describe an instance of a problem. The running time of an algorithm on a specific input is defined to be the sum of times taken by each instruction executed. The *worst case time complexity* or simply *time complexity* of an algorithm is the function $T(n)$ which is the maximum running time taken over all inputs of size n (cf. [AHU74, GJ79, GLS88]).

worst case time complexity

alphabet

An *alphabet* Σ is a nonempty set of characters. By Σ^* we denote the set of all strings over Σ including the empty word. We will assume that every problem Π has an (encoding independent) associated function *length* : $D_\Pi \to \mathbb{N}$, which is polynomially related to the input lengths that would result from a "reasonable encoding scheme". Here, $D_\Pi \subseteq \Sigma^*$ is the set of *instance*s of the problem Π, expressed as words over the alphabet Σ. For a more formal treatment of the input length and also of the notion of a "reasonable encoding scheme" we refer to [GJ79].

encoding scheme

decision problem

A *decision problem* is a problem where each instance has only one of two outcomes from the set $\{\text{yes}, \text{no}\}$. For a nondecreasing function $f\colon \mathbb{N} \to \mathbb{N}$ the *deterministic time complexity class* $\mathsf{DTIME}(f(n))$ consists of the decision problems for which there exists a deterministic Turing machine deciding the problem in $\mathcal{O}(f(n))$ time. Its nondeterministic counterpart $\mathsf{NTIME}(f(n))$ is defined analogously. The most important complexity classes with respect to this thesis are

$\mathsf{DTIME}(f(n))$

$\mathsf{NTIME}(f(n))$

$$\mathsf{P} := \bigcup_{k=1}^{\infty} \mathsf{DTIME}(n^k) \qquad \text{and} \qquad \mathsf{NP} := \bigcup_{k=1}^{\infty} \mathsf{NTIME}(n^k)\,.$$

Suppose we are given two decision problems Π and Π'. A *polynomial time transformation* is an algorithm t which, given an encoded instance I of Π, produces in polynomial time an encoded instance $t(I)$ of Π' such that the following holds: For every instance I of Π, the answer to Π is "yes" if and only if the answer to the transformation $t(I)$ (as an instance of Π') is "yes". A decision problem Π is called NP-*complete* if $\Pi \in$ NP and every other decision problem in NP can be transformed to Π in polynomial time.

polynomial time transformation

NP-complete

To tackle also optimization problems rather than just decision problems it is useful to extend the notion of a transformation between problems. Informally, a *polynomial time Turing reduction* (or just *Turing reduction*) from a problem Π to a problem Π' is an algorithm ALG, which solves Π by using a hypothetical subroutine ALG′ for solving Π' such that if ALG′ were a polynomial time algorithm for Π', then ALG would be a polynomial time algorithm for Π. More precisely, a polynomial time Turing reduction from Π to Π' is a deterministic polynomial time oracle Turing machine (with oracle Π') solving Π.

polynomial time Turing reduction

An optimization problem Π is called NP-*hard* ("at least as difficult as any problem in NP"), if there is an NP-complete decision problem Π' such that Π' can be Turing reduced to Π. Results from complexity theory (see e.g. [GJ79]) show that such an NP-hard optimization problem can not be solved in polynomial time unless P $=$ NP.

NP-hard

Bibliography

[ABF96] B. Awerbuch, Y. Bartal, and A. Fiat, *Distributed paging for general networks*, Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms, 1996, pp. 574–583.

[AC+86] F. Afrati, C. Cosmadakis, C. Papadimitriou, G. Papageorgiou, and N. Papakostantinou, *The complexity of the traveling repairman problem*, Informatique Theorique et Applications **20** (1986), no. 1, 79–87.

[AC+99] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, *Complexity and approximation. combinatorial optimization problems and their approximability properties*, Springer, 1999.

[ACM98] S. Albers, M. Charikar, and M. Mitzenmacher, *Delayed information and action in on-line algorithms*, Proceedings of the 30th Annual ACM Symposium on the Theory of Computing, 1998, pp. 416–425.

[Ada96] S. Adams, *The Dilbert principle*, HarperCollins, New York, 1996.

[AF+94] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo, *Serving request with on-line routing*, Proceedings of the 4th Scandinavian Workshop on Algorithm Theory, Lecture Notes in Computer Science, vol. 824, July 1994, pp. 37–48.

[AF+95] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo, *Competitive algorithms for the traveling salesman*, Proceedings of the 4th Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science, vol. 955, August 1995, pp. 206–217.

[AF+01] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo, *Algorithms for the on-line traveling salesman*, Algorithmica (2001), To appear.

[AG+98] N. Ascheuer, M. Grötschel, S. O. Krumke, and J. Rambau, *Combinatorial online optimization*, Proceedings of the International Conference of Operations Research (OR'98), Springer, 1998, pp. 21–37.

[AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The design and analysis of computer algorithms*, Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1974.

[AK88] M. J. Atallah and S. R. Kosaraju, *Efficient solutions to some transportation problems with applications to minimizing robot arm travel*, SIAM Journal on Computing **17** (1988), no. 5, 849–869.

[AK99] 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.

[AKR00] 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, Springer, 2000, pp. 639–650.

[Alb93] S. Albers, *The influence of lookahead in competitive paging algorithms*, Proceedings of the 1st Annual European Symposium on Algorithms, Lecture Notes in Computer Science, vol. 726, Springer, 1993, pp. 1–12.

[Alb97] S. Albers, *Better bounds for online scheduling*, Proceedings of the 24th Annual ACM Symposium on the Theory of Computing, 1997, pp. 130–139.

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

[BC$^+$94] 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.

[BDB$^+$94] S. Ben-David, A. Borodin, R. M. Karp, G. Tardos, and A. Wigderson, *On the power of randomization in on-line algorithms*, Algorithmica **11** (1994), 2–14.

[BEY98] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*, Cambridge University Press, 1998.

[BI$^+$91] A. Borodin, S. Irani, P. Raghavan, and B. Schieber, *Competitive paging with locality of reference*, Proceedings of the 23th Annual ACM Symposium on the Theory of Computing, 1991, pp. 249–259.

[BI$^+$95] A. Borodin, S. Irani, P. Raghavan, and B. Schieber, *Competitive paging with locality of reference*, Journal of Computer and System Sciences **50** (1995), 244–258.

[BK$^+$96] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson, *Adversarial queueing theory*, Proceedings of the 23rd Annual ACM Symposium on the Theory of Computing, 1996, pp. 376–385.

[BK$^+$01] M. Blom, S. O. Krumke, W. E. de Paepe, and L. Stougie, *The online-TSP against fair adversaries*, Informs Journal on Computing **13** (2001), no. 2, 138–148, A preliminary version appeared in the Proceedings of the 4th Italian Conference on

Algorithms and Complexity, 2000, vol. 1767 of Lecture Notes in Computer Science.

[Bro79] D. J. Brown, *A lower bound for on-line one-dimensional bin packing algorithms*, Technical Report R-864, Coodinated Science Lab, University of Illinous at Urbana-Champaign, 1979.

[CGJ97] E. G. Coffman, M. R. Garey, and D. S. Johnson, *Approximation algorithms for bin packing: a survey*, In Hochbaum [Hoc97].

[Chr76] N. Christofides, *Worst-case analysis of a new heuristic for the traveling salesman problem*, Tech. report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, 1976.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to algorithms*, MIT Press, 1990.

[CP^{+}96] 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, Springer, 1996, pp. 646–657.

[CR98] M. Charikar and B. Raghavachari, *The finite capacity dial-A-ride problem*, Proceedings of the 39th Annual IEEE Symposium on the Foundations of Computer Science, 1998.

[CVW97] 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.

[CW98] J. Csirik and G. J. Woeginger, *On-line packing and covering problems*, In Fiat and Woeginger [FW98].

[DGS98] D. R. Dooley, S. A. Goldman, and S. D. Scott, *TCP dynamic acknowledgement delay: Theory and practice*, Proceedings of the 30th Annual ACM Symposium on the Theory of Computing, 1998, pp. 389–398.

[dP01] W. E. de Paepe, Personal communication, June 2001.

[DST87] M. Dror, H. Stern, and P. Trudeau, *Postman tour on a graph with precedence relation on the arcs*, Networks **17** (1987), 283–294.

[EN^{+}99] L. Epstein, J. Noga, S. S. Seiden, J. Sgall, and G. J. Woeginger, *Randomized on-line scheduling on two uniform machines*, Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, 1999, pp. 317–326.

[Fel68] W. Feller, *An introduction to probability theory and its applications*, 3 ed., vol. 1, John Wiley & Sons, Inc., 1968.

[Fel71] W. Feller, *An introduction to probability theory and its applications*, 2 ed., vol. 2, John Wiley & Sons, Inc., 1971.

[FG93] G. N. Frederickson and D. J. Guan, *Nonpreemptive ensemble motion planning on a tree*, Journal of Algorithms **15** (1993), no. 1, 29–60.

[FHK78] G. N. Frederickson, M. S. Hecht, and C. E. Kim, *Approximation algorithms for some routing problems*, SIAM Journal on Computing **7** (1978), no. 2, 178–193.

[FS01] E. Feuerstein and L. Stougie, *On-line single server dial-a-ride problems*, Theoretical Computer Science (2001), To appear.

[FW98] A. Fiat and G. J. Woeginger (eds.), *Online algorithms: The state of the art*, Lecture Notes in Computer Science, vol. 1442, Springer, 1998.

[GH$^+$99] M. Grötschel, D. Hauptmeier, S. O. Krumke, and J. Rambau, *Simulation studies for the online dial-a-ride problem*, Preprint SC 99-09, Konrad-Zuse-Zentrum für Informationstechnik Berlin, March 1999.

[GH$^+$00] M. Grötschel, D. Hauptmeier, S. O. Krumke, and J. Rambau, *Simulation studies for the online control of an automated pallet transportation system*, Presented at Odysseus 2000 First International Workshop on Freight Transportation and Logistics, Chania, Crete, Grece, May 2000, 2000.

[GJ79] M. R. Garey and D. S. Johnson, *Computers and intractability (a guide to the theory of* NP*-completeness)*, W.H. Freeman and Company, New York, 1979.

[GK96] 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.

[GLS88] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric algorithms and combinatorial optimization*, Springer-Verlag, Berlin Heidelberg, 1988.

[Glü00] B. Glück, *Online-Steuerungen automatischer Transportsysteme bei vertretbarer Belastung*, Diplomarbeit, Technische Universität Berlin, 2000, in German.

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

[Gra69] Ronald L. Graham, *Bounds on multiprocessing timing anomalies*, SIAM Journal on Applied Mathematics **17** (1969), 263–269.

[Gua98] D. J. Guan, *Routing a vehicle of capacity greater than one*, Discrete Applied Mathematics **81** (1998), no. 1, 41–57.

[Har72] F. Harary, *Graph theory*, Addison-Wesley Publishing Company, Inc., 1972.

[Hau99] D. Hauptmeier, *Online algorithms for industrial transport systems*, Diplomarbeit, Technische Universität Berlin, 1999, available for download at `http://www.zib.de/groetschel/students/diplomhauptm.pdf`.

[HK$^+$99] D. Hauptmeier, S. O. Krumke, J. Rambau, and H.-C. Wirth, *Euler is standing in line*, Proceedings of the 25th International Workshop on Graph-Theoretic Concepts in Computer Science, Ascona, Switzerland, Lecture Notes in Computer Science, vol. 1665, Springer, June 1999, Journal version to appear in Discrete Applied Mathematics, pp. 42–54.

[HKR00] D. Hauptmeier, S. O. Krumke, and J. Rambau, *The online dial-a-ride problem under reasonable load*, Proceedings of the 4th Italian Conference on Algorithms and Complexity, Lecture Notes in Computer Science, vol. 1767, Springer, 2000, pp. 125–136.

[HKR01] D. Hauptmeier, S. O. Krumke, and J. Rambau, *The online dial-a-ride problem under reasonable load*, Theoretical Computer Science (2001), A preliminary version appeared in the Proceedings of the 4th Italian Conference on Algorithms and Complexity, 2000, vol. 1767 of Lecture Notes in Computer Science.

[Hoc97] D. S. Hochbaum (ed.), *Approximation algorithms for* NP*-hard problems*, PWS Publishing Company, 20 Park Plaza, Boston, MA 02116–4324, 1997.

[Höf01] A. Höft, *Online-Optimierung einer halbautomatischen Glückwunschkarten-Kommissionierungsanlage*, Diplomarbeit, Technische Universität Berlin, 2001, in German.

[HS$^+$97] 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.

[HSW96] L. Hall, D. B. Shmoys, and J. Wein, *Scheduling to minimize average completion time: Off-line and on-line algorithms*, Proceedings of the 7thAnnual ACM-SIAM Symposium on Discrete Algorithms, 1996, pp. 142–151.

[HT84] D. Harel and R. E. Tarjan, *Fast algorithms for finding nearest common ancestors*, SIAM Journal on Computing **13** (1984), no. 2, 338–355.

[IKP92] S. Irani, A. Karlin, and S. Phillips, *Strongly competitive algorithms for paging with locality of reference*, Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms, 1992, pp. 228–236.

[Joh74] D. S. Johnson, *Fast algorithms for bin packing*, Journal of Computer and System Sciences **8** (1974), 272–314.

[Joh01] D. S. Johnson, *A theoretician's guide to the experimental analysis of algorithms*, Preliminary partial draft available at URL: `http://www.research.att.com/~dsj/papers.html`, 2001.

[JRR95] M. Jünger, G. Reinelt, and G. Rinaldi, Network Models, Handbooks in Operations Research and Management Science, ch. The traveling salesman problem, Handbooks in Operations Research and Management Science, Elsevier Science B. V., 1995.

[KdP+01a] 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, 2001.

[KdP+01b] S. O. Krumke, W. E. de Paepe, L. Stougie, and J. Rambau, *Online bincoloring*, Proceedings of the 9th Annual European Symposium on Algorithms, Lecture Notes in Computer Science, 2001.

[KM+88] A. Karlin, M. Manasse, L. Rudolph, and D. D. Sleator, *Competitive snoopy caching*, Algorithmica **3** (1988), 79–119.

[KP94] 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.

[KRW00] S. O. Krumke, J. Rambau, and S. Weider, *An approximation algorithm for the non-preemptive capacitated dial-a-ride problem*, Preprint 00-53, Konrad-Zuse-Zentrum für Informationstechnik Berlin, March 2000.

[KTW96] H. Kellerer, Th. Tautenhahn, and G. J. Woeginger, *Approximability and nonapproximability results for minimizing total flow time on a single machine*, Proceedings of the 28th Annual ACM Symposium on the Theory of Computing, 1996, pp. 418–426.

[Lia80] F. M. Liang, *A lower bound for online bin packing*, Information Processing Letters **10** (1980), 76–79.

[Lip99] M. Lipmann, *The online traveling salesman problem on the line*, Master's thesis, Department of Operations Research, University of Amsterdam, The Netherlands, 1999.

[LL74] J. W. S. Liu and C. L. Liu, *Performance analysis of heterogenous multi-processor computign systems*, Computer architectures and networks (E. Gelenbe and R. Mahl, eds.), North Holland, 1974, pp. 331–343.

[Mot92] R. Motwani, *Lecture notes on approximation algorithms: Volume I*, Tech. Report CS-TR-92-1435, Department of Computer Science, Stanford University, Stanford, CA 94305-2140, 1992.

[MR95] R. Motwani and P. Raghavan, *Randomized algorithms*, Cambridge University Press, 1995.

[Pap94] C. M. Papadimitriou, *Computational complexity*, Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1994.

[PK95] 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.

[PS$^+$97] C. Phillips, 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.

[Ric91] M. B. Richey, *Improved bounds for harmonic-based bin packing algorithms*, Discrete Applied Mathematics **34** (1991), 203–227.

[Rud76] W. Rudin, *Principles of mathematical analysis*, 3 ed., McGraw Hill, 1976.

[Sei00] 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.

[Sei01] S. Seiden, *On the online bin packing problem*, Proceedings of the 28th International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science, Springer, 2001.

[ST85] 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.

[SV88] B. Schieber and U. Vishkin, *On finding lowest common ancestors: Simplification and parallelization*, SIAM Journal on Computing **17** (1988), no. 6, 1253–1262.

[SWW95] D. B. Shmoys, J. Wein, and D. P. Williamson, *Scheduling parallel machines on-line*, SIAM Journal on Computing **24** (1995), no. 6, 1313–1331.

[Tar77] R. E. Tarjan, *Finding optimum branchings*, Networks **7** (1977), 25–35.

[Vaz01] V. Vazirani, *Approximation algorithms*, Springer, 2001.

[Ves94] A. P. A. Vestjens, *On-line machine scheduling*, Ph.D. thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 1994.

[Wei00] S. Weider, *Steuerung von Mehrplatz-Aufzügen*, Master's thesis, Technische Universität Berlin, 2000.

Index

If there is more than one reference, the page number in **boldface** refers to the definition of the entry.