# Online algorithms for transport systems

Diplomarbeit
bei Prof. Dr. M. Grötschel

vorgelegt von Dietrich Hauptmeier
am Fachbereich Mathematik der
Technischen Universität Berlin

Berlin, den 31. Mai 1999

# Acknowledgments

The research presented in this thesis is motivated by a joint project of the Herlitz PBS AG and the Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB). It was partly sponsored by the "Deutsche Forschungsgemeinschaft" (DFG). I appreciate all the support these institutions offered.

A number of persons made this thesis possible:

First of all, I wish to mention *Sven Oliver Krumke* and *Jörg Rambau*, in cooperation with whom the new results presented in this thesis were derived. I could not have wished to be part of a better team. I have learned a lot while working on this project and—almost equally important—I have also had a lot of fun. Thanks!

Special thanks to Jörg Rambau for meticulously reading and re-reading this thesis. He made many valuable suggestions as to the presentation of the material and helped to eliminate many factual, grammatical and typographical errors.

I wish to thank *Martin Grötschel* for supervising this thesis and also for being an outstanding teacher and motivator.

At Herlitz, *Mr. Eckle* and *Mr. Rüstau* proved to be excellent partners, open for discussing new ideas and never failing to provide information.

I further wish to thank everybody from the research group *Combinatorial Optimization* at ZIB for the friendly atmosphere at the institute and also for the many enjoyable discussions at lunch and other occasions.

Dietrich Hauptmeier
Berlin, May 1999

# Contents

# Introduction

Automation of production and transport logistics is an ongoing process in industry. Increasingly complex manufacturing systems have prompted a growing interest in designing efficient control algorithms. Also, knowledge of the difficulties and bottle-necks involved in controlling material flow should yield new insights for designing more efficient layouts for automated logistics systems.

In this thesis, we study aspects of a particular pallet transportation system which is installed at the logistics center of the Herlitz PBS AG in Falkensee near Berlin. We will focus on the control of a subsystem providing vertical transportation between the eight floors of the facility via five elevators. The elevators are on each floor connected to a conveyor belt on which pallets can circle around the elevators. Both for entering and exiting elevators, there are waiting areas, holding one pallet each. The system is described in detail in Chapter 1.

We generalize the problem of scheduling elevators in the following fashion: We consider a server that has to carry objects between points in a metric space. The requests will become known and available at their release times. This problem is the online version of the well-known *Dial-a-ride problem*, which we call ONLINE-DARP. In this thesis we are dealing with the case where there is one server that can carry one object at a time. We do not allow preemption, the server has to deliver an object to its destination once the server has picked up the object, without dropping it in between. We also study a new problem, which we call ONLINE-FIFO-DARP. Here we allow precedence constraints between requests starting at the same point in the space. We use this to model FIFO waiting queues feeding the elevators. ONLINE-DARP and ONLINE-FIFO-DARP also model a number of other tasks in transportation systems, such as the scheduling of stacker cranes in automated storage systems and the control of Automated Guided Vehicles.

A special difficulty is to devise a suitable objective function. The intuitive goal would be to maximize "throughput". In discussions with our partners in industry we formalized this as the task of minimizing average flow times. However, it is also important that the system is "reliable" and "predictable" which leads to the goal of minimizing maximal flow times, to ensure that no request is postponed for too long.

Many of the online algorithms which we developed need to solve offline instances of DARP and FIFO-DARP, where all requests are known and available at the start. For the offline case, all results consider minimizing the completion time of the schedule as objective function. We first prove a new result on the complexity of DARP: We show that DARP on caterpillars is NP-hard. We also show that the same applies to FIFO-DARP. Both DARP and FIFO-DARP on paths can be solved to optimality in polynomial time. We then describe existing combinatorial approximation algorithms for DARP and design new algorithms for FIFO-DARP. Practical tests of one of the algorithms suggest, that for instances of DARP and FIFO-DARP encountered in transport problems these heuristics work very well. We have not studied DARP and FIFO-DARP from a polyhedral perspective[1]. Initially we did formulate a Mixed Integer Program modelling an elevator connected to circular conveyor belts. This particular model, however, contained a number of weakly linked blocks and required "big M" inequalities. These are known to cause difficulties while seeking for integral solutions. We therefore did not further pursue the study of this program. Polyhedral studies of variants of DARP and FIFO-DARP surely present a direction for future research, however the good results achieved with our combinatorial approximation algorithms seem to justify our approach. The results for DARP are described in Chapter 2 and our new results for FIFO-DARP are contained in Chapter 3.

Roughly speaking, there are two paradigms for designing and analyzing algorithms that deal with uncertainty about the future: Firstly, one can make assumptions about the underlying probability distributions of the events and derive stochastical optimization problems. The second approach, used in this thesis, compares solutions computed by online algorithms with optimal offline solutions in a worst-case fashion. This method of designing and analyzing algorithms is called competitive analysis. We believe that the combinatorial nature of the decisions facing algorithms in transport systems cannot be adequately dealt with, when making assumptions on probability distributions: A split second delay in an event can lead to radically different situations, e.g., a new pallet arrives on a floor just after the elevator has started to move away from that floor. In our opinion, such situations are better dealt with using worst-case techniques, such as competitive analysis. Competitive Analysis is described in more detail in Appendix C.

In Chapter 4 we describe algorithms and theoretical results concerning online algorithms for ONLINE-DARP and ONLINE-FIFO-DARP. The main focus is on studying two online strategies: REPLAN computes a new optimal offline schedule, each time a new request arrives. IGNORE on the other hand executes its current schedule and temporarily "ignores" all requests arriving in the mean time. When completing the schedule, IGNORE computes a new schedule serving all the previously ignored requests, then serves this schedule

[1]DARP can be modeled as the well-known Asymmetric Traveling Salesman problem. This approach has been applied in the past to logistical problems, e.g., N. Ascheuer studied Asymmetric Hamiltonian Path Problems with time window constraints with the application of scheduling a stacker crane [Asc95].

and so. Both algorithms are competitive, and they can not be distinguished by their competitive ratios. Furthermore, all the competitiveness results are for the objective of minimizing the total completion time—we prove that the competitive ratio for average and maximal flow times, which we are really interested in, is unbounded for all algorithms.

This together with the desire to find theoretical criteria distinguishing REPLAN and IGNORE lead to the development of a new concept for studying online algorithms in continuously operating systems: In Chapter 5, we introduce the new notion of *reasonable load*. This concept restricts request sequences to sequences that "make sense" in continuously operating systems: There may be no arbitrarily long periods of time, where the requests arriving in this period cannot be served (by an optimal offline algorithm) in a time period at most as long as that period. Using this new concept, we prove that the maximal and average flow times of IGNORE are bounded under reasonable load, which—as we show—is not the case for REPLAN.

We finally tested the online algorithms in simulations of both single elevators fed by FIFO-waiting queues and also in simulations of the integrated elevator system, which motivates this thesis. The results are described in Chapter 6. The simulations suggest, that their is a conflict between the objectives of minimizing average and maximal flow times. Algorithms such as REPLAN achieve very good average flow times, but can have disastrous maximal flow times. On the other hand, IGNORE achieves the best maximal flow times, however its average flow times are only average. Yet, in terms of balancing both objectives, a slightly modified version of IGNORE, called IGGREEDY, seems to be the most suitable algorithm. IGGREEDY inserts ignored requests into its schedule, when they simply substitute empty moves. These experimental results seem to confirm the theoretical results on the performance of IGNORE and REPLAN gained from the new concept of reasonable load.

The simulation of the integrated elevator system generally supports the results concerning the elevator algorithms. However, it also suggests that the layout of the subsystem, with the circular conveyor belt around the elevators, is not suitable. The differences in the performance of the algorithms is very prominent when looking at the individual elevators, however the effects are leveled off when considering the whole system. We believe that this is due to the fact, that pallets moving on the conveyor belt allow only little room for advance planning with respect to which elevator they should use. An interesting project for future research should be to consider other layouts such as longer waiting queues in front of the elevators. We believe that such research might lead to general insights into the design of efficient transport systems.

# Chapter 1

# Description of the Considered Transport System

In this chapter we describe the integrated vertical pallet transportation systems at the Herlitz logistics center in Falkensee near Berlin, which is studied in this thesis.

## 1.1 The Herlitz plant in Falkensee

The Herlitz PBS AG is the major European office supply provider. The company established in 1994 a new logistics and production center in Falkensee near Berlin, investing about 350 million DM. This plant covers an area of 5,000 square meters and employs some 700 people. The center includes a production facility for greeting cards (with an annual output of almost 120 million cards featuring 4,000 different images), a facility for printing gift wrapping paper, a warehouse and various commissioning and loading facilities.

The material flow through the plant is fully automatic. The pallet transportation system consists of truck loading stations, pallet sorters, conveyor belts (horizontal transportation), elevators (vertical transportation), registration and inspection stations, and an automatic storage system. The flow of pallets from the automatic storage system to the truck loading stations is controlled by a central computer unit. Online decisions with respect to control have to be made, e.g., which way a pallet should travel to its destination, or the time when a pallet is to be stored in the automatic storage system. There are constraints restricting the organization of the system, for instance the FIFO-principle which requires that pallets must be delivered in the order of their manufacturing date.

The overall goal is to control the system in such a way that there is a quick and congestion free flow of pallets which satisfies all restrictions. Violating constraints inhibits the throughput. For instance, if the capacity of a route is exceeded this might lead to blocking of the automatic transport system.

The pallets that have to be transported during one day of production are not known in advance. Thus, decisions have to be made *online* without any knowledge of the future. Upon the arrival of a new transportation request a (temporarily) locally optimal control might have bad consequences which even reoptimization can not cure.

Decisions do not only have to be made online, they also must be made within severe time restrictions which brings up the *realtime aspect* of the problem.

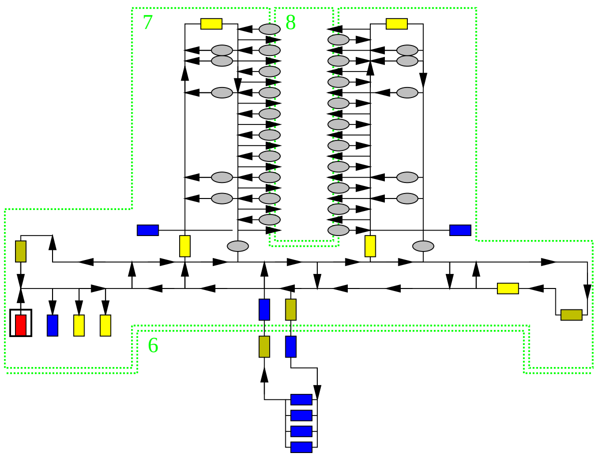

Figure 1.1: Layout of one floor of the pallet transportation system (from [GHKR99]).

Figure 1.1 displays a layout diagram of the pallet transportation system at Herlitz.

A research team of the ”Konrad-Zuse-Zentrum für Informationstechnologie” in Berlin is working in cooperation with partners from the Herlitz PBS AG on studying the mathematical background of controlling this complex transportation system and on developing suitable algorithms which are adequate for the online and real-time nature of the environment.

In this thesis we focus on the control of the elevator system of the facility. The next section describes this subsystem in greater detail.

## 1.2 The integrated elevator system

Attached to the logistics center is a warehouse. On both sides of this warehouse are two elevator towers. Each of these two subsystems consists of five elevators, which are connected to a conveyor belt on each floor. On this conveyor belt, pallets can circle around the elevators. Figure 1.2 shows a top view of one of the eight floors. The arrows indicate the direction of the conveyor

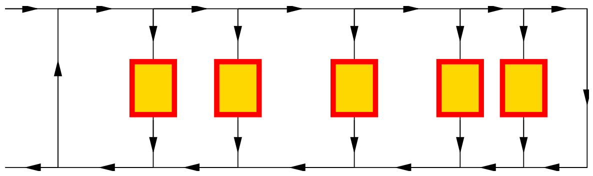
Figure 1.2: Top view of one of the eight floors (from [GHKR99]).

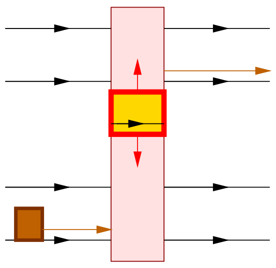
Figure 1.3: Horizontal view of an elevator (from [GHKR99]).

belts which move pallets between the five elevators. A schematic horizontal view of a single elevator is shown in Figure 1.3.

Each elevator can carry one pallet at a time. For each elevator there is on each floor a waiting space for entering pallets and a waiting space for exiting pallets. All waiting spaces can hold one pallet. Part of one of the circular conveyor belts together with the switches leading to the elevators is shown in Figure 1.4.

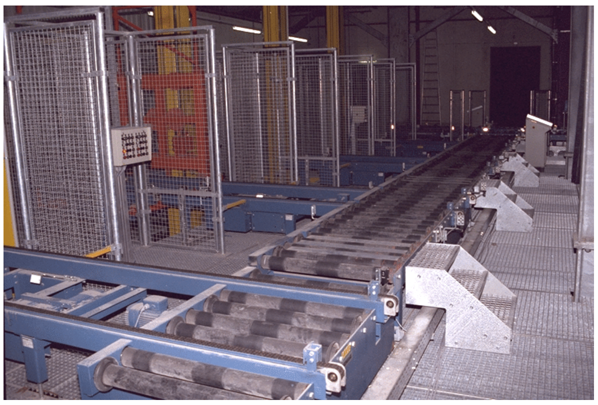
Figure 1.4: One of the circular conveyor belts of the elevator system (courtesy of Herlitz PBS AG).

The conveyor belt for the pallets works in a discrete fashion: It is separated into segments of the size of one pallet. A pallet moves from one segment to the next segment only if the next segment is empty. This can lead to a blockage of the conveyor system, if all segments are currently filled. Figure 1.5 shows part of a conveyor belt at the facility in action.

An important aspect of the system is that horizontal transportation within the elevator subsystem is much slower than vertical transportation: A pallet takes about 4s to move from one floor to another (not counting another 5s for boarding the elevator and the same time for exiting from the elevator). However traveling from the system entrance to the first elevator takes 15s and completing a whole journey around the conveyor belt takes more than 2 minutes.

On each of the eight floors there is one entrance and one exit connecting the elevator subsystem to the plant. On the second and on the third floor there are connections to the ten aisles of the storage system which is shown in Figure 1.6. Here pallets are picked up or delivered by the stacker cranes serving the warehouse.

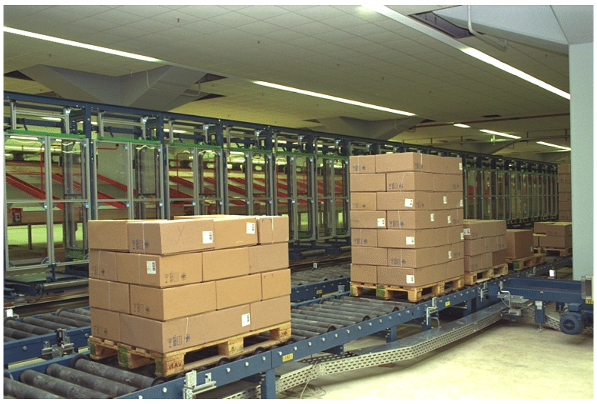

Figure 1.5: A conveyor belt in action (courtesy of Herlitz PBS AG).

## 1.3 Algorithms used to control the elevator system

Within the elevator subsystem, there are two tasks that require algorithmic control: Firstly, decisions need to be made concerning which elevator a pallet should use. Secondly, the elevators need to be scheduled. Currently, the pallets are assigned to the elevators using a simple first-fit type algorithm. Whenever a pallet that needs to be transported to a different floor passes an empty elevator waiting area, it will enter this area and request transportation from the corresponding elevator. Additionally, the current Herlitz strategy requires, that the waiting area for exiting pallets of the respective elevator is empty on the destination floor of the pallet.

Elevators are also controlled using a first-fit type strategy. Whenever an elevator has delivered a pallet, it will next serve the pallet which is located on the nearest floor, i.e., the algorithm tries to minimize in a greedy fashion the length of it's empty moves. To guard against single pallets being left for

Figure 1.6: The automatic storage system (courtesy of Herlitz PBS AG).

an arbitrarily long time, each pallet is associated with a counter. This counter is increased by one each time the elevator chooses another pallet for serving and leaves this pallet waiting. When the counter of a pallet reaches a certain threshold, the elevator has to serve this pallet next.

Clearly, a setting of this threshold parameter which is too low, leads to a FIFO pattern, where all pallets have to be served in order of their arrival in the waiting areas. On the other hand, if the parameter is set too high, it will fail to achieve the desired result.

## 1.4 Objectives for the elevator system

Discussion with our partners in industry showed that there are two main objectives for the elevator system, and, in fact, the entire transport system. These two objectives try to reflect the intuitive goals of "maximal throughput", which proves difficult to formalize in a discrete system, and of "reliability" in the sense of "predictable behavior".

- Global efficiency—pallets should arrive quickly. This can be modeled by minimizing average flow times for pallets.

- Reliability—pallets traveling under similar conditions should take about the same time for their journey. This can be modeled by minimizing maximal flow times.

It seems intuitively clear, that there is a conflict between the two objectives. This is confirmed by our simulation studies, which we describe in Chapter 6: Algorithms which do particularly well with respect to one objective, show a less satisfying behavior in terms of the other objective.

# Chapter 2

# Dial-a-ride Problems

In this chapter we study a variant of the combinatorial Dial-a-ride problem. With DARP we denote the case where a single server with unit capacity moves on a graph. We are dealing with closed schedules, i.e., the server has to return to its origin. The objective is to minimize overall server movements.

We will first show that DARP can also be expressed as a graph augmentation problem. We then discuss balancing, which is a general technique for tackling DARP on paths and trees introduced in [AK88] and [FG93]. From this we derive the polynomial solvability of our variant of DARP on paths. We then show that the problem is NP-hard on caterpillar graphs. This is a new result, which strengthens the complexity results for this variant of DARP from [FG93], where it is shown that the variant is NP-hard on trees. Next, we explain the approximation algorithms for this problem on general graphs from [FHK78] and also on trees, published in [FG93].

We summarize the main results for DARP discussed in this chapter:

| | Hardness | Best approx. result |
|---|---|---|
| Path | P | – |
| Caterpillar | NP-hard | 5/4 |
| Tree | NP-hard | 5/4 |
| Graph | NP-hard | 9/5 |

## 2.1 DARP on graphs

We study in this chapter the subclass $1, cap1||G|\sum m$ of the Dial-a-ride problem, employing the notation described in Appendix B. We use the term DARP in this thesis to denote this particular case of the general problem. The space for this subclass is a (finite) graph with edge weights, as indicated by the entry $G$. The entry $1$ specifies, that a unique server moves on this graph. The server has to deliver a (finite) set of transportation requests. We consider the requests to be given as arcs in the graph, directed from their source vertex to their destination vertex. The server has unit capacity, as indicated by the entry *cap1*. This means that it can only serve one request at a time. The entry $\sum m$ specifies that the objective is to find a shortest transportation for the jobs. We also require the server to start and end its movements at a designated location.

### 2.1.1 DARP on graphs is a graph augmentation problem

We show in this section that DARP can be formulated as a graph augmentation problem. To this end, we will first define more formally what we mean by DARP. For notation from graph theory see Appendix A.1.

**Definition 2.1.1 (Dial-a-Ride Problem (DARP)).** The input for DARP consists of a finite mixed graph $G = (V, E, A)$, an origin vertex $o \in V$ and a nonnegative cost function $c\colon E \cup A \to \mathbb{R}_{\geq 0}$. It is assumed that $G[E]$ is connected and contains all endpoints of arcs from $A$. Also, for any arc $a = (u, v) \in A$, its cost $C(a)$ equals the length of a shortest path from $u$ to $v$ in $G[E]$.

The goal of DARP is to find a closed walk in $G$ of minimum cost which starts in $o$ and traverses each arc in $A$.

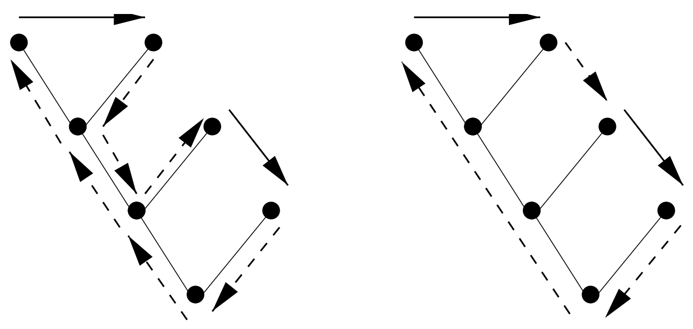

Figure 2.1: An instance of DARP as a graph augmentation problem. The dotted arcs are a solution with arcs from $A(E)$ and a solution from $V \times V$ respectively.

An important observation is that DARP can be equivalently formulated as a *graph augmentation problem* as illustrated in Figure 2.1. Let $A(E)$ be the set of arcs containing a pair of antiparallel arcs between the endpoints of each edge in $E$. We can then extend the cost function $c$ to $A(E)$ in a natural way by defining the cost of an arc in $A(E)$ to be the cost of the corresponding undirected edge in $E$.

Let $W$ be any feasible solution to a given instance of DARP, i.e., a closed walk that starts in $o$ and traverses each arc in $A$. Then $W$ induces a multiset $S$ of arcs in the following way: For each time an undirected edge $e = [u, v] \in E$ is traversed by $W$ from $u$ to $v$, the multiset $S$ contains a copy of the directed arc $(u, v)$. The graph $G[A \cup S]$ consisting of the arcs in $A \cup S$ and their endpoints (which include the origin $o$) is then Eulerian. This follows from the following observation: When tracing $W$ and traversing for each undirected edge in $W$ the corresponding directed arc in $S$ in the respective direction we get an Eulerian cycle.

Conversely, let $S$ be a multiset of arcs from $A(E)$ such that $G[A \cup S]$ is Eulerian and contains the origin $o$. Then we can easily obtain a feasible solution $W$ for our instance of DARP as follows: Choose an Eulerian cycle $C$ in $G[A \cup S]$ which starts and ends in $o$. The walk $W$ starts at $o$ and then follows $C$. If the current arc $r$ from $C$ is in $A$ then $W$ traverses this arc in $G$, otherwise $W$ traverses the undirected edge corresponding to $r$.

Thus, any feasible solution for DARP corresponds to an augmenting multiset $S$ of arcs such that $G[A \cup S]$ is Eulerian and contains $o$ and vice versa. This enables us to reformulate DARP equivalently as the problem of finding a multiset $S$ containing arcs from $A(E)$ minimizing the cost $C(A \cup S)$ such that $G[A \cup S]$ is Eulerian and contains $o$.

We can also allow augmentation with arcs from $V \times V$ as illustrated in Figure 2.1. We extend the cost function $c$ to arcs from $V \times V$ by defining the cost of an arc to be the cost of a shortest path in $G[E]$ from its source to its destination.

**Definition 2.1.2 (Graph augmentation version of DARP).** An instance of the problem DARP $I = (G = (V, E, A), C, o)$ consists of a finite mixed graph $G = (V, E, A)$, an origin vertex $o \in V$ and a non-negative cost function $c \colon E \cup V \times V \to \mathbb{R}_{\geq 0}$. It is assumed that for any arc $a = (u, v) \in V \times V$ its cost $C(a)$ equals the length of a shortest path from $u$ to $v$ in $G[E]$.

The goal of DARP is to find a multiset $S$ of arcs in $V \times V$ minimizing $C(A \cup S)$ such that $G[A \cup S]$ is Eulerian and contains the vertex $o$.

### 2.1.2 Simplifying technical assumptions

We start with some technical assumptions about the input instances depending on the structure of the undirected graph $G[E]$ given in an instance $I = (G = (V, E, A), C, o)$ of DARP. While all these assumptions are without loss of generality they greatly simplify the presentation of our algorithms in Sections 2.3, 2.5 and 2.6.

Suppose that $G[E]$ is a tree. Let $v \in V \setminus \{o\}$ be a vertex of degree at most two in $G[E]$ which is neither source nor target of an arc from $A$. If the degree of $v$ is one, i.e., if $v$ is a leaf, we can remove $v$ and its incident edge without affecting the optimal solution. Similarly, if the degree of $v$ is two, we can replace $v$ and its incident edges by a single edge with cost equal to the sum of the two edges. Thus, for trees we can make the following assumption without loss of generality:

**Assumption 2.1.3 (Technical assumption for DARP on trees).** *Each vertex $v \in V$ of degree one or two is either the origin $o$ or incident to at least one arc from $A$.*

If $G[E]$ is a path, then every vertex has degree one or two. It is easy to see that in this case we can make an even stronger assumption without loss of generality:

**Assumption 2.1.4 (Technical assumption for DARP on paths).** *Each vertex $v \in V$ is either the origin $o$ or it is incident to at least one arc from $A$.*

We now turn to DARP on general graphs.

**Assumption 2.1.5 (Technical assumption for DARP on general graphs).**

*(i)* *Each vertex $v \in V$ is incident to at least one arc from $A$.*

*(ii)* *$G[E]$ is complete and the cost function $c$ obeys the triangle inequality, i.e., for any edge $[u, v] \in E$ the cost $C(u, v)$ does not exceed the length of a shortest path in $G[E]$ between $u$ and $v$.*

Note that Assumption 2.1.5 can be enforced without increasing the value of an optimal solution. If the start vertex $o$ is not incident to any arc from $A$ we can add a new vertex $o'$, a new arc $(o, o')$ and a new edge $[o, o']$, each of cost zero. The new vertex $o'$ is joined by undirected edges to all neighbors of $o$ where the cost of an edge $[o', v]$ equals $C(o, v)$. We can then remove vertices which are not source or target of an arc and for every pair $u$ and $v$ of vertices insert new edges of cost equal to the shortest path in $G[E]$ between $u$ and $v$.

Assumption 2.1.5 can not be made without loss of generality for DARP on trees, since removing vertices and later completing the graph as described will in general destroy the "tree-property". For further properties of DARP on trees see Section 2.5.1.

## 2.2 Balancing

An important concept for tackling DARP on paths and trees is *balancing*. Balancing is based on the observation that every edge in a tree is a cut edge. This implies that any closed walk on a tree has to traverse each edge as often in one direction as it traverses it in the other direction. Given an instance of DARP, we can exploit this observation and construct a multiset of balancing arcs which is contained in an optimal solution. Balancing multisets have been studied in [AK88, FG93].

### 2.2.1 Balancing DARP on trees

We begin by formally defining the concept of a balancing multiset. We then will explain a simple method for constructing a balancing multiset for a given instance of DARP on a tree, such that the balancing multiset is contained in an optimal solution.

**Definition 2.2.1 (Balancing multiset).** Let $G = (V, A)$ be a directed multigraph. A multiset $B$ of arcs from $V \times V$ is called a *balancing multiset* if in $H = G[A \cup B]$ we have $d_H^+(v) = d_H^-(v)$ for all vertices $v \in H$.

We now describe a method for generating a balancing set which is illustrated in Figure 2.2. Suppose that for a mixed graph $G = (V, E, A)$ the graph $G[E]$ is a tree and that Assumption 2.1.3 is satisfied. Let $[x, y]$ be an arbitrary edge from $E$. The removal of $[x, y]$ cuts $V$ into the sets $X$ and $Y := V \setminus X$ with $x \in X$ and $y \in Y$. Any closed walk $W$ in $G = (V, E, A)$ which traverses each arc from $A$ must traverse the cut $(X, Y)$ the same number of times in each direction. Denote by $\Phi(X, Y)$ the number of arcs emanating from $X$, i.e., $\Phi(X, Y) := |\{r = (x, y) \in A | x \in X, y \in Y\}|$. Hence, $W$ must traverse edge $[x, y]$ from $x$ to $y$ at least $b(x, y)$ times, where

$$b(x,y) := \begin{cases} 1 & \text{if } \phi(X,Y) = \phi(Y,X) = 0 \\ \phi(Y,X) - \phi(X,Y) & \text{if } \phi(Y,X) > \phi(X,Y) \\ 0 & \text{otherwise.} \end{cases}$$

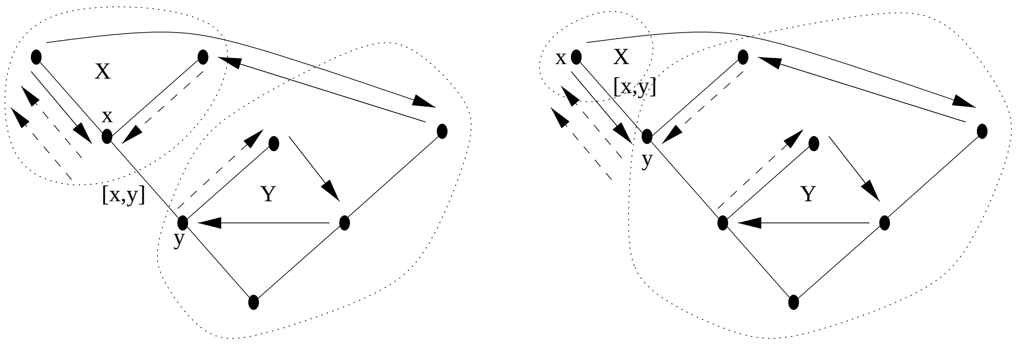

Figure 2.2: An instance of DARP on a tree with a balancing set (dashed arcs). On the left hand side, the edge $[x, y]$ defines vertex sets $X$ and $Y$ (dotted regions) with $\Phi(X, Y) = \Phi(Y, X) = 1$. Therefore $b(x, y) = b(y, x) = 0$, and no balancing arc is added. On the right hand side, we have $\Phi(X, Y) = 2$ and $\Phi(Y, X) = 0$, therefore $b(x, y) = 0$ and $b(y, x) = 2$, two balancing arcs from $y$ to $x$ are added.

The above observation has the following consequence for the equivalent graph augmentation version: If $B$ is a multiset of arcs from $A(E)$ such that $B$ contains exactly $b(x, y)$ copies of the directed arc $(x, y)$, then for every solution $S$ which uses only arcs from $A(E)$ we have that $B \subseteq S$. This leads to the following lemma, which is proved in [AK88].

**Lemma 2.2.2.** *Let* $I = (G = (V, E, A), C, o)$ *be an instance of* DARP *such that* $G[E]$ *is a tree. Then in time* $\mathcal{O}(nm_A)$ *one can find a balancing set* $B$ *of arcs from* $A(E)$ *such that* $B \subseteq S$ *for every feasible solution* $S$ *which uses only arcs from* $A(E)$. □

### 2.2.2 An $\mathcal{O}(n + m_A)$ time algorithm for balancing on trees

In the last section we showed how to compute a balancing multiset with arcs from $A(E)$. We will now consider the general case where solutions contain arcs from $V \times V$.

Lemma 2.2.2 obviously does not apply when allowing augmentation with arcs from $V \times V$: Any directed path $P$ of arcs in a balancing set $B$ can be replaced with a single arc which has the same cost. If there is for every arc in $P$ a parallel arc with the same endpoints in $B$, then the new balancing set obviously visits the same vertices and is still a balancing set. Given a solution $S$ containing $B$, we can also in $S$ replace $P$ with a single arc and have constructed a solution which does not contain $B$. However it is intuitive that these balancing sets are in some way equivalent, since "embedding" them into a closed walk in $G[E]$ yields the same walk.

We now define formally what we mean by an embedding and will use this to derive an equivalence relation between arc multisets. The two concepts are illustrated in Figure 2.3.

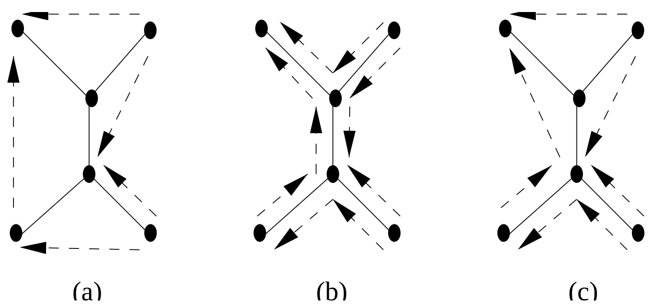

Figure 2.3: The arcs of the graph in (b) are an embedding of both the arcs in (a) and (c) into the set of edges. The multiset of arcs in (a) and (c) are equivalent, they are however not equivalent with the multiset of arcs in (b).

**Definition 2.2.3 (embedding).** Let $G = (V, E, A)$ be a mixed graph. A multiset $D$ of arcs from $A(E)$ is an embedding of $A$ into $E$ if

1. for every arc $a = (u, v) \in A$ there is a path $P_a$ in $D$ from $u$ to $v$,
2. the multiset $D$ is the disjoint union of all these paths, $D = \bigcup_{a \in A} P_a$.

Let $c$ be a cost function $c\colon E \to \mathbb{R}_{\geq 0}$. A multiset $D$ of arcs from $A(E)$ is a *minimum cost embedding* of $A$ into $E$, if it is an embedding and all the paths $P_a$ have minimum cost.

It is easy to see that an embedding is not unique if $G[E]$ is a general graph. However, if $G[E]$ is a tree, the path connecting any two vertices in this graph is unique, and therefore an embedding into $E$ is also unique.

We now define an equivalence relation on multisets of arcs.

**Definition 2.2.4 (equivalence).** Let $G = (V, E)$ be a graph and let $c$ be a cost function $c\colon E \to \mathbb{R}_{\geq 0}$. Let $N, M$ be multisets of arcs from $V \times V$. Then $N$ and $M$ are equivalent ($N \sim M$) if

1. there exists a multiset $D$ of arcs from $A(E)$ which is a minimum cost embedding of both $N$ and $M$,
2. the set $V(N)$ containing all endpoints of arcs in $N$ equals the set $V(M)$ containing all endpoints of arcs in $M$.

We will now derive the result, that for a solution to a DARP, all multisets of arcs equivalent to this set are also solutions of the same cost. However, we notice that even on paths not all optimal solutions are equivalent multisets!

**Lemma 2.2.5.** *Let $I = (G = (V, E, A), C, o)$ be an instance of* DARP *and let $S$ be a solution. Let $D$ be a minimum cost embedding of $S$. Then $D$ is also a solution and $C(S) = C(D)$.*

*Proof.* By definition of embedding, each arc $s \in S$ corresponds to a path $P_s \in D$ and $D = \bigcup_{s \in S} P_s$. In an Euler tour through $A \cup S$ we replace each arc $s \in S$ by the corresponding path $P_s$. This yields an Euler tour traversing $A \cup D$. The claim $C(S) = C(D)$ follows from the way we extended the cost function from $E$ to $V \times V$: The cost $C(s)$ of an arc $s = (u, v) \in S$ was defined as the cost of a shortest path from $u$ to $v$ in $E$. However the path $P_s$ is a shortest path from $u$ to $v$ in $A(E)$, therefore $C(s) = C(P_s)$. This yields $C(S) = C(D)$. □

**Corollary 2.2.6.** *Let $I = (G = (V, E, A), C, o)$ be an instance of* DARP *and let $S$ be a solution of* DARP*. Let $C$ be a submultiset of $S$ and $C'$ be a multiset of arcs such that $C' \sim C$. Then $S' := (S \setminus C) \cup C'$ is also a solution of* DARP *and $C(S) = C(S')$.*

*Proof.* By definition of equivalence we can find a multiset of arcs from $A(E)$ which is an embedding of both $S$ and $S'$. The claim follows therefore directly from Lemma 2.2.5. □

Notice that the converse of Corollary 2.2.6 is not true: Two solutions of an instance of DARP with the same cost are not necessarily equivalent. This holds even for instances of DARP on trees, if there are edges of cost zero.

We use Lemma 2.2.5 together with the earlier results on balancing for DARP restricted to $A(E)$ to prove the following result from [AK88]:

**Lemma 2.2.7.** *Let $I = (G = (V, E, A), C, o)$ be an instance of* DARP *such that $G[E]$ is a tree. Then in time $\mathcal{O}(nm_A)$ one can find a balancing multiset $B$ of arcs from $V \times V$ such that there exists an optimal solution $S^*$ with $B \subseteq S^*$.*

*Proof.* Let $S^*$ be an optimal solution. Let $S'$ be the embedding of $S^*$ into $E$. Then by Lemma 2.2.5 $S'$ is also an optimal solution. However $S'$ uses only arcs from $A(E)$, therefore by Lemma 2.2.2 we can find in time $\mathcal{O}(nm_A)$ a balancing multiset $B$ contained in an optimal solution. □

```
Input:  A mixed graph $G = (V, E, A)$, such that $G[E]$ is a tree
1 Choose an arbitrary leaf $r$ of tree $G[E]$ and root the tree relative to this vertex (that is,
  calculate the parent of each vertex other than $r$).
2 Let $B = \emptyset$. Compute the function $b(x, y)$ for each ordered pair of adjacent vertices in $G[E]$
  using procedure calculate-b.
3 For each $b(x, y) > 0$, add to $B$ an arc from $x$ to $y$ and subtract 1 from $b(x, y)$.
4 Add a new vertex $r'$, connected via an edge to the root $r$ of the tree. Set $b(r, r') =$
  $b(r', r) = 0$.
5 Let $B_1$ be the set of edges with $b(u, v) > 0$ and $u$ is a parent of $v$, let $B_2$ be the set of all
  other edges with $b(u, v) > 0$.
6 Add balancing arcs to $B$ generated by calling algorithm arcs-towards-root($r', r$) using the
  edges in $B_1$.
7 Generate balancing arcs using the edges in $B_2$ with algorithm arcs-away-from-root($r', r$).
8 return $B$ which is a multiset of $\mathcal{O}(n + m_A)$ balancing arcs for $G = (V, E, A)$.
```

**Algorithm 1:** Algorithm Balance for DARP on trees.

We will now discuss Balance (Algorithm 1) as described in [AK88, FG93]. This algorithm calculates a balancing multiset with arcs from $V \times V$ and has an improved time bound of $\mathcal{O}(n + m_A)$. The balancing multiset $B$ returned by the algorithm has at most $\mathcal{O}(n + m_A)$ elements.

In the first step of Balance the values $b(u, v)$ for all $(u, v) \in A(E)$ are calculated. This can be accomplished using procedure compute-b (Algorithm 2). Clearly the algorithm correctly calculates the values of $b$. Note that the modified version of compute-$\Phi$ referred to in Step 14 is basically the same algorithm as compute-$\Phi$, only replacing all occurrences of *destcount* with *sourcecount* and vice versa. We will now consider the time complexity of algorithm compute-b.

**Lemma 2.2.8.** *The algorithm compute-b runs in time $\mathcal{O}(n + m_A)$, where $n = |V|$ and $m_A = |A|$.*

*Proof.* The tree $G[E]$ can be rooted at $r$ in time $\mathcal{O}(n)$. Using $\mathcal{O}(n)$ preprocessing time, the nearest common ancestor of any pair of vertices can be found in $\mathcal{O}(1)$ time [SV88, HT84]. Replacing some arcs in $A$ and partitioning the resulting multiset into $A_1$ and $A_2$ can therefore be accomplished in $\mathcal{O}(m_A)$ time. Hence all the preprocessing steps take together $\mathcal{O}(n + m_A)$ time.

Calculating the values for *destcount* and *sourcecount* takes $\mathcal{O}(m_A)$ time. The recursive procedure compute-$\Phi$ takes $\mathcal{O}(1)$ time for its calculations. Including recursive calls, compute-$\Phi$ is executed twice for each edge. Therefore the algorithm compute-b runs in time $\mathcal{O}(n + m_A)$. $\square$

After calculating the values of $b$, in Step 3 algorithm Balance adds one copy of each arc in $A(E)$ which is contained in the embedding of the balancing arcs into $A(E)$. This ensures that this algorithm returns a balancing multiset with the same set of endpoints as the balancing set using arcs from $A(E)$ considered in the last section.

Now the other balancing arcs are computed, using two recursive procedures, arcs-towards-root and arcs-away-from-root, which respectively compute

```
Input:  A mixed graph $G = (V, E, A)$, and a vertex $r \in V$, such that $G[E]$ is a tree with
        root $r \in V$
 1 for all $(u, v) \in A$ do
 2   Let $n \in V$ be the nearest common ancestor of $u$ and $v$ in $G[E]$.
 3   if $n$ is neither $u$ nor $v$ then
 4     Replace $a$ by two arcs, one from $u$ to $n$ and one from $n$ to $v$.
 5   end if
 6 end for
 7 Partition the arcs into multiset $A_1$ containing arcs directed towards the root and multiset $A_2$
   of arcs directed away from the root.
 8 for all $v \in V$ do
 9   Set $destcount(v)$ to be the number of arcs in $A_1$ that have $v$ as destination vertex and
     set $sourcecount(v)$ to be the number of arcs in $A_1$ emanating from $v$.
10 end for
11 for all $v \in V$ such that $[r, v] \in E$ do
12   $\Phi(r, v) =$ compute-$\Phi(r, v)$
13 end for
14 Repeat steps 8 and 11 with the arcs in $A_2$ using a modified version of compute-$\Phi(r, v)$ that
   deals with arcs directed away from the root.
15 for all $[u, v] \in E$ do
16   Calculate $b(u, v)$ from $\Phi(u, v)$ and $\Phi(v, u)$ using formula 2.2.1.
17 end for
18 return $b$
```

**Algorithm 2:** Procedure compute-b of algorithm Balance

```
Input:  A pair of vertices $u$ and $v$ such that $[u, v] \in E$.
Input:  A mixed graph $G = (V, E, A)$, such that $G[E]$ is a tree with root $r \in V$,
        values $sourcecount$ and $destcount$ for each vertex and values $\Phi(x, y)$ for each
        ordered pair of adjacent vertices $(x, y)$.
 1 Set $\Phi(u, v) = sourcecount(v) - destcount(v)$.
 2 for all $w \in V$ such that $w$ is child of $v$ do
 3   Compute $\Phi(v, w) =$compute-$\Phi(v, w)$.
 4   Increment $\Phi(u, v)$ by $\Phi(v, w)$.
 5 end for
 6 return $\Phi$
```

**Algorithm 3:** Procedure compute-$\Phi$ of algorithm Balance

balancing arcs directed towards the root and balancing arcs directed away from the root. We will only discuss procedure arcs-towards-root. The procedure arcs-away-from-root is similar.

Algorithm arcs-towards-root keeps for each vertex $v \in V$ a list of vertices $L(v)$ containing possible initial vertices for balancing arcs whose embedding into $E$ contains the arc $(u, v)$ where $u$ is the parent of $v$. We assume that these lists are empty when the procedure is called first. Recursive procedure calls will then manipulate the lists. Note that any vertex added to a list will become an initial vertex of a balancing arc. Also note that for any vertex $v$, either copies of $v$ are added to the list $L(v)$ or balancing arcs with destination $v$ are generated, but not both.

**Lemma 2.2.9.** *Let* $I = (G = (V, E, A), C, o)$ *be an of* DARP *where* $G[E]$ *is a tree. Then algorithm* Balance *calculates in time* $\mathcal{O}(n + m_A)$ *(where* $n = |V|$ *and* $m_A =$

```
Input:  An arc $(u, v) \in B_1$.
Input:  A mixed graph $G = (V, E, A)$, such that $G[E]$ is a tree with root $r \in V$, a
        multiset of arcs $B_1 \subset A(E)$, values $b(x, y)$ for each $(x, y) \in B_1$ and a list of
        vertices $L(v)$ for each vertex $v \in V$.
if $v$ is a leaf then
    Append $b(v, u)$ copies of $v$ to $L(v)$.
else
    Set $L(v) = \emptyset$.
    for all $w \in V$ such that $w$ is a child of $v$ do
        Call arcs-towards-root$(w, v)$.
        Append $L(w)$ to $L(v)$.
    end for
    if $b(v, u) > |L(v)|$ then
        Append $b(v, u)$ copies of $v$ to $L(v)$.
    end if
    if $b(v, u) < |L(v)|$ then
        for $|L(v)| - b(v, u)$ elements $x$ from the list $L(v)$ do
            Add a balancing arc $(x, v)$ to $B$.
            Remove $x$ from $L(v)$.
        end for
    end if
end if
return $B$
```

**Algorithm 4:** Procedure arcs-towards-root of algorithm Balance

$|A|$*) a balancing multiset* $B$ *of arcs from* $V \times V$ *such that* $B \subseteq S^*$ *for some optimal solution* $S^*$. *The multiset* $B$ *has* $\mathcal{O}(n + m_A)$ *elements.*

*Proof.* First, we note that the artificial edge $[r, r']$ with $b(r, r') = 0$ is the last edge considered by the procedure arcs-towards-root. Therefore the procedure generates an arc for each initial vertex that has been inserted into one of the lists.

It is easy to see that the multiset of arcs $B$ generated by the algorithm has the property that for the multiset $A \cup B$ and for all ordered pairs of adjacent vertices $(u, v)$ the value $b(u, v) = 0$. This implies that $B$ is a balancing multiset.

Let $\bar{B}$ be the balancing set from $A(E)$ considered in the last section. We already noticed that Step 3 of Balance ensures that $B$ and $\bar{B}$ have the same set of endpoints. The procedures arcs-towards-root and arcs-away-from-root yield arcs which are chains of arcs from $\bar{B}$. In fact, the algorithm generates a set $B$ such that $B \sim \bar{B}$. With Corollary 2.2.6 we conclude that there is an optimal solution that contains $B$.

We now show that the multiset $B$ generated by Balance contains at most $\mathcal{O}(n + m_A)$ elements. In Step 3, algorithm Balance adds $\mathcal{O}(n)$ arcs to $B$. In each connected component in $G[A \cup B]$, no more than half the arcs are balancing arcs generated by arcs-towards-root. Otherwise in an Euler tour of the component, two arcs generated by arcs-towards-root would have to be traversed directly after each other. This is not possible, since we already noted that a vertex can never be both start and stop vertex for arcs generated by arcs-towards-root.

The same argument applies to the number of arcs generated by arcs-away-from-root. We also notice that the targets of arcs generated by arcs-away-from-root can not be sources of arcs generated by arcs-towards-root. If this would be the case, then the edge from this vertex towards the root would be traversed in both directions by the two shortest path connecting the endpoints of both arcs. This, however, can not happen due to the definition of $b$. Therefore the number of balancing moves generated by the two procedures is $\mathcal{O}(m_A)$. This yields an overall number of arcs of $\mathcal{O}(n + m_A)$.

We finally consider the time that Balance takes. For this discussion we assume the following times for each list operation: For each list, we save additionally its size. The size is updated in $\mathcal{O}(1)$ for each list operation. Appending a list to another list takes $\mathcal{O}(1)$ time and so does adding one element and also removing the first element. We already know that procedure compute-b takes $\mathcal{O}(n+m_A)$ time. Step 3 of algorithm Balance clearly takes $\mathcal{O}(n)$ time. We now consider the time for executing arcs-towards-root and arcs-away-from-root.

The algorithm appends at each vertex $v \in V$ at most $d(v)$ lists to $L(v)$ where $d(v)$ is the degree of the vertex. This yields $\mathcal{O}(n)$ appending operations. For each arc generated the algorithm once inserts the initial vertex into a list and once deletes it from the list and generates the arc. We already showed that the two procedures generate $\mathcal{O}(n + m_A)$ arcs. We conclude that algorithm Balance runs in time $\mathcal{O}(n + m_A)$. □

## 2.3 A polynomial time algorithm for DARP on paths

In this section we give a polynomial time algorithm for DARP on paths, which was presented in [AK88]. To this end, let $G = (V, E, A)$ be a mixed graph such that $G[E]$ is a path. We assume throughout this section that Assumption 2.1.4 holds, i.e., each vertex is either the origin or it is incident to at least one arc.

The algorithm uses the *component graph* $\hat{G}[A] = (\hat{V}, \hat{E})$ of a multiset $A$ of arcs. This graph is constructed as follows: $\hat{V}$ is the set of all strongly connected components $G_i$ in $G[A]$. For every vertex $v \in V$ that is not contained in one of the connected components, add a vertex $G_i$ to $\hat{V}$. Add an edge to $\hat{E}$ between two vertices $G_i$ and $G_j$ in $\hat{V}$ if there is an edge in $G[A \cup S]$ from some vertex in $G_i$ to some vertex in $G_j$. Its cost is set to the shortest edge connecting $G_i$ and $G_j$.

Note that the number of edges in $\hat{G}[A]$ is at most $|E|$, the number of edges in the original graph. If $G[E]$ is a tree, this implies that there are at most $n - 1$ edges in $\hat{G}[A]$ where $n = |V|$.

**Theorem 2.3.1.** *Algorithm OptPath finds an optimal solution for* DARP *on paths.*

*Proof.* Clearly the algorithm constructs a feasible solution to the problem. It remains to show optimality. Let $S^*$ be an optimal solution such that $B \subseteq S^*$.

```
Input: A mixed graph $G = (V, E, A)$, such that $G[E]$ is a path, a cost function $c$ on $E$,
       an initial vertex $o \in V$
1 Compute a balancing multiset $B$ of arcs from $V \times V$ such that $B \subseteq S^*$ for some optimal
  solution $S^*$.
2 Construct the graph of strongly connected components $\hat{G}[A \cup B] = (\hat{V}, \hat{E})$ of $G[A \cup B]$.
3 Compute a minimum spanning tree $T$ of $\hat{G}$.
4 Let $N = \emptyset$. For each edge in $T$ add a pair of antiparallel arcs between the endpoints of the
  corresponding edge in $G[E]$ to $N$.
5 return the multiset $B \cup N$.
```

**Algorithm 5:** Algorithm OptPath for DARP on paths.

Then $N^* = S^* \setminus B$ is a multiset of arcs from $V \times V$ which strongly connect the components of $G[A \cup B]$. Since $A \cup B$ is balanced and $G[A \cup S^*]$ is Eulerian (and therefore also balanced), we can conclude that the multiset $N^*$ is balanced. Since the underlying graph is a path, every edge is a cut edge. This implies that the embedding of $S^* \setminus B$ into $E$ consists of pairs of antiparallel arcs. The edges in $\hat{G}[A \cup B]$ traversed by arcs in the embedding of $S^* \setminus B$ into $E$ connect all vertices in $\hat{G}$ and hence contain a spanning tree. The cost of $N^*$ is therefore at least twice the cost of a MST in $\hat{G}[A \cup B]$. Since the multiset $N$ calculated by OptPath has exactly twice the cost of an MST, the multiset $B \cup N$ is an optimal solution. □

We briefly comment on the running time of algorithm OptPath for DARP. Computing a balancing multiset $B$ can be accomplished in time $\mathcal{O}(n + m_A)$ (where $n = |V|$ and $m_A = |A|$) using algorithm Balance. The graph of connected components can be constructed in $\mathcal{O}(n + m_A)$ time. Let $q$ be the number of connected components. The MST algorithm from [FT84] takes time $\mathcal{O}(n\beta(n, q))$ where $\beta(n, q) = \min\{i | \log^i n \leq n/q\}$ is a very slowly growing function. This implies an overall time complexity of $\mathcal{O}(m_A + n\beta(n, q))$.

## 2.4 DARP on caterpillars is NP-hard

In the last section we saw that DARP can be solved in polynomial time if the underlying graph $G[E]$ is a path. Frederickson and Guan proved that DARP is NP-hard when the graph $G[E]$ is a tree [FG93]. We will strengthen this result and show that DARP is NP-hard on a caterpillar graph. This will allow us in Section 3.2 to show that introducing time penalties for starting and stopping at a given vertex leads to the resulting problem PENALTY-DARP being NP-hard even on paths.

**Theorem 2.4.1.** DARP *on caterpillars is NP-hard to solve. This result continues to hold when the sources and destinations of all arcs are leafs of the caterpillar and when there is at most one request starting from each leaf.*

*Proof.* To show that DARP on a caterpillar graph is NP-complete, we reduce the Steiner tree problem on bipartite graphs BIPARTITE-STP to it, which is known to be NP-complete [GJ79, Problem ND12]. An instance of BIPARTITE-STP consists of a bipartite graph $H = (X \cup Y, F)$ with bipartitions $X, Y$ and a nonnegative number $k$. The problem is then to decide, whether there exists a tree $T$ contained in $H$ which spans all vertices in $X$ and has at most $k$ edges.

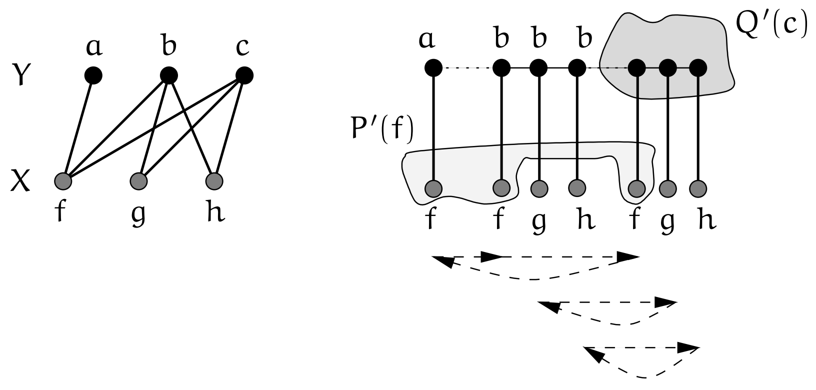

Figure 2.4: Transformation of a Steiner tree problem on a bipartite graph to a DARP on a caterpillar graph. Optimal solutions correspond to each other (from[HKRW99]).

We begin with a short outline of the proof as illustrated in Figure 2.4: Each vertex is split into sufficiently many copies, such that each of these copies is incident to exactly one of the edges incident to the original vertex. These edges will be the hairs of the caterpillar and we assign them a weight of one. We then add additional edges of weight zero, so that for each vertex in $Y$ all of its copies lie on a path segment of weight zero. These path segments are then joined with edges with a very large weight. The resulting paths is the backbone of the caterpillar. Finally, for each vertex in $X$ its copies are joined by a cycle of arcs. An solution of the resulting instance of DARP connects all these cycles. We will show that all the (original) edges traversed by an optimal solution correspond to a Steiner tree of minimum weight spanning $X$ in the original graph.

Without loss of generality, we assume that each vertex in $X$ has degree at least two. For a vertex in $X$ with a single edge incident to it, this edge has to be contained in any tree in $H$ that spans $X$. Also without loss of generality, we assume that $H$ is connected. Otherwise two cases can apply: If there is a connected component of $H$ that contains $X$, then we can restrict our discussion to this component. If there is no such component, then it follows that there is no tree connecting all vertices in $X$. We further assume that $|X| \leq k \leq |F|$. Since $X$ is an independent set in $H$, any tree connecting the vertices in $X$ has to contain at least one vertex from $Y$. Therefore such a tree must contain at least $|X|$ edges. The assumption $k \leq |F|$ follows from the observation that $T$ is a subgraph of $H$ and therefore it can not contain more than $|F|$ edges.

We now describe how to construct an instance $I = (G = (V, E, A), C, o)$ of DARP where $G[E] = (V, E)$ is a caterpillar, the arcs $A$ describe the requests, $c$ is a positive cost function and $o \in V$ is the origin of the server. The method is illustrated in Figure 2.4. It is based on the following ideas: First, we construct a new graph which contains the edges from $G[E]$, but these edges are all not adjacent. We further add a backbone to this graph which traverses one vertex from each edge. The resulting graph is a caterpillar. We add arcs between the leafs of the caterpillar, so that all those edges which in the original graph were incident to a single vertex from $X$ are now connected with a cycle of arcs. By choosing appropriate weights for the edges, we can then show that a solution for DARP on the graph constructed in this way corresponds to a solution of the BIPARTITE-STP.

We start our construction of $G$ with the backbone of the caterpillar. We first construct a path of $|Y|$ vertices, where for each vertex in $Y$ the path contains a copy of this vertex. The weight of the edges on the path are all set to $M = |F|+1$. We then replace each vertex $y$ on this path by a path of $d_H(x)$ vertices of cost zero. Here $d_H(y)$ denotes the degree of $y$ in $H$. The set of copies of all vertices in $Y$ is called $Q'$ and the set of copies of vertex $y \in Y$ is called $Q'(y)$.

In the second step we add for each edge in $F$ a hair of cost one to the caterpillar. Iteratively we choose the edges $f \in F$. Let the endpoints of $f$ be $x \in X$ and $y \in Y$. We choose a copy $y' \in Q'(y)$ of vertex $y$ which is not yet incident to a hair, and add a new hair to this vertex (note that there is always a "free" vertex in $Q'$, since the number of copies of each vertex equals to its degree in $H$). The other endpoint of the hair is a new vertex $x'$ which is a copy of the vertex $x \in X$. This construction will add $d_H(x)$ many copies of each $x \in X$ to the caterpillar. Again we denote with $P'$ the set of copies of all vertices in $X$ and we denote with $P'(x)$ the set of copies of vertex $x \in X$. Clearly, the graph $G[E]$ constructed in this way is a caterpillar graph.

We proceed to construct the transportation requests $A$ on the caterpillar $G[E]$: For each vertex $x \in X$, we add arcs between copies of $x$, such that these arcs constitute a simple directed cycle through the vertices in $P'(x)$. Finally the origin of the server is chosen to be the source of an arbitrary arc in $A$. Notice that there is at most one arc leaving each leaf of the caterpillar and that there are no arcs incident to vertices on the backbone.

Let $K = \sum_{a \in A} C(a)$. We will show, that $H$ contains a tree $T$ spanning $X$ which has less than $k$ edges, if and only if there is a feasible solution to the instance $I$ of DARP with cost less than $K + 2k$.

Notice that the graph $G[A]$ is by construction degree balanced, i.e., the in-degree of each vertex is equal to its out-degree. Also each set $P'(x)$ containing the copies of a vertex $x \in X$ corresponds to a strongly connected component in $G[A]$. Due to the property that $G[A]$ is degree balanced, each strongly connected component is Eulerian and there are no arcs between different components.

Let $T \subseteq H$ be a tree with $k$ edges spanning $X$. We now describe how to construct a multiset of arcs $S$ such that $G[A \cup S]$ is Eulerian, $o \in G[A \cup S]$ and which has cost $C(A \cup S)$ at most $K + 2k$.

Let $Y_T$ be the subset of vertices in $Y$ which is spanned by $T$. For each edge in $F$ with cost zero that connects copies of vertices in $Y_T$ we add to $S$ two antiparallel arcs between the endpoints of this edge. This does not incur any cost. Notice that these arcs ensure, that for each vertex $y \in Y_T$ its copies $Q'(y)$ are strongly connected.

For each edge $t$ in the tree $T$ we add two antiparallel arcs between the endpoints of the corresponding edge in $E$ (remember that the hairs of the caterpillar correspond to edges in $F$, such that for each edge $f \in F$ there is a unique hair whose endpoints are copies of the endpoints of $f$). The resulting multiset of arcs $S$ has therefore cost $C(S) = 2k$.

Since $S$ contains only pairs of antiparallel arcs, $G[S]$ is degree balanced. Therefore $G[A \cup S]$ is also degree balanced. It remains to show that $G[A \cup S]$ is connected. Recall that the arcs in $A$ were constructed such that the copies of a vertex in $X$ are a strongly connected component of $G[A]$. We already remarked that the copies of each vertex in $Y_T$ are strongly connected in $G[S]$. Since $T$ is a tree with vertex set $X \cup Y_T$, the pairs of antiparallel arcs in $S$ corresponding to edges in $T$ connect in $G[A \cup S]$ the (strongly connected) sets of copies of vertices in $X \cup Y_T$.

Conversely, let $S$ be a multiset of arcs with the properties that $G[A \cup S]$ is Eulerian and $C(A \cup S)$ is less or equal to $K + 2k$. Without loss of generality, we assume that $S$ contains only arcs from $A(E)$—otherwise we replace it with its embedding into $A(E)$ which, by Lemma 2.2.5, is a solution of the same cost.

The graph $G[S]$ is degree balanced, since $G[A \cup S]$ is Eulerian and $G[A]$ is degree balanced. We claim that each arc in $S$ has cost of less than $M = |F| + 1$. Since all arcs correspond to edges in $E$, an arc can have at most cost $M$. If an arc $a$ of cost $M$ exists, then $a$ corresponds to an edge $e \in E$ of cost $M$. We choose the connected component in $G[S]$ that contains $a$ (such a component exists, since $G[A]$ is degree balanced). Since $G[S]$ consists only of arcs corresponding to edges in $G[E]$, an Euler tour through this component describes a closed walk in $G[E]$, which is a caterpillar. Each edge in a caterpillar is a cut edge. A closed walk has to traverse a cut edge as often in one direction as it traverses it in the other direction. Therefore $G[S]$ contains another arc corresponding to $e$ but traversing it in the opposite direction than $a$. Since $k \leq |F|$, the cost of the arcs $A$ plus the cost of two $M$-weighted arcs already exceed the alleged cost of $K + 2k$.

We denote by $T$ the undirected graph induced by the hairs of the caterpillar $G[E]$ that correspond to an arc in $S$. We showed that all other arcs in $S$ have weight zero. Since $C(S) = 2k$ and the weight of arcs corresponding to hairs is one, there are $2k$ arcs corresponding to hairs. Using again the observation that for an arc corresponding to an edge there must be a second arc traversing this edge in the opposite direction, we conclude that there are at most $k$ elements in $T$.

We recall that there is a one-to-one correspondence between the hairs of the caterpillar $G[E]$ and the edges $F$ in the bipartite graph. We call $H[T]$ the graph in $H$ induced by those edges. It only remains to show, that $H[T]$ contains a path between any two vertices $u, v \in X$. In this case, $H[T]$ contains a tree connecting all vertices in $X$. This tree can have at most $k$ edges since $H[T]$ itself consists of at most $k$ edges.

To prove that any two vertices $u, v \in X$ are connected via a path in $H[T]$, we pick any two copies $u', v' \in P'$ of $u$ and $v$ respectively. There is a chain of arcs $L \subseteq (A \cup S)$ such that $G[L]$ is a walk connecting $u'$ and $v'$. There are only three types of arcs in $L$: The arcs in $A$ connect copies of the same vertex in $X$, the arcs in $S$ with weight zero connect copies of the same vertex in $Y$ and the arcs in $S$ which are hairs of the caterpillar correspond to edges in $H[T]$. This implies that traversing the edges in $H[T]$ that correspond to arcs in $S$ traversing hairs of the caterpillar yields a walk in $H[T]$ connecting $u$ and $v$. □

## 2.5 Approximation algorithms for DARP on trees

In this section we discuss approximation algorithms for DARP on trees. We have already seen that DARP on caterpillars (and therefore also on trees) is NP-hard (see Section 2.4).

We first show, that in order to solve DARP on trees we have to solve a Steiner tree problem on the component graph, which is a general graph. We will then study two approximation algorithms for DARP on trees. Frederickson and Guan [FG93] showed that a combination of these two algorithms achieves a performance of 5/4.

### 2.5.1 Notes about balanced instances of DARP on trees

A balanced instance of DARP dissects $G$ into strongly connected components $G_i$ each of which can either be traversed by an Euler tour in $A$ or consists of a single unused vertex in $G$. We have seen in Section 2.3, that we can compute an optimal solution for DARP on paths by finding a MST of the component graph, which has as vertices the components and unused vertices of $G$. Why does this not yield an optimal solution on trees? The answer to this question is that we can not eliminate the unused nodes without destroying the tree-property. Thus, we need to solve a Steiner tree problem rather than a MST problem on the component graph. In some instances where the "doubled MST" is not optimal, the Steiner points to use are still canonical (see Figure 2.5) because, e.g., $\hat{G}$ happens to be a tree, and hence the problem is efficiently solvable. In general, however, we can not expect this (see Figure 2.6)—not even on a path. The component graph can also contain instances of $K_4$ (otherwise it could be solved in linear time using Wald and Colbourn's algorithm [WC83]).

In fact, for an arbitrary graph $H = (W, F)$ we can easily construct a mixed graph $G = (V, E, A)$ where $G[E]$ is a tree such that $H$ is the component graph of $G$: Initially, let $G$ be a spanning tree of $F$. For each vertex $u$, we add a copy $u'$ of this vertex connected via an edge to $u$. Then, for each edge $[u, v]$ in $F$ which is not used by the spanning tree we add a copy $u'$ of vertex $u$ to $V$ and add an edge between $v$ and $u'$ to $E$. Finally we construct an arc set $A$ as follows: For any vertex $v \in W$ we add a cycle of arcs connecting all its copies in $V$. This construction clearly yields a mixed graph $G = (V, E, A)$ such that $G[E]$ is a tree and $\hat{G}[A] = F$. We conclude that $\hat{G}$ has in general no properties that can be exploited for solving the Steiner problem.

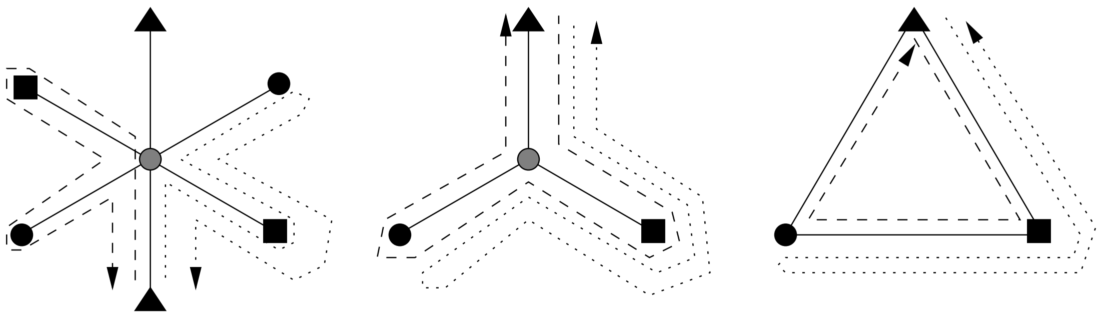

Figure 2.5: The "doubled MST" solution (pointed) does not equal the optimal tour (dashed) on trees. The given requests are: bring a unit from each symbol to its counterpart; this induces back-and-forth arcs between equally shaped nodes; the grey node is neither start nor end point of any request. From left to right: the instance with four connected components, the component graph $\hat{G}$, and $\hat{G}$ after removal of Steiner points and shortest-path completion (from[HKRW99]).

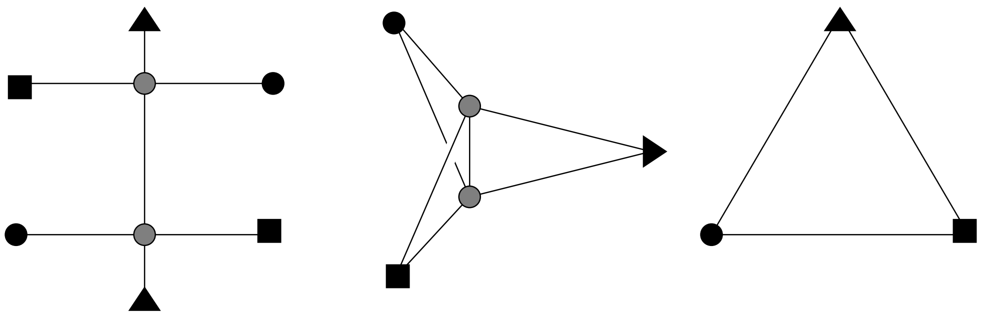

Figure 2.6: A small instance on a tree where $\hat{G}$ is not a tree (from[HKRW99]).

A possible approximation algorithm for solving DARP on trees would be to employ approximation algorithms for the corresponding Steiner tree problems on the component graph. This has been studied by Frederickson and Guan [FG93] using the approximation algorithms for the Steiner tree problem by Zelikovsky [Zel93] and by Berman and Rahmaiyer [BR92].

However, Frederickson and Guan achieved even better approximation results using a combination of two other approximation algorithms. We will discuss these two approximation algorithms and the combined strategy in the remainder of this chapter.

```
Input:   A mixed graph $G = (V, E, A)$, a cost function $c$ on $E$, an initial vertex $o \in V$
1 Compute a balancing multiset $B$ of arcs from $V \times V$ such that $B \subseteq S^*$ for some optimal
  solution $S^*$.
2 Compute the set of edges $X$ in $G[E]$ that are traversed by the shortest paths connecting the
  source and destination of arcs from $A$ which are in different components of $G[A \cup B]$.
3 Generate the instance $I' = (G' = (V', E', A'), C, o)$ that results from instance $I$ when
  contracting edges in $X$. Let $B'$ be a multiset of arcs in $G'$ corresponding to $B$.
4 Compute the component graph $\hat{G}'[A' \cup B']$.
5 Compute a minimum-cost Steiner tree $T$ for $\hat{G}'[A' \cup B']$ that spans at least the set of
  vertices corresponding to strongly connected components of $G'[A' \cup B']$.
6 Let $N = \emptyset$. For each edge in $T \cup X$ add two corresponding antiparallel arcs to $N$.
7 Find an Euler tour $C$ in $G[A \cup B \cup N]$.
8 return the multiset $B \cup N$ and the cycle $C$.
```

**Algorithm 6:** Algorithm SteinerSpecial for DARP on trees

### 2.5.2 Approximation algorithm SteinerSpecial

The algorithm SteinerSpecial first reduces the problem of connecting the components of $G[A \cup B]$ to a special case of the Steiner tree problem, which it can then efficiently solve in linear time.

The algorithm exploits a property of instances $I = (G = (V, E, A), C, o)$ of DARP on trees where $A$ is balanced and where each edge in $E$ is contained in the shortest paths in $G[E]$ between the sources and targets of arcs from at most one component. Frederickson and Guan prove in [FG93] that the component graph of such an instance contains no subgraph homeomorphic to $K_4$.

Using this result we can easily prove the following lemma:

**Lemma 2.5.1.** *Let* $I = (G = (V, E, A))$ *be an instance of* DARP *where* $G[E]$ *is a tree and* $A$ *is balanced. If every edge* $e \in E$ *is contained in the shortest paths between sources and sinks of arcs from at most one connected component of* $G[A]$ *then the corresponding Steiner tree problem on the component graph can be solved in time* $\mathcal{O}(n + m_A)$ *where* $n := |V|$ *and* $m_A := |A|$.

*Proof.* In time $\mathcal{O}(n + m_A)$ we can compute the component graph $\hat{G}[A]$. As shown in [FG93] the graph $\hat{G}[A]$ does not contain a subgraph homeomorphic to $K_4$. We can therefore use Wald and Colbourn's algorithm [WC83] to compute a Steiner tree in $\mathcal{O}(n)$ time. □

To prove the linear time complexity of algorithm SteinerSpecial, it remains to show that the set $X$ computed in Step 2 can be found in $\mathcal{O}(n + m_A)$. Frederickson and Guan describe a linear time algorithm for this [FG93].

**Theorem 2.5.2.** *For an instance* $I = G = (V, E, A), C, o)$ *where* $G[E]$ *is a tree, the algorithm SteinerSpecial computes multisets* $B$ *and* $N$ *such that* $G[A \cup B \cup N]$ *is Eulerian and* $C(A \cup B \cup N) \leq \frac{3}{2}C(A \cup S^*)$ *where* $S^*$ *is an optimal solution for the instance of* DARP. *The algorithm takes* $\mathcal{O}(n + m_A)$ *time where* $n = |V|$ *and* $m_A = |A|$.

*Proof.* Clearly the algorithm computes a feasible solution. The worst case performance ratio is established as follows: Since X is the set of edges traversed by the shortest paths connecting sources and sinks of arcs from more than one component, this edge is clearly traversed by at least four paths connecting the source and sink of arcs, twice in each direction. Note that the edges in X and the edges in T are disjoint. Therefore $4\,C(X) + 2\,C(T) \leq C(A \cup B)$. Let $S^*$ be an optimal solution such that $B \subseteq S^*$. We define $N^* := S^* \setminus B$. Using the observation that $C(T) \leq C(N^*)$ we get

$$\begin{aligned} C(A \cup B \cup N) &= C(A \cup B) + 2\,C(X) + 2\,C(T) \\ &\leq C(A \cup B) + \Big(\frac{C(A \cup B)}{2} - C(T)\Big) + 2\,C(T) \\ &\leq \frac{3}{2}\,C(A \cup B) + C(T) \leq \frac{3}{2}\,C(A \cup B) + C(N^*) \\ &\leq \frac{3}{2}\,\mathrm{OPT}. \end{aligned}$$

We now consider the computation time of the algorithm. Frederickson and Guan [FG93] show that it is possible to find the set X in time $\mathcal{O}(n + m_A)$. We also know that we can find a multiset of balancing arcs in time $\mathcal{O}(n + m_A)$ (see section 2.2.2). Clearly contraction of the edges in X and the subsequent computation of the component graph can take place in $\mathcal{O}(n + m_A)$ time. The Steiner tree can be computed in $\mathcal{O}(n)$ time. Finding the cycle C will take $\mathcal{O}(n + m_A)$ time. This concludes the proof that SteinerSpecial has time complexity $\mathcal{O}(n + m_A)$. □

We will later use the following intermediary result from the last proof:

**Corollary 2.5.3.** *With premises as in the last theorem, the inequality*

$$C(A \cup B \cup N) \leq \frac{3}{2}\,C(A \cup B) + C(T)$$

*holds.*

### 2.5.3 Approximation algorithm MinSpanTree

The approximation algorithm MinSpanTree is adapted from algorithm OptPath, which computes optimal solutions for DARP on paths (see section 2.3). However this time we can not simply calculate a MST for the component graph since in the case of trees, the component graph in general contains Steiner vertices which do not correspond to connected components of the balanced problem instance. However we approximate the Steiner tree by calculating a MST of a reduced graph.

The algorithm uses the *reduced component graph* $\bar{G}[A] = (\bar{V}, \bar{E})$ of a multiset A of arcs. This graph is constructed as follows: $\bar{V}$ is the set of all strongly

```
Input:  A mixed graph $G = (V, E, A)$, such that $G[E]$ is a tree, a cost function $c$ on $E$,
        an initial vertex $o \in V$
1 Compute a balancing multiset $B \subseteq A[E]$ such that $B \subseteq S^*$ for some optimal solution $S^*$.
2 Compute the reduced component graph $\bar{G}[A]$.
3 Compute a MST $\bar{T}$ of $\bar{G}[A]$.
4 Let $N = \emptyset$. For each edge in $\bar{T}$ add a pair of antiparallel arcs between the endpoints of the
  corresponding edge in $G[E]$ to $N$.
5 Find an Euler tour $C$ in $G[A \cup B \cup N]$.
6 return the multiset $B \cup N$ and the cycle $C$.
```

**Algorithm 7:** Algorithm MinSpanTree for DARP on trees.

connected components $G_i$ in $G[A]$. Edges in the graph have weight equal to the shortest path connecting any two vertices from the two components in $G[E]$. To allow calculation of a MST in the reduced component graph, we can simply generate an edge between any two vertices.

Kou and Makki [KM87] and Mehlhorn [Meh88] showed the following result and we refer to these authors for a proof:

**Lemma 2.5.4.** *There is an algorithm that computes in $\mathcal{O}(n)$ time a reduced component graph which has at most $\mathcal{O}(n)$ edges and has the property that a MST of this graph is also a MST of the complete reduced component graph, containing an edge between any pair of vertices.*

In order to establish a worst-case ratio for algorithm MinSpanTree, we use the following lemma, that a MST of the reduced component graph has cost at most twice that of an optimal Steiner tree in the component graph. This is proved in [FG93].

**Lemma 2.5.5.** *Let $G = (V, E, A)$ be a mixed graph such that $A$ is balanced. Let $c$ be a cost function on the edges. Let $\hat{T}$ be a Steiner tree in $\hat{G}[A] = (\hat{V}, \hat{E})$, the component graph of $A$ which spans the vertices corresponding to connected components in $G[A]$. Then we can construct a MST $\bar{T}$ of $\bar{G}[A] = (\bar{V}, \bar{E})$, the reduced component graph, with cost $C(\bar{T}) \leq 2\,C(\hat{T})$.*

We can now prove the following theorem on the performance and time complexity of algorithm MinSpanTree.

**Theorem 2.5.6.** *For an instance $I = (G = (V, E, A), C, o)$ where $G[E]$ is a tree, the algorithm MinSpanTree computes multisets $B$ and $N$ such that $G[A \cup B \cup N]$ is Eulerian and $C(A \cup B \cup N) \leq \frac{4}{3}\,C(A \cup S^*)$ where $S^*$ is an optimal solution for the instance of DARP. The algorithm takes $\mathcal{O}(m_A + n \log \beta(n, q))$ time, where $m_A := |A|, n := |V|$ and $q$ is the number of connected components in $A \cup B$ and $\beta(n, q) = \min\{i | \log^i(n) \leq n/q\}$*

*Proof.* Clearly the algorithm computes a feasible solution. The worst-case performance ratio of 4/3 can be shown as follows:

Let $S^*$ be an optimal solution for the instance of DARP such that $B \subseteq S^*$. Let $N^* := S^* \setminus B$. In a balanced problem instance for which assumption 2.1.3

holds, each edge is traversed by at least two antiparallel arcs from $A \cup B$. Therefore $C(A \cup B) \geq C(N)$. We will now show that $C(N) \leq 2\,C(N^*)$. The cost of $C(N)$ is at most twice the cost of a MST of the reduced component graph. In section 2.5.1 we discussed, that the cost of $N^*$ is twice the cost of a Steiner tree in the component graph spanning the vertices corresponding to connected components in $G[A]$. Using lemma 2.5.5 we get the claimed bound $C(N) \leq 2\,C(N^*)$.

Together with the first result that $C(A \cup B) \geq C(N)$ we get that $C(A \cup B) + C(N^*) \geq C(N) + \frac{1}{2}C(N) = \frac{3}{2}C(N)$ and therefore

$$\begin{aligned} C(A \cup B \cup N) &\leq C(A \cup B) + C(N) + C(N^*) - C(N^*) \\ &= C(A \cup B \cup N^*) + C(N) - C(N^*) \\ &\leq C(A \cup B \cup N^*) + \frac{1}{2}C(N) \\ &\leq \frac{4}{3}C(A \cup B \cup N^*) \\ &= \frac{4}{3}\text{OPT} \end{aligned}$$

With algorithm Balance from section 2.2.2 we can find $B$ in time $\mathcal{O}(n + m_A)$ when allowing balancing arcs from $V \times V$. Lemma 2.5.4 tells us that in time $\mathcal{O}(n)$ we can compute a reduced component graph with $q$ vertices and $\mathcal{O}(n)$ edges where $q$ is the number of connected components in $G[A \cup B]$. A minimum spanning tree in the reduced component graph can be computed in time $\mathcal{O}(m_A + n\log\beta(n,q))$ time, where $\beta(n,q) = \min\{i \mid \log^i(n) \leq n/q\}$ [GGST86]. □

We will later use the following intermediary result from the last proof:

**Corollary 2.5.7.** *With premises as in the last theorem, the inequality*

$$C(A \cup B \cup N) \leq C(A \cup B \cup N^*) + \frac{1}{2}C(N)$$

*holds.*

### 2.5.4 A mixed strategy algorithm

We finally present Frederickson and Guan's result that a combination of the algorithms MinSpanTree and SteinerSpecial yields an improved worst-case performance ratio of 5/4.

For an instance of DARP on a tree we compute solutions with both algorithm MinSpanTree and algorithm SteinerSpecial and select the one with lower cost.

**Theorem 2.5.8.** *Given an instance* $I = (G = (V, E, A), C, o)$ *of* DARP *where* $G[E]$ *is a tree we compute solutions* $(S_{MST})$ *and* $(S_{Steiner})$ *with algorithms* MinSpanTree *and* SteinerSpecial *respectively. Let* $S^*$ *be an optimal solution for this instance. Then the following holds:*

$$\min\{C(A \cup S_{MST}), C(A \cup S_{Steiner})\} \leq \frac{5}{4} C(A \cup S^*)$$

*Proof.* Since both MinSpanTree and SteinerSpecial calculate a balancing multiset $B$, we can assume without loss of generality, that both algorithms return the same balancing multiset $B$. Let $N_{MST} := S_{MST} \setminus B$ and $N_{Steiner} := S_{Steiner} \setminus B$. With $T$ we denote the Steiner tree calculated by SteinerSpecial. Further, let $S^*$ be an optimal solution such that $B \subseteq S^*$ and let $N^* := S^* \setminus B$.

If $C(A \cup S^*) \geq 2\, C(N_{MST})$ then from Corollary 2.5.7 we get

$$\begin{aligned} C(A \cup B \cup N_{MST}) &\leq C(A \cup B \cup N^*) + \frac{1}{2} C(N_{MST}) \\ &\leq \frac{5}{4} C(A \cup B \cup N^*) \\ &= \frac{5}{4} \text{OPT} \end{aligned}$$

Otherwise $C(A \cup S^*) \leq 2C(N_{MST})$ and with Corollary 2.5.3 it follows that

$$\begin{aligned} C(A \cup B \cup N_{Steiner}) &\leq \frac{3}{2} C(A \cup B) + C(T) \\ &\leq \frac{3}{2} C(A \cup B) + \frac{1}{2} C(N^*) \\ &\leq \frac{3}{2} C(A \cup B \cup N^*) - C(N^*) \\ &\leq \frac{6}{4} C(A \cup B \cup N^*) - \frac{1}{2} C(N_{MST}) \\ &\leq \frac{5}{4} C(A \cup B \cup N^*) \\ &= \frac{5}{4} \text{OPT} \end{aligned}$$

□

## 2.6 Approximation algorithm for DARP on general graphs

We present two approximation algorithms for DARP on general graphs by Frederickson, Hecht and Kim [FHK78]. They showed, that combining the two algorithms yields a performance ratio of 9/5.

Our discussion makes use of Assumption 2.1.5 for DARP on general graphs.

```
Input:  A mixed graph $G = (V, E, A)$ satisfying assumption 2.1.5, a cost function $c$
        on $E$, an initial vertex $o \in V$
1 Compute a balancing multiset $B$ of arcs from $V \times V$ of minimum cost.
2 Compute the reduced component graph $\bar{G}[A]$.
3 Compute a MST $\bar{T}$ of $\bar{G}[A]$.
4 Let $N = \emptyset$. For each edge in $\bar{T}$ add a pair of antiparallel arcs between the endpoints of the
  corresponding edge in $G[E]$ to $N$.
5 Find an Euler tour $C$ in $G[A \cup B \cup N]$.
6 return the multiset $B \cup N$ and the cycle $C$.
```

**Algorithm 8:** Algorithm LargeArcs for DARP on general graphs.

### 2.6.1 Approximation algorithm LargeArcs

The algorithm LargeArcs is basically the same algorithm as MinSpanTree, which is an approximation algorithm for DARP on trees (see section 2.5.3). However MinSpanTree computed a balancing multiset $B$ which is contained in an optimal solution using algorithm Balance.

On a general graph such a multiset can not be computed efficiently. The set $B$ computed in Step 1 is in general not contained in an optimal solution.

Before we analyze the performance and time complexity of LargeArcs we briefly comment on the computation of a minimum cost balancing multiset. Frederickson, Guan and Hecht [FHK78] compute a minimum cost bipartite matching between the set of sources of arcs and the set of destinations of arcs. They show that this can be done in time $\mathcal{O}(m_A^3)$ (where $m_A = |A|$) when using the weighted general matching algorithm by Edmonds and Johnson [EJ73] as implemented by [GL75]. The balancing arcs are from $V \times V$. There are at most $m_A$ balancing arcs in $B$.

Alternatively Step 1 can be carried out by performing a minimum cost flow computation in the auxiliary graph $F = (V, A(E))$. A vertex $v$ has charge $d_G^-(v) - d_G^+(v)$ and the cost of sending one unit of flow over arc $r \in A(E)$ equals its cost $C(r)$. We then compute an integral minimum cost flow in $F$. If the flow on an arc $r$ is $t \in \mathbb{N}$, we add $t$ copies of arc $r$ to the multiset $B$. It is easy to see that this yields in fact a balancing multiset of minimum cost.

With this approach the balancing multiset of arcs $B$ contains $\mathcal{O}(m_E)$ elements (where $m_E = |E|$). The arcs in $B$ are all from $A(E)$. The minimum cost flow computation 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].

For both methods of computing $B$ the running time of LargeArcs is clearly dominated by this computation.

**Theorem 2.6.1.** *For an instance $I = (G = (V, E, A), C, o)$ of* DARP *the algorithm LargeArcs computes multisets $B$ and $N$ such that $G[A \cup B \cup N]$ is Eulerian and $C(A \cup B \cup N) \leq 3\,C(A \cup S^*) - 2\,C(A)$ where $S^*$ is an optimal solution for the instance of* DARP.

```
Input: A mixed graph $G = (V, E, A)$ satisfying assumption 2.1.5, a cost function $c$
       on $E$, an initial vertex $o \in V$
 1 Compute the arc-distance graph $G'[A] = (V', E')$.
 2 Compute a TSP tour $T$ in $G'[A]$. Consider $T$ to be a set of arcs.
 3 Let $A_1 \subset A$ be the multisubset of arcs $a$ such that $T$ enters the vertex representing $a$
   in $G'[A]$ through an edge representing a shortest path in $G[E]$ incident to the destination
   vertex of $a$ and leaves the vertex representing $a$ through an edge representing a shortest
   path on $G[E]$ incident to the source vertex of $a$.
 4 Let $A_2 \subset A$ be the multisubset of arcs $a$ such that $T$ enters the vertex representing $a$
   through an edge representing a shortest path in $G[E]$ incident to the destination vertex of $a$
   and leaves the vertex representing $a$ through an edge representing a shortest path on $G[E]$
   incident to the source vertex of $a$.
 5 Let $A_3 \subset A$ be the multisubset of arcs $a$ such that the two edges in $T$ incident to the vertex
   representing $a$ represent shortest path incident to a single endpoint of $a$.
 6 Let $N_1 := \emptyset$ and $N_2 := \emptyset$.
 7 while traversing the TSP tour $T$ once do
 8     Let $t$ be the current edge from $E'$ in the tour.
 9     Let $M$ be the embedding of $t$ into $A(E)$ and let $\bar{M}$ be the multiset of arcs computed
       from $M$ by reversing the direction of each arc in $M$.
10     Add $M$ to $N_1$ and add $\bar{M}$ to $N_2$.
11 end while
12 Add to $N_1$ a copy and an antiparallel copy of each arc in $A_2$.
13 Add to $N_2$ a copy and an antiparallel copy of each arc in $A_1$.
14 Add to both $N_1$ and $N_2$ an antiparallel copy of each arc in $A_3$.
15 Let $S$ be the set among $N_1$ and $N_2$ which has lower cost.
16 Find an Euler tour $C$ in $G[A \cup S]$.
17 return the multiset $S$ and the cycle $C$.
```

**Algorithm 9:** Algorithm SmallArcs for DARP on general graphs.

*Proof.* Clearly algorithm LargeArcs computes a feasible solution.

The balancing multiset $B$ found in Step 1 of algorithm LargeArcs has cost at most $\mathrm{OPT} - C(A)$: The optimal solution $S^*$ is degree balanced, hence a minimal set of balancing arcs has at most cost $C(S^*) = \mathrm{OPT} - C(A)$.

The cost of the spanning tree edges must also be smaller than $\mathrm{OPT} - C(A)$, therefore $C(N) \leq 2\,\mathrm{OPT} - 2\,C(A)$. Adding together both inequalities, we get that $C(A \cup B \cup N) \leq 3\,\mathrm{OPT} - 2\,C(A)$. □

From the Theorem 2.6.1 we can immediately derive the following result:

**Corollary 2.6.2.** *Algorithm LargeArcs computes a solution of cost at most 3 OPT.*

### 2.6.2 Approximation algorithm SmallArcs

The algorithm SmallArcs uses the *arc-distance graph* $G'[A]$ of a multiset of arcs $A$ in a mixed graph $G = (V, E, A)$. The graph $G'[A] = (V', E')$ is constructed as follows: For each arc $a \in A$ we include a vertex $v'_a$ into $V'$. The graph $V'$ is complete. The weight of an edge $(v'_a, v'_b)$ is set to be the minimum distance in $G[E]$ between either source and sink of arc $a$ and source or sink of arc $b$.

**Theorem 2.6.3.** *If in Step 2 a $\rho_{TSP}$-approximation algorithm for computing a TSP-tour is employed, then algorithm SmallArcs finds a solution of cost at most $\rho_{TSP} \cdot (OPT - C(A)) + 2\,C(A)$.*

*Proof.* Clearly algorithm SmallArcs computes a feasible solution. We now show that $C(A \cup S) \leq \frac{3}{2}\,C(A \cup S^*) + \frac{1}{2}\,C(A)$. An optimal TSP solution on the arc reduced graph has at most cost $C(S^*)$ (a tour of this cost is implied by $A \cup S^*$). Therefore a $\rho_{TSP}$-approximation algorithm finds a solution of cost at most $\rho_{TSP} \cdot (\mathrm{OPT} - C(A))$. The cost of the smaller multiset of arcs added to $N_1$ and $N_2$ respectively is clearly no greater than $C(A)$. This yields $C(A \cup S) \leq \rho_{TSP} \cdot (\mathrm{OPT} - C(A)) + 2\,C(A)$. □

The TSP tour computation in Step 2 can be accomplished with Christofides' approximation algorithm [Chr76]. This yields a tour of length not more then 3/2 times the optimum.

The time that algorithm SmallArcs takes is then clearly dominated by running Christofides' approximation algorithm, which can be implemented to take time $\mathcal{O}(|A|^3)$.

We can therefore derive the following Corollary:

**Corollary 2.6.4.** *Algorithm SmallArcs employing Christofides' algorithm computes a solution of cost at most $2\,OPT$.*

*Proof.* From Theorem 2.6.3 we get $C(A \cup S) \leq \frac{3}{2}\,\mathrm{OPT} + \frac{1}{2}\,C(A)$. Since we know that $\frac{1}{2}\,C(A) \leq \frac{1}{2}\,\mathrm{OPT}$ the claim follows. □

### 2.6.3 A mixed strategy

Similarly to DARP on trees, we get an improved performance ratio of 9/5 by combining the algorithms LargeArcs and SmallArcs. This result was shown by Frederickson, Hecht and Kim [FHK78]

Given an instance of DARP we compute solutions with both algorithm LargeArcs and algorithm SmallArcs and select the one with lower cost.

**Theorem 2.6.5.** *Given an instance $I = (G = (V, E, A), C, o)$ of DARP and solutions $(S_{Small})$ and $(S_{Large})$ computed with algorithms SmallArcs and LargeArcs respectively, let $S^*$ be an optimal solution for this instance. Then the following holds:*

$$\min\{C(A \cup S_{Small}), C(A \cup S_{Large})\} \leq \frac{9}{5}\,C(A \cup S^*)$$

*Proof.* Let OPT := $C(A \cup S^*)$. If $C(A) \geq \frac{3}{5} C(A \cup S^*)$ then from Theorem 2.6.1 we get

$$\begin{aligned} C(A \cup S_{Large}) &\leq 3\,\mathrm{OPT} - 2\,C(A) \\ &\leq 3\,\mathrm{OPT} - \frac{6}{5}\mathrm{OPT} \\ &= \frac{9}{5}\mathrm{OPT}. \end{aligned}$$

Otherwise $C(A) \leq \frac{3}{5} C(A \cup S^*)$ and with Theorem 2.6.3 it follows that

$$\begin{aligned} C(A \cup S_{Small}) &\leq \frac{3}{2}\mathrm{OPT} + \frac{1}{2}C(A) \\ &\leq \frac{3}{2}\mathrm{OPT} + \frac{3}{10}\mathrm{OPT} \\ &= \frac{9}{5}\mathrm{OPT}. \end{aligned}$$

□

# Chapter 3

# Extensions of the Dial-a-ride Problem

In this chapter we consider extensions of the combinatorial optimization problem DARP studied in Chapter 2. All results presented in this chapter are new. They have been published together with S. O. Krumke, J. Rambau and H. C. Wirth in [HKRW99].

First we deal with FIFO-DARP, where a certain type of precedence relations is present. Similarly to DARP, we show that FIFO-DARP can be solved polynomially on paths and is NP-hard for caterpillars. Again, approximation algorithms for trees and general graphs are presented.

We then look at PENALTY-DARP, where time penalties for starting and stopping are considered. We show that PENALTY-DARP reduces to DARP on a slightly larger graph. This, however, yields that PENALTY-DARP is NP-hard even on paths.

Finally, we show how to derive open schedule solutions for DARP and FIFO-DARP, given approximate algorithms for the closed schedule case.

We summarize the main results for FIFO-DARP discussed in this chapter:

| | Hardness | Best approx. result |
|---|---|---|
| Path | P | – |
| Caterpillar | NP-hard | 5/3 |
| Tree | NP-hard | 5/3 |
| Graph | NP-hard | 9/4 |

## 3.1 DARP with FIFO precedence constraints

We now study FIFO-DARP, which is an extension of DARP. In FIFO-DARP we are given for each vertex a partial ordering on the arcs emanating from this vertex. The arcs have to be traversed by the Euler tour according to this ordering. With this we can model cargo elevators fed via conveyor belts: The elevator has to transport the first pallet waiting on a specific floor before being able to access the next pallet on this floor.

### 3.1.1 Precedence constraints

In FIFO-DARP for each vertex $v \in V$ we are additionally given a partial order $\prec_v$ on the arcs in $A_v$. For each feasible solution we require that the arcs from $A_v$ are traversed according to that partial order: whenever $a \prec_v a'$, then arc $a$ must be traversed before $a'$ in any feasible solution.

A partial order $\prec$ on the arc multiset of a graph $H = (V, R)$ is a *FIFO-order*, if it satisfies: $r \prec r'$ implies that $r$ and $r'$ have the same source. The partial order $\prec$ on the arc multiset $A$ of $G = (V, E, A)$ resulting from the disjoint union of the partial orders $\prec_v$ is clearly a FIFO-order. In the sequel $\prec$ is extended to multisets containing arcs from $V \times V$ by defining that arcs which are not in $A$ are incomparable to each other and to those of $A$.

FIFO-orders are useful to model situations in applications when FIFO waiting lines are present at each source and objects can be picked only from the head of the queue. The elevator system described in Chapter 1 has waiting areas of capacity one and is therefore not directly an example for FIFO-DARP—however we use FIFO-DARP for modelling a generalization of the elevators in this system where the waiting spaces are conveyor belts holding more than one pallet.

Figure 3.1 shows an instance where a FIFO-respecting transportation is strictly longer than a transportation neglecting the FIFO-order. The undirected edges of the graph (which is a path) are drawn as solid lines, and the arcs corresponding to the transportation jobs are shown as dashed arcs. If no constraints have to be obeyed, then the jobs can be served without any "empty move", i.e., without traversing any undirected edge. If the constraint $a' \prec a$ must be obeyed then two empty moves are necessary.

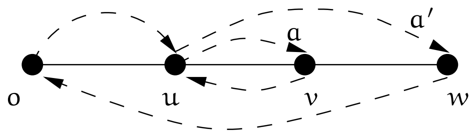

Figure 3.1: One precedence constraint increases the cost.

Again, FIFO-DARP can be reformulated as a graph augmentation problem. To do this, we need some additional notations:

**Definition 3.1.1 ($\prec$-respecting Eulerian Cycle, $\prec$-Eulerian).** Let $H = (V, R)$ be a directed graph, $\prec$ be a FIFO-order on the arcs $R$, and $o \in V$. A *$\prec$-respecting*

*Eulerian cycle* in $H$ *with start* $o$ is a Eulerian Cycle $C$ in $G$ such that $a \prec a'$ implies that in the 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 classical Eulerian cycles, in the case of $\prec$-respecting Eulerian cycles it is meaningful to specify a start node explicitly. Consider the graph in Figure 3.1, with solid edges removed. Then, for $a \prec a'$, there is a $\prec$-respecting Eulerian cycle with start vertex $o$, but there is none starting at $v$.

**Definition 3.1.2 (Graph Augmentation Version of FIFO-DARP).** An instance of FIFO-DARP consists of the same input as for DARP and additionally a FIFO-order $\prec$ on the arc multiset $A$. The goal is to find a multiset $S$ of arcs from $V \times V$ minimizing the weight $C(A \cup S)$ such that $G[A \cup S]$ is $\prec$-Eulerian with start $o$ and to determine a $\prec$-respecting Eulerian cycle in $G[A \cup S]$.

In the sequel we consider FIFO-DARP as a graph augmentation problem. We use $S^*$ to denote an optimal solution and $\mathrm{OPT} := C(A \cup S^*)$ to denote its cost. Notice that if $C^*$ is a $\prec$-respecting Eulerian cycle in $G[A \cup S^*]$ with start $o$, then the length of $C^*$ is equal to OPT.

### 3.1.2 Technical assumptions and balancing

In this section we will first explain some technical assumptions for FIFO-DARP. We will then show that the technique of balancing described in Section 2.2 for DARP can also be applied to FIFO-DARP on paths and trees.

We note that the assumptions made for DARP in Section 2.1.2 are also without loss of generality valid for FIFO-DARP. For the sake of completeness we repeat these assumptions here:

**Assumption 3.1.3 (Technical assumption for FIFO-DARP on trees).** *Each vertex $v \in V$ of degree one or two is either the origin $o$ or incident to at least one arc from $A$.*

For FIFO-DARP on paths we can make the following even stronger assumption:

**Assumption 3.1.4 (Technical assumption for FIFO-DARP on paths).** *Each vertex $v \in V$ is incident to at least one arc from $A$.*

We now turn to FIFO-DARP on general graphs.

**Assumption 3.1.5 (Technical assumption for FIFO-DARP on general graphs).**

*(i) Each vertex $v \in V$ is incident to at least one arc from $A$.*

*(ii)* $G[E]$ *is complete and the cost function* $c$ *obeys the triangle inequality, i.e., for any edge* $[u, v] \in E$ *the cost* $C(u, v)$ *does not exceed the length of a shortest path in* $G[E]$ *between* $u$ *and* $v$.

In Section 2.2 we discussed the concept of balancing, which turned out to be very useful for solving DARP on paths and trees. Remember that for a mixed graph $G = (V, E, A)$, we called a multiset $B \subseteq V \times V$ of arcs *balancing multiset* if in $H = G[A \cup B]$ we have $d^+_H(v) = d^-_H(v)$ for all vertices $v \in H$.

In Section 2.2.1 we showed for DARP on trees that we can construct in time $\mathcal{O}(nm_A)$ a balancing multiset which is contained in every feasible solution using only arcs from $A(E)$. Since a feasible solution for FIFO-DARP is clearly also a feasible solution for DARP, we can immediately conclude that this multiset is also contained in every feasible solution for FIFO-DARP which uses only arcs from $A(E)$.

When allowing augmentation with arcs from $V \times V$ we have seen in Section 2.2.2 that we can compute with algorithm Balance in $\mathcal{O}(n + m_A)$ a multiset $B$ of balancing arcs. We proved in that section that there exists an optimal solution containing that balancing set $B$. The proof of this result is equally valid when considering FIFO-DARP. We can therefore conclude that, when allowing augmentation with arcs from $V \times V$, Balance computes for an instance of FIFO-DARP on trees a balancing multiset which is contained in an optimal solution.

### 3.1.3 Euler tours respecting FIFO-orders

In this section we prove some new structural results about Eulerian cycles which respect a given FIFO-order. Notice that it is easy to decide whether a given graph $H$ is $\prec$-Eulerian with start $o$, provided the restriction of $\prec$ to each multiset $A_v$ is total: The $\prec$-respecting cycle (if it exists) is uniquely determined and can be found by a 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.

Let $C$ be an Eulerian cycle starting at $o$ in a directed graph. We define the *set of last arcs* of $C$, denoted by $L$, to contain for each vertex $v \in V$ the unique arc emanating from $v$ which is traversed last by $C$. Observe that $L$ contains a directed spanning tree rooted towards $o$.

Let $\prec$ be a FIFO-order. We denote the set of maximal elements with respect to $\prec$ by $M_\prec$, that is, $M_\prec := \{ a \in A : \text{there is no arc } a' \text{ such that } a \prec a' \}$.

**Definition 3.1.6 (Possible set of last arcs).** Let $H = (V, R)$ be a directed graph and $o \in V$ be a distinguished vertex. A set $L \subseteq R$ is called a *possible set of last arcs*, if it satisfies the following conditions:

(i) $d^+_L(v) = 1$ for all $v \in V$, and

(ii) for each $v \in V$ there is a path from $v$ to $o$ in $H[L]$.

**Theorem 3.1.7.** *Let* $H = (V, R)$ *be a directed Eulerian graph with distinguished vertex* $o \in V$ *and let* $L \subseteq R$ *be a possible set of last arcs.*

*(i) There exists an Eulerian cycle* $C$ *in* $H$ *such that for each vertex* $v \in V$ *the (unique) arc from* $L$ *emanating from* $v$ *is traversed last at* $v$ *by* $C$.

*(ii) Let* $\prec$ *be any FIFO-order with* $L \subseteq M_{\prec}$. *Then there exists a* $\prec$*-respecting Eulerian cycle with start* $o$ *in* $H$. *This cycle can be found in time* $\mathcal{O}(|V| + |R|)$.

*Proof.* We first show (i). Color the arcs from $L$ red and the arcs in $R \setminus L$ blue. We claim that by the following procedure we construct an Eulerian cycle $C$ in $H$ with the desired properties. Start with current vertex $o$. If possible, choose an arbitrary (but yet untraversed) blue arc emanating from the current vertex, otherwise choose the red arc. Traverse the 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 in-degree equals its out-degree. Therefore, $C$ must end in the origin $o$ and forms in fact a cycle.

We show that there is no arc in $H$ which is not traversed by $C$. For a node $v \in V$, let $\text{dist}(v, o)$ be the distance (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 the 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 $C$. For $d^+_H(w) = d^-_H(w)$, it follows that all arcs entering $w$, in particular arc $a$, are also contained in $C$. Since red arc $a$ is chosen last by the 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.

We proceed to show (ii). Analogously to (i) construct a Eulerian cycle with the sole difference that at each node $v$ we choose the next arc according to the $\prec$-constraint at $v$. Since by assumption $L \subseteq M_{\prec}$ this yields a valid $\prec$-respecting Eulerian cycle with start $o$. □

**Corollary 3.1.8.** *Let* $H = (V, R)$ *be a graph,* $o \in V$ *and* $\prec$ *a FIFO-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.*

```
Input:  A mixed graph $G = (V, E, A)$, such that $G[E]$ is a path, a cost function $c$ on $E$,
        an initial vertex $o \in V$, and a FIFO-order $\prec$
1 Compute a balancing multiset $B$ of arcs from $V \times V$ such that $B \subseteq S^*$ for some optimal
  solution $S^*$.
2 Let $M_\prec$ be the set of maximal elements with respect to $\prec$.
3 Set $H = G[B \cup M \cup A(E)]$ with cost function $C'$ on the arcs defined by
      $C'(r) = \begin{cases} 0 & \text{if } r \in B \cup M_\prec \\ C(r) & \text{if } r \in A(E) \setminus (B \cup M_\prec) \end{cases}$
4 Compute a directed spanning tree $D$ rooted towards $o$ of minimum weight $C'(D)$ in $G[B \cup$
  $M_\prec \cup A(E)]$.
5 Set $N := \emptyset$. For each directed arc $r \in D$ which is not in $B \cup M_\prec$, add $r$ and its anti-
  parallel $r^{-1}$ to $N$.
6 Define $L := D \cup \{r\}$, where $r$ is an arbitrary arc from $A_o \cap (M_\prec \cup B)$
    {Notice that such an arc must exist since $o$ is source or target of at least one job
    and $G[A \cup B]$ is degree-balanced.}
7 Use the method from Theorem 3.1.7 to find a $\prec$-respecting Eulerian cycle $C$ with start $o$
  in $G[A \cup B \cup N]$ such that $L$ is the last set of arcs of $C$.
8 return the multiset $B \cup N$ and the cycle $C$.
```

**Algorithm 10:** Algorithm FifoOptPath for FIFO-DARP on paths.

*Proof.* If $H$ is $\prec$-Eulerian with start $o$, then the set of last arcs of any $\prec$-respecting Eulerian cycle forms a possible set of last arcs which must be contained in $M_\prec$. Thus Statement 1 implies 2. The other direction is an immediate consequence of part (ii) of Theorem 3.1.7. □

Observe that Corollary 3.1.8 in fact 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 directed spanning tree $D$ rooted towards $o$ (which can be done in linear time). Adding to $D$ an arbitrary arc from $A_o \cap M_\prec$ then yields indeed a possible set of last arcs.

### 3.1.4 A polynomial time algorithm for FIFO-DARP on paths

In this section we consider FIFO-DARP on paths and show that the problem can be solved in polynomial time. To this end, let $G = (V, E, A)$ be a mixed graph such that $G[E]$ is a path. We assume throughout this section that Assumption 3.1.4 holds. Our algorithm FifoOptPath is an extension of OptPath for DARP on paths, as discussed in Section 2.3. The algorithm FifoOptPath has been published in [HKRW99].

**Lemma 3.1.9.** *Let $B \cup N$ be the multiset returned by algorithm FifoOptPath. The multiset $B \cup N$ is a feasible solution for FIFO-DARP, i.e., $G[A \cup B \cup N]$ is $\prec$-Eulerian with start $o$.*

*Proof.* In $G[A \cup B]$ each node has in-degree equal to its out-degree. Since $N$ consists of pairs of anti-parallel arcs, the graph $G[A \cup B \cup N]$ is also degree balanced. Since by construction $G[A \cup B \cup N]$ contains a directed spanning tree rooted towards o and $G[A \cup B \cup N]$ is degree balanced it follows that this graph is strongly connected and hence Eulerian.

The set L of arcs determined in Step 6 is clearly a set of possible last arcs. By Theorem 3.1.7 (ii) there exists indeed a $\prec$-respecting Eulerian cycle with start o in $G[A \cup B \cup N]$. □

**Theorem 3.1.10.** *Algorithm* **FifoOptPath** *finds an optimal solution for* FIFO-DARP *on paths.*

*Proof.* The idea behind this result is is based on the following two observations: Each solution has to include a possible set of last arcs and, since $G[E]$ is a tree, each arc from $A(E)$ has to be traversed as often in one direction as it is traversed in the other direction. Therefore, we get a solution of minimum cost by constructing an minimum directed spanning tree (where existing arcs can be used "for free") and "doubling" this tree.

More formally, let $S^*$ be an optimal solution such that $B \subseteq S^*$. Without loss of generality, we assume that all arcs in $S^* \setminus B$ are copies of arcs from $A(E)$ (we can otherwise replace $S^* \setminus B$ with its embedding into $A(E)$, which by Lemma 2.2.5 yields a solution of the same cost). By feasibility of $S^*$ the graph $G[A \cup S^*]$ is $\prec$-Eulerian with start o.

We now consider the multiset $Z := (A \cup S^*) \setminus (A \cup B) = S^* \setminus B$. Since $G[A \cup B]$ and $G[A \cup S^*] = G[A \cup B \cup Z]$ are degree balanced and $Z \cap (A \cup B) = \emptyset$, we can decompose the set Z into arc disjoint cycles $C_1, \dots, C_p$. Since Z contains only (multiple) copies of arcs from $A(E)$ and $G[E]$ is a tree it follows that $r \in Z$ implies that $r^{-1} \in Z$.

Let C be a $\prec$-respecting Eulerian cycle in $G[A \cup S^*]$ and let L be its last set of arcs. Notice that $L \subseteq B \cup M \cup Z$, where the set M is defined in Step 2 of the algorithm. The set L must contain a directed spanning tree $D'$ rooted towards o. We partition $D'$ into the multiset $D'_{B \cup M_\prec} := D' \cap (B \cup M_\prec)$ and $D'_Z := D' \cap Z$. Thus, $C'(D'_{B \cup M_\prec}) = 0$ and $C'(D'_Z) = C(D'_Z)$. 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

$$C(Z) \geq 2C(D'_Z) = 2C'(D'_Z) + 2\underbrace{C'(D'_{B \cup M_\prec})}_{=0} = 2C'(D') \geq 2C'(D). \tag{3.1}$$

Here, D is the directed spanning tree of minimum weight computed in Step 4. The multiset N computed in Step 5 has cost

$$C(N) = 2C(D \setminus (B \cup M_\prec)) = 2C'(D \setminus (B \cup M_\prec)) = 2C'(D) \overset{(3.1)}{\leq} C(Z). \tag{3.2}$$

Using this result yields that

$$\begin{aligned} C(A \cup B \cup N) &= C(A \cup B) + C(N) = C(A \cup (S^* \setminus Z)) + C(N) \\ &\overset{(3.2)}{\leq} C(A \cup (S^* \setminus Z)) + C(Z) = C(A \cup S^*). \end{aligned}$$

Thus, $B \cup N$ is an optimal solution as claimed. □

We briefly comment on the running time of algorithm FifoOptPath. Computing a balancing multiset $B$ with Balance takes $\mathcal{O}(n + m_A)$ time. A rooted spanning 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 FifoOptPath can be implemented to run in time $\mathcal{O}(n+m_A+\min\{(m_A+n)\log n, n^2\})$.

### 3.1.5 Algorithms for general graphs

In this section we present an approximation algorithm for FIFO-DARP on general graphs. The algorithm uses ideas similar to the algorithms for DARP on general graphs discussed in Section 2.6. We will assume that Assumption 3.1.5 is satisfied, i.e., each vertex is incident to at least one arc and the cost function obeys the triangle inequality.

Clearly FIFO-DARP is NP-hard to solve on caterpillars, since it contains DARP as a special case. Even if we assume that the FIFO-orders for each vertex are total, we notice that in Theorem 2.4.1 we showed that DARP on caterpillars is NP-hard to solve even when restricted to instances when there is at most one request starting from each leaf. However for an instance of FIFO-DARP on a caterpillar with at most one arc starting from each leaf, the precedence relation $\prec$ is empty.

Once again, the algorithm consists of *two* different sub-algorithms, FifoTSP and FifoLastArcs, which are run both and the best solution is picked. The first sub-algorithm, FifoTSP, is extremely simple: It computes a shortest tour which visits each vertex from which emanates an arc at least once. Then, it uses this TSP-tour to obtain a feasible solution for FIFO-DARP in the most obvious way. The algorithm is shown in Algorithm 11.

**Lemma 3.1.11.** *If in Step 3 a $\rho_{TSP}$-approximation algorithm for computing a TSP-tour is employed, then algorithm FifoTSP finds a solution of cost at most $\rho_{TSP} \cdot OPT + 2C(A)$.*

*Proof.* Let $S^*$ be an optimum augmenting multiset and $C^*$ be a $\prec$-respecting Eulerian cycle in $G[A \cup S^*]$ starting at $o$. Since $C^*$ visits all vertices from $V_s$, the length of $C^*$ (which equals OPT) is at least that of a shortest TSP-tour on the vertices $V_s$. Thus, if a $\rho_{TSP}$-approximation is used, the tour computed in Step 3 will have length at most $\rho_{TSP} \cdot \mathrm{OPT}$. The additional cost incurred in Step 7 is not greater than $2C(A)$, since each path added has weight not greater than the corresponding arc from $A$. Hence, the total cost of the cycle $C$ found by algorithm FifoTSP is bounded from above by $\rho_{TSP} \cdot \mathrm{OPT} + 2C(A)$ as claimed. □

Since the cost of the optimum tour serving all jobs is at least $C(A)$, we get from Lemma 3.1.11 that FifoTSP is a $(\rho_{TSP} + 2)$-approximation algorithm for

```
Input:  A mixed graph $G = (V, E, A)$, a cost function $c$ on $E$, an initial vertex $o \in V$,
        and a FIFO-order $\prec$
 1 Let $V_s$ be the set of vertices which are sources of arcs from $A$.
 2 Compute a complete undirected auxiliary graph $U$ with vertex set $V_s$. 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 $w$.
 3 Find an approximately shortest Traveling Salesperson tour $P$ in $U$ starting and ending in $o$.
   Let the order in which the vertices of $V$ are visited by $P$ be $v_0 = o, v_1, \dots, v_{|V_s|}, v_{|V_s|+1} = o$.
 4 Construct a feasible tour $C$ for FIFO-DARP as follows:
 5 Start with the empty tour $C$.
 6 for $i := 0, \dots, |V_s|$ do
 7     Let $a_1, \dots, a_k$ be the arcs from $A$ emanating from vertex $v_i$. Set $C \leftarrow C +$
       $(a_1, p_1, \dots, a_k, p_k)$, where $p_j$ is a shortest path in $G[E]$ from the endpoint of $a_j$ to $v_i$.
 8     Append to $C$ the shortest path in $G[E]$ from $v_i$ to $v_{i+1}$.
 9 end for
10 Let $S$ be the multiset of directed edges used in $C$ which are not contained in $A$.
11 return the multiset $S$ and the cycle $C$.
```

**Algorithm 11:** TSP-based approximation algorithm FifoTSP for FIFO-DARP.

```
Input:  A mixed graph $G = (V, E, A)$, a cost function $c$ on $E$, an initial vertex $o \in V$,
        and a FIFO-order $\prec$
 1 Compute a balancing multiset $B$ of arcs from $V \times V$ of minimum cost.
 2 Follow steps 2 to 7 of algorithm FifoOptPath to compute a multiset $N$ of arcs and a $\prec$-
   respecting Eulerian cycle $C$ with start $o$.
 3 return the multiset $B \cup N$ and the cycle $C$
```

**Algorithm 12:** Algorithm FifoLastArcs "mimicking" the algorithm for paths.

FIFO-DARP. Using Christofides' algorithm [Chr76] achieves $\rho_{TSP} = 3/2$ and thus FifoTSP provides a 7/2-approximation for FIFO-DARP. In the sequel we will improve this bound by providing a second algorithm and combining this algorithm with FifoTSP.

Our second algorithm, FifoLastArcs, is based on similar ideas as the algorithm from Section 3.1.4 for paths. We first compute a multiset of balancing arcs $B$ which makes $G[A \cup B]$ degree balanced. Again, we then compute a rooted tree directed towards the origin $o$ of minimum cost, double the arcs which are not yet in $A \cup B$ and add the resulting multiset $N$ to the solution. Our second algorithm, FifoLastArcs is shown in Algorithm 12.

By a proof similar to Lemma 3.1.9 it follows that the multiset $B \cup N$ found by algorithm FifoLastArcs is indeed a feasible solution.

**Lemma 3.1.12.** *The balancing multiset $B$ found in Step 1 of algorithm FifoLastArcs has cost at most OPT $- C(A)$. Step 1 can be accomplished in the time needed for one minimum cost flow computation on a graph with $n$ vertices and $2m_E$ arcs.*

*Proof.* Let $S^*$ be an optimal solution for an instance of FIFO-DARP, i.e., an augmenting multiset of arcs from $V \times V$ with minimum cost. Then the graph $G[A \cup S^*]$ is $\prec$-Eulerian with start $o$. Thus, in particular, the addition of the arcs from $S^*$ turns $G$ Eulerian, which means that each vertex has an in-degree equal to its out-degree. Thus, the cost $C(S^*) = \text{OPT} - C(A)$ is at least that of a minimum cost multiset $B$ of arcs from $V \times V$ which achieves the degree balance.

Step 1 can be carried out by performing a minimum cost flow computation in the auxiliary graph $F = (V, A(E))$. A vertex $v$ has charge $d^-_G(v) - d^+_G(v)$ and the cost of sending one unit of flow over arc $r \in A(E)$ equals its cost $C(r)$. We then compute an integral minimum cost flow in $F$. If the flow on an arc $r$ is $t \in \mathbb{N}$, we add $t$ copies of arc $r$ to the multiset $B$. It is easy to see that this yields in fact a balancing multiset of minimum cost. $\square$

**Lemma 3.1.13.** *The cost of the multiset* $N$ *computed in algorithm* FifoLastArcs *is at most* $2(OPT - C(A))$.

*Proof.* The proof of the lemma is similar to the one for Theorem 3.1.10. The major difference is that in general we can not assure that the balancing multiset $B$ computed in Step 1 is a subset of an optimal solution.

Let again be $S^*$ an optimal augmenting and $L$ be the set of last arcs of a $\prec$-respecting Eulerian cycle in $G[A \cup S^*]$. We can find a directed spanning tree rooted towards $o$ in $L$. The only arcs from $A$ that $L$ can contain are those from the set $M_\prec$. Thus $L \setminus (A \cup B) = L \setminus (M_\prec \cup B)$. Similar to Theorem 3.1.10 we can now conclude that

$$\mathrm{OPT} - C(A) = C(S^*) \geq C(L \setminus (A \cup B)) = C(L \setminus (M_\prec \cup B)) = C'(L) \geq \frac{C(N)}{2}.$$

This shows the claim. $\square$

**Corollary 3.1.14.** *Algorithm* FifoLastArcs *finds a solution of cost at most* $3OPT - 2C(A)$.

*Proof.* By Lemma 3.1.12, $C(A \cup B) \leq \mathrm{OPT}$. Lemma 3.1.13 establishes that $C(N) \leq 2\mathrm{OPT} - 2C(A)$. Thus $C(A \cup B \cup N) \leq 3\mathrm{OPT} - 2C(A)$ as claimed. $\square$

We are now ready to combine the algorithms into one with an improved performance guarantee. The combined algorithm Combine simply runs both algorithms and picks the better solution.

**Theorem 3.1.15.** *The combined algorithm* Combine *has a performance of* $\frac{\rho_{\mathrm{TSP}}+3}{2}$.

*Proof.* Let $\beta := \frac{4}{3-\rho_{\mathrm{TSP}}}$. If $\mathrm{OPT} \leq \beta C(A)$, then the solution returned by FifoTSP has cost at most

$$(\rho_{\mathrm{TSP}} + \frac{2}{\beta})\,\mathrm{OPT} = \left(\rho_{\mathrm{TSP}} + 2\,\frac{3-\rho_{\mathrm{TSP}}}{4}\right)\mathrm{OPT} = \frac{\rho_{\mathrm{TSP}}+3}{2}\mathrm{OPT}.$$

If $\mathrm{OPT} > \beta C(A)$, then the cost of the solution found by FifoLastArcs is bounded from above by

$$(3 - \frac{2}{\beta})\,\mathrm{OPT} = \left(3 - 2\,\frac{3-\rho_{\mathrm{TSP}}}{4}\right)\mathrm{OPT} = \frac{\rho_{\mathrm{TSP}}+3}{2}\mathrm{OPT}.$$

This shows the claim of the theorem. $\square$

Using Christofides' algorithm [Chr76] with $\rho_{\text{TSP}} = 3/2$ results in a performance guarantee of $3/4 + 3/2 = 9/4$ for algorithm Combine.

**Corollary 3.1.16.** *There is an approximation algorithm for* FIFO-DARP *with performance* 9/4*. This algorithm can be implemented to run in time* $\mathcal{O}(\max\{n^3 + m_A m_E + m_A n \log n, m_E^2 \log n + m_E n \log^2 n\})$.

*Proof.* The performance has already been proved. The running time of algorithm FifoTSP is dominated by that of Christofides' algorithm, which can be implemented to run in time $\mathcal{O}(n^3)$, and the time needed for the addition of the paths in Step 7 which can be done in total time $\mathcal{O}(m_A m_E + m_A n \log n)$. The running time of FifoLastArcs 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]. □

### 3.1.6 Improved performance on trees

For graph classes where the TSP can be approximated within a factor better than 3/2 the performance improves over the one stated in Corollary 3.1.16. In particular for trees, where the TSP can be solved in polynomial time, Theorem 3.1.15 already implies a 2-approximation algorithm. However, we can still improve this performance guarantee. This is again analogous to DARP on trees as discussed in Section 2.5.

**Theorem 3.1.17.** *There exists a polynomial time approximation algorithm for* FIFO-DARP *on trees with performance* 5/3*. This algorithm can be implemented to run in time* $\mathcal{O}(n m_A + n^2 \log n)$.

*Proof.* Our algorithm for trees uses a modified version of FifoLastArcs. We defer removal of the vertices in $V$ which are neither start nor endpoint of an arc from $A$ and the completion of $G$ via shortest paths until after the (modified) balancing step. The balancing step Step 1 of FifoLastArcs is modified so that we find a balancing subset $B \subseteq S^*$ as in Lemma 2.2.2. After the balancing we remove all vertices which are not incident to the arcs in $A \cup B$ and continue with FifoLastArcs from Step 2 on.

Let $I = (G = (V, E, A), C, o, \prec)$ be the original instance given such that $G[E]$ is a tree. We can consider the instance $I' = (G = (V, E, A \cup B), C, o, \prec)$ of FIFO-DARP (still on a tree) which results from adding the balancing arcs $B$ as new transportation jobs. Since any feasible solution to $I$ will have to use the arcs from $B$ anyway (cf. Lemma 2.2.2), we get that $\text{OPT}(I) = \text{OPT}(I')$.

Now look at the instance $I''$ of FIFO-DARP which is obtained by removing vertices and completing $G$ along shortest paths as in the algorithm. It is easy to see that $\text{OPT}(I'') = \text{OPT}(I')$. Notice also that a feasible solution to $I''$ is also a feasible solution to $I'$. Let $S^*$ and $S''$ be optimal solutions for $I$ and $I''$, respectively. Define $Z := S^* \setminus B$ and $Z'' := S'' \setminus B$. Since $\text{OPT}(I) = C(A \cup B) + C(Z) = \text{OPT}(I'') = C(A \cup B) + C(Z'')$, we have that $C(Z) = C(Z'')$.

Let $A \cup B \cup N$ be the solution found by the modified version of FifoLastArcs. Then, using the arguments of Lemma 3.1.13 we get that

$$\begin{aligned} C(A \cup B \cup N) &= C(A \cup B) + C(N) = C(S^*) - C(Z) + C(N) \\ &\leq C(S^*) - C(Z) + 2C(Z'') \leq C(S^*) + C(Z) \\ &= 2\mathrm{OPT}(I) - C(A). \end{aligned}$$

As noted before, FifoTSP finds a solution of cost at most $\mathrm{OPT} + 2C(A)$, since we can solve the TSP on the tree $G[E]$ in polynomial time. We can estimate the cost of the best of the two solutions returned by FifoTSP and the modified FifoLastArcs by the techniques from Theorem 3.1.15 where this time $\beta = 3$. This yields a performance of 5/3 as claimed.

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 FifoTSP in such a way that the invocations of Step 7 take total time $\mathcal{O}(nm_A)$. This means that FifoTSP can be implemented to run in time $\mathcal{O}(nm_A)$.

The balancing in the modified version of FifoLastArcs can be accomplished in time $\mathcal{O}(n + m_A)$. 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 directed spanning tree. □

## 3.2 Fifo-Darp with start and stop penalties

In this section we show how to incorporate additional start- and stop-penalties into the problem Fifo-Darp. In the Dial-a-Ride-Problem with Penalties, short Penalty-Fifo-Darp, we are given additional 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 objective is to find a closed walk serving all requests, such that the cost of the walk plus the cost of starting and stopping is minimized.

**Definition 3.2.1 (Graph augmentation version of Penalty-Fifo-Darp).** An instance of Penalty-Fifo-Darp consists of the same input as for Fifo-Darp together with additional penalty functions $p^+, p^- \colon V \to \mathbb{R}_{\geq 0}$ on the set of vertices $V$. The objective is to find a multiset of arcs $S$ from $V \times V$ minimizing the weight

$$C(A \cup S) + \sum_{u \in U^+} d^+(u) p^+(u) + \sum_{u \in U^-} d^-(u) p^-(u)$$

such that $G[A \cup S]$ is $\prec$-Eulerian with start $o$. Here, $U^+$ is the set of sources of arcs in $A \cup S$ and $U^-$ is the set of endpoints of arcs in $A \cup S$.

In the sequel we show that an instance $I = (G = (V, E, A), C, o, \prec, p^-, p^+)$ of PENALTY-FIFO-DARP can be transformed into an equivalent instance of FIFO-DARP $I' = (G' = (V', E', A'), C', o', \prec')$ on a slightly larger graph.

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 $C'(e_v) = 1/2(p^+(v) + p^-(v))$. The cost function $C'$ coincides with $c$ on the set $E$. For each arc $a = (u, v) \in A$ we add an arc $a' = (u^{(\pm)}, v^{(\pm)})$ to $A$ (the arcs in $A$ are not contained in $A'$). The partial order on the multiset $A_{u^{(\pm)}}$ is induced in the obvious way by that on $A_u$. Finally, the start vertex $o'$ equals $o$.

**Lemma 3.2.2.** *Let* $I = (G, C, o, \prec, p^+, p^-)$ *be an instance of* PENALTY-FIFO-DARP *and* $I' = (G', \prec, C', o')$ *be the instance of* 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.*

*Proof.* Let $S'$ be a valid solution for problem instance $I'$ of FIFO-DARP where $S'$ is an augmenting multiset of arcs. Let $C'$ be a $\prec$-respecting Eulerian cycle in $G'[A' \cup S']$ with start $o'$.

We first construct an auxiliary multiset $M$ 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 to $S'$ for each arc $(u, v)$ in $S$ the arc $(u^{(\pm)}, v^{(\pm)})$.

The time bound is obvious. □

It follows from the construction that if $G[E]$ is a tree then $G'[E']$ is also a tree. Thus, the last lemma implies that approximation results for FIFO-DARP on trees can be applied directly to PENALTY-FIFO-DARP on tree. Similarly, approximation results for general graphs carry over immediately. Hence, we obtain the following result:

**Theorem 3.2.3.** *The problem* PENALTY-FIFO-DARP *can be approximated on trees with performance* $5/3$ *and with performance* $9/4$ *on general graphs.* □

However, transforming an instance of PENALTY-FIFO-DARP where $G[E]$ is a path yields an instance of FIFO-DARP where $G'[E']$ is a caterpillar graph. This seems unfortunate, since we know from Theorem 2.4.1 that FIFO-DARP is NP-hard to solve on caterpillars. Is there a better transformation? More general, is PENALTY-FIFO-DARP on paths still polynomial time solvable?

The caterpillar constructed in the proof of Theorem 2.4.1 has the property that jobs have sources and targets only in the feet of the caterpillar and all hairs have the same length. Actually every instance of FIFO-DARP on caterpillars with these properties can be transformed into an equivalent instance of PENALTY-FIFO-DARP on a path: Let $f$ be a foot and $v$ be its unique adjacent vertex on the backbone. We replace all arcs from $A$ which are incident to $f$ by corresponding arcs with source or target $v$. We then remove foot $f$. The start- and stop-penalty on $v$ are set to the length $C(f, v)$ of the hair between $v$ and the foot $f$. It follows by arguments similar to those given in Lemma 3.2.2 that the constructed instance of PENALTY-FIFO-DARP on the path (which corresponds to the former backbone) is in fact an equivalent instance to the instance of FIFO-DARP on the caterpillar. Thus, we obtain the following result which contrasts with the polynomial solvability of FIFO-DARP on paths:

**Lemma 3.2.4.** PENALTY-FIFO-DARP *on paths is NP-hard to solve.* □

## 3.3 Open schedule problems

All our results for DARP from Chapter 2 and for FIFO-DARP and PREC-DARP from this chapter are derived for the closed schedule case, where the server has to return to the origin at the end.

In this section we will briefly describe how algorithms for the closed schedule case can be used to compute solutions for the open schedule case—however the method described in this section does not use any special insight into the open schedule problem. Dedicated research into open schedule DARP (and FIFO-DARP) should yield much better algorithms.

Let $I = (G = (V, E, A), C, o)$ be an instance of DARP (or FIFO-DARP, when given an additional FIFO-order $\prec_v$) and let $A$ be a $\rho$-approximate algorithm for DARP (or FIFO-DARP) with closed schedules. We will run algorithm $A$ $|V| - 1$ times. In each iteration we pick one of the vertices $v \in V \setminus \{o\}$—each time a different one. We then add a new arc $a_v = [v, o]$ to the multiset of arcs and run algorithm $A$ on this instance. At the end, we choose the shortest schedule, remove the corresponding additional request and return the resulting open schedule solution. When solving instances of FIFO-DARP in this manner, the artificial arc will get the highest precedence with respect to $\prec_v$ of all arcs emanating from $v$, i.e., the "dummy request" has to be served last.

It is easy to see, that, given an optimal algorithm for solving closed schedule instances, the above procedure yields an optimal algorithm for solving the open schedule case. For $\rho$-approximate algorithms, it is easy to see that we get the following result:

**Theorem 3.3.1.** *Let $I = (G = (V, E, A), C, o)$ be an instance of* DARP *(or* FIFO-DARP*, when given an additional FIFO-order $\prec_v$) and let $A$ be a $\rho$-approximate algorithm for* DARP *(or* FIFO-DARP*) with closed schedules. The procedure described above yields a $2\rho - 1$-approximate algorithm for open schedules.*

*Proof.* Let $C_{\text{open}}$ be the cost of the solution returned by $A$, let $C_{\text{closed}}$ be the cost of the closed schedule, which yielded the open schedule solution, i.e, there exists an arc $a = [v, o], v \in V$ such that $C_{\text{open}} = C_{\text{closed}} - C(a)$. Further, let $\text{OPT}_{\text{open}}$ be the optimal solution for the open schedule problem and let $\text{OPT}_{\text{closed}}$ be the optimal solution for the closed schedule problem with addition-l arc $a$. Clearly, $C(a) \leq \text{OPT}_{\text{open}}$—this follows directly from our assumptions for any of the graph classes (paths, trees and general graphs). We therefore get

$$\begin{aligned} C_{\text{open}} &= C_{\text{closed}} - C(a) \\ &\leq \rho(\text{OPT}_{\text{closed}}) - C(a) \\ &\leq \rho(\text{OPT}_{\text{open}} + C(a)) - C(a) \\ &\leq \rho\text{OPT}_{\text{open}} + (\rho - 1)C(a) \\ &\leq (2\rho - 1)\text{OPT}_{\text{open}}. \end{aligned}$$

□

# Chapter 4

# Online Dial-a-ride Problems

In this chapter we study the online versions of the combinatorial optimization problems DARP and FIFO-DARP. We begin by defining the problems more formally and we also introduce the special case ONLINE-TSP, where sources and targets of requests are identical. We then show lower bounds on the competitive ratio of algorithms for ONLINE-DARP and ONLINE-TSP which have been proved in [AFL+95, AFL+94, AKR98a]. For all these results the objective function is—as in DARP—the *total completion time* of the schedule. However we also show that there can be no competitive algorithm for the task of minimizing the maximal or the average flow or waiting times. We then present the algorithms REPLAN and IGNORE and prove competitive ratios for these algorithms. We conclude the chapter by discussing two other competitive algorithms, namely algorithm PAH for ONLINE-TSP and algorithm SLEEP for ONLINE-DARP.

We summarize the main results discussed in this chapter:

| | Lower bound | REPLAN | IGNORE | Best other |
|---|---|---|---|---|
| ONLINE-TSP | $\approx 1.64$ | 5/2 | 5/2 | 2 |
| ONLINE-DARP | $\approx 1.71$ | 5/2 | 5/2 | 5/2 |
| ONLINE-FIFO-DARP | $\approx 1.71$ | 3 | 5/2 | – |

## 4.1 The problems ONLINE-TSP and ONLINE-DARP

The problem ONLINE-DARP studied in this chapter is the online version of the Dial-a-ride variant $1, cap1 || M | \sum m$, using the notation from Appendix B. An instance consists of a metric space $M$ with a distance function $d \colon M \times M \to \mathbb{R}^+$ which is symmetric and obeys the triangle inequality (we allow non-identical points to have distance zero—e.g., vertices in a graph connected via a zero-weight edge). Additionally, we require the metric space to be *continuous*: For any pair $x, y \in M$ there must be a continuous map from the interval $[0, 1]$ to the shortest path in $M$ from $x$ to $y$.

Each *request* is a triple $r_i = (t_i, a_i, b_i)$. The real number $t_i$ is the *release time*, when request $r_i$ becomes known. The points in the metric space $a_i, b_i \in M$ are the source and target, respectively, between which an object is to be transported. We will consider each set $\sigma = \{r_1, \dots, r_m\}$ of transportation requests as ordered by their release times. We assume that the online algorithm does neither have information about when the last request occurs nor about the total number of requests.

The server can move at constant unit speed and is positioned at location $o \in M$ at time $0$. The server has *unit capacity*, i.e., it can carry at most one object at a time. Finally, we do *not allow preemption*: once the server has picked up an object, it is not allowed to drop it at any place other than its destination.

A *server move* $m_j = (t_j, a_j, b_j, R_j)$ is a quadruple, where the real number $t_j$ is the time when the server starts the move at location $a_j$. It's *arrival time* at location $b_j$ is $t_j + d(a_j, b_j)$. The set $R$ contains all requests that are served by the move—for the unit capacity problems studied in this thesis, $R$ is either empty or contains one request.

An *open transportation schedule* serving a sequence of requests $\sigma$ is a sequence of moves of the server $\Sigma = (m_1, m_2, m_3 \dots)$ with the following properties:

- The first move starts in the origin $o$ ($a_1 = o$),
- The starting point of $m_j$ is the end point of $m_{j-1}$ ($a_j = b_{j-1}$),
- The starting time of $m_j$ carrying $R_j$ is greater than the release times of all requests $r_i \in R$ ($t_j \geq \max\{t_i\colon (t_i, a_i, b_i) \in R\}$)
- For every request $r_i \in \sigma$ there is a first move $m_k$ and a last move $m_l$ carrying $r_i$. The move $m_k$ starts at the source of $r_i$ and the move $m_l$ ends at the destination of $r_i$, the request is carried by all moves in between $m_k$ and $m_l$ ($j < k : r_i \notin R_j$; $j > l : r_i \notin R_j$; $k \leq j \leq l : r_i \in R_j$; $a_k = a_i$ and $b_l = b_i$).

For the unit capacity case considered in this thesis we simplify the last property and require that there is exactly one move serving each request, which starts at the source of the request and ends at the destination of the request.

For a *closed schedule* we additionally require that the last move has the origin as destination. If there is no restriction on the final destination of the server we are dealing with an *open schedule* problem.

For an algorithm $A$ and a request sequence $\sigma$, let $C_A(\sigma)$ denote the *completion time* of the sequence of moves that algorithm $A$ generates for serving $\sigma$. With $\mathrm{OPT}(\sigma)$ we denote the completion time of an optimal (offline) solution for serving the request sequence.

The problem ONLINE-TSP studied by Aussiello et.al. is a special case of ONLINE-DARP: The source and the destination of each request are identical

[AFL$^+$95, AFL$^+$94]. We will denote requests for ONLINE-TSP as tuples $r_i = (t_i, a_i)$ where $t_i$ is the time when the request occurs and $a_i \in M$ is the position of the request. Using the notation from Appendix B, ONLINE-TSP is the online version of the Dial-a-ride variant $1, \mathit{cap1} | s = t | M | \sum m$. There are two main differences between ONLINE-TSP and the (more general) ONLINE-DARP:

- In ONLINE-TSP the server can change its schedule at any time and, if necessary, move in a different direction straight away. In ONLINE-DARP this is not always possible: Once the server has picked up an object to be transported from $a$ to $b$ it is not allowed to drop that object at any other place than $b$.

- The distances in the metric space of ONLINE-TSP are assumed to be symmetric. Thus, if starting and ending in the origin, a set $\sigma = \{r_1, \dots, r_m\}$ of requests can be served in the order $r_1, \dots, r_m$ or $r_m, \dots, r_1$ at the same cost (ignoring release times). In ONLINE-DARP changing the "direction" of service could increase the cost by a factor of two.

Clearly, ONLINE-DARP generalizes the online variant of the Dial-a-ride problem on graphs (DARP) studied in Chapter 2, which we formulated as a graph augmentation problem in a mixed graph $G(V, E, A)$ together with an origin $o$ and a cost function $c$: Given a graph with positive edge weights, we can restrict server moves to the shortest path in the graph. Hence we can assume that the graph is complete, with edge-weights satisfying the triangle inequality. This in turn yields a metric space with the properties specified above.

However, we have to allow the server to move "continuously" in the space, i.e., to move continuously from one endpoint of an edge $(u, v)$ to the other endpoint and possibly change its direction while at some location $s$ on the edge $(u, v)$. The problem is different when the server may not change direction while traversing an edge and results stated in this chapter may not apply in this case.

Given an offline instance of the problem studied in this chapter where servers move in metric spaces, we can transform this into an instance of DARP on graphs: The set of vertices $V = \bigcup_{r_i \in \sigma} \{a_i, b_i\} \cup \{o\}$ is the set of endpoints of requests together with the origin. For each pair of vertices $u, v \in V$ we add an edge of weight $d(u, v)$. Finally the set of arcs $A = \bigcup_{r_i \in r} \{(a_i, b_i)\}$ contains one arc for each request, directed from the source of the request to its destination.

Yet we note that even though the offline problems are equivalent, the online problem on metric spaces is more general than the online problem on finite graphs: In an offline setting a server will never move towards a point and then "change its mind", therefore the direct paths between all sources and destinations and the origin fully describe the feasible walks of the server. However in an online setting, where new requests can occur at any time, the server may decide to move into a different direction at any location—moving only along the edges of a finite graph is therefore a restriction on the server movements.

Most of the results for ONLINE-DARP presented in this chapter apply equally to the online case of the problem FIFO-DARP studied in Chapter 3: All requests with a common source point in the metric space have to be served in the order of their release times. We call this problem ONLINE-FIFO-DARP.

## 4.2 Lower bounds

In this section we address the question of how well an online algorithm for ONLINE-DARP and for ONLINE-TSP can perform compared to the optimal offline adversary. Since DARP is a special case of FIFO-DARP, all the lower bounds derived in this section apply equally to ONLINE-FIFO-DARP.

For ONLINE-TSP with closed schedules (the server has to return to the origin), Aussiello et.al. have shown the following result [AFL$^+$95]:

**Theorem 4.2.1.** *For closed schedules, no deterministic algorithm for* ONLINE-TSP *on the real line can achieve a competitive ratio* $c < \frac{9+\sqrt{17}}{8} \approx 1.640388$.

*Proof.* The underlying continuous metric space is the real line with 0 as the origin of the server. Suppose that A is a deterministic online algorithm with competitive ratio c. We can assume that $c \leq \frac{9+\sqrt{17}}{8}$, since otherwise there is nothing left to prove. It is also easy to see that $c \geq 1.5$ [1].

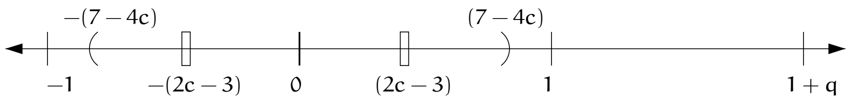

Figure 4.1: The requests and intervals from the proof of the lower bound for ONLINE-TSP.

Figure 4.1 illustrates the requests and the intervals which we use in this proof. Before time 1 no request occurs. At this time the online server must be located within the interval $[-(2c-3),(2c-3)]$: Otherwise, if for example the online server is positioned to the right of $(2c-3)$, we let a request occur at time 1 at position $-1$. The optimal offline server incurs cost 2, whereas the online server incurs cost $C_A > 1 + (2c-3) + 2 = 2c$, which contradicts the assumption that it is c-competitive. The case when it is located to the left of $-(2c-3)$ is symmetric. We conclude that the online server must be within $[-(2c-3),(2c-3)]$ which is contained in $(-1,1)$ since $c < \frac{9+\sqrt{17}}{8}$.

We now present at time 1 two simultaneous requests at 1 and $-1$. Due to the above observations, at time 3 the online server can not have served both requests. W.l.o.g. we assume that the server has not served the request at $-1$.

[1]The first request occurs at time 1. W.l.o.g. the online server is either at zero or to the right of zero. Let a request occur at $-1$. Then the offline server can serve this request and return to the origin at time 2 whereas the online server will incur cost of at least 3.

**Case 1:** The server is at time 3 positioned within $(-(7-4c),(7-4c))$ (and has either served the request at 1 or not).

At time 3 another request at position 1 is presented. The offline server can serve all requests and return to the origin in time 4. The online server still has to move to both extremes and return to the origin, therefore $C_A > 3+4-(7-4c) = 4c$. This contradicts the assumption that A is c-competitive.

**Case 2:** The server has at time 3 served request one and is not within $(-(7-4c),(7-4c))$.

We show that the online server is then within $[(7-4c),1]$: It remains to show that the online server is not within $(-1,-(7-4c)]$. Since at time 1 the online server has been to the right of $(2c-3)$ and it then moves to 1, it must be at time 3 to the right of $-(2c-3)$ which is to the right of $-(7-4c)$ for any $c \leq 5/3$. Therefore the claim holds for our range of c.

We already noted that the optimal offline solution for the request sequence can serve both requests and return to the origin in time 4. Our online algorithm is now at time 3 positioned to the right of $(7-4c) > 0$ and still has to serve the request at $-1$. To be c-competitive it must therefore pass the origin at some time before $4c-2$. Let $3+q$ be the time when the online server passes the origin. We know that $q \leq 4c-5$. At time $3+q$ we issue a new request at position $1+q$. An optimal offline solution takes time $4+2q$ to serve all requests. The online algorithm, however, is at time $3+q$ at position 0 and has to serve requests at $-1$ and at $1+q$. Therefore it can not finish before time $7+3q$. This yields the following condition for the competitive ratio:

$$c \leq \frac{7+3q}{4+2q}$$

This is a monotonically decreasing function in q, and, using $q \leq 4c-5$, we get:

$$c \leq \frac{7+3(4c-5)}{4+2(4c-5)}$$

The lowest value for c satisfying this inequality is $c = \frac{9+\sqrt{17}}{8}$.

**Case 3:** The server has neither served the request at 1, nor the request at $-1$ and is not within $(-(7-4c),(7-4c))$.

If the server is within $[(7-4c),1]$, then by the last case, the server can not even serve the request at $-1$ and return to zero (even when ignoring the request at 1) in sufficient time. Otherwise we can apply the same argument by symmetry. □

For ONLINE-DARP with closed schedules, Ascheuer, Krumke and Rambau proved the following lower bound [AKR98a]:

**Theorem 4.2.2.** *For closed schedules no deterministic algorithm for* ONLINE-DARP *on the real line can achieve a competitive ratio* $c < 1+\sqrt{2}/2 \approx 1.7071068$. □

*Proof.* The underlying continuous metric space for the instance of ONLINE-DARP is the real line. The server is initially positioned at the origin $0$. Suppose that $A$ is a deterministic online algorithm with competitive ratio $c$. We can assume that $c < 1 + \sqrt{2}/2$, since otherwise there is nothing left to prove.

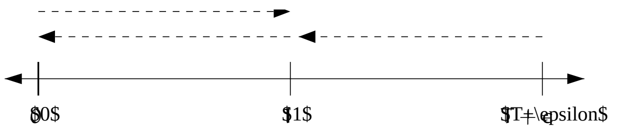

Figure 4.2: The requests from the proof of the lower bound for ONLINE-DARP on the real line.

Figure 4.2 illustrates the requests which are used for this proof. At time $t = 0$, the algorithm $A$ is faced with two requests $r_1 = (0, 0, 1)$ and $r_2 = (0, 1, 0)$. Thus $\mathrm{OPT}(r_1, r_2) = 2$ and the server operated by $A$ must start serving request $r_2$ at some time $1 \leq T \leq 2c - 1$. Since $c < 1 + \sqrt{2}/2$, we have $1 \leq T \leq 2c - 1 < 1 + \sqrt{2}$.

We issue at time $T + \varepsilon$ another request $r_3 = (T + \varepsilon, T + \varepsilon, 1)$. The optimal offline server will need total time $2(T + \varepsilon)$. The online server however will need total time $T + 1 + 2(T + \varepsilon) = 1 + 3T + 2\varepsilon$.

Thus the ratio of the time needed by $A$ and the optimal offline algorithm is

$$\frac{C_A(r_1, r_2, r_3)}{\mathrm{OPT}(r_1, r_2, r_3)} \geq \frac{1 + 3T + 2\varepsilon}{2T + 2\varepsilon} \geq \frac{1 + 3(2c - 1) + 2\varepsilon}{2(3c - 2) + 2\varepsilon} = \frac{6c - 2 + 2\varepsilon}{4c - 2 + 2\varepsilon}$$

It is easy to check that the expression given above is increasing in $\varepsilon$. Thus the competitive ratio $c$ is bounded from below by the value of the above expression for $\varepsilon$ approaching $0$:

$$c \geq \lim_{\varepsilon \to 0} \left( \frac{6c - 2 + 2\varepsilon}{4c - 2 + 2\varepsilon} \right) = \frac{6c - 2}{4c - 2} \tag{4.1}$$

The smallest value $c \geq 1$ satisfying Equation (4.1) is $c = 1 + \frac{\sqrt{2}}{2} \approx 1.7071068$, which contradicts the assumption that $A$ is c-competitive with $c < 1 + \frac{\sqrt{2}}{2}$. □

Ascheuer, Krumke and Rambau have also proved a (slightly weaker) lower bound for continuous metric spaces where requests can only occur at discrete points, namely to paths where the ratio of the diameter and the minimum distance between any two vertices is bounded by 4 [AKR98a]. The construction in the following proof is similar to the proof of Theorem 4.2.2.

**Theorem 4.2.3.** *For closed schedules no deterministic algorithm for* ONLINE-DARP *can achieve a competitive ratio* $c < 5/3$.

*Proof.* The underlying metric space is a path $G = (V, E)$ of 5 vertices $v_0, \dots, v_4$ with the origin being the "leftmost" vertex $v_0$ at distance $D = 4$ from the right end. All edge-weights are equal to one, hence $d(o, v_i) = i$.

Suppose that A is a deterministic online algorithm with competitive ratio c. We can assume that $c \le 5/3$, since otherwise there is nothing left to be proved.

At time $t = 0$, the algorithm A is faced with two requests $r_1 = (0, o, v_2)$ and $r_2 = (0, v_2, o)$. Thus $\mathrm{OPT}(r_1, r_2) = 4$ and the server operated by A must start serving request $r_2$ at some time $2 \le T \le 4c - 2$. Since $c \le 5/3$, we have $2 \le T \le 4c - 2 \le 4\frac{2}{3}$.

**Case 1:** $2 \le T \le 3$.

At time T the adversary issues another request $r_3 = (T, v_3, v_2)$. Thus, the online server can not finish before time $T + 8 \ge 10$. On the other hand, the offline server first handles $r_1$, then continues to move to $v_3$ which it reaches no earlier than the time when $r_3$ becomes known. Hence, the optimal offline server incurs a total cost of 6, which gives us that

$$\frac{C_A(r_1, r_2, r_3)}{\mathrm{OPT}(r_1, r_2, r_3} \ge \frac{5}{3}.$$

**Case 2:** $3 < T \le 4c - 2$.

In this case, at time T the adversary issues another request $r_3 = (T, v_{\lfloor T \rfloor}, v_2)$. Notice that since $T > 3$ and $\lfloor T \rfloor \le 4$ we have that $v_{\lfloor T \rfloor} \in \{v_3, v_4\}$.

Let $\lfloor T \rfloor = T + \varepsilon$ for some $-1 \le \varepsilon \le 0$. The online algorithm will need total time at least $T+2+2\lfloor T \rfloor = 3T+2+2\varepsilon$. The offline server first serves request $r_1$ and then moves to vertex $v_{\lfloor T \rfloor}$ where it sits until time T. It then serves $r_3$ and, finally, $r_2$ at a total cost of $\mathrm{OPT}(r_1, r_2, r_3) = T + \lfloor T \rfloor = 2T + \varepsilon$.

Thus in the second case the ratio between the time needed by A and the optimal offline algorithm is

$$\frac{C_A(r_1, r_2, r_3)}{\mathrm{OPT}(r_1, r_2, r_3)} \ge \frac{3T + 2 + 2\varepsilon}{2T + \varepsilon} \ge \frac{3(4c-2) + 2 + 2\varepsilon}{2(4c-2) + \varepsilon} = \frac{12c - 4 + 2\varepsilon}{8c - 4 + \varepsilon}$$

It is easy to check that the expression given above is increasing in $\varepsilon$. Thus we have that the competitive ratio c is bounded from below by the value of the above expression for $\varepsilon = -1$:

$$c \ge \frac{12c - 6}{8c - 5}. \tag{4.2}$$

The smallest value $c \ge 1$ satisfying Equation (4.2) is $c = \frac{17+\sqrt{97}}{16} \approx 1.678 > \frac{5}{3}$, which contradicts the assumption that A is c-competitive with $c < 5/3$. □

We now establish a (trivial) lower bound on the competitive ratio of any deterministic algorithm for ONLINE-TSP with open schedules.

**Theorem 4.2.4.** *For open schedules no deterministic algorithm for* ONLINE-TSP *can achieve a competitive ratio* $c < 2$.

*Proof.* The metric space for the instance of ONLINE-TSP consists of a path $G = (V, E)$ of 3 vertices $v_{-1}, v_0$ and $v_1$ with the origin $v_0$ being the middle vertex at a distance of 1 from each end. Let A be a deterministic online algorithm with competitive ratio $c < 2$.

There is no request until time 1. At this time the server can not be between $v_0$ and $v_1$ since otherwise we could issue a request $(1, v_{-1},)$ which the optimal server could serve at a cost of 1 by moving at time zero towards $v_{-1}$. However the online server would reach $v_{-1}$ not before time 2, contradicting the assumption that $c < 2$.

Therefore we know that the online server must be between $v_0$ and $v_{-1}$. We then issue at time one a request at $v_1$ and by the same argument as before we once again get a contradiction to the assumption that $c < 2$. □

Aussiello et.al. gave a slightly more complicated proof of the last result, where the first request is issued at time zero [AFL+95]—showing that the lower bound also holds when defining the completion time as the difference between the time when the last request is served and the time that the first request is issued. No better lower bound for ONLINE-DARP with open schedules is known.

All the lower bounds we proved so far have dealt with the objective of minimizing the completion time. To conclude this section, we show that the same simple idea that we used for proving Theorem 4.2.4 also yields that there can be no competitive algorithms for the task of minimizing the *maximal waiting time* or the *maximal flow time* and that there can also be no competitive algorithms for minimizing the *average flow time* or the *average waiting time*.

**Theorem 4.2.5.** *The competitive ratio for online algorithm for* ONLINE-DARP *on the real line minimizing the maximal flow- or waiting time is unbounded.*

*Proof.* The metric space is the real line, with 0 as the origin of the server. The first request occurs at time 1. At this time the online server is either to the right of 0, at 0, or to the left of 0. W.l.o.g. we assume that the position of the online server is less or equal to 0. Then the following request occurs: $(1, 1, 1 + \varepsilon)$. Clearly the optimal offline server begins to move at time 0 towards 1 and achieves a waiting time of 0 and a flow time of $\varepsilon$. The waiting time of the online server, however, is not less then 1 and the flow time is not less then $1 + \varepsilon$. This shows that the competitive ratio for both the maximal flow time and the maximal waiting time are unbounded. □

**Corollary 4.2.6.** *The competitive ratio for online algorithm for* ONLINE-DARP *on the real line minimizing the average flow- or waiting time is unbounded.*

*Proof.* Notice that the proof of Theorem 4.2.5 uses an instance of ONLINE-DARP with just one request. Therefore the maximal flow and waiting time and the average flow and waiting time are equal for this instance. □

## 4.3 Strategies FIFO and FIRSTFIT are not competitive

In this section we will study two very simple algorithms, namely FIFO and FIRSTFIT. We show that both algorithms can not achieve a constant competitive ratio.

**Definition 4.3.1 (Algorithm FIRSTFIT).** After serving a request, select the unserved request with source nearest to the current position of the server and serve this request. In the closed schedule case, the server moves towards the origin when there is currently no unserved request.

**Definition 4.3.2 (Algorithm FIFO).** Serve the requests in the order of their release times. In the closed schedule case, the server moves towards the origin when there is currently no unserved request.

We will now prove that neither FIRSTFIT nor FIFO are competitive with a constant ratio.

**Theorem 4.3.3.** *For the algorithm FIRSTFIT for* ONLINE-TSP *there does not exist any constant* $c$ *such that FIRSTFIT is* $c$*-competitive.*

*Proof.* An instance of ONLINE-TSP where all requests are known in advance, is also an instance of the (offline) TSP in metric spaces (where we have to find a shortest closed walk connecting a set of points in a metric space). The online algorithm FIRSTFIT is in an offline setting equivalent to the Nearest Neighbor heuristic (NN) for the TSP. Rosenkrantz, Stearns and Lewis showed that for each $m > 3$, there exists a traveling salesman graph with $n = 2^m - 1$ nodes and having only positive edge weights such that $C_{NN} \geq (1/3 \log(n+1) + 4/9)\mathrm{OPT}$ [RSL77]. □

**Theorem 4.3.4.** *For the algorithm FIFO for* ONLINE-TSP *there does not exist any constant* $c$ *such that FIRSTFIT is* $c$*-competitive.*

*Proof.* The metric space is a graph with two vertices $v_0$ and $v_1$ connected via an edge of weight 1. The server is initially at the origin $v_0$. We present the following request sequence:

$$r_i := \begin{cases} (1/2(i-1), 0) & \forall i = 1, \ldots, N \text{ with } i \text{ odd} \\ (1/2(i-1), 1) & \forall i = 1, \ldots, N \text{ with } i \text{ even} \end{cases}$$

The FIFO algorithm will incur a cost of N whereas the optimal server will incur a cost of less than $N/2 + 1$. □

## 4.4 Preliminaries for the competitive analysis of ONLINE-DARP

In this section we will derive some results which we will use for the competitiveness proofs in the following sections.

First we introduce some notation for analyzing the closed schedule case of both ONLINE-DARP and ONLINE-FIFO-DARP. For a set $\sigma$ of requests and a point $x$ 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 $t$) which starts in $x$ at time $t$, serves all requests from $\sigma$ and ends in the origin (for ONLINE-FIFO-DARP we additionally require the schedule to serve requests with a common source in the order of their release times). Clearly for any $t' \geq t$ it is true that $L^*(t', x, \sigma) \leq L^*(t, x, \sigma)$. Moreover, $\text{OPT}(\sigma) = L^*(0, o, R)$ and thus we get that $\text{OPT}(\sigma) \geq L^*(t, o, \sigma)$ for any time $t \geq 0$.

Let $r_m = (t_m, a_m, b_m)$ be the last request from $\sigma$. Since the optimum offline server can not serve $r_m$ before it is released we get that

$$\text{OPT}(\sigma) \geq \max\{L^*(t, o, \sigma), t_m + d(a_m, b_m) + d(b_m, o)\} \quad \text{for any } t \geq 0. \tag{4.3}$$

Notice that all of these claims are equally true for ONLINE-DARP and for ONLINE-FIFO-DARP. However the following lemma does not hold when there are precedence constraints present, since we explicitly reorder requests in a schedule.

**Lemma 4.4.1.** *For an instance of* ONLINE-DARP, *let* $\sigma = \{r_1, \dots, r_m\}$ *be the set of requests. Then for any* $t \geq t_m$ *and any request* $r_i = (t_i, a_i, b_i)$ *from* $\sigma$

$$L^*(t, b_i, \sigma \setminus r_i) \leq L^*(t, o, \sigma) - d(a_i, b_i) + d(a_i, o).$$

*Proof.* Consider an optimum schedule $S^*$ which starts at the origin $o$ at time $t$, serves all requests in $\sigma$ and has length $L^*(t, o, \sigma)$. It suffices to construct another schedule $S$ which starts in $b_i$ no earlier than time $t$ serves all requests in $\sigma \setminus r_i$ and has length at most $L^*(t, o, \sigma) - d(a_i, b_i) + d(a_i, o)$.

Let $S^*$ serve the requests in the order $r_{j_1}, \dots, r_{j_m}$ such that $r_i = r_{j_k}$. Notice that if we start in $b$ at time $t$ and serve the requests in the order

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

and then move back to the origin, the resulting schedule $S$ has the desired properties. $\square$

We will now derive similar results for the open schedule case of ONLINE-DARP and ONLINE-FIFO-DARP. For a set $\sigma$ of requests, a time $t$ and a point $x$, let $\tilde{L}^*(t, x, \sigma)$ denote the length of a shortest schedule (i.e., the time difference between its completion time and $t$) which starts in $x$ at time $t$ and serves all requests from $\sigma$ (again, for ONLINE-FIFO-DARP we additionally require the schedule to serve request with a common source in the order of their release times). The difference to $L^*(t, x, \sigma)$ 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, $\mathrm{OPT}(\sigma) = \tilde{L}^*(0, o, \sigma)$ and thus $\mathrm{OPT}(\sigma) \geq \tilde{L}^*(t, o, \sigma)$ for any time $t \geq 0$. Since the optimum offline server can not serve the last request $r_m = (t_m, a_m, b_m)$ from $\sigma$ before this request is available we get that

$$\mathrm{OPT}(\sigma) \geq \max\{L^*(t, o, \sigma), t_m + d(a_m, b_m)\} \quad \text{for any } t \geq 0. \tag{4.4}$$

These results hold for both ONLINE-DARP and ONLINE-FIFO-DARP. The following lemma can be proved similarly to Lemma 4.4.1, and its proof again requires that no precedence constraints are present (i.e., the lemma is *not* valid for ONLINE-FIFO-DARP).

**Lemma 4.4.2.** *For an instance of* ONLINE-DARP, *let* $\sigma = \{r_1, \ldots, r_m\}$ *be the set of requests. Then for any* $t \geq t_m$ *and any request* $r_i = (t_i, a_i, b_i)$ *from* $\sigma$

$$\tilde{L}^*(t, b_i, \sigma \setminus r_i) \leq \tilde{L}^*(t, o, \sigma) - d(a_i, b_i) + d(b, o),$$

*where* $b$ *is the endpoint of a path with length* $\tilde{L}^*(t, o, \sigma)$. *In particular*

$$\tilde{L}^*(t, b_i, \sigma \setminus r_i) \leq 2\tilde{L}^*(t, o, \sigma) - d(a_i, b_i).$$

## 4.5 The REPLAN algorithm

**Definition 4.5.1 (Algorithm REPLAN).** Algorithm REPLAN may assume the following states (initially it is IDLE):

**IDLE** Wait until a request is released, then goto PLAN.

**BUSY** Execute the current schedule. When a new request is released goto PLAN. When the current schedule is completed, goto IDLE.

**PLAN** Produce a preliminary optimal transportation schedule starting in the current position of the server for all currently available unserved requests $R$. Then goto BUSY.

Ascheuer, Krumke and Rambau proved that the REPLAN algorithm is 5/2 competitive:

**Theorem 4.5.2.** *Algorithm REPLAN is* $5/2$*-competitive.*

*Proof.* Let $\sigma = \{r_1, \ldots, r_m\}$ be any set of requests. We distinguish between two cases depending on the current load of the REPLAN-server at the time $t_m$ (the last request becomes known).

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,

$$C_{\text{REPLAN}}(\sigma) \leq t_m + d(o, s(t_m)) + L^*(t_m, o, \sigma) \overset{(4.3)}{\leq} t_m + d(o, s(t_m)) + \text{OPT}(\sigma). \tag{4.5}$$

We now consider the second case, when the server is currently serving a request $r = (t, a, b)$. The time needed to complete the move is $d(s(t_m), b)$. Then a shortest schedule starting at $b$ serving all unserved requests is computed which has length at most $L^*(t_m, b, \sigma \setminus r)$. Thus in the second case

$$\begin{aligned} C_{\text{REPLAN}}(\sigma) &\leq t_m + d(s(t_m), b) + L^*(t_m, b, \sigma \setminus r) \\ &\leq t_m + d(s(t_m), b) + L^*(t_m, o, \sigma) - d(a, b) + d(a, o) \quad \text{(by 4.4.1)} \\ &\leq t_m + \text{OPT}(\sigma) - d(a, b) + \underbrace{d(s(t_m), b) + d(a, s(t_m))}_{=d(a,b)} + d(s(t_m), o) \\ &= t_m + d(o, s(t_m)) + \text{OPT}(\sigma). \end{aligned}$$

This means that inequality (4.5) holds in both cases. Since the REPLAN server has traveled to position $s(t_m)$ at time $t_m$, there must be a request $r_j = (t_j, a_j, b_j)$ in $\sigma$ where either $d(o, a_i) \geq d(o, s(t_m))$ or $d(o, b_i) \geq d(o, s(t_m))$. But this means that the optimal offline server will have to travel at least twice the distance $d(o, s(t_m))$ during its schedule. Thus, $d(o, s(t_m)) \leq \text{OPT}(\sigma)/2$ and using this result together with (4.5) we get that the total time the REPLAN server needs is no more than $5/2$ times that of the offline server. ☐

The REPLAN algorithm is in general not a polynomial time algorithm, since it has to solve a number of instances of (offline) DARP, which we showed in Section 2.4 to be NP-hard.

If the REPLAN algorithm uses approximate solutions instead of the optimal solution for instances of DARP, it is easy to see that we can modify the proof of the last theorem to yield the following result:

**Corollary 4.5.3.** *Let A be a $\rho$-approximate algorithm for the following problem: Given an instance of (offline)* DARP *and an additional point $x$ in the continuous metric space, compute a shortest sequence of moves starting in $x$ and terminating in the origin that serves all requests. REPLAN computing schedules with A is $5/2\rho$-competitive.* ☐

However, when using the $\rho$-approximate algorithms for closed-schedule DARP described in Chapter 2, the proof of the last theorem does not carry through: Given a $\rho$-approximate algorithm for DARP with closed schedules, we can calculate an approximate schedule starting in $x$ and terminating in the origin by inserting a "dummy request" between $x$ and $o$. However this will not necessarily yield a $\rho$-approximate solution! Also for ONLINE-FIFO-DARP the proof is not applicable, since it uses Lemma 4.4.1 which is in general not true for ONLINE-FIFO-DARP.

The following theorem proves a different competitive ratio for these cases:

**Theorem 4.5.4.** *Let B be a $\rho$-approximate algorithm for (offline)* FIFO-DARP *with closed schedules. Then REPLAN for* ONLINE-FIFO-DARP *using B to compute approximate offline solutions is $(2+\rho)$-competitive.*

*Proof.* We make the assumption, that the (offline) schedule for a set of requests $\sigma$ starting in a point $x$ and terminating at the origin, which is computed using B, is at most as long as the distance from $x$ to the origin plus a $\rho$-approximate closed schedule solution for $\sigma$ (we can always achieve this bound with the—rather impractical—strategy of returning to the origin before starting on a new tour).

Let $\sigma = \{r_1, \dots, r_m\}$ be any set of requests. Again, we distinguish between two cases depending on the current load of the REPLAN-server at the time $t_m$ (the last request becomes known).

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 (by our initial comments) length at most $d(s(t_m), o) + \rho L^*(t_m, o, \sigma)$. Thus,

$$C_{\text{REPLAN}}(\sigma) \le t_m + d(o, s(t_m)) + \rho L^*(t_m, o, \sigma) \le 3/2\mathrm{OPT}(\sigma) + \rho\mathrm{OPT}(\sigma). \tag{4.6}$$

We now consider the second case, when the server is currently serving a request $r = (t, a, b)$. The time needed to complete the move is $d(s(t_m), b)$. Then a shortest schedule starting at $b$ serving all unserved requests is computed which has length at most $d(o, b) + \rho L^*(t_m, 0, \sigma \setminus r)$. Thus in the second case

$$\begin{aligned} C_{\text{REPLAN}}(\sigma) &\le t_m + d(s(t_m), b) + d(b, o) + \rho L^*(t_m, b, \sigma \setminus r) \\ &\le 2\mathrm{OPT}(\sigma) + \rho\mathrm{OPT}(\sigma) \end{aligned}$$

□

Using the approximation algorithms from Chapter 2 and Chapter 3, we immediately get the following results:

**Corollary 4.5.5.** *REPLAN with suitable approximation algorithms is a polynomial time algorithm that can achieve the following competitive ratios:*

| | |
|---|---|
| ONLINE-DARP *on the real line:* | 5/2 |
| ONLINE-DARP *on trees:* | 13/4 |
| ONLINE-DARP *on general continuous metric spaces:* | 19/5 |
| ONLINE-FIFO-DARP *on the real line:* | 3 |
| ONLINE-FIFO-DARP *on trees:* | 11/3 |
| ONLINE-FIFO-DARP *on general continuous metric spaces:* | 17/4 |

□

Ascheuer, Krumke and Rambau also showed how to modify the proof of Theorem 4.5.2 to obtain a result about the competitiveness of REPLAN for open schedules [AKR98a]:

**Theorem 4.5.6.** *In the case of open schedules algorithm REPLAN is 3-competitive.*

*Proof.* If at the time $t_m$ when the last request $r_m = (t_m, a_m, b_m)$ from $\sigma$ is issued, the REPLAN server does not perform a carrying move, then the total time needed by REPLAN is no more than

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

The distance $d(o, s(t_m))$ can be bounded from above by $\mathrm{OPT}(\sigma)$ (instead of $\mathrm{OPT}(\sigma)/2$ as for closed schedules) and thus REPLAN needs time at most 3 times the optimum in this case.

If at time $t_m$ REPLAN performs a carrying move from $a$ to $b$, it will finish its move at time $t_m + d(s(t_m), b)$. We thus have

$$\begin{aligned} C_{\mathsf{REPLAN}}(\sigma) &\le t_m + d(s(t_m), b) + \tilde{L}^*(t_m, b, \sigma) \\ &\le t_m + d(s(t_m), b) + 2\,\tilde{L}^*(t_m, o, \sigma) - d(a, b) \quad \text{(by Lemma 4.4.2)} \\ &\le t_m + 2\,\tilde{L}^*(t_m, o, \sigma) \\ &\le 3\mathrm{OPT}(\sigma). \end{aligned}$$

This completes the proof. □

Given a $\rho$-approximate algorithm for solving offline DARP with open schedules, we can easily adapt the above proof to yield a $(1 + 2\rho)$-competitive algorithm. However, the last proof uses Lemma 4.4.2 which is not true for ONLINE-FIFO-DARP. It is easy to see that we therefore get the following results:

**Corollary 4.5.7.** *REPLAN for* ONLINE-DARP *with open schedules that solves offline instances with a $\rho$-approximate algorithm for* DARP *with open schedules is* $\min\{1 + 2\rho, 3 + \rho\}$*-competitive.*

*REPLAN for* ONLINE-FIFO-DARP *with open schedules that solves offline instances with a $\rho$-approximate algorithm for* FIFO-DARP *with open schedules is* $(3 + \rho)$*-competitive.* □

For ONLINE-TSP Aussiello et.al. showed that the REPLAN algorithm (which they call "greedy") is 5/2-competitive for the open schedule case:

**Theorem 4.5.8.** *REPLAN for* ONLINE-TSP *with open schedules is* $5/2$*-competitive on vector spaces.*

*Proof.* At time $t_m$ the last request $r_m = (t_m, a_m, b_m)$ from $\sigma((t_1, a_1) \dots (t_m, a_m))$ is issued. Let $x$ be the current position of the server. It will then need at most time $\tilde{L}^*(t_m, x, \sigma)$ to finish. Let $V(\sigma) := \{a_i : 1 \le i \le m\} \cup \{o\}$ be the set of points including the origin and the points in $M$ where requests occur. Let $P^*$ be the shortest path through all points in $V(\sigma)$, and let $d(P^*)$ be its length. Notice that $x$ lies on the shortest path between two points $a, b \in V(\sigma)$. It is easy to see that the distance between $x$ and one of the two endpoints of $P^*$ is therefore at most half the length of $P^*$. Let $y$ be this endpoint, i.e., $d(x, y) \le d(P^*)$. Then

$$\begin{aligned} C_{\mathsf{REPLAN}}(\sigma) &\le t_m + \tilde{L}^*(t_m, x, \sigma) \\ &\le t_m + d(x, y) + d(P^*) \\ &\le 5/2\mathrm{OPT}(\sigma) \end{aligned}$$

□

## 4.6 The IGNORE algorithm

**Definition 4.6.1 (Algorithm IGNORE).** Algorithm IGNORE has an internal buffer for requests. It may assume the following states (initially it is IDLE):

**IDLE** Wait until a request is released, then goto PLAN.

**BUSY** Execute the current schedule. While it is in work store the upcoming requests in a buffer ("ignore them"). When the current schedule is completed, goto IDLE if the buffer is empty, else goto PLAN.

**PLAN** Produce a preliminary optimal transportation schedule for all currently available requests $R$ (taken from the buffer). Then goto BUSY.

Ascheuer, Krumke and Rambau showed that IGNORE is a 5/2-competitive algorithm for ONLINE-DARP [AKR98a]. Just like REPLAN, the IGNORE algorithm has to solve offline instances of DARP and is therefore in general not a polynomial time algorithm. However, the proof by Ascheuer, Krumke and Rambau can be applied directly to IGNORE using a $\rho$-approximate algorithm for solving (offline) DARP and is even applicable to ONLINE-FIFO-DARP. We also note that the competitive ratio for IGNORE holds also for discrete metric spaces—we do not require the metric space to be continuous.

**Theorem 4.6.2.** *IGNORE employing a $\rho$-approximate algorithm for solving instances of (offline)* DARP *(or* FIFO-DARP*) with closed schedules is a $5/2\rho$-competitive algorithm for* ONLINE-DARP *(or* ONLINE-FIFO-DARP*)*

*Proof.* Consider $t_m$, the moment in time when the last request $r_m$ becomes known. If the IGNORE server is currently idle at the origin $o$, then its completes its last schedule no later than

$$\begin{aligned} t_m + \rho L^*(t_m, o, \{r_m\} &\le \rho\,(t_m + d(o, a_m) + d(a_m, b_m) + d(b_m, o)) \\ &\le \rho\,(\mathrm{OPT}(\sigma) + d(o, a_m)) \\ &\le 3/2\rho\mathrm{OPT}(\sigma). \end{aligned}$$

Here, we have used again the fact that the optimum offline server will have to travel at least twice the distance $d(o, a_m)$.

It remains the case that at time $t_m$ the IGNORE server is currently working on a schedule $S$ for a subset $\sigma_S$ of the requests. Let $t_S$ denote the starting time of this schedule. Thus, the IGNORE-server will complete $S$ latest at time $t_S + \rho L^*(t_S, o, \sigma_S)$. Denote by $\sigma_{\geq t_S}$ the set of requests presented after time $t_S$. Notice that $\sigma_{\geq t_S}$ 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 $t_S + \rho L^*(t_S, o, \sigma_S) + \rho L^*(t_m, o, \sigma_{\geq t_S})$.

Let $r_f \in \sigma_{\geq t_S}$ be the first request from $\sigma_{\geq t_S}$ served by the offline server. Thus

$$\mathrm{OPT}(\sigma) \geq t_f + L^*(t_f, a_f, \sigma_{\geq t_S}) \geq t_S + L^*(t_m, a_f, \sigma_{\geq t_S}). \tag{4.7}$$

Now $L^*(t_m, o, \sigma_{\geq t_S}) \leq d(o, a_f) + L^*(t_m, a_f, \sigma_{\geq t_S})$ and $L^*(t_S, o, \sigma_S) \leq \mathrm{OPT}(\sigma)$. This gives us that

$$\begin{aligned} C_{\mathsf{IGNORE}}(\sigma) &\leq t_S + \rho\,(\mathrm{OPT}(\sigma) + d(o, a_f) + L^*(t_m, a_f, \sigma_{\geq t_S})) \\ &\overset{(4.7)}{\leq} \rho\,(t_S + \mathrm{OPT}(\sigma) + d(o, a_f) + L^*(t_m, a_f, \sigma_{\geq t_S})) \\ &\leq 2\rho\,\mathrm{OPT}(\sigma) + \rho d(o, a_f) \\ &\leq 5/2\rho \cdot \mathrm{OPT}(\sigma). \end{aligned}$$

This completes the proof. □

Using the approximation algorithms from Chapter 2 and Chapter 3, we immediately get the following results:

**Corollary 4.6.3.** *IGNORE with suitable approximation algorithms is a polynomial time algorithm that can achieve the following competitive ratios:*

| | |
|---|---|
| ONLINE-DARP *on the real line:* | 5/2 |
| ONLINE-DARP *on trees:* | 25/8 |
| ONLINE-DARP *on general metric spaces:* | 9/2 |
| ONLINE-FIFO-DARP *on the real line:* | 5/2 |
| ONLINE-FIFO-DARP *on trees:* | 25/6 |
| ONLINE-FIFO-DARP *on general metric spaces:* | 45/8 |

□

Ascheuer, Krumke and Rambau proved that the open schedule case of ONLINE-DARP is 4-competitive [AKR98a]. Again, their proof can be applied directly to IGNORE using a ρ-approximate algorithm for solving (offline) DARP with open schedules and is also applicable for ONLINE-FIFO-DARP:

**Theorem 4.6.4.** *In the case of* ONLINE-DARP *(or* ONLINE-FIFO-DARP*) with open schedules, the competitive ratio of algorithm IGNORE using a ρ-approximate algorithm for (offline)* DARP *(or* FIFO-DARP*) is $2 + 2\rho$.*

*Proof.* The only interesting case is that at the time $t_m$ when the last request 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 ends at point $y$.

Then the schedule will be completed no later than time $t_S + \rho\tilde{L}^*(t_S, x, \sigma_S)$, where $\sigma_S$ denotes the subset of requests served in the current schedule $S$. The IGNORE server will complete its total work no later than time

$$t_S + \rho\tilde{L}^*(t_S, x, \sigma_S) + \rho\tilde{L}^*(t_m, y, \sigma_{\geq t_S}),$$

where $\sigma_{\geq t_S}$ denotes the set of requests presented after time $t_S$.

Let $r_f$ be the first request from the set $\sigma_{\geq t_S}$ of ignored requests served by the optimal offline server (note that therefore in the case of ONLINE-FIFO-DARP the online server is also allowed to serve $r_f$ as the first among the requests in $\sigma_{\geq t_S}$). Then

$$\mathrm{OPT}(\sigma) \geq t_f + \tilde{L}^*(t_S, a_f, \sigma_{\geq t_S}) \geq t_S + \tilde{L}^*(t_S, a_f, \sigma_{\geq t_S}).$$

Thus, we have that

$$\begin{aligned} C_{\mathsf{IGNORE}}(\sigma) &\leq \rho\tilde{L}^*(t_S, x, \sigma_S) + d(y, a_f) + t_S + \rho\tilde{L}^*(t_m, a_f, \sigma_{\geq t_S}) \\ &\leq \rho\tilde{L}^*(t_S, x, \sigma_S) + d(y, a_f) + \mathrm{OPT}(\sigma) \\ &\leq 2\rho\,\mathrm{OPT}(\sigma) + d(x, o) + d(y, a_f). \end{aligned}$$

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

## 4.7 Other competitive algorithms

In this section we will consider the algorithm PAH for closed schedule ONLINE-TSP and the SLEEP algorithm for ONLINE-DARP. The strategy PAH—which abbreviates "Plan at Home"—is basically an adaption of the IGNORE strategy. It is an open question whether a similar algorithm for ONLINE-DARP can perform better than IGNORE. The SLEEP algorithm on the other hand is not suitable for practical implementations—as will become apparent when we explain how it works. However, it is a competitive strategy, which once again illustrates the gap between competitiveness results and practical relevance.

**Definition 4.7.1 (Algorithm PAH).** If the server is at the origin, it plans an optimal schedule that serves all requests (and terminates again in the origin) and the server begins to follow this schedule.

If a new request $(t, a)$ is presented, then the server takes one of the two following actions, depending on its current position $x$: If $d(a, o) > d(x, o)$ then the server goes back to the origin (along a shortest path), otherwise the server ignores the request until it arrives at the origin.

Aussiello et.al. showed that the algorithm PAH is a 2-competitive algorithm for ONLINE-TSP with closed schedules.

**Theorem 4.7.2.** *Algorithm PAH is 2-competitive for* ONLINE-TSP *with closed schedules.*

*Proof.* Let $\sigma = (r_1, \dots, r_m)$ be a set of requests. Consider $t_m$, the moment in time when the last request $r_m = (t_m, a_m)$ becomes known. Let $x$ be the current position of the server. There are three cases:

**Case 1:** At time $t_m$ the algorithm is at the origin. It then requires time $t_m + L^*(t_m, o, \{r_m\})$ to serve all requests, which is at most 2OPT.

**Case 2:** At time $t_m$ the algorithm is currently not at the origin (i.e., it is serving a schedule or moving back to the origin) and $d(a_m, o) > d(x, o)$. The server goes back to the origin where it will arrive at time $t_m + d(x, o)$ and then it serves all remaining requests. We notice that OPT $\geq t_m + d(o, a_m)$. Therefore

$$\begin{aligned} C_{\mathsf{PAH}} &\leq t_m + d(x, o) + L^*(t_m, o, \sigma) \\ &\leq t_m + d(a_m, o) + L^*(t_m, o, \sigma) \\ &\leq 2\text{OPT} \end{aligned}$$

**Case 3:** At time $t_m$ the algorithm is currently moving back to the origin due to an earlier occurrence of a request $r_j = (t_j, a_j)$ with $d(a_j, o) < d(x_j, o)$ where $x_j$ is the position of the server at time $t_j$. We can apply the argument from Case 2 to get the result.

**Case 4:** At time $t_m$ the algorithm is currently serving a schedule and $d(a_m, o) < d(x, o)$. Let $\sigma_{\text{cur}}$ be the set of requests that are currently being served and let $\sigma_{\text{ig}}$ be the set of requests that have been temporarily ignored since the server last left the origin. Let $r_j = (t_j, a_j)$ be the first request in $\sigma_{\text{ig}}$ that is served by an optimal offline solution, then $L^*(t_m, a_j, \sigma_{\text{ig}})$ is the length of a shortest path starting in $a_j$, visiting all points where requests in $\sigma_{\text{ig}}$ occur and finishing at $o$. Clearly, OPT $\geq t_j + L^*(t_m, a_j, \sigma_{\text{ig}})$. Let $x_j$ be the position of the server at time $t_j$, we note that $d(a_j, o) < d(x_j, o)$ since $r_j$ was ignored. We further note that the current tour has started at the origin not after time $t_j - d(x_j, o)$. Therefore we get

$$\begin{aligned} C_{\mathsf{PAH}} &\leq t_j - d(x_j, o) + L^*(0, o, \sigma_{\text{cur}} + L^*(t_m, o, \sigma_{\text{ig}}) \\ &\leq t_j - d(a_j, o) + L^*(0, o, \sigma_{\text{cur}}) + L^*(t_m, a_j, \sigma_{\text{ig}}) + d(a_j, o) \\ &\leq t_j + L^*(t_m, a_j, \sigma_{\text{ig}}) + L(0, o, \sigma_{\text{cur}}) \\ &\leq 2\text{OPT} \end{aligned}$$

This completes the proof. □

Rather surprisingly, Ascheuer, Krumke and Rambau showed that the following algorithm for ONLINE-DARP is competitive [AKR98a]:

**Definition 4.7.3 (Algorithm SLEEP).** The algorithm has a fixed "waiting scaling" parameter $\theta > 1$. The algorithm also records a "base time" $T$ which is initially equal to zero. From time to time the algorithm consults its "work-or-sleep" routine: this subroutine computes a shortest schedule for all unserved requests. If this schedule can be completed before time $\theta T$ the subroutine returns $(S, \text{work})$, otherwise it returns $(S, \text{sleep})$.

The server of algorithm SLEEP can be in four 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 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 four states.

- If the algorithm is idle and a new request arrives, it updates the base time $T$ to the release time of the new request (which equals the current time). It then calls "work-or-sleep". If the result is $(S, \text{work})$, the algorithm resets $T$ to the completion time of $S$ and enters the working state.

  If the result of "work-or-sleep" is $(S, \text{sleep})$, then the algorithm enters the sleeping state.

- In the sleeping state the algorithm resets its base time to $T' = \theta T$ and simply does nothing (or sleeps) until time $T'$. At time $T'$ the algorithm reconsults its "work-or-sleep" subroutine. This process is continued until the server eventually enters the working state (since the number of requests is finite).

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

**Theorem 4.7.4.** *Algorithm SLEEP for* ONLINE-DARP *with closed schedules has a competitive ratio of* $\max\left\{\frac{5}{2}, 2 + \frac{1}{\theta - 1}\right\}$.

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

**Case 1:** The server is idle.

In this case the algorithm computes a shortest schedule for the requests in $\sigma_{=t_m}$ and resets its base time to $T = t_m$. The SLEEP-server will start its work at time $T\theta^i$, where $i$ is the smallest nonnegative integer such that $T\theta^i + L^*(T\theta^i, o, \sigma_{=t_m}) \le T\theta^{i+1}$. Notice that for all $i$ we have $L^*(T\theta^i, o, \sigma_{=t_m}) = L^*(t_m, o, \sigma_{=t_m}) \le \mathrm{OPT}(\sigma)$.

In other words, the server starts its work at time $T\theta^i$ where $i \ge 0$ is the smallest nonnegative integer such that

$$L^*(t_m, o, \sigma_{=t_m}) \le T\theta^i(\theta - 1). \tag{4.8}$$

The server will complete its work at time $T\theta^i + L^*(t_m, o, \sigma_{=t_m})$.

If $i = 0$, then $C_{\mathsf{SLEEP}}(\sigma) = t_m + L^*(t_m, o, \sigma_{=t_m}) \le 2\mathrm{OPT}(\sigma)$. If $i > 0$, then the minimality of $i$ yields that Equation (4.8) is false for $i - 1$. Therefore we get that

$$\mathrm{OPT}(\sigma) \ge L^*(t_m, o, \sigma_{=t_m}) > T\theta^{i-1}(\theta - 1). \tag{4.9}$$

On the other hand

$$\begin{aligned} C_{\mathsf{SLEEP}}(\sigma) &\le T\theta^i + L^*(t_m, o, \sigma_{=t_m}) \\ &\overset{(4.9)}{<} \frac{\theta}{\theta - 1}\mathrm{OPT}(\sigma) + L^*(t_m, o, \sigma_{=t_m}) \\ &\le \left(1 + \frac{\theta}{\theta - 1}\right)\mathrm{OPT}(\sigma) \\ &= \left(2 + \frac{1}{\theta - 1}\right)\mathrm{OPT}(\sigma). \end{aligned}$$

**Case 2:** The server is sleeping.

Let $T\theta^p$ for some $p \ge 0$ be the time when the algorithm went to sleep the last time. At time $T\theta^{p+1}$ the algorithm will reconsult its "work-or-sleep" subroutine. We have that $T\theta^p \le t_m \le T\theta^{p+1}$.

The server will start its last schedule at time $T\theta^{p+1+i}$, where $i$ is the smallest integer such that $T\theta^{p+1+i} + L^*(t_m, o, \sigma') \le \delta\theta^{p+i+2}$, where $\sigma'$ denotes the set of yet unserved requests. By similar arguments as in Case 1 we get that

$$C_{\mathsf{SLEEP}}(\sigma) \le \left(2 + \frac{1}{\theta - 1}\right)\mathrm{OPT}(\sigma).$$

**Case 3:** The algorithm is working.

If after completion of the current schedule $S$ the server enters the sleep state then the arguments given above establish that the completion time does not exceed $\left(2 + \frac{1}{\theta-1}\right)\mathrm{OPT}(\sigma)$.

The remaining case is that the SLEEP-server starts its final schedule immediately after having completed $S$. Thus, from the time $t_S$ where the server started $S$, the SLEEP algorithm behaves exactly like the IGNORE strategy and the arguments given in the proof of Theorem 4.6.2 show that in this case $C_{\mathsf{SLEEP}}(\sigma) \le 5/2\mathrm{OPT}(\sigma)$. This completes the proof. $\square$

We note that the second factor $1 + \frac{\theta}{\theta-1} = 2 + \frac{1}{\theta-1}$ is at most $5/2$ provided that $\theta \geq 3$. We thus obtain the following corollary.

**Corollary 4.7.5.** *If $\theta \geq 3$, then algorithm* SLEEP *is* $5/2$*-competitive.* □

# Chapter 5

# ONLINE-DARP under Reasonable Load

In this chapter we introduce a new concept for studying online algorithms in continuously operating systems. The main idea is to restrict the request sets to sequences which "make sense" in a continuously operating system and at the same time to measure "how difficult" they are. For this twin task we introduce the new concept of $\Delta_0$-*reasonable* request sets.

We begin by motivating the basic idea behind the concept. We then formally define what we mean by a $\Delta_0$-reasonable request set. Finally we will study the algorithms REPLAN and IGNORE for ONLINE-DARP under reasonable load. Our main result is, that for ONLINE-DARP (and also for ONLINE-FIFO-DARP) under $\Delta_0$-reasonable load, IGNORE yields a maximal and an average flow time of at most $2\Delta_0$, whereas the maximal and the average flow time of REPLAN are unbounded. The results presented in this chapter have been published together with S. O. Krumke and J. Rambau in [HKR99].

## 5.1 Introducing reasonable load

In Chapter 4 we studied the problem ONLINE-DARP, focusing on the REPLAN and IGNORE algorithm. Competitive analysis of ONLINE-DARP provided the following:

- There are competitive algorithms (IGNORE and REPLAN) for the goal of minimizing the *total completion time* of the schedule.
- For the task of minimizing the *maximal flow or waiting time* or the *average flow or waiting time* there can be no competitive algorithm.

When considering a continuously operating system with a possibly infinite request sequence, the total completion time is meaningless. In this case, the existing positive results cannot be applied and the negative results tell us, that we cannot hope for performance guarantees that are of practical relevance. In particular, the two algorithms IGNORE and REPLAN cannot be distinguished by classical competitive analysis.

The objective function value for an offline optimal solution that is used in classical competitive analysis can be thought of measuring how difficult a particular instance of a problem is. We are looking for a similar measure telling us how difficult it is to serve a set of possibly infinitely many requests.

At the same time, a "good" algorithm should show a stable behavior: If possible, it should keep up with its work. However, when is it possible for an algorithm to do that in a continuously operating system? In queuing theory this is usually modeled by a stability assumption: the rate of incoming requests is at most the rate of requests served. Since in many 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.

For the twin task of "measuring" the combinatorial complexity of a request set, and deciding whether a "good" algorithm can be stable when serving it, we introduce the notion of $\Delta_0$-*reasonable* request sets. A set of requests is $\Delta_0$-reasonable if—roughly speaking—requests released during a period of time $\Delta \geq \Delta_0$ can be served in time at most $\Delta$. A set of requests $R$ is *reasonable* if there exists a $\Delta_0 < \infty$ such that $R$ is $\Delta_0$-reasonable. That means, 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.

We already remarked that $\Delta_0$ measures in a sense the combinatorial difficulty of a request set. Thus, it is natural to ask for performance guarantees for algorithms in terms of $\Delta_0$. Our main result on the ONLINE-DARP under $\Delta_0$-reasonable load is the following:

**Theorem 5.1.1.** *For the* ONLINE-DARP *under $\Delta_0$-reasonable load, IGNORE yields a maximal and an average flow time of at most $2\Delta_0$, whereas the maximal and the average flow time of REPLAN are unbounded.*

We prove this result in Sections 5.4 and 5.5.

We will also show how we can derive results for IGNORE when using an approximate algorithm for solving offline instances of DARP. For this we refine the notion of reasonable request sets again, introducing a second parameter that tells us, how "fault tolerant" the request sequence is. In other words, the second parameter tells us, how "good" the algorithm has to be, to show stable behavior. Again, roughly speaking, a set of requests is $(\Delta_0,\rho)$-reasonable if requests released during a period of time $\Delta \geq \Delta_0$ can be served in time at most $\Delta/\rho$. If $\rho = 1$, we get the notion of $\Delta_0$-reasonable as explained above. For $\rho > 1$, the algorithm can work "sloppy" or have break-downs to an extend measured by $\rho$ and still show a stable behavior.

We finally remark, that all the results proved in this chapter are equally applicable to ONLINE-FIFO-DARP—again, as for the competitiveness results concerning IGNORE proved in Section 4.6, additional precedence constraints on the requests do not matter for the proofs.

## 5.2 Preliminaries

We start with some useful notation. The goal is to avoid the special meaning for time 0 in the analysis of online algorithms with time stamped requests.

**Definition 5.2.1.** The *time shift* of $r$ by $\tau \in \mathbb{R}$ is the request

$$r + \tau := (t + \tau, a, b)$$

The *offline version* of $r$ is the request

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

**Definition 5.2.2.** Let $R$ be a request set for ONLINE-DARP. The *time shift* of $R$ by $\tau$ is the request set

$$R + \tau := \{r + \tau \ : \ r \in R\}.$$

The *offline version* of $R$ is the request set

$$R^{\textit{offline}} := \left\{r^{\textit{offline}} \ : \ r \in R\right\}.$$

An important characteristic of a request set with respect to system load considerations is the time period in which it is released.

**Definition 5.2.3.** Let $R$ be a finite request set for ONLINE-DARP. The *release span* $\Delta(R)$ of $R$ is defined as

$$\Delta(R) := \max_{r=(t,a,b)\in R} t - \min_{r=(t,a,b)\in R} t.$$

The next definition describes a class of objectives that discard the special meaning of time 0.

**Definition 5.2.4.** A cost function $C$ for the ONLINE-DARP is *translation invariant* if for any set of requests $R = r_1, r_2, \dots$ and for all algorithms $A$ we have

$$C_A(R) = C_A(R + \tau) \text{ forall } \tau \in \mathbb{R}.$$

**Example 5.2.5.** We consider some objective functions which are relevant in this chapter and remark whether they are translation invariant:

- The *completion time* or *makespan* of a request set $R$, i.e., the time when all requests from $R$ are served, is not translation invariant. However, if we set all release times in a request set $R$ to $0$ then we get the translation invariant objective $C^{comp}(R^{offline})$. This is the *offline completion time* of $R$.
- Assume the first request in $R$ is released at time $t_1$. Then the *flow time of* $R$ is the translation invariant objective $C^{comp}(R - t_1)$.
- The *average flow time* and the *maximal flow time* of a request set $R$, i.e., the average resp. maximum time taken over all request $r \in R$ that $r$ spends in the system, are translation invariant. We consider these objectives as especially important for a continuously operating system.

## 5.3 Reasonable load

We will now derive formally the notion of a $(\Delta_0,\rho)$-reasonable set, where $\Delta_0 > 0$ serves as a measure of the combinatorial complexity of the instance, whereas $\rho > 1$ measures how stable the sequence is, in a worst-case fashion.

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

**Definition 5.3.1.** Let $R$ be a (possibly infinite) set of request for ONLINE-DARP. A weakly monotone function

$$f\colon \begin{cases} \mathbb{R} & \to & \mathbb{R}, \\ \Delta & \mapsto & f(\Delta); \end{cases}$$

is a *load bound* on $R$ if for any $\Delta \in \mathbb{R}$ and any finite subset $S$ of $R$ with $\Delta(S) \leq \Delta$ we have

$$C^{comp}_{OPT}(S^{offline}) \leq f(\Delta).$$

*Remark 5.3.2.*

- If the whole request set $R$ is finite then there is always the trivial load bound given by the total completion time of $R$.
- For every load bound $f$ we may set $f(0)$ to be the maximum completion time we need for a single request, and nothing better can be achieved.

A stable situation would be characterized by a load bound equal to the identity on $\mathbb{R}$. In that case we would never get more work to do than we can accomplish. If it has a load bound equal to a function $id/\rho$, where $id$ is the identity and where $\rho \geq 0$, then $\rho$ measures the tolerance of the request set: An algorithm that is by a factor $\rho$ worse then optimal will still accomplish all the work that it gets. However, we cannot expect that the identity (or any linear function) is a load bound for ONLINE-DARP or ONLINE-FIFO-DARP because of the following observation: a request set consisting of one single request has a release span of $0$ whereas in general it takes non-zero 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.

**Definition 5.3.3.** A load bound $f$ is *($\Delta_0$,$\rho$)-reasonable* for some $\Delta_0, \rho \in \mathbb{R}$ if

$$\rho f(\Delta) \leq \Delta \quad \text{for all } \Delta \geq \Delta_0$$

A request set $R$ is *($\Delta_0$,$\rho$)-reasonable* if it has a ($\Delta_0$,$\rho$)-reasonable load bound.

For $\rho = 1$, we say that the request set is $\Delta_0$-reasonable.

In other words, a load bound is *($\Delta_0$,$\rho$)-reasonable*, if it is bounded from above by $1/\rho \cdot id(x)$ for all $x \geq \Delta_0$ and by the constant function with value $1/\rho\Delta_0$ otherwise.

*Remark 5.3.4.* If $\Delta_0$ is sufficiently small so that all request sets consisting of two or more requests have a release span larger than $\Delta_0$ 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. (By "stable" we mean that the number of unserved requests in the system does not become arbitrarily large.)

If for the same setting, we also have $\rho > 1$, then the server only has to work for a time period $1/\rho \cdot \Delta$ in any interval $\Delta \geq \Delta_0$ and it will still provide for a stable system.

## 5.4 Performance guarantees for IGNORE

We are now in a position to prove the performance guarantees for minimizing the maximal resp. average flow time in ONLINE-DARP (and ONLINE-FIFO-DARP) for algorithm IGNORE stated in Theorem 5.1.1. The algorithm IGNORE has been defined in 4.6.1. We assume that IGNORE solves instances of (offline) DARP (or FIFO-DARP) using a $\rho$-approximate algorithm.

The algorithm IGNORE induces a dissection of the time axis $\mathbb{R}$ in the following way: Because maximal flow time is a translation invariant objective function, we can assume, w.l.o.g., that the first set of requests arrives at time $0$. Let $\Delta_1$ be the time period the server works on the first available set of requests. Moreover, for $i > 1$ let $\Delta_i$ be the time period the server is working on the requests that have been ignored during the last $\Delta_{i-1}$ time units. Then the time axis is split into the intervals

$$[0, \Delta_1], [\Delta_1, \Delta_1 + \Delta_2], [\Delta_1 + \Delta_2, \Delta_1 + \Delta_2 + \Delta_3], \ldots$$

Let us denote these intervals by $I_1, I_2, I_3, \ldots$. Moreover, let $R_i$ be the set of those requests that come up in $I_i$. Clearly, the complete set of requests $R$ is the 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 beginning of interval $I_{i+1}$) and is done $\Delta_{i+1}$ 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 $\rho$ times the optimal completion time of $R_i^{\mathit{offline}}$. In other words:

**Lemma 5.4.1.** *For all* $i > 0$ *we have*

$$\Delta_{i+1} \leq \rho C^{\mathit{comp}}_{\mathsf{OPT}}(R_i^{\mathit{offline}}).$$

Let us now recall and prove the first statement of Theorem 5.1.1.

**Theorem 5.4.2.** *Let* $\Delta_0 > 0$ *and* $\rho \geq 1$*. For all instances of* ONLINE-DARP *(*ONLINE-FIFO-DARP*) with* $(\Delta_0, \rho)$*-reasonable request sets, IGNORE employing a* $\rho$*-approximate algorithm for solving offline instances of* DARP *(*FIFO-DARP*) yields a maximal flow time of no more than* $2\Delta_0$*.*

*Proof.* Let $r$ be an arbitrary request in $R_i$, i.e., $r$ is released in $I_i$. By construction, the schedule containing $r$ is finished at the end of interval $I_{i+1}$, i.e., at most $\Delta_i + \Delta_{i+1}$ time units later than $r$ was released. Thus, for all $i > 0$ we get that

$$C^{\mathit{maxflow}}_{\mathsf{IGNORE}}(R_i) \leq \Delta_i + \Delta_{i+1}.$$

If we can show that $\Delta_i \leq \Delta_0$ for all $i > 0$ then we are done. To this end, let $f : \mathbb{R} \to \mathbb{R}$ be a $\Delta_0$, $\rho$-reasonable load bound for $R$. Then $C^{\mathit{comp}}_{\mathsf{OPT}}(R_i^{\mathit{offline}}) \leq f(\Delta_i)$ because $\Delta(R_i) \leq \Delta_i$.

By Lemma 5.4.1, we get for all $i > 0$

$$\Delta_{i+1} \leq \rho C^{\mathit{comp}}_{\mathsf{OPT}}(R_i^{\mathit{offline}}) \leq \rho f(\Delta_i) \leq \max\{\Delta_i, \Delta_0\}.$$

Because the release span of the requests served during the time period $\Delta_1$ is $0$ by definition of IGNORE, we know that $\Delta_1 \leq \max\{0, \Delta_0\} = \Delta_0$. It follows by induction on $i$ that $\Delta_i \leq \Delta_0$, and we are done. $\square$

The statement of Theorem 5.1.1 concerning the average flow time of IGNORE follows from the fact that the average is never larger then the maximum.

**Corollary 5.4.3.** *Let* $\Delta_0 > 0$ *and* $\rho \geq 1$*. For all instances of* ONLINE-DARP *(*ONLINE-FIFO-DARP*) with* $(\Delta_0, \rho)$*-reasonable request sets, algorithm IGNORE using a* $\rho$*-approximate algorithm for solving offline instances of* DARP *(*FIFO-DARP*) yields an average flow time of no more than* $2\Delta_0$*.*

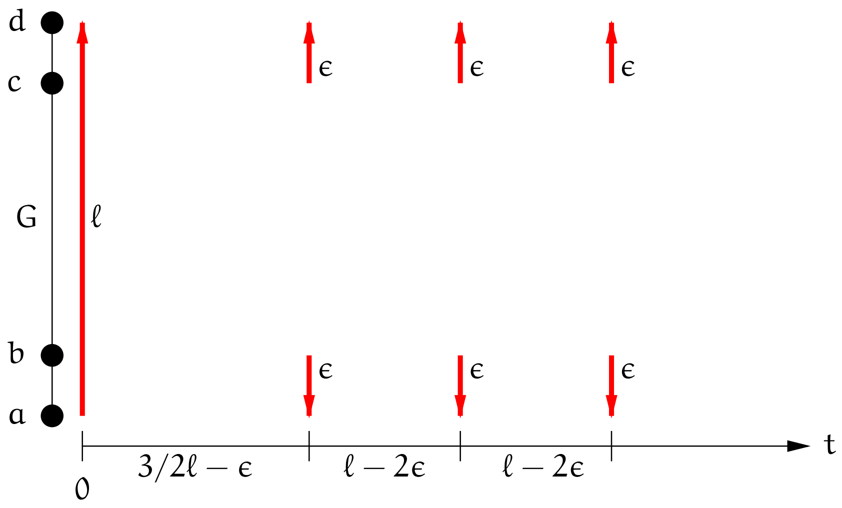

Figure 5.1: A sketch of a $(2\frac{2}{3}\ell)$-reasonable instance of ONLINE-DARP ($\ell = 18\epsilon$) (from[HKR99]).

## 5.5 A disastrous example for REPLAN

We now provide an instance of ONLINE-DARP and a $\Delta_0$-reasonable request set $R$ such that the maximal and the average flow time of the REPLAN algorithm $C^{maxflow}_{\text{REPLAN}}(R)$ is unbounded, thereby proving the remaining assertions of Theorem 5.1.1. The algorithm REPLAN is defined in 4.5.1

**Theorem 5.5.1.** *There is an instance of* ONLINE-DARP *under reasonable load such that the maximal and the average flow time of REPLAN is unbounded.*

*Proof.* In Figure 5.1 there is a sketch of an instance for the ONLINE-DARP. The graph $G$ is a path on four nodes $a, b, c, d$; the length of the path is $\ell$, the distances are $d(a,b) = d(c,d) = \epsilon$, and hence $d(b,c) = \ell - 2\epsilon$. At time $0$ a request from $a$ to $d$ is issued; at time $3/2\ell - \epsilon$, the remaining requests periodically come in pairs from $b$ to $a$ and form $c$ to $d$, resp. The time distance between them is $\ell - 2\epsilon$.

We show that for $\ell = 18\epsilon$ the request set $R$ indicated in the picture is $2\frac{2}{3}\ell$-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 $a$ resp. from $c$ to $d$. Then the release span $\Delta(R_k)$ of $R_k$ is

$$\Delta(R_k) = (k-1)(\ell - 2\epsilon).$$

The offline version ${R_k}^{offline}$ of $R_k$ can be served in time

$$C^{comp}_{\text{OPT}}({R_k}^{offline}) = 2\ell + (k-1) \cdot 4\epsilon.$$

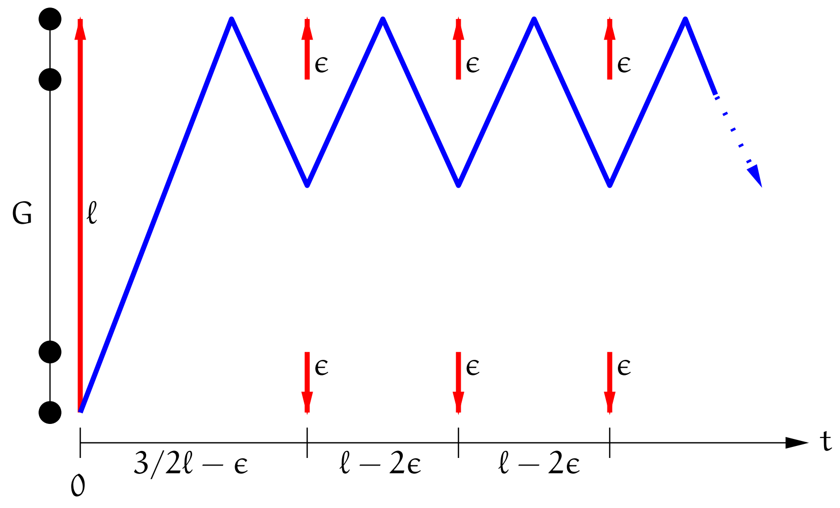

Figure 5.2: The track of the REPLAN-server. Because a new pair of requests is issued exactly when the server is still closer to the requests at the top all the requests at the bottom will be postponed in an optimal preliminary schedule. Thus, the server always returns to the top when a new pair of requests arrives (from[HKR99]).

In order to find the smallest parameter $\Delta_0$ for which the request set $R_k$ is $\Delta_0$-reasonable we solve for the integer $k-1$ and get

$$k-1 = \left\lceil \frac{2\ell}{\ell - 6\epsilon} \right\rceil = 3.$$

Hence, we can set $\Delta_0$ to

$$\Delta_0 := C_{\text{OPT}}^{comp}({R_4}^{offline}) = 2\tfrac{2}{3}\ell.$$

Now we define

$$f : \begin{cases} \mathbb{R} & \to \quad \mathbb{R}, \\ \Delta & \mapsto \quad \begin{cases} \Delta_0 & \text{for } \Delta < \Delta_0, \\ \Delta & \text{otherwise.} \end{cases} \end{cases}$$

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 additional time we need to serve it (offline), $f$ is also a load bound for $R$. Thus, $R$ is $\Delta_0$-reasonable, as desired.

Now: how does REPLAN perform on this instance? In Figure 5.2 we see the track of the server following the preliminary schedules produced by REPLAN on the request set $R$.

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

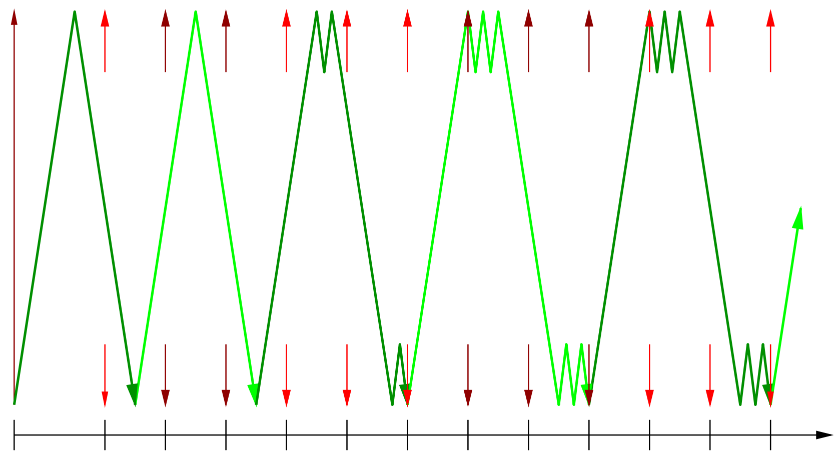

Figure 5.3: The track of the IGNORE-server (from[HKR99]).

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. $\square$

In Figure 5.3 we show the track of the server under the control 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 5.4 is sharp.

## 5.6 Reasonable load as a general framework

In this chapter we introduced the new concept of reasonable request sequences, using as example the problem ONLINE-DARP. However, the concept can be applied to any combinatorial online problem with (possibly infinte) sequences of time stamped requests, such as online scheduling, e.g., as described by Sgall [Sga98], or ONLINE-TSP, studied by Ausiello et.al. [AFL+95, AFL+94] (see also Chapter 4).

IGNORE and REPLAN represent general "online paradigms" which can be used for any online problem with time-stamped requests. We notice that the proof of the result that the average and maximal flow and waiting times of IGNORE are bounded by $2\Delta_0$ has not explicetly drawn on any specific property of ONLINE-DARP—this result holds for all combinatorial problems with time-stamped requests.

The proof that the maximal flow and waiting time of a $\Delta_0$-reasonable request sequence is unbounded for REPLAN is equally applicable to ONLINE-TSP. We expect that the same is true for any "sufficiently difficult" online problem with release times—for very simple problems, such as ONLINE-TSP on a zero dimensional space, the result trivially does not hold.

# Chapter 6

# Simulation Studies

We now discuss experimental results concerning the performance of the online algorithms studied in Chapters 4 and 5. We tested variants of the algorithms both in a simulation of single elevators with FIFO waiting queues and also in a simulation of the integrated transportation system described in Chapter 1. For our studies we used randomly generated data for various load situations and also real life data provided by our partners in industry. Preliminary simulation results have been published together with M. Grötschel, S. Krumke and J. Rambau in [GHKR99].

We will first provide some information on the simulation model and on the implemented algorithms. We will then present our results, focusing on the performance with respect to minimizing the average and the maximal flow times. Our main results are:

- Algorithms like FIRSTFIT or REPLAN, that seek for highest possible "global efficiency" (low average flow times) may leave single requests unserved for an unacceptable long period of time.
- Algorithms like FFMAXAGE or FFDYNAGE seem to require different parameter settings for different load situations. Parameters suitable for normal load have both algorithms imitate the inefficient FIFO strategy under high load.
- The algorithms IGNORE and IGGREEDY achieve a good balance between the two objectives in every load situation.
- The integrated transportation system suffers from the poor performance of the conveyor system. The optimization effects on the single elevators do not lead to a substantially higher overall performance.

## 6.1 The simulation model

Our simulation programs are built on top of AMSEL [Asc], a callable C-library to design event-based simulation programs. The input data consists of a set of *event points*, a set of *modules*, and a collection of *requests* with release times.

Every request becomes an *object* which flows through the system via the event points. For every event point, a subroutine is specified that derives a successor event from the current state of the system. If an object is on an event point then the event is stored in the global *event list* together with a time stamp, and the object stores it as its current event. The basic flow of objects is modeled as follows: the currently next event in time is read from the event list. Then the successor event is derived together with the point in time when this event should be processed. Now, the object updates its current event point to the successor event, and the successor event is entered into the event list; the old event is deleted from the list.

Modules are closed regions in the system where the number of objects inside is constrained by a capacity value. Modules are entered through entry events and left through exit events. For more details on AMSEL see [Asc95].

All our studies were produced on a Sun UltraSparc 10 workstation. The running time was approx. 2% of the simulated time. This shows that our simulation environment is fast and that the algorithms under consideration are all real-time compliant.

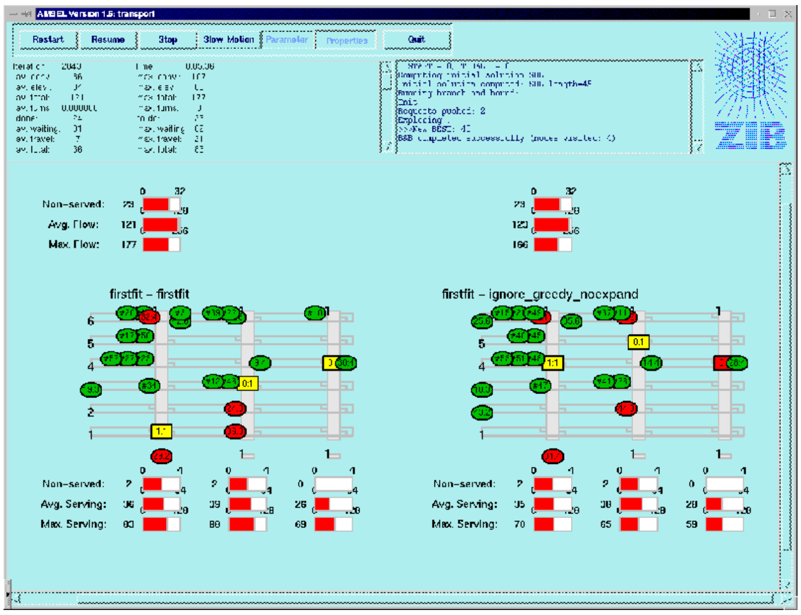

Figure 6.1: The graphical interface of the simulation program

## 6.2 The algorithms

We used the simulation environment to test the algorithms studied in Chapters 4 and 5 as well as some modifications of these algorithms. Since we are mainly concerned with average flow and waiting times, we let all algorithms run open schedules rather than closed schedules, i.e., we did not require the server to return to the origin at the end. For easier reference, we list all algorithms tested in the simulation environment together with the most relevant theoretical results from Chapters 4 and 5:

**FIRSTFIT** The server always serves a "nearest request". A request is "nearest", if the corresponding empty move, i.e., the move of the empty server to the source of the request, is shortest.

In Section 4.3, we show that this algorithm does not achieve a constant competitive ratio for minimizing the total completion time.

**FIFO** Requests are served in the order of their occurrence.

In Section 4.3, we show that this algorithm does not achieve a constant competitive ratio for minimizing the total completion time.

**REPLAN** As soon as a new request arrives, the server completes the current carrying move (if it is performing one), then the server stops and replans: it computes a new shortest schedule which starts at the current position of the server and takes care of all yet unserved requests.

In Section 4.5 we show that this algorithm is $(3 + \rho)$-competitive for the objective of minimizing the total completion time when solving instances of offline FIFO-DARP with open schedules using a $\rho$-approximate algorithm. In Section 5.5 we show for the closed schedule case that for a $\Delta_0$-reasonable request set, the maximal and average flow and waiting times for single requests are not bound.

**IGNORE** The server remains idle until the first request becomes known. It then serves the first request immediately. All requests that arrive during the service of the first request are temporarily ignored. After the first request has been served, the server 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 this way.

In Section 4.6 we show that this algorithm is $(2 + 2\rho)$-competitive for the objective of minimizing the total completion time with open schedules when solving instances of offline FIFO-DARP using a $\rho$-approximate algorithm. In Section 5.4 we show for the closed schedule case that for a $\Delta_0$-reasonable request set, the maximal and average flow and waiting times for single requests are bound by $2\Delta_0$.

**IGGREEDY** The algorithm works basically like IGNORE. However, if a new request becomes known and this request can be inserted into the current schedule without additional moves, then the new request is added to the schedule.

Notice that inserting a request can still increase the cost of the tour, because it may require additional stops of the elevator.

**FFMAXAGE** The algorithm works like FIRSTFIT, but additionally for each unserved request records its waiting time. If the waiting time for a request exceeds the grace period parameter then this request is served next.

**FFDYNAGE** This is a modification of FFMAXAGE which does not have a fixed grace period parameter, but which adjusts the parameter dynamically during the run to a fraction of the maximal waiting time of all served request so far. The value of this fraction has to be given to the algorithm as a parameter.

The current strategy for controlling the elevators in the Herlitz transport system as described in Chapter 1 is similar to our FFMAXAGE algorithm.

## 6.3 Solving offline instances

Both the algorithm IGNORE and the algorithm REPLAN have to solve offline instances of the problem PENALTY-FIFO-DARP on paths with open schedules. We studied the closed schedule case in Section 3.2. There we showed that PENALTY-FIFO-DARP and PENALTY-DARP are both NP-hard. However we also showed that both problems can be transformed into instances of FIFO-DARP and DARP on caterpillar graphs, respectively. This means that the problems with closed schedules can be tackled using the approximation algorithms for FIFO-DARP and DARP on trees, which we studied in Section 3.1.6 and Section 2.5, respectively.

In Section 3.3 we considered the open schedule case of (offline) DARP and FIFO-DARP and derived a simple strategy that yielded a $(2\rho - 1)$-approximate algorithm for open schedules, given a $\rho$-approximate algorithm for closed schedules.

For the simulation of the integrated system, the waiting areas for elevators have a capacity of one—there will never be two requests waiting for an elevator on the same level. Therefore we can employ an approximation algorithm for solving DARP on trees for this simulation, rather than an approximation algorithm for FIFO-DARP.

For solving the closed schedule problems, we used the MinSpanTree algorithm, which is studied in Section 2.5.3. This algorithm is 4/3-approximate as we showed in Section 2.5.3. This means, the open schedule algorithm which we use for both REPLAN and IGNORE is 5/3-approximate. However, during our simulations, the balancing step of algorithm MinSpanTree has yielded at most two strongly connected components—which implies that MinSpanTree has for all instances of (offline) DARP encountered in the simulations computed an optimal solution (the technique of balancing is explained in Section 2.2). This in turn implies that the all solutions for the open schedule problems have been optimal.

Moreover, we also tested the algorithm for a number of instances of DARP on caterpillars with up to 100 nodes and 300 requests. The number of connected components after balancing has always been fairly small (between one and six). Our only comparison has been the Branch and Bound algorithm described in the next paragraph. MinSpanTree has (for a maximal Branch and Bound time of 1 hour) always achieved equal or better results.

For the simulation of single elevators with FIFO queues, we allow the waiting queues to be arbitrarily large. For this simulation, REPLAN and IGNORE therefore need to solve instances of FIFO-DARP on a caterpillar. We have not yet implemented any of the heuristics for FIFO-DARP on trees, which we studied in Section 3.1.6, and can therefore give no details on their behavior in practice. Instead, in our studies we used a simple Branch and Bound algorithm for solving offline instances of PENALTY-FIFO-DARP. This algorithm uses a random order of all requests as initial solution. It then enumerates all allowed orders of subsets of requests. Whenever the length of a path defined by an ordering of a subset of requests plus the net cost of the remaining arcs is greater than the current best solution, all orderings of the requests containing this "suborder" are eliminated. We let this algorithm run for at most one second. We compared the performance of the Branch and Bound running for at most one second with Branch and Bound solved to optimality on a number of instances occurring in the simulation. In most cases the 1-second Branch and Bound achieved optimality. The largest gap observed was 10%.

## 6.4 Simulation studies of a single elevator with random data

The basic layout investigated in this section consists of one elevator of unit capacity, connecting eight floors. The distances between the floors as well as the start and stop times and the loading time of the elevator are all based on the elevators that are used in the transportation system described in Chapter 1.

Waiting requests line up in front of the elevator; the requests on one level have to be served in a First-In-First-Out fashion (FIFO). Requests are generated randomly as follows: at every point, the time until the next requests occur is taken uniformly at random on the interval $(0, t_\epsilon]$, where $t_\epsilon$ is an adjustable parameter. The number of requests issued at that time is taken uniformly at random from the integers in the interval $[1, n_{\max}]$, where $n_{\max}$ is another parameter of the simulation program. With these two parameters it is possible to control the load of the system: a smaller $t_\epsilon$ yields a higher load; a larger $n_{\max}$ leads to a stronger peakedness of the input data.

Under low load almost all algorithms yield the same performance concerning the minimization of the average flow time. For the minimization of the maximal flow time we see that FIRSTFIT and REPLAN perform considerably worse than the rest.

| Algorithm | average flow | | | maximal flow | | | completion | | |
|---|---|---|---|---|---|---|---|---|---|
| | mean | high | low | mean | high | low | mean | high | low |
| FIRSTFIT | 64 | 77 | 54 | 449 | 649 | 340 | 28879 | 28954 | 28832 |
| FIFO | 84 | 129 | 65 | 401 | 651 | 249 | 28909 | 29093 | 28815 |
| REPLAN | 65 | 79 | 56 | 439 | 695 | 293 | 28876 | 28954 | 28832 |
| IGNORE | 72 | 95 | 59 | 367 | 610 | 212 | 28885 | 28959 | 28815 |
| IGGREEDY | 70 | 91 | 58 | 356 | 557 | 211 | 28883 | 28954 | 28815 |
| FFMAXAGE | 69 | 92 | 58 | 359 | 539 | 227 | 28893 | 28998 | 28832 |
| FFDYNAGE | 65 | 79 | 55 | 346 | 512 | 227 | 28881 | 28954 | 28832 |

System Parameter: .................................. 7 systems, 1 elevator each, 8 FIFO levels.
Request Parameter: ..................................................... $t_\epsilon$: 165, and $n_{max}$: 3.
Statistics Parameter: .......................................simulated time: 8h, sample size: 20.

Table 6.1: Simulation studies for low load.

If the system works under medium load then we observe that algorithms performing especially well for the average flow time (FIRSTFIT, REPLAN) yield approx. 30% worse results for the maximal flow time than the other algorithms. Note that FIFO is still a feasible strategy with comparably poor performance for the average flow time and acceptable performance for the maximal flow time. The algorithms IGNORE, IGGREEDY, FFMAXAGE, and FFDYNAGE all show a balanced behavior in the sense that average and maximal flow times are not too much apart.

| Algorithm | average flow | | | maximal flow | | | completion | | |
|---|---|---|---|---|---|---|---|---|---|
| | mean | high | low | mean | high | low | mean | high | low |
| FIRSTFIT | 84 | 97 | 74 | 721 | 1407 | 492 | 28915 | 29229 | 28819 |
| FIFO | 224 | 409 | 144 | 754 | 1064 | 475 | 29046 | 29704 | 28819 |
| REPLAN | 88 | 103 | 76 | 736 | 1096 | 490 | 28900 | 29343 | 28583 |
| IGNORE | 107 | 135 | 89 | 528 | 766 | 355 | 28927 | 29413 | 28819 |
| IGGREEDY | 100 | 118 | 88 | 522 | 790 | 348 | 28926 | 29413 | 28819 |
| FFMAXAGE | 119 | 164 | 90 | 551 | 819 | 340 | 28936 | 29485 | 28819 |
| FFDYNAGE | 92 | 122 | 78 | 502 | 785 | 315 | 28921 | 29349 | 28819 |

System Parameter: .................................. 7 systems, 1 elevator each, 8 FIFO levels.
Request Parameter: ..................................................... $t_\epsilon$: 120, and $n_{max}$: 3.
Statistics Parameter: .......................................simulated time: 8h, sample size: 20.

Table 6.2: Simulation studies for medium load.

Under high load the FIFO strategy breaks down because it runs in such an inefficient way that requests are issued faster than FIFO can serve them. Algorithm FFMAXAGE almost breaks down because of the following: if many requests are in the system then at some point there will be many requests whose time has run out. All these requests are then scheduled first-in-first-out, which is inefficient. Thus, the number of requests whose age is too large will increase, resulting in an even worse situation. We conclude that FFMAXAGE is unstable under heavy load and under large peakedness. While the average performance on 20 instances of FFDYNAGE is acceptable, we see that there is a large deviation. Hence, we consider FFDYNAGE not very robust. For both FF-MAXAGE and FFDYNAGE we have to cope with the problem that parameters have to be adjusted to the actual system; this makes both algorithms difficult to use in praxis. Algorithm FIRSTFIT works very efficiently under high load (small average flow time). This is plausible because if many requests are in the system then the probability that FIRSTFIT can proceed without non-carrying moves is large. A similar argument explains why REPLAN also yields a small average flow time: the potential for the optimization of a schedule is the larger the more requests can be planned. Both algorithms, however, trade the flow times of individual requests for the global efficiency. In this respect, IGNORE and IGGREEDY show the most balanced behavior, where the average flow time results of IGGREEDY show that the insertion of additional requests at no extra costs pays off.

| | average flow | | | maximal flow | | | completion | | |
|---|---|---|---|---|---|---|---|---|---|
| Algorithm | mean | high | low | mean | high | low | mean | high | low |
| FIRSTFIT | 375 | 619 | 224 | 2729 | 4893 | 1521 | 29373 | 29794 | 29085 |
| FIFO | 6942 | 8203 | 5479 | 13990 | 15870 | 10849 | 42777 | 44677 | 39656 |
| REPLAN | 411 | 701 | 294 | 2611 | 5549 | 1428 | 29374 | 29906 | 29028 |
| IGNORE | 933 | 1520 | 629 | 2367 | 3730 | 1636 | 30189 | 30852 | 29550 |
| IGGREEDY | 740 | 1055 | 457 | 2097 | 3133 | 1210 | 29904 | 30834 | 29386 |
| FFMAXAGE | 3651 | 4713 | 2664 | 7538 | 9357 | 5308 | 36245 | 38163 | 34079 |
| FFDYNAGE | 3310 | 4648 | 901 | 7055 | 9289 | 3286 | 35763 | 38095 | 32057 |

System Parameter: .......................................... 7 systems, 1 elevator each, 8 FIFO levels.
Request Parameter: .......................................... $t_\epsilon$: 75, and $n_{\max}$: 3.
Statistics Parameter: .......................................... simulated time: 8h, sample size: 20.

Table 6.3: Simulation studies for high load.

We conclude that online control of elevator systems shows a trade-off between "global efficiency" (small average flow time) and "balanced behavior" (small maximal flow time). The load of the system has a substantial influence on the performance of the algorithms under consideration. Our simulation renders FIFO unusable under high load, while FIRSTFIT and REPLAN often show an unacceptable imbalance between average and maximal flow time. The performance of the algorithms FFMAXAGE and FFDYNAGE heavily depends on their parameter settings. We were not able to find parameters for them that worked equally well in all load situations.

By ignoring upcoming requests IGNORE and IGGREEDY do not ignore waiting requests for an arbitrarily long period of time, thereby showing a stable behavior with respect to the maximal flow time; the local improvement procedure of IGGREEDY enhances it with a better performance for the average flow time than IGNORE. We propose IGGREEDY for robust control of elevators under varying load.

| Elevator | Passengers |
|---|---|
| 1st elevator | 683 |
| 2nd elevator | 188 |
| 3rd elevator | 45 |
| 4th elevator | 8 |
| 5th elevator | 5 |

Table 6.4: Passenger distribution on elevators—simulation with 24h real-world traffic data

## 6.5 Simulation studies of an integrated elevator system

The layout investigated in this section integrates five elevators into a conveyor system. This is the layout of the elevator system in the distribution center of Herlitz which is described in more detail in Chapter 1. On each level, pallets are transported on circuits which have connections to single-capacitated waiting slots in front of the elevators. See Figure 1.2 in Chapter 1 for a sketch. For the following simulation studies, the simulation parameters, such as the capacities of the components and the times that pallets take for traveling between components, were adjusted to reflect the real situation at the Herlitz plant.

Additionally to controlling the elevators, the system has to decide on which elevator a pallet is to be transported. For all the results presented in this Chapter, this is done using a "first fit" strategy: pallets enter the first free waiting slot which they encounter. This obviously leads to the first elevators being used much more extensively than the further elevators as shown in Table 6.4.

We have also implemented some simple strategies that balance the load on the elevators. However all these algorithms lead to longer overall flow- and waiting times then the first-fit strategy. This is not surprising, since in this particular system vertical transportation is much faster than horizontal transportation and therefore choosing the first possible elevator proves to be efficient.

This observation together with the fact that there is at most one pallet waiting for an elevator on each floor could lead to the conclusion that the scheduling of elevators has a rather small influence on the performance of the system.

In fact, looking at a simulation using real data as shown in Table 6.5, we see that the systems behave very similar, when different algorithms are used for controlling the elevators.

A similar pattern emerges, when running the system on randomly generated data. The load used for generating Table 6.6 is chosen similar to the maximum load observed in the real data.

However, when we increase the load of the system, some trends appear which are similar to the observations for elevators without a connecting conveyor system. The following observations draw on Tables 6.7 and 6.8.

| Algorithm | average flow | maximal flow | completion |
|---|---|---|---|
| FIRSTFIT | **108** | **312** | **86363** |
| conveyor | **78** | **303** | |
| elevator | **29** | **104** | |
| FIFO | **108** | **312** | **86363** |
| conveyor | *79* | **303** | |
| elevator | **29** | **104** | |
| REPLAN | **108** | **312** | **86363** |
| conveyor | **78** | **303** | |
| elevator | **29** | *170* | |
| IGNORE | **108** | **312** | **86363** |
| conveyor | *79* | **303** | |
| elevator | **29** | **104** | |
| IGGREEDY | **108** | **312** | **86363** |
| conveyor | *79* | **303** | |
| elevator | **29** | **104** | |
| FFMAXAGE | **108** | **312** | **86363** |
| conveyor | **78** | **303** | |
| elevator | **29** | **104** | |
| FFDYNAGE | **108** | **312** | **86363** |
| conveyor | *79* | **303** | |
| elevator | **29** | **104** | |

System Parameter: ......... 7 systems, 5 elevators each, 8 levels.
Requests: ............... Real request data, simulated data: $24h$.

Table 6.5: Simulation study using real data.

| | average flow | | | maximal flow | | | completion | | |
|---|---|---|---|---|---|---|---|---|---|
| Algorithm | mean | high | low | mean | high | low | mean | high | low |
| FIRSTFIT | **120** | **123** | **119** | 342 | 425 | 266 | *28931* | **28973** | **28884** |
| conveyor | **87** | *89* | **87** | **272** | **352** | **155** | | | |
| elevator | **32** | **34** | **31** | 176 | 231 | 137 | | | |
| FIFO | *122* | *125* | *120* | 342 | **402** | 216 | **28929** | **28973** | **28884** |
| conveyor | **87** | *89* | **87** | *286* | **352** | **155** | | | |
| elevator | *33* | *36* | *32* | **137** | 209 | **105** | | | |
| REPLAN | 121 | 124 | **119** | *406* | *799* | *316* | **28929** | **28973** | **28884** |
| conveyor | **87** | **88** | **87** | 277 | **352** | *156* | | | |
| elevator | *33* | 35 | **31** | *274* | *680* | *146* | | | |
| IGNORE | 121 | **123** | **119** | **337** | **402** | **215** | **28929** | **28973** | **28884** |
| conveyor | **87** | *89* | **87** | 277 | **352** | **155** | | | |
| elevator | *33* | **34** | *32* | 141 | 241 | 112 | | | |
| IGGREEDY | 121 | **123** | **119** | **337** | **402** | **215** | **28929** | **28973** | **28884** |
| conveyor | **87** | *89* | **87** | 277 | **352** | **155** | | | |
| elevator | *33* | **34** | **31** | 142 | 241 | 112 | | | |
| FFMAXAGE | **120** | **123** | **119** | 340 | 425 | 266 | *28931* | **28973** | **28884** |
| conveyor | **87** | *89* | **87** | **272** | **352** | **155** | | | |
| elevator | **32** | **34** | **31** | 163 | **198** | 137 | | | |
| FFDYNAGE | **120** | **123** | **119** | 340 | 425 | 255 | *28931* | **28973** | **28884** |
| conveyor | **87** | *89* | **87** | **272** | **352** | **155** | | | |
| elevator | **32** | **34** | **31** | 170 | 289 | 132 | | | |

System Parameter: ..................................... 7 systems, 5 elevators each, 8 levels.
Request Parameter: ..................................................... $t_\epsilon$: 90, and $n_{max}$: 3.
Statistics Parameter: .................................... simulated time: 8h, sample size: 20.

Table 6.6: Simulation study with low random load.

| | average flow | | | maximal flow | | | completion | | |
|---|---|---|---|---|---|---|---|---|---|
| Algorithm | mean | high | low | mean | high | low | mean | high | low |
| FIRSTFIT | **160** | 168 | **157** | 638 | 985 | 549 | **29000** | 29112 | 28921 |
| conveyor | **103** | 107 | **101** | 526 | 719 | **369** | | | |
| elevator | **56** | **60** | **54** | 461 | 534 | 363 | | | |
| FIFO | *210* | *220* | *202* | *819* | *1089* | *680* | *29054* | *29167* | *28933* |
| conveyor | *120* | *125* | *116* | *706* | *975* | *565* | | | |
| elevator | *89* | *95* | *85* | 302 | 342 | 270 | | | |
| REPLAN | 163 | **167** | 160 | 645 | 945 | 558 | 29006 | 29112 | 28912 |
| conveyor | **103** | **106** | 102 | **507** | 731 | **369** | | | |
| elevator | 59 | 62 | 56 | *492* | *874* | *417* | | | |
| IGNORE | 171 | 179 | 167 | 626 | 763 | 556 | 29028 | 29120 | 28913 |
| conveyor | 107 | 111 | 105 | 551 | 731 | 499 | | | |
| elevator | 64 | 67 | 61 | 274 | 319 | 238 | | | |
| IGGREEDY | 169 | 176 | 165 | **603** | 803 | **500** | 29008 | **29077** | **28908** |
| conveyor | 106 | 110 | 104 | 523 | 766 | 376 | | | |
| elevator | 63 | 66 | 60 | 276 | 353 | 247 | | | |
| FFMAXAGE | 168 | 175 | 162 | 638 | 846 | 516 | 29009 | 29112 | 28917 |
| conveyor | 105 | 109 | 104 | 547 | 720 | **369** | | | |
| elevator | 62 | 65 | 57 | **262** | **314** | **224** | | | |
| FFDYNAGE | 162 | **167** | 159 | 612 | **713** | 554 | 29005 | 29108 | 28917 |
| conveyor | 104 | 107 | 102 | 536 | **586** | **369** | | | |
| elevator | 58 | 62 | 56 | 278 | 318 | 246 | | | |

System Parameter: .................................... 7 systems, 5 elevators each, 8 levels.
Request Parameter: .................................................... $t_\epsilon$: 30, and $n_{\max}$: 3.
Statistics Parameter: .................................... simulated time: 8h, sample size: 20.

Table 6.7: Simulation study with medium random load.

The algorithms FIRSTFIT and REPLAN once again show the best average flow behavior. We also notice again that these algorithms are performing badly when considering maximal flow times through the elevator subsystem, i.e., the maximal time it takes a pallet from entering the elevator waiting slot until emerging from the elevator on its target floor. However, in the special settings of the Herlitz system the conveyor belt levels off most of these effects, so that the overall maximal flow times of FIRSTFIT and REPLAN are not much worse than those achieved by other algorithms.

Once again FIFO performs initially quite well, however with rising load both its average and maximum flow times deteriorate compared to other algorithms.

The performance of the modified first fit algorithms FFMAXAGE and FFDYNAGE is for all samples run in the original Herlitz configuration similar to that of IGGREEDY. However Table 6.9 displays the results of a simulation under high load, when increasing the number of floors from eight to twenty. Here we again notice for the elevator systems a deterioration of the performance of both FFMAXAGE and FFDYNAGE, even though there is still at most one request waiting on each floor. This suggests that even though FFMAXAGE and FFDYNAGE are stable algorithms for small systems under low load, their reliance on parameters does not allow them to "scale" well when using them for larger systems and higher load situations.

When comparing the elevator results with the overall system results, it

| Algorithm | average flow | | | maximal flow | | | completion | | |
|---|---|---|---|---|---|---|---|---|---|
| | mean | high | low | mean | high | low | mean | high | low |
| FIRSTFIT | **174** | **180** | **169** | 764 | **919** | 647 | **29041** | 29172 | 28943 |
| conveyor | **110** | **114** | **107** | 634 | 822 | 547 | | | |
| elevator | **62** | **66** | **61** | *561* | *720* | *419* | | | |
| FIFO | *297* | *356* | *256* | *1934* | *3276* | *1157* | *29205* | *29434* | *29087* |
| conveyor | *174* | *220* | *145* | *1778* | *3079* | *993* | | | |
| elevator | *122* | *141* | *111* | 321 | 348 | 294 | | | |
| REPLAN | 179 | 184 | 173 | **753** | 1072 | **569** | 29042 | 29147 | 28958 |
| conveyor | 112 | 115 | 109 | **606** | **797** | **513** | | | |
| elevator | 66 | 70 | 63 | 535 | 682 | 407 | | | |
| IGNORE | 191 | 199 | 185 | 792 | 1082 | 645 | 29056 | 29148 | 28962 |
| conveyor | 117 | 122 | 114 | 711 | 1024 | 577 | | | |
| elevator | 73 | 77 | 71 | **285** | **310** | 264 | | | |
| IGGREEDY | 189 | 194 | 183 | 786 | 1045 | 627 | 29051 | 29172 | **28941** |
| conveyor | 116 | 119 | 113 | 704 | 975 | 553 | | | |
| elevator | 72 | 75 | 70 | 299 | 350 | 262 | | | |
| FFMAXAGE | 192 | 200 | 184 | 858 | 1139 | 677 | 29058 | **29132** | 28983 |
| conveyor | 118 | 122 | 114 | 739 | 1023 | 559 | | | |
| elevator | 73 | 77 | 70 | **285** | 319 | **245** | | | |
| FFDYNAGE | 177 | 183 | 173 | 756 | 1058 | 611 | 29052 | 29157 | 28949 |
| conveyor | 112 | 115 | 108 | 664 | 1011 | 536 | | | |
| elevator | 65 | 69 | 63 | 297 | 358 | 275 | | | |

System Parameter: .................................... 7 systems, 5 elevators each, 8 levels.
Request Parameter: .................................................... $t_\epsilon$: 25, and $n_{max}$: 3.
Statistics Parameter: .................................... simulated time: 8h, sample size: 20.

Table 6.8: Simulation study with high random load.

| Algorithm | average flow | | | maximal flow | | | completion | | |
|---|---|---|---|---|---|---|---|---|---|
| | mean | high | low | mean | high | low | mean | high | low |
| FIRSTFIT | **385** | **417** | **361** | 2895 | 4628 | 2034 | 29401 | **29525** | **29120** |
| conveyor | **123** | **137** | **117** | 1201 | 2280 | 779 | | | |
| elevator | **261** | **280** | **243** | *2777* | *4343* | 1963 | | | |
| FIFO | *4968* | *5775* | *3913* | *30098* | *33768* | *25034* | *38595* | *39988* | *36302* |
| conveyor | *3926* | *4698* | *2888* | *29602* | *33713* | *24788* | | | |
| elevator | *1041* | *1076* | *1002* | 1558 | 1634 | 1469 | | | |
| REPLAN | 418 | 475 | 390 | 2876 | 3820 | 2049 | **29397** | 29549 | 29174 |
| conveyor | 128 | 145 | 121 | **1061** | **1683** | **713** | | | |
| elevator | 289 | 330 | 268 | 2680 | 3749 | *1970* | | | |
| IGNORE | 479 | 544 | 436 | 1901 | 2586 | 1297 | 29460 | 29709 | 29198 |
| conveyor | 144 | 169 | 132 | 1458 | 2111 | 797 | | | |
| elevator | 334 | 375 | 303 | 1192 | 1357 | 999 | | | |
| IGGREEDY | 458 | 517 | 421 | **1726** | **2558** | **1295** | 29468 | 29687 | 29196 |
| conveyor | 138 | 158 | 128 | 1294 | 2287 | 779 | | | |
| elevator | 319 | 359 | 293 | 1197 | 1398 | 1089 | | | |
| FFMAXAGE | 1239 | 1720 | 849 | 11113 | 19491 | 4016 | 30979 | 31921 | 30027 |
| conveyor | 597 | 1006 | 301 | 10644 | 18906 | 3757 | | | |
| elevator | 641 | 717 | 548 | 1277 | 1411 | 1141 | | | |
| FFDYNAGE | 512 | 738 | 440 | 2825 | 6370 | 1687 | 29543 | 29937 | 29130 |
| conveyor | 161 | 279 | 135 | 2254 | 5733 | 1025 | | | |
| elevator | 350 | 459 | 305 | **1095** | **1223** | **993** | | | |

System Parameter: ...................................... 7 systems, 5 elevators each, 20 levels.
Request Parameter: ........................................................ $t_\epsilon$: 35, and $n_{max}$: 3.
Statistics Parameter: ...................................... simulated time: 8h, sample size: 20.

Table 6.9: Simulation study with 20 floors under high random load.

seems that the conveyor belts in fact harm the performance of the overall system. Since pallets move constantly on the belt, advance planning becomes impossible, or at least very difficult. A pallet can only enter a waiting slot, when it is empty at the exact point in time when the pallet passes by. Once a pallet passes the last elevator, it has to move back to the first elevator until it gets the chance to find an empty waiting slot. This takes a comparatively very long time.

A further problem of the circular conveyor belts is that they can get blocked, as explained in Chapter 1: A pallet can only move forward when the next segment of the conveyor is empty. When there is a pallet standing on each segment, then one of the pallets has to be removed (by five workers!) before anything can move again. Herlitz currently uses a "defensive" strategy for running the elevator system: Only a limited number of pallets is allowed into the system, which therefore works always under low load. In fact, in our simulation studies under medium and high load, the conveyor belts did get regularly blocked (for high load, approximately every 200 hours). The results in the tables of this chapter were all generated by simulation runs which terminated without blocking of the conveyor belts.

It seems to us, that longer waiting queues in front of the elevators would lead to a much better performance of the system. However, then one would also have to employ more sophisticated algorithms for assigning pallets to elevators (when simply using the first-fit strategy, as for single capacity waiting slots, simulation studies show that the system performance deteriorates—which is hardly surprising). Studying how the system behaves when certain features of the (logical) layout are changed should be an interesting topic for future investigations.

# Conclusions and Outlook

In this thesis we studied an elevator system in an industrial facility, which needs to be controlled *online* and in *real-time*: The elevators have to move pallets between floors without knowledge about the transportation requests that appear in the future. Schedules need to be computed in seconds, so that the running time of the algorithm does not impede the material flow. Good algorithms should perform well with respect to two (conflicting) objectives: They should achieve low average flow times and should also achieve low maximal flow times of pallets.

We began by studying the offline case, where all requests are known and available in advance. We concentrated on the problem DARP, where the objective is to minimize the completion time when running a closed schedule. When considering open schedules, where the server does not have to return to the origin, this corresponds to an instance of ONLINE-DARP with the objective of minimizing the maximal completion time where all requests are known and available from the beginning. We proved a new complexity result, showing that DARP on caterpillar graphs is NP-hard. We also described existing combinatorial algorithms for this problem. We then looked at new extensions of DARP. For the case of FIFO-DARP, where there are FIFO precedence constraints on requests emanating from the same source, we developed new approximation algorithms. Further, we could show that the problem PREC-DARP, which includes penalties for starting and stopping, can be transformed into DARP on a slightly larger graph. We finally considered how closed schedule algorithms for DARP and FIFO-DARP can be used to derive open schedule solutions.

We believe the following topics in this context should prove interesting for future research:

- Find lower bounds for polynomial time approximation algorithms for DARP and FIFO-DARP on trees and general graphs.
- Develop better approximation algorithms for open schedule DARP and FIFO-DARP.
- Consider other objective functions than completion time—namely, the weighted sum of the completion times of jobs. This covers the offline problem resulting from an instance of ONLINE-DARP with the objective

of minimizing average flow times, where all requests are available and known in advance.

- There are many polyhedral results for the Asymmetrical Hamiltonian Path (and Asymmetrical Traveling Salesman) problems, which are similar to DARP with open (and closed) schedules (see, e.g., [Asc95]). Study FIFO-DARP and DARP from a polyhedral perspective.

We then studied the online case of DARP and FIFO-DARP, looking at both closed and open schedules. Our main interest lies in the algorithms REPLAN and IGNORE, which are both competitive. These algorithms need to solve offline instances, for which the offline algorithms developed in the first part of this thesis can be employed. We also showed that there are no competitive algorithms for the objectives of minimizing average or maximal flow times.

To get theoretical results for distinguishing REPLAN and IGNORE in a practically relevant manner, we conceived a new framework for studying online algorithms in continuously operating systems. The idea is, to exclude request sequences which cannot be served by any algorithm without an ever increasing number of requests left waiting. In this framework, we showed that the maximal and average flow times of requests are bounded when using IGNORE and that they are not bounded when using REPLAN. The new concept, which we call "reasonable load", can be employed to study online algorithms with time-stamped requests.

The following interesting theoretical questions remain open:

- Close the gap between the lower bounds and the best competitive ratios proved for ONLINE-DARP and ONLINE-FIFO-DARP.
- Study the case, when the elevator has a greater capacity than one, and also the case, when there is more than one elevator serving the same requests.
- Further refine the concept of "reasonable load" and apply it to other problems and algorithms.

Finally, we tested various online algorithms in simulations of single elevators and also of the integrated elevator system that motivates this thesis. The results clearly suggest that there is a trade-off between minimizing average and maximal flow times. The results for REPLAN and IGNORE seem to confirm the theoretical results for these algorithms under reasonable load: IGNORE shows a more balanced behavior with much better maximal flow times. However, REPLAN achieves better average flow times. The most appropriate algorithm for controlling elevators under varying load while balancing the two objectives of minimizing average and maximal flow times, seems to be IGGREEDY, which is an extension of IGNORE with a 1-OPT like heuristic.

The simulation studies of the integrated elevator system suggest, that the layout of this system is inappropriate. The circular conveyor belt connected to unit capacity waiting slots seems to seriously harm efficient processing of requests.

For future research, we suggest studying different layouts, e.g., with longer waiting queues in front of the elevators, more then one elevator serving a waiting queue and also higher capacitated elevators. This might lead to general design principles for transportation systems.

# Appendix A

# Basic Notation

## A.1 Graphs

In this section we introduce the graph theoretical notation used in this thesis. In particular, we explain the definitions and assumptions concerning the multigraphs studied in Chapter 2, which are not entirely canonical. For a more detailed treatment of graph theory, we refer to, e.g., Bondy and Murty [BM76].

A *multiset* $A$ is a finite sequence of elements $A = (a_1, a_2, a_3, \ldots, a_k)$ that may contain repeated elements. The order of the elements is irrelevant, but their multiplicities are part of the structure. The *size of a multiset* is the number of elements, counting multiplicities. A *submultiset* $A' \subseteq A$ is a multiset where each element has a lower or equal multiplicity than in $A$.

A *graph* $G = (V, E)$ consists of a set $V$ of vertices and a set $E$ of edges. A graph is *simple*, if there is at most one edge between a pair of vertices and if there is no edge connecting a vertex with itself.

A *digraph* $G = (V, A)$ consists of a set $V$ of vertices and a set $A$ of arcs. A digraph is simple, if there is at most one arc from $u$ to $v$ for every pair of vertices and if there is no arcs connecting a vertex with itself.

A multigraph $G = (V, E)$ consists of a set $V$ of vertices together with a multiset of arcs. Similarly a multidigraph $G = (V, A)$ consists of a set of vertices together with a multiset of arcs.

A *mixed graph* $G = (V, E, A)$ consists of a set $V$ of vertices, a set (or multiset) $E$ of undirected edges, and a set (or multiset) $A$ of directed arcs. An edge with endpoints $u$ and $v$ will be denoted by $[u, v]$, an arc from $u$ to $v$ by $(u, v)$. In Chapter 2 we will study graph augmentation problems in mixed graphs with a *set* of edges and a *multiset* of arcs.

For (di)graphs and multi(di)graphs, we denote by $n := |V|$, $m_E := |E|$ and $m_A := |A|$ the number of vertices, edges and arcs, respectively. For a vertex $v \in V$ we let $A_v$ be the set of arcs in $A$ emanating from $v$.

If $X \subseteq E \cup A$, then 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 endpoints.

The *out-degree* of a vertex $v$ in $G$, denoted by $d^+_G(v)$, equals the number of arcs in $G$ leaving $v$. Similarly, the *in-degree* $d^-_G(v)$ is defined to be the number of arcs entering $v$. If $X \subseteq A$ we briefly write $d^+_X(v)$ and $d^-_X(v)$ instead of $d^+_{G[X]}(v)$ and $d^-_{G[X]}(v)$. A graph $G$ is called *degree balanced* if $d^+_G(v) = d^-_G(v)$ for all vertices $v \in V$.

A *directed spanning tree rooted towards* $o \in V$ is a subgraph $D = (V, Y)$ of a directed (multi)graph $H = (V, R)$ which is a tree and which for each $v \in V$ contains a directed path from $v$ to $o$.

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 *leafs* of the caterpillar. The edges between vertices on the path and leafs are called *hairs*. Notice that caterpillars have maximum degree three.

A *walk* in a (multi)graph $(v_0, e_1, v_2, e_2, \dots, e_k, v_k)$ is a sequence of edges and vertices such that each edge $e_i, 1 \leq i \leq k$ is incident to both vertices $v_{i-1}$ and $v_i$. For a *closed walk* in a (multi)graph, we additionally require the first and the last vertex to be identical.

A *walk* in a (multi)digraph is a sequence of arcs $a_1, \dots, a_k$, such that for any two arcs $a_i = (u_i, v_i)$ and $a_{i+1} = (u_{i+1}, v_{i+1})$ with $0 \leq i \leq k-1$ we have that $v_i = u_{i+1}$. For a *closed walk* in a (multi)digraph, we additionally require the last target vertex of the last arc to be identical to the source vertex of the first arc.

A *cycle* is a closed walk containing no two edges (or arcs) twice. A *tour* is a cycle containing for each vertex an edge (or arc) incident to that vertex.

## A.2 Approximation algorithms

Since most of the problems studied in this thesis are NP-hard, we are interested in approximation algorithms for them.

Let $\Pi$ be a minimization problem. A polynomial-time algorithm $A$ is said to be a $\rho$*-approximation algorithm* for $\Pi$, if for every problem instance $I$ of $\Pi$ with optimal solution value $\mathrm{OPT}(I)$ the solution of value $A(I)$ returned by the algorithm satisfies $A(I) \leq \rho \cdot \mathrm{OPT}(I)$.

Notice that other authors often define a $\sigma$-approximate algorithm independent of whether the problem $\Pi$ is a minimization or a maximization problem: A polynomial-time algorithm for an optimization problem $\Pi$ is then said to be a $\sigma$-approximate algorithm, if for every problem instance $I$ of $\Pi$ with optimal solution value $\mathrm{OPT}(I)$ the solution of value $A(I)$ returned by the algorithm satisfies $\frac{|A(I)-OPT(I)|}{OPT(I)} \leq \rho$. For a minimization problem, a $\sigma$-approximation algorithm according to the last definition is then a $\rho$-approximation algorithm with $\rho = 1 + \sigma$ according to the definition of approximation algorithms used in this thesis.

# Appendix B

# Feuerstein and Stougie's Classification of DARP

## B.1 Classifying DARP

An instance of the Dial-a-ride problem DARP consists of a set of servers which have to transport objects between sources and destinations in some metric space or graph. As in scheduling, there are many variants of this problem. They differ, for example, in the number of servers, the capacity of the servers, or the nature of the space in which the servers move. Recently, a classification of DARP has been proposed by Feuerstein and Stougie. It was first described by de Paepe [dP98] and is based on the classification of scheduling problems by Lawler et.al. [LLKS93].

Using this notation, a variant of DARP is given as a tuple of four fields with a total of eight entries:

$$\beta_1, \beta_2 | \beta_3, \beta_4, \beta_5, \beta_6 | \beta_7 | \beta_8$$

- The first field describes the servers. The entry $\beta_1$ indicates the type and number of servers, $\beta_2$ specifies the capacity of the servers.
- The second field contains information about the transportation requests. Constraints on their sources and destinations are given in $\beta_3$. The entry $\beta_4$ indicates whether preemption is allowed. Possible release times and deadlines for serving the requests are specified in $\beta_5$. Finally $\beta_6$ indicates whether there are precedence constraints on the requests.
- The third field, consisting of $\beta_7$, specifies the metric space or graph in which the servers move.
- Finally $\beta_8$ contains the objective function of the problem.

## B.2 The servers

The servers are described by the two entries $\beta_1$ and $\beta_2$. The entry $\beta_1$ specifies the type of the servers. As for scheduling, the notation allows three different values for this entry:

| | |
|---|---|
| $P$ | identical unit speed servers working in parallel; |
| $Q$ | uniform parallel servers, each traveling at its own fixed speed; |
| $R$ | unrelated parallel servers; |
| | each server travels at a speed that depends on the rides it is serving. |

If the number of servers $k$ is given, we write $Pk$, $Qk$ or $Rk$ respectively. If there is only one server, the entry receives *1* as its value .

The second entry $\beta_2$ specifies the capacity of the servers. The capacity of a server is given by the number of requests it can serve simultaneously. If this entry is left blank, each server has its own specific capacity. Otherwise the following constraints on the capacities may be given:

| | |
|---|---|
| *capc* | all servers have identical capacity $c$; |
| *cap1* | all servers have unit capacity; |
| *cap*$\infty$ | all servers have infinite capacity, |
| | each server can handle all requests simultaneously. |

The notation as described by de Paepe does not allow to convey information about the origins of the servers. The servers can either start at one common origin or from distinct origins. The origins can either be given as part of the problem instance or they can be chosen arbitrarily by the algorithm. In the context of this thesis we will assume that servers may start from distinct vertices, which are given as part of the problem instance.

## B.3 The requests

The four entries $\beta_3 \ldots \beta_6$ contain information concerning the transportation requests. Each request $j$ has a source $s_j$ and a destination $t_j$. Constraints on the sources and destinations are contained in $\beta_3$:

| | |
|---|---|
| $S$ | all requests have a common source |
| $T$ | all requests have a common destination |
| $s = t$ | source and destination are identical for each request |

By default, sources and destination are not constrained.

The entry $\beta_4$ specifies whether preemption is allowed when serving requests. By default, preemption is not allowed. That means that a transportation request has to be served without interruption (once a server has started serving a request, this request will decrease the free capacity of the server until the server reaches the destination of the request). When preemption is allowed, $\beta_4$ contains the value *pmtn*. In this case a server can transport a request from its source to any other point. There the request may be picked up later, possibly by a different server, which in turn delivers it to some other place until it finally reaches its destination.

By default, requests have neither release times nor deadlines. The release time $r_j$ of a request j is the time when it can first be served. A deadline $d_j$ of a request j is the latest time when it may be delivered to its destination. If release times or deadlines are given, $\beta_5$ contains the value $r_j$ or $d_j$ respectively.

Finally the entry $\beta_6$ specifies whether there are precedence constraints on serving the requests. By default there are no precedence constraints. To indicate that there are (general) precedence constraints present, $\beta_6$ receives value *prec*.

## B.4 The space

The entry $\beta_7$ indicates on what sort of metric space or graph the DARP is defined. De Paepe allows the following entries:

| | |
|---|---|
| *G* | undirected (finite) graph with positive edge weights |
| *line* | (finite) path with positive edge weights |
| *tree* | (finite) tree with positive edge weights |
| $\mathbb{R}^n$ | the $\mathbb{R}^n$ with Euclidean metric |

## B.5 The objective function

To define objective functions, we first have to take a a closer look at the constituents of the time that a request spends in the transportation system: The *completion time* $C_j$ of a request j is the point in time when the request has been processed, i.e., when it arrives at its destination. The *flow time* $F_j$ of a job is the time that a request spends in the transportation system. This is equal to the difference between the job's completion time and its release time $F_j = C_j - R_j$. If we are dealing with a DARP without release times, flow times and completion times are equal. The flow time consists of two components, one being the *service time* which is the time that the server dealing with the request takes for serving it. The remainder is the *waiting time* $W_j$ of a request. Notice that the service time of a request can be different from the minimal service time, when there are multiple servers with different speeds. The minimal service time is the time for serving the request using the fastest server.

De Paepe suggested three possible objective functions:

| | |
|---|---|
| $C_{max}$ | the maximum of the completion times, also called *makespan* |
| $\sum C_j$ | the sum of the completion times, also called *latency*, a measure for the average completion time |
| $\sum w_j C_j$ | the sum of the completion times, weighted according to their priority |

Further objective functions result from using flow times or waiting times instead of completion times.

The classification as described by de Paepe does not specify whether servers have to return to their origin. If it is required that the servers return to their origin after serving all requests, the problem is called a *closed schedule problem* according to [AKR98b]. Otherwise we deal with an *open schedule problem*.

For all of the above objective functions, the closed schedule and open schedule optimum will have the same value. However there are objective functions where this is not the case. An example for an objective function which leads to different solutions in the closed and open schedule case is to minimize the total cost of server movements. We will use this as an objective for closed schedule problems and denote it by $\sum m$.

## B.6 Examples

All variants of DARP studied in this thesis lie in the subclass described by $P, \mathit{capc}|r_j|G|$. $P$ specifies that the problem has some identical unit speed servers. The next entry *capc* means that each of these servers has capacity $c$. The token $r_j$ means that the requests have release times. The entry $G$ tells us, that the metric space is a general (edge weighted) graph.

The *traveling salesman problem* of finding a shortest tour through a graph $G$ traversing all vertices can also be expressed as a DARP, given by $1|\mathit{cap1}|s = t|G|\sum m$: One unit capacity server has to serve requests where the destination of each request is equal to its source (additionally we require that the requests occur on all vertices). The server moves on an edge weighted graph. The objective is to minimize the cost of the server moves.

The traveling salesman problem is a good example that there may be multiple representations for the same problem. The following also represents the same TSP: $1, \mathit{cap}\infty|S|G|\sum m$. Here a server with infinite capacity has to serve requests which have a common source.

# Appendix C

# Online Algorithms and Competitive Analysis

In this appendix we introduce the notion of online algorithms, which are algorithms that operate under uncertainty about the future. We will then discuss how the performance of such algorithms can be determined using competitive analysis. For a survey of the current state of research on online algorithms we refer to the recent book edited by Fiat and Woeginger [FW98]. Another book on the subject has been published by Borodin and El-Yaniv [BEY98]. A number of articles have been published that introduce online algorithms and competitive analysis, for example [Alb96], [Alb97], [IK97].

## C.1 Online algorithms

An *online algorithm* in the strict sense can be characterized as an algorithm responding to events over time without prior knowledge of these events.

Online problems in this strict sense exhibit therefore the following properties:

1. One or more servers,
2. Operating in a known environment,
3. Dealing with a sequence of requests arriving over time.

Many strict online problems can be reduced to the following model: Requests arrive as a sequence $\sigma = \sigma_1, \ldots, \sigma_n$. Once a request has been dealt with, the next request becomes known to the algorithm. This model has the advantage of encapsulating the notion of time within the event sequence, time does not appear explicitly in the analysis.

The online algorithms studied in Chapters 4 and 5 of this thesis are also online algorithms in the strict sense. However, we deal with time-stamped requests, that become known at their release time and can be served at any later time and in an order not necerssarily dependent on their release times. In this case, therefore, time has to be considered explicitly.

In a wider sense, an *online algorithm* is an algorithm that is suitable for competitive analysis. This applies to algorithms which operate with input data that becomes known incrementally as the algorithm proceeds. The difference to a strict online algorithm is, that the information revealed to the algorithm depends not only on time but also on the actions of the algorithm.

Fiat and Woeginger [FW98] describe problems for which such algorithms are appropriate as systems that possess the following properties:

1. Some notion of time progression,
2. A memory state,
3. An environment,
4. Respond in some way to changes in the environment.

Examples for online problems in the wider sense are navigation or exploration problems, where information is gained through actions of the algorithm rather than the passage of time. Distributed algorithms also fall in this category since each component has only partial knowledge of the complete system with this knowledge depending on its actions.

Some authors speak of *online optimization* if the aim of the algorithm is to maximize or minimize a given objective function. In case of maximization, we speak of *benefit problems*, when the objective is to minimize a function we are dealing with a *cost problem*.

## C.2 Competitive analysis

Competitive analysis addresses the question of how to evaluate the performance of online algorithms. Sleator and Tarjan suggested to compare the performance of the online algorithm with the performance of its offline counterpart, i.e., the optimal solution given complete knowledge of the problem instance.

An online algorithm is called $k$**-competitive**, if for all problem instances the solution is only off by factor $k$. More formally, given a cost problem $P$ with problem instances $I \in P$, an algorithm $A$ is $k$-competitive, if and only if

$$C_{OPT}(I) \leq k \cdot C_A(I) + c \qquad \forall\, I \in P \tag{C.1}$$

where $C_{OPT}$ denotes the cost of an optimal (offline) solution to the problem instance I, $C_A$ is the cost of the solution delivered by A and c is a constant that does not depend on the problem instance I. Similarly competitiveness can be defined for benefit problems.

If the constant c equals to zero in the above definition, the algorithm is said to be **strictly** $k$**-competitive**

The **competitive ratio** of the algorithm is defined as

$$\inf\{k : \text{ A is k-competitive }\} \tag{C.2}$$

Competitive analysis also illustrates the connections between online algorithms and approximation algorithms: By definition, any c-competitive algorithm is also c-approximate, using the definition of approximate algorithms from Appendix A.2.

Competitive analysis can be interpreted as a game between two players, where one player, the adversary, invents a sequence of requests, which the other player has to serve. The two players have opposite objectives in the sense that the adversary tries to make it as difficult as possible for the serving player. In deterministic competitive analysis, the adversary is often called **offline adversary**, since he posses complete knowledge about the (deterministic) reactions of the online algorithms.

## C.3 Beyond competitive analysis

Competitive Analysis is however not the answer to all ills. Since it is a worst case measure, competitiveness is overly pessimistic—competitive algorithms regularly perform much better in practice than their competitive ratio would suggest.

Many authors have questioned the relevance of competitive ratios, both as an absolute measure for the effectiveness of an algorithm and for comparing different algorithms. Authors who stress the importance of timeliness in connection with online algorithms also criticize that competitive analysis takes no account of the time complexity of the algorithms [WZ98].

One possible way to improve the bounds of the online algorithm is to randomize it, so that the adversary has restricted information on the algorithm's behavior. Competitive analysis of randomized online algorithm has first been studied by Ben-David et.al.[BDBK+94].

A number of different approaches have also been suggested to refine competitive analysis both for deterministic and randomized algorithms with the aim of yielding more meaningful results, for example:

- Restricted classes of input: The possible input is restricted by some constraints [BIRS95]

- Statistical adversary: The input has to fulfill certain statistical properties [Rag91]
- Diffuse adversary: The input has to be generated AC-coding to a probability distribution D belonging to a class of "allowed" distributions $\Delta$ [KP94]

In this thesis we introduce a new concept for studying online algorithms with time-stamped requests, which falls into the category of "restricted classes of input". For details on this concept of reasonable request sequences see Chapter 5.

# Bibliography

[AFL$^+$94] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo. Serving request with on-line routing. In *Proceedings of the 4th Scandinavian Workshop on Algorithm Theory (SWAT'94)*, volume 824 of *Lecture Notes in Computer Science*, pages 37–48, July 1994.

[AFL$^+$95] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo. Competitive algorithms for the traveling salesman. In *Proceedings of the 4th Workshop on Algorithms and Data Structures (WADS'95)*, volume 955 of *Lecture Notes in Computer Science*, pages 206–217, August 1995.

[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(5):849–869, October 1988.

[AKR98a] N. Ascheuer, S. O. Krumke, and J. Rambau. Competitive scheduling of elevators. Preprint, Konrad-Zuse-Zentrum für Informationstechnik Berlin, November 1998. To appear.

[AKR98b] N. Ascheuer, S. O. Krumke, and J. Rambau. The online transportation problem. Manuscript, submitted for publication, 1998.

[Alb96] S. Albers. Online algorithms. Lecture Series LS-96-2, BRICS, 1996.

[Alb97] S. Albers. Competitive online algorithms. *OPTIMA*, 54:2–8, June 1997.

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

[Asc] N. Ascheuer. *Amsel—a modelling and simulation environment library*. Developed at Konrad-Zuse-Zentrum für Informationstechnik, Berlin. Online-Documentation at `http://www.zib.de/ascheuer/AMSEL.html`.

[Asc95] N. Ascheuer. *Hamiltonian path problems in the on-line optimization of flexible manufacturing systems*. PhD thesis, Technische Universität Berlin, 1995.

[BDBK$^{+}$94] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson. On the power of randomization in online algorithms. *Algorithmica*, 11:2–14, 1994.

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

[BIRS95] A. Borodin, S. Irani, P. Raghavan, and B. Schieber. Competitive paging with locality of reference. *Journal of Computer Systems Science*, 50:244–258, 1995.

[BM76] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. American Elsevier, New York and Macmillan, London, 1976.

[BR92] P. Berman and V. Rahmaiyer. Improved approximations for the steiner tree problem. In *Proceedings, 3rd ACM-SIAM Symposium on Discrete Algorithms*, pages 325–334, Orlando, FL, 1992.

[Chr76] N. Christofides. Worst case analysis of a new heuristic for the travelling salesman problem. Technical report, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburg, PA, 1976.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. MIT Press, 1990.

[dP98] W. E. de Paepe. Computer-aided complexity classification of dial-a-ride problems. Master's thesis, University of Amsterdam, 1998.

[EJ73] J. Edmonds and E. L. Johnson. Matching, euler tours and the chinese postman. *Mathematical Programming*, 5:88–124, 1973.

[FG93] G. N. Frederickson and D. J. Guan. Nonpreemptive ensemble motion planning on a tree. *Journal of Algorithms*, 15(1):29–60, July 1993.

[FHK78] G. N. Frederickson, M. S. Hecht, and C. E. Kim. Approximation algorithms for some routing problems. *SIAM Journal on Computing*, 7(2):178–193, May 1978.

[FT84] M. F. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. In *Proc. 25th Annual IEEE Symposium on Foundations of Computer Science*, pages 338–346, 1984.

[FW98] A. Fiat and G. J. Woeginger, editors. *Online Algorithms: The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*. Springer, 1998.

[GGST86] H. N. Gabow, Z. Galil, T. Spencer, and R. E. Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6(2), 1986.

[GHKR99] M. Grötschel, D. Hauptmeier, S. O. Krumke, and J. Rambau. Simulation studies for the online dial-a-ride problem. Technical report, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1999.

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

[GL75] H. Gabow and E. L. Lawler. An efficient implementation of edmonds' algorithm for maximum weight matching on graphs. Technical Report TR CU-CS-075-75, Department of Computer Science, University of Colorado, 1975.

[HKR99] D. Hauptmeier, S. O. Krumke, and J. Rambau. The online dial-a-ride problem under reasonable load. Technical report, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1999.

[HKRW99] D. Hauptmeier, S. O. Krumke, J. Rambau, and H. C. Wirth. Euler is standing in line. Technical report, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1999.

[HT84] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

[IK97] S. Irani and A. R. Karlin. Online computation. In D. S. Hochbaum, editor, *Approximation algorithms for NP-hard problems*, chapter 13, pages 521–564. PWS Publishing Company, 1997.

[KM87] L. T. Kou and Y. K. Makki. An even faster approximation algorithm for the steiner tree problem in graphs. In *Proceedings, Congressus Numerantium, Eighteenth Southeastern International Conference on Combinatorics, Graph Theory and Computing*, volume 59, pages 147–154, Boca Raton, Florida, 1987.

[KP94] E. Koutsoupias and C. H. Papadimitriou. Beyond competitive analysis. In *35th Annual Symposium on Foundations of Computer Science*, Foundations of Computer Science, pages 394–400. IEEE, November 1994.

[LLKS93] E. L. Lawler, J. K. L Lenstra, A. H. G. Rinnooy Kan, and S. B. Shmoys. Sequencing and scheduling: Algorithms and complexity. In *Handbooks in OR & MS*, volume 4, pages 445–522. 1993.

[Meh88] K. Mehlhorn. A faster approximation algorithm for the steiner tree problem in graphs. *Information Processing Letters*, pages 125–128, 1988.

[Rag91] P. Raghavan. A statistical adversary for online algorithms. In L. A. McGeoch and D. Sleator, editors, *Online algorithms*, volume 7 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 79–84. AMS/ACM, February 1991.

[RSL77] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis. An analysis of several heuristics for the travelling salesman problem. *SIAM Journal on Computing*, 6(3):563–581, September 1977.

[Sga98] J. Sgall. Online scheduling. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms: The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*. Springer, 1998.

[SV88] B. Schieber and U. Vishkin. On finding lowest common ancestors: Simplification and parallelization. *SIAM Journal on Computing*, 17(6):1253–1262, 1988.

[Tar77] R. E. Tarjan. Finding optimum branchings. *Networks*, 7:25–35, 1977.

[WC83] J. A. Wald and C. J. Colbourn. Steiner trees, partial 2-trees and minimum ifi networks. *Networks*, 13(2):159–167, 1983.

[WZ98] T. Winter and U. Zimmermann. Discrete online and real-time optimization. In *Proceedings of the 15th IFIP World Computer Congress, Budapest/Vienna, August 31 - September 4*, 1998. To appear.

[Zel93] A. Z. Zelikovsky. The 11/6-approximation algorithm for the steiner tree problem. *Algorithmica*, 9(5):463–470, 1993.

# Zusammenfassung

In dieser Diplomarbeit untersuchen wir Online-Algorithmen zur Steuerung industrieller Transportsysteme. Als Praxisbeispiel dient ein integriertes Lastaufzugssystem der Herlitzwerke in Falkensee bei Berlin. Das System wird in Kapitel 1 beschrieben.

Wir formulieren das Problem mathematisch als Online Dial-a-ride-Problem (ONLINE-DARP): Ein Server muß Transportaufträge zwischen jeweils zwei Punkten in einem metrischen Raum bedienen. Die Aufträge werden im Laufe der Zeit bekannt und verfügbar. Der Server kann zu jedem Zeitpunkt nur einen Auftrag transportieren. Wenn der Server angefangen hat, einen Auftrag zu bedienen, dann muß er ihn zu seinem Zielort befördern, ohne den Auftrag zwischendurch abzusetzen. Außerdem formulieren wir eine neue Erweiterung von ONLINE-DARP, bei der Reihenfolgebedingungen zwischen Aufträgen mit dem selben Startpunkt bestehen können. Dieses Problem nennen wir ONLINE-FIFO-DARP. Dies erlaub beispielsweise die Modellierung von Aufzügen, bei denen Paletten über Förderbänder angeliefert werden.

Einige unserer Online-Algorithmen müssen regelmäßig Offline-Instanzen der Probleme ONLINE-DARP und ONLINE-FIFO-DARP lösen. Hier sind von Anfang an alle Aufträge bekannt und verfügbar. In Kapitel 2 zeigen wir ein neues Komplexitätsresultat für DARP (die offline Version von ONLINE-DARP): DARP auf Tausendfüßlergraphen ist NP-schwer. Außerdem beschreiben wir verschiedene Approximationsalgorithmen für DARP aus der Literatur. In Kapitel 3 beschäftigen wir uns mit dem neuen Problem FIFO-DARP (der Offline-Version von ONLINE-FIFO-DARP). Wir beschreiben ein Strukturresultat für dieses Problem und geben einige Approximationsalgorithmen an.

Für die Entwicklung und Analyse der Online-Algorithmen verwenden wir das Konzept der kompetitiven Analyse, beschrieben in Anhang C. Wir geben untere Schranken für die Kompetitivität von Algorithmen aus der Literatur an. Unsere Diskussion konzentriert sich auf die beiden Algorithmen IGNORE und REPLAN, die generelle Ansätze für Online-Algorithmen darstellen. REPLAN berechnet jedesmal wenn ein neuer Auftrag eintrifft ein optimales Schedule für alle bekannten, aber noch nicht abgearbeiteten Aufträge und fängt an, dieses auszuführen. IGNORE dagegen führt ein berechnetes Schedule immer bis zum Ende aus und „ignoriert“ vorläufig alle Aufträge die in der Zwischenzeit entreffen. Wenn IGNORE ein Schedule abgearbeitet hat, berechnet

der Algorithmus ein optimales Schedule für alle „ignorierten“ Aufträge und führt dann dieses aus.

Sowohl IGNORE als auch REPLAN sind kompetitiv. Die Resultate der Kompetitivitätsanalyse liefern keine praxisrelevanten Unterscheidungskriterien für IGNORE und REPLAN. Außerdem gelten alle Aussagen nur für die Zielfunktion der minimalen Gesamtfertigstellungszeit, eigentlich interessieren uns aber die durschnittlichen und die maximalen Flußzeiten von Aufträgen. Für diese Zielfunktionen kann es keine kompetitiven Algorithmen geben, wie wir zeigen.

Dies führt zu der Entwicklung eines neuen Ansatzes zur Untersuchung von Online-Algorithmen in kontinuierlich arbeitenden Systemen. Auftragssequenzen werden beschränkt auf solche, die zu einer „vertretbaren Belastung“[1] führen. Damit ist – grob gespochen – gemeint, daß alle Aufträge die in einem ausreichend großen Zeitintervall eintreffen, von einem optimal arbeitenden Offline-Server in einem ebenso großen Zeitintervall bedient werden können. Unter vertretbarer Belastung läßt sich beweisen, daß für IGNORE sowohl die maximale als auch die durschnittliche Flußzeit beschränkt sind und daß dieses für REPLAN nicht gilt.

Schließlich testen wir verschiedene Algorithmen in Simulationen von einfachen Aufzügen mit FIFO-Warteschlangen und auch in Simulationen des integrierten Aufzugsystems bei Herlitz. Es zeigt sich, daß offenbar ein Zielkonflikt besteht zwischen der Minimierung der maximalen und der durchschnittlichen Flußzeiten. Außerdem werden die theoretischen Ergebnisse bzgl. IGNORE und REPLAN bestätigt – IGNORE erzielt bedeutend bessere maximale Flußzeiten als REPLAN und zeigt insgesamt ein ausgewogeneres Verhalten. Jedoch erreicht REPLAN meistens bessere durschnittliche Flußzeiten. Der erfolgreichste Algorithmus ist anscheinend IGGREEDY, eine Erweiterung von IGNORE, bei der neue Aufträge nicht ignoriert, sondern in den aktuellen Plan eingefügt werden, falls sie eine Leerfahrtstrecke ersetzen.

[1]Deutsche Übersetzung von *reasonable load*, vorgeschlagen von Rainer E. Burkard.