DISS. ETH Nr. 20323

Conflict-free Vehicle Routing with an Application to Personal Rapid Transit

Abhandlung zur Erlangung des Titels

DOKTOR DER WISSENSCHAFTEN

der

ETH Zürich

vorgelegt von

KASPAR SCHÜPBACH

MSc RW, ETH Zürich

geboren am 22. Juni 1981

aus Landiswil, BE

Angenommen auf Antrag von

Prof. Dr. Hans-Jakob Lüthi

Prof. Dr. Martin Skutella

Dr. Rico Zenklusen

2012

Acknowledgments

My time at IFOR started with my Master's project, after which I received the opportunity to continue with doctoral studies. It was the combination of theoretical work (puzzle solving) and industry collaboration which I found most fascinating, together with the often enriching student assistance duties. The first year at IFOR I was mainly working on projects related to timetabling in railways. It was then, through an industry collaboration, that I got in contact with the routing of automated staplers in a logistics center. Theoretical questions on this problem led to the results in the first part of this dissertation. The idea for the case study in the second part came up remembering a project I once wrote about Personal Rapid Transit in high school.

For the opportunity to experience all this I would like to express my heartfelt gratitude towards my doctoral advisor, Professor Hans-Jakob Lüthi. I owe him many thanks for the possibility to bring in my own research ideas and for his support, which I could experience also in difficult times. He has my admiration for his expertise in building bridges between theoretical work and applications. I hope that he will soon find more time to fully enjoy his retirement.

Many many thanks also go to Rico Zenklusen, for kick-starting my theoretical work, for the papers I could write with him, for his support as a referee of this dissertation and for the hospitality of Rico and his wife Sarah during my stay in Boston. I wish all the best to Rico for the continuation of his rocketing academic career.

Further I am very grateful to Martin Skutella for accepting the referee role and for having me in Berlin in November 2010. The stay was definitely a highlight of my doctoral studies. I would also like to thank the crowd from COGA at TU Berlin for the good discussions and for hosting me and showing me around the city.

Further, I would like to express my thanks for the support coming from industry partners, in particular to John Lees-Miller from Ultra, who gave me very interesting insights on the current developments in Personal Rapid Transit.

During the years at IFOR, I enjoyed very much the good vibe among the companions, and the common leisure activities such as skiing days, barbecues, Friday beers, half marathon challenges and more. I feel that I have made many friends and hope these contacts will persist. Many thanks go to everyone for making the time at the institute so enjoyable. The following names I would like to mention in particular: Michael Guarisco and David Adjiashvili as my office mates in the IFOR outpost. With each of you I had a splendid time and I thank you for sharing many delights and sorrows. Gabrio Caimi, Marco Laumanns and Martin Fuchsberger for the close and fruitful collaborations on the railway project in the beginning of my time at IFOR.

Special thanks go to all persons who have been around outside the institute. There are the sports guys, who always are a great source of distraction and motivation for me. In particular, I would like to mention the frequent lunch joggers Markus and Stefan. Then there are the party people from ESN with whom I could organize and experience many fun activities. A particular thank goes to Lars who checked the comprehensibility of a part of this thesis for persons from outside the research field.

A big big hug goes to my girlfriend Arnika for her patience, for her understanding and for cheering me up many times. Thank you so much for sharing this journey with me, and for being there for me.

Finally, I would like to thank my family for their enduring support along all the way.

Abstract

This thesis investigates the conflict-free routing of vehicles through a track network, a problem frequently encountered in many applications in transportation and logistics. The most common routing approach for conflict-free routing problems in various settings is a sequential one, where requests are greedily served one after the other in a quickest way without interfering with previously routed vehicles. There is a need for a better theoretical understanding as guarantees on the quality of the routings are often missing. Conflict-free vehicle routing also is of inherent interest as a sister problem of the well-studied packet routing problem.

In the first part, we present new theoretical results for the case of bidirectional networks. We consider a natural basic model for conflict-free routing of a set of k vehicles. Previously, no efficient algorithm is known with a sublinear (in k) approximation guarantee and without restrictions on the graph topology. We show that the conflict-free vehicle routing problem is hard to solve to optimality even on paths. Building on a sequential routing scheme, we present an algorithm for trees with makespan bounded by $O(\mathrm{OPT}) + k$. Combining this result with ideas known from packet routing, we obtain a first efficient algorithm with sublinear approximation guarantee, namely an $O(\sqrt{k})$-approximation. Additionally, a randomized algorithm leading to a makespan of $O(\mathrm{polylog}(k)) \cdot \mathrm{OPT} + k$ is presented that relies on tree embedding techniques applied to a compacted version of the graph to obtain an approximation guarantee independent of the graph size.

The second part is about routing in the Personal Rapid Transit (PRT) application. PRT is a public transportation mode in which small automated vehicles transport passengers on demand. Central control of the vehicles leads to interesting possibilities for optimized routings. Routing in PRT is an online problem where transit requests appear over time and where routing decisions need to be taken without knowledge of future requests. Further, the network in PRT is directed. The complexity of the routing problems together with the fact that routing algorithms for PRT essentially have to run in real-time often leads to the choice of a fast sequential scheme. The simplicity of such schemes stems from the property that a chosen route is never changed later. This is as well the main drawback of it, potentially leading to large detours.

It is natural to ask how much one could gain by using a more adaptive routing strategy. This question is one of the core motivations of this second part.

We first suggest a variation to the routing model used in the first part which is suitable for PRT. We show that the routing problem remains hard in the directed setting. Further, we introduce a capacity notion for PRT networks and derive a bound for it. Computational results show that the capacity bound is close to the achievable throughput. It therefore is a useful quantity for estimating network capacity in PRT system design. We then introduce a new adaptive routing algorithm that repeatedly uses solutions to an LP as a guide to route vehicles. New requests are integrated into the LP as soon as they appear and the routing is reoptimized over all vehicles concurrently. We provide computational results that give evidence of the potential gains of an adaptive routing strategy. For this we compare the presented adaptive strategy to sequential routing and to a simple distributed routing strategy in a number of scenarios.

Zusammenfassung

Diese Dissertationsarbeit untersucht das konfliktfreie Befördern von Fahrzeugen durch Schienennetzwerke - eine Herausforderung, wie sie oft in verschiedensten Anwendungen in den Transport- und Logistikbranchen vorkommt. Der bekannteste Ansatz für die konfliktfreie Fahrplanung ist ein sequentieller, in dem eine Anfrage nach der anderen mit einer möglichst kurzen Reisezeit eingeplant wird, ohne mit den früheren Anfragen in Konflikt zu geraten. Solche Ansätze haben oft keine Qualitätsgarantien für den resultierenden Fahrplan, und es besteht ein Bedarf für ein besseres theoretisches Verständnis. Das Problem der konfliktfreien Fahrzeugbeförderung ist auch von Interesse durch die enge Verwandtschaft mit dem besser bekannten und untersuchten Problem der Datenbeförderung durch Kommunikationskanäle.

Im ersten Teil der Dissertation präsentieren wir neue theoretische Resultate für den Fall von ungerichteten (in beiden Richtungen befahrbaren) Schienennetzwerken. Wir betrachten ein natürliches Modell für die konfliktfreie Beförderung eines Sets von k Fahrzeugen. Zuvor war kein effizienter Algorithmus mit einer sublinearen (in k) Approximationsgarantie für allgemeine Netzwerktopologien bekannt. Wir zeigen, dass es NP-schwer ist, eine optimale Lösung zu finden, sogar wenn das Netzwerk nur aus einem Pfad besteht. Wir präsentieren einen Algorithmus für Baum-Netzwerke, der auf dem sequentiellen Ansatz aufbaut und zu einem Fahrplan führt, der alle Fahrzeuge in Zeit $O(\text{OPT}) + k$ ans Ziel befördert. Indem wir dieses Resultat mit Methoden aus dem Bereich der Datenbeförderung kombinieren, erhalten wir den ersten effizienten Algorithmus mit sublinearer Approximationsgarantie von $O(\sqrt{k})$. Zusätzlich präsentieren wir einen randomisierten Algorithmus der zu Lösungen der Länge $O(\text{polylog}(k)) \cdot \text{OPT} + k$ führt. Der Ansatz generiert Baumgraphen, die in eine komprimierte Version des Netzwerks eingebettet werden, womit eine Approximationsgarantie erreicht werden kann die unabhängig von der Graphgrösse ist.

Der zweite Teil der Dissertation handelt von der Beförderungen von Fahrzeugen in Personal Rapid Transit (PRT). PRT ist ein öffentliches Verkehrsmittel in welchem Passagiere auf Verlangen durch kleine automatisierte Fahrzeuge befördert werden. Zentrale Steuerung der Fahrzeuge führt zu interessanten Möglichkeiten für die optimierte Nutzung der Schienenressourcen. Die

Fahrplanerstellung für PRT ist ein Online-Problem in dem die Transportaufträge über die Zeit eintreffen. Entscheidungen müssen ebenfalls über die Zeit getroffen werden, ohne Kenntnis der zukünftigen Aufträge. Ein weiterer Unterschied zum ersten Teil der Dissertation ist, dass die Schienen in PRT nur in eine Richtung befahren werden. Die Komplexität des PRT Beförderungs-Problems und die Anforderung, dass die Fahrpläne in Echtzeit berechnet werden müssen, führt oft zur Wahl von schnellen sequentiellen Algorithmen. Die Einfachheit dieser Ansätze ist bedingt durch die Eigenschaft, dass ein geplanter Fahrplan nicht mehr geändert wird sobald er einmal berechnet wurde. Diese Eigenschaft ist gleichzeitig der grösste Nachteil eines solchen Ansatzes, da sie zu grossen Umwegen oder Verzögerungen führen kann. Eine natürliche Frage ist, wie viel man gewinnen kann wenn man adaptive Strategien verwendet. Diese Frage steht im Zentrum dieses zweiten Dissertationsteils.

Zuerst stellen wir eine auf PRT zugeschnittene Variation des mathematischen Modells aus dem ersten Dissertationsteils vor. Wir zeigen, dass das Beförderungs-Problem auch in dieser Variante NP-schwer bleibt. Weiter führen wir einen Kapazitätsbegriff für PRT Netzwerke ein und beweisen eine obere Schranke für diese. Mit Hilfe von Simulationen können wir zeigen, dass die Kapazitätsschranke nicht weit vom erreichbaren Netzwerk-Durchsatz entfernt ist. Sie kann deshalb eine nützliche Grösse zur Abschätzung der Netzwerkkapazität in der Designphase von neuen PRT Systemen sein. Dann präsentieren wir einen neuen adaptiven Algorithmus, der den Fahrplan auf der Grundlage von LP Lösungen erstellt. Neue Aufträge werden gleich beim Eintreffen in das LP integriert und der Fahrplan wird neu berechnet, für alle Fahrzeuge gleichzeitig. Wir zeigen mit Hilfe von Simulationsresultaten das Potential solch adaptiver Strategien. Wir vergleichen den neuen adaptiven Ansatz mit dem sequentiellen Ansatz und mit einem einfachen dezentralisierten Algorithmus in einer Anzahl von Szenarien.

Preface

In this dissertation we present results on the conflict-free vehicle routing problem from two different perspectives. The results are presented in two self-contained parts, each with a separate introduction and conclusion. The first view is a mainly theoretical view on routing on bidirectional networks (two-way traffic). The second view looks into routing strategies for Personal Rapid Transit (PRT), on directed networks (one-way traffic), and reveals a compilation of theoretical results and observations obtained in a computational study. Readers interested mainly in the results on PRT are referred directly to the second part.

In the first part, we investigate the routing problem on bidirectional networks. The main challenge here is to avoid delays from opposing traffic. If, for example, two vehicles use the same route but in opposite direction, one of the two needs to wait until the other has finished the trip. We present a simple and natural model for this setting which is similar to the models in earlier related work and which also has a strong connection to the standard model for packet routing. We consider here an offline setting in which all vehicles are ready to depart at the same time and in which the goal is to find a routing moving all vehicles to their destinations in minimal time. This objective, for which only the time span between earliest departure and latest arrival is relevant, is known as makespan optimization. Most of the results from this first part are published in [SZ11].

The second part studies routing in PRT. This part is written in a language more accessible also for persons from the application side. It starts with an introduction to PRT for readers not familiar with the concept and is followed by a detailed discussion of the model used for routing in PRT. The model used here is adapted from the one used in the first part such to fit the application while keeping the focus on routing questions. The first important adaptation is that the vehicles (resp. transportation requests) now appear over time and that routing decisions need to be taken online. When a new request is released, the routing algorithm includes it into the routing plan without knowledge of the requests to appear in the future. A second adaptation is the change of the objective function towards minimization of the total travel time. The third adaptation is that all tracks now have a designated driving direction, as it is standard in PRT designs.

For this setting, we present a new routing scheme and compare it to two algorithms which are known from the literature. We evaluate the performance of each by computational comparison in several scenarios. The main question addressed is whether it is beneficial to use adaptive algorithms, i.e. algorithms which can change the routing plans for earlier requests when new requests appear. On the theoretical side, we present a method for bounding the capacity of a PRT network. Additionally, we could show that also the PRT routing problem is NP-hard to solve to optimality. A preliminary version of the results from the second part of the thesis are published in [SZ12].

Contents

Part I

Approximation Algorithms for Conflict-free Vehicle Routing on Bidirectional Networks

Chapter 1

Introduction

In the first part of this thesis, we investigate the conflict-free routing of vehicles through a network of bidirectional guideways. Conflicts are defined in a natural way, i.e., vehicles cannot occupy the same resource at the same time, hence forbidding crossing and overtaking. The task is to find a routing consisting of a route selection and a schedule for each vehicle, in which they arrive at their destinations as quickly as possible.

Such conflict-free routing algorithms are needed in various applications in logistics and transportation. A prominent example is the routing of Automated Guided Vehicles (AGVs). AGVs are often employed to transport goods in warehouses (for survey papers we recommend [GHS98, Vis06]), or to move containers in large-scale industrial harbors [SV08]. The guideways can be tracks or any sort of fixed connected and bidirectional lane system. Other related application settings are the routing of ships in canal systems [PT88], locomotives in shunting yards [FLKH05], or airplanes during ground movement at airports [GBM$^+$02, ABR10].

Conflict-free vehicle routing problems can be divided into online problems, where new vehicles with origin-destination pairs are revealed over time, and offline problems, where the vehicles to route are known in advance together with their origin-destination pairs. Here, we concentrate on the offline problem, which is also often a useful building block for designing online algorithms.

Algorithms for conflict-free routings either follow a *sequential* or *concurrent* routing paradigm. Sequential routing policies consider the vehicles in a given order, and select a route and schedule for each vehicle such that no conflict occurs with previously routed vehicles (see [KT91, MKGS05, KJR07] for sequential routing examples in the context of AGVs). Concurrent approaches take into account multiple or all vehicles at the same time. Whereas the higher flexibility of those approaches opens up possibilities to obtain stronger routings than the sequential paradigm, they usually lead to very hard optimization problems. Furthermore, they are often difficult to implement in practice.

Typically, the routing problem is modeled as an Integer Program (IP) which is tackled by IP solvers [Oel08], column generation methods [FLKH05] or heuristics without optimality guarantee [PT88, GBM$^+$02, KBK93].

Sequential algorithms are thus often more useful in practice due to their computational efficiency but suffer from the difficulty of finding a good sequence to route the vehicles. Furthermore, the theoretical guarantees of these approaches are often weak. The goal of this work is to address these shortcoming of sequential routing algorithms. Most of the results presented in the following are also published in [SZ11].

1.1 Problem Formulation

We consider the following problem setting which captures common structures of many conflict-free vehicle routing problems.

Conflict-Free Vehicle Routing Problem (CFVRP). *Given is a undirected connected graph $G = (V, E)$, and a set of k vehicles Π with origin-destination pairs (s_π, t_π) for all $\pi \in \Pi$. Origins and destinations are also called* terminals. *A discretized time setting is considered with vehicles residing on vertices. At each timestep, every vehicle can either stay (wait) on its current position or move to a neighboring vertex. Vehicles are forbidden to traverse the same edge at the same timestep, also when driving in opposite directions, and no two vehicles are allowed to be on the same node at the same time. A routing not violating the above rules is called* conflict-free. *The goal is to find a conflict-free routing minimizing the* makespan, *i.e. the number of timesteps needed until all vehicles reach their destination.*

The CFVRP is a natural first candidate for modelling and analyzing routing problems in a variety of contexts. Clearly, it omits application-specific details and makes further simplifying assumptions.

As a relaxation of the conflict definition above, we assume that vehicles can only be conflicting while in transit, i.e., no conflict is possible before departure and after arrival. The *departure time* of a vehicle is the last timestep that the vehicle is still at its origin, and the *arrival time* is the earliest time when the vehicle is at its destination. We call this relaxation the *parking assumption.* The parking assumption is natural in many of the listed applications since the terminal node occupations are often managed by separate procedures. In AGV systems the dispatching (task assignment) is usually separated from the routing process and takes care of terminal node occupations. In airport ground movement problems, airplanes are assigned to runways and gates before airplane routing starts. When routing ships through a canal system, the terminals represent harbors with usually enough space for conflict-free parking of all arriving and departing vessels.

1.2 Related Work

The model setting investigated in [KBK93, Spe06, Ste08] is very similar to the one used here. The differences lie mostly in the modelling of waiting vehicles, which block edges in their setting. In [KBK93], only designated edges can be used for waiting. However, these variations do not significantly change the problem, and the results can easily be transferred. The main reason why we use the CFVRP setting introduced above is that it leads to a simplified presentation of the algorithms.

CFVRP has many similarities with packet routing [Sch98, PSW09], where the goal is a conflict-free transmission of data packets through cable networks. The crucial difference is that the conflict notion in packet routing is relaxed. It allows for several packets to occupy a node at the same time, as nodes represent network routers with large storage capacity. The concept of edge-conflicts is essentially the same as in the present setting and models the limited bandwidth of the transmission links. Hence, the CFVRP setting can as well be seen as a packet routing problem with unit capacities on every node.[1]

Some sequential routing approaches. We briefly discuss some variants of sequential routing schemes, emphasizing on approaches used later when presenting the algorithms.

The presumably simplest approach is to serially send one vehicle after another on a shortest route to the destination, such that a vehicle departs as soon as the previous one has arrived. The obtained makespan is bounded by $k \cdot L$, where L is used as the maximum origin-destination distance over all vehicles. Since L is a lower bound on the optimal makespan OPT, this is a k-approximation. Interestingly, for general graph topologies, no efficient algorithm was known to substantially beat this approach, i.e. with a $o(k)$ approximation guarantee.

[1]There are approximation results for packet routing with buffer size 1 in [adHS95]. However, contrary to the CFVRP, they consider bidirectional edges on which two packages can be sent concurrently in opposite directions.

Still, several stronger routing paradigms are known and commonly used in practice. An improved sequential routing policy, which we call simply *sequential routing*, is the following procedure as introduced in [KT91, MKGS05]. Vehicles are considered in a given order, and for each vehicle a route and a schedule (timetable) is determined with earliest arrival time, avoiding conflicts with previously scheduled vehicles. For a fixed ordering of the vehicles, a sequential routing can be obtained efficiently, e.g. by finding shortest paths in a time-expanded graph. Sequential routing is often applied with given origin-destination paths for all vehicles, in which case the task is only to find a schedule for each vehicle that determines how to traverse its origin-destination path over time.

For given origin-destination paths, the following restricted version of sequential routing algorithms, called *direct routing*, often shows to be useful. In direct routing, see e.g. [BMIMS04] for the corresponding approach in packet routing, vehicles are not allowed to wait while in transit, i.e., once a vehicle leaves its origin, it has to move to its destination on the given origin-destination path without waiting. An advantage of direct routing is that vehicles only block a very limited number of vertex/time slot combinations.

Combining this concept with the sequential routing, the *direct sequential* algorithm is obtained. Here an ordering of the vehicles is given, as well as a source-destination path for each vehicle. Considering vehicles in the given ordering, the routing of a vehicle is determined by finding the earliest possible departure that allows for advancing non-stop to its destination on the given path, without creating conflicts with previously routed vehicles.

When fixing the origin-destination paths to be shortest paths, sequential routing and its direct variant perform at least as good as the trivial serial algorithm. However, for unfortunate choices of the routing sequence, one can observe that the resulting makespan of both approaches can still be a factor of $\Theta(k)$ larger than the optimum (see [Ste08] for details).

Further related results. Spenke [Spe06] showed that the CFVRP is NP-hard on grid graphs. The proof implies that finding the optimal priorities for sequential routing is also NP-hard.

Polynomial routing policies with approximation quality sublinear in k are known for grid graphs. Spenke introduces a method for choosing a routing sequence with a makespan bounded by $4\text{OPT} + k$. An online version of the problem was investigated by Stenzel [Ste08], again for grid topologies.

Computational results published in [Ste08] indicate that sequential algorithms can have bad performance when the number of route choices of near-shortest-path lengths are limited. For grid topologies, the above-mentioned algorithm of Stenzel [Ste08] takes advantage of the fact that grid graphs contain at least two disjoint routes of almost the same length for each pair of vertices.

1.3 Outline

On the negative side we present in Chapter 2 hardness results showing that there is not much hope to obtain exact solutions even for seemingly simple settings. The results give a theoretical explanation for the difficulties encountered in practice when looking for good orderings for sequential routings.

On the positive side, we consider in Section 3.1 the CFVRP problem on trees, and present a priority ordering of the vehicles leading to a direct routing algorithm with a makespan bounded by $4\mathrm{OPT} + k$. This is achieved by dividing the vehicles into two groups, and showing that each group admits an ordering which leads only to very small delays stemming from vehicles driving in opposite directions.

For general instances, without restrictions on the graph topologies, we show how the tree algorithm can be leveraged to obtain a $O(\sqrt{k})$-approximation, thus leading to the first sublinear approximation guarantee for the CFVRP problem. A crucial step of the algorithm is to *discharge* high-congestion vertices by routing vehicles on a well-chosen set of trees. The purely multiplicative approximation guarantee is obtained despite the $+k$ term in the approximation guarantee for the tree algorithm by exploiting results from the packet routing literature. More precisely, using an approach of Srinivasan and Teo [ST97], we determine routes for the vehicles with a congestion C bounded by $C = O(\mathrm{OPT})$, and never route more than C vehicles over a given tree. These results are presented in Section 3.2.

Additionally, an efficient randomized method with makespan $O(\log^3 k)\mathrm{OPT}+k$ is presented for general graph topologies in Section 3.3. This approach relies on obtaining strong tree embeddings in a compacted version of the graph, therefore avoiding a dependency of the approximation guarantee on the size of the graph, which would result by a straightforward application of tree embeddings.

Chapter 2

Hardness Results

In this chapter we present new hardness results for the CFVRP. Hardness results are important in the sense that they allow to shift the focus towards the development of approximation algorithms and heuristics, as they imply that the optimal solution cannot be found in polynomial time unless P=NP.

Our first result shows hardness for the special case of path topologies. It may be surprising that already this appearingly simple setting turns out to be hard to solve. The optimal coordination of vehicles with opposing driving directions corresponds to a difficult packing type problem, as we will show in the first section.

The question arises whether it is due to the two-way traffic that the CFVRP is hard. To answer this question, we consider in the second section of this chapter a problem variant in which the graph is directed and opposing traffic is hence not an issue. It turns out that this problem variant is still hard to solve for tree topologies.

2.1 On Paths

Theorem 2.1. *The CFVRP problem on paths is NP-hard.*

Proof of Theorem 2.1. We reduce from the 3-Partition problem, where a vector $A = (a_1, \ldots, a_{3m})$ of $3m$ positive integers and a bound $B \in \mathbb{Z}^+$ are given such that $\sum_{i \in \{1,\ldots,3m\}} a_i = mB$ and $B/4 < a_i < B/2$ for all $i \in \{1, \ldots, 3m\}$. The Problem is to decide whether the coefficients of the vector can be partitioned into m disjoint vectors $A_0, \ldots, A_{m-1}$ each of length 3 and with coefficients summing up to B. 3-Partition is known to be strongly NP-hard [GJ79].

Given an instance of 3-Partition, we construct an instance of the CFVRP on a path, such that for some value $T \in \mathbb{N}$, the constructed CFVRP instance has an optimal makespan of at most T if and only if the underlying 3-Partition instance is feasible.

We set $\beta = 2B + 9$ and $T = (m + 1)\beta$. Without loss of generality, we assume that a_i to be even for all $i \in \{1, \dots, 3m\}$. The graph of the CFVRP instance is a path of length $2T + \beta$. The nodes are labeled with increasing numbers from left to right starting with 0. The nodes with labels T to $T + \beta$, are referred to as *critical* nodes. We introduce vehicles running in both directions of the path, as listed in Table 2.1.

The general idea is to use the right-to-left ($\leftarrow$) vehicles to generate separated areas of unoccupied time-space slots, which we call *time-space windows*, or simply *windows*. These can then be used to route the left-to-right ($\rightarrow$) vehicles. More precisely, between the vehicles of types I-III, there are m time-space windows, forming stripes going diagonally down from top-right to bottom-left. Each of these stripes has a width of $\beta - 1 = 2B + 8$. We call the width of such a stripe, the *size* of the window. Since the vehicles of type I-III have a route length identical to the makespan threshold T, there is no flexibility in scheduling, i.e., in any routing with makespan T, they must depart immediately at time zero using direct routing (no waiting or driving backwards). There are two more $\leftarrow$ vehicles passing the critical nodes between every two consecutive vehicles of type III, one of type IV and one of type V. They cut each of the m time-space windows into three smaller windows (see Figure 2.1).

The sizes of those smaller windows is not fix, as vehicles of type IV have a route length smaller than T which allows for a certain flexibility in scheduling. Their departure time can be chosen within the range $[0, B + 2]$. We assume here, without consequences for the result, that vehicles of type IV are also routed directly. As we will see later, the only property we need is that vehicles of type IV do not wait at critical nodes, and this property must be fulfilled to obtain a makespan of T since for every critical node v, there are $T + 1$ vehicles whose paths contain v. Therefore, at each timestep, i.e. at start and the T following timesteps, each of those $T + 1$ vehicles must occupy v for precisely one time slot.

Let us denote the sizes of the resulting $3m$ time-space windows by $\alpha_1, \dots, \alpha_{3m}$, in temporal order. By appropriate choice of the departure times for the vehicles of types IV-V, the window sizes can attain any value between 1 and $B + 3$, as long as $\alpha_{3i+1} + \alpha_{3i+2} + \alpha_{3i+3} = 2B + 6$ for all $i \in \{0, \dots, m - 1\}$.

Table 2.1: Each line of the table corresponds to a group of vehicles to be routed. Vehicles described in the same line have the same direction (indicated in the corresponding column), and the same length, i.e., distance between origin and destination. The origins of a group of vehicles described by one line are given by an arithmetic progression, starting at the value *First*, ending at *Last*, and with step size *Step*. For example, the third line of the table contains a vehicle with origin $T+\beta$, one with origin $T+2\beta$ and so on, with the last one having its origin at node $T+(m+1)\beta = 2T$. The last column summarizes the number of vehicles described by each line. Note that the vehicles of type VI are introduced in identical pairs.

		Origin				
Type	Direction	First	Step	Last	Length	Number
I	$\leftarrow$	T	1	$T+\beta-1$	T	β
II	$\leftarrow$	$2T+1$	1	$2T+\beta$	T	β
III	$\leftarrow$	$T+\beta$	β	$T+(m+1)\beta$	T	$m+1$
IV	$\leftarrow$	$T+\beta+2$	β	$T+m\beta+2$	$T-(B+2)$	m
V	$\leftarrow$	$T+\beta+B+6$	β	$T+m\beta+B+6$	$T-(B+2)$	m
	$\rightarrow$	$T-a_1$	1	$T+\beta$	a_1	$\beta+a_1+1$
	$\rightarrow$	$T-a_1$	1	$T+\beta$	a_1	$\beta+a_1+1$
	$\rightarrow$	$T-a_2$	1	$T+\beta$	a_2	$\beta+a_2+1$
VI	$\rightarrow$	$T-a_2$	1	$T+\beta$	a_2	$\beta+a_2+1$
	$\vdots$		$\vdots$		$\vdots$	$\vdots$
	$\rightarrow$	$T-a_{3m}$	1	$T+\beta$	a_{3m}	$\beta+a_{3m}+1$
	$\rightarrow$	$T-a_{3m}$	1	$T+\beta$	a_{3m}	$\beta+a_{3m}+1$

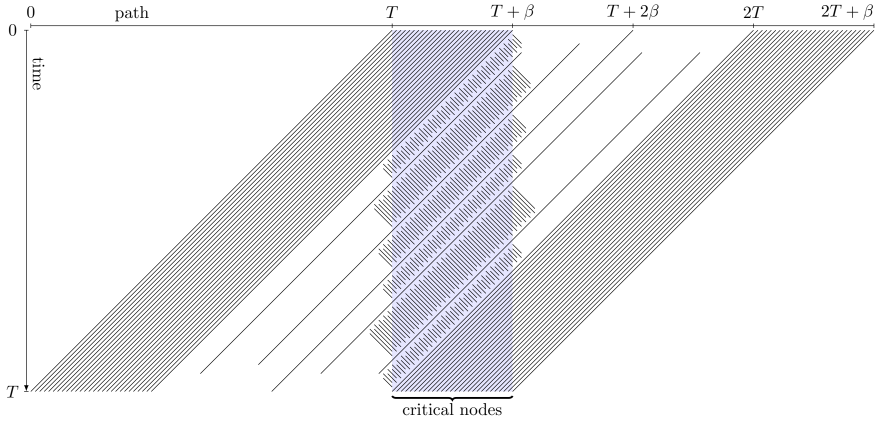

Figure 2.1: Sketch of the time-space diagram for $m = 2$. The lines going from top-right to bottom-left indicate schedules of vehicles of types I-V. The block of adjacent parallel lines to the left corresponds to vehicles of type I and the block to the right to vehicles of type II. The other parallel lines between these blocks correspond to vehicles of types III-IV. The short lines from top-left to bottom-right indicate schedules for the vehicles of type VI.

This is true because the three consecutive time-space windows result from the subdivision of a larger window of size $2B+8$, and two timesteps are occupied by the two vehicles of types IV and V. Notice that all time-space windows are non-empty: as vehicles of types III-V cannot be scheduled one right after the other, there always must be a gap of at least one timestep in-between.

Vehicles of type VI run in opposite direction. They were introduced such that for each critical node v, $6m$ vehicles have their origin at v, twice a set of $3m$ vehicles with trip lengths $\{a_1, \ldots, a_{3m}\}$.

We now show how to find a conflict-free direct routing plan with makespan T from a feasible 3-Partition $A_0, \ldots, A_{m-1}$. Vehicles of types I-III clearly must depart at time zero. The departure times of vehicles of types IV-V are chosen such that the window sizes fulfill $\alpha_{3i+j} = 2(a_{i,j} + 1)$ for every $i \in \{0, \ldots, m-1\}$, $j \in \{1, 2, 3\}$ and $a_{i,j}$ corresponding to the j^{th} element of A_i. It is possible to choose such departure times, as $a_{i,1} + a_{i,2} + a_{i,3} = B$ leads to $\alpha_{3i+1} + \alpha_{3i+2} + \alpha_{3i+3} = 2B + 6$, and $B/4 < a_{i,j} < B/2$ leads to the bounds $B/2 + 2 < \alpha_{3i+j} < B + 2$.

The vehicles of type VI can now be routed using these time-space windows. A time-space window of size α, delimited by $\leftarrow$ vehicles, can be used to send two $\rightarrow$ vehicles of trip length $\alpha/2 - 1$ from each critical node (see Figure 2.2). By the choice of the window sizes above, all vehicles of types VI can be routed in the respective windows, leading to a routing plan with makespan T.

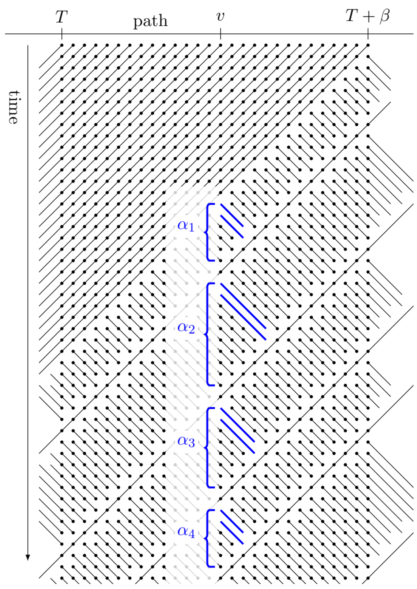

Figure 2.2: A closer view on the critical nodes, showing how the vehicles corresponding to a_i can be scheduled within a window of size $\alpha_i = 2(a_i + 1)$. In a schedule with makespan T, each critical node is always occupied. For the critical node v, the set of $\rightarrow$ vehicles departing from v are marked in blue. There are exactly two per window.

It remains to show the converse, that, if there is a solution to the CFVRP instance with makespan T, there necessarily exists a feasible solution to the underlying 3-Partition instance. Remember that the instance was constructed such that all critical nodes are on the route of exactly $T + 1$ vehicles. A routing with makespan T is therefore only possible if every such node is occupied on each timestep, because there is no spare capacity. Also a routing involving waiting or detours on the critical nodes can be excluded.

We now argue that any feasible routing with makespan T solves implicitly the underlying 3-Partition problem. For this purpose, look at some critical node v. We claim that a time-space window of size α must contain two $\rightarrow$ vehicles of trip length $\alpha/2 - 1$ departing from v. This follows from the fact that v and all other critical nodes can never be unoccupied. Note that α needs to be even as the lack of vehicles of odd trip length would lead to unoccupied nodes otherwise. Scheduling less than two vehicles departing from v into a time-space window also necessarily would lead to unoccupied nodes. However, this as well implies that it is not possible to schedule more than two vehicles departing from v into a time-space window, since there are exactly $6m$ vehicles departing from v to be distributed over $3m$ windows. Scheduling shorter vehicles than claimed would also lead to unoccupied nodes. Combining several smaller vehicles is again not possible as there are no spare vehicles.

One can hence only route all $\rightarrow$ vehicles if the time-space windows have sizes $2(a_1 + 1), 2(a_2 + 1), \ldots, 2(a_{3m} + 1)$. With the additional restriction that the sizes of any three time-space windows $\alpha_{3i+1}, \alpha_{3i+2}, \alpha_{3i+3}$ for $i \in \{0, \ldots, m-1\}$ must sum up to $2B + 6$. Hence, the window sizes correspond to a solution to the 3-Partition problem. $\square$

The proof of Theorem 2.1 considers a problem instance where the optimal routing can be chosen to be a sequential routing, thus implying the following result.

Corollary 2.2. *Choosing an optimal ordering of the vehicles for sequential routing is NP-hard, even if the underlying graph is a path.*

2.2 On Directed Trees

We have seen that the optimal routing of vehicles on a path with two-way traffic is hard. We now ask the question whether there exist efficient algorithms if we restrict the problem to instances without opposing traffic. For this purpose we consider a variant of CFVRP on directed graphs and will show that this variant is hard to solve also for the special case of trees.

Theorem 2.3. *Conflict-free vehicle routing is NP-hard on directed trees.*

Proof. We show this by reduction from 3-Bounded-3-SAT (3B3S), which is known to be NP-complete [GJ79]. A SAT instance is a boolean formula consisting of disjunctive (OR) clauses over a set of n variables. The following small example consist of the variables x, y, z and two clauses containing three *literals* each, where $\bar{y}$ denotes the negation of variable y.

$$(x \vee \bar{y} \vee z) \wedge (x \vee y \vee \bar{z})$$

The objective is to find an assignment of true/false to the variables such that each of the clauses in a SAT instance is satisfied. A clause is satisfied if one of the variables occurring as unnegated (respectively negated) literal is set to true (false).

3B3S is a variant of SAT in which each clause consists of at most three variables and each variable occurs in at most three clauses. We can assume that each variable occurs at least once as a negated and once as an unnegated literal. Otherwise, one could simply set the variable such that all clauses containing it are satisfied and the variable could be removed from the problem together with the clauses containing it. Consequently, a variable appears at least once and at most twice as negated or unnegated literal. Furthermore, one can assume that each clause consists of at least two literals, as single-literal clauses can also be removed easily. Let in the following c_3 and c_2 be the numbers of clauses consisting of three respectively two literals in the 3B3S instance.

In the following, we relate instances of 3B3S to instances of CFVRP on a tree network such that the first has a satisfying assignment if and only if there exists a routing plan with a certain makespan T for the second. The tree network with its corresponding 3B3S instance is shown in Figure 2.3. There are four vehicles per variable, labeled $x, x', \bar{x}, \bar{x}'$ for a variable x, and one additional for each appearance of a literal in a clause, labeled c^1x for the first appearance of x in a fixed ordering of clauses, respectively c^2x for the second. The tree topology for other 3B3S instances is generated accordingly. The number of tree branches depends on the numbers of clauses and variables in the 3B3S instance under consideration. The tree is directed in the obvious way.

We introduce additional vehicles with the purpose of blocking certain edges for certain timesteps. An edge e can be blocked for a time slot $[t, t+1]$ in the following way. We merge the tree graph with a path of length T whose nodes are labeled consecutively $(v_0, \dots, v_T)$. The merge is done by fusion of edge e from the tree with edge $\{v_t, v_{t+1}\}$ from the path. By introducing a vehicle with origin v_0 and destination v_T, we enforce that in any feasible routing plan, for time slot $[t, t+1]$, the edge e is occupied by this vehicle. Note that the merged graph is also a tree. We will in the following not mention explicitly the blocking vehicles and their paths but only indicate which edges are blocked for which time slots. In particular, we introduce blockings on the colored edges (blue, green, brown and red) of Figure 2.3. Note that the colored edges do not share nodes and therefore no conflicts between blocking vehicles occur.

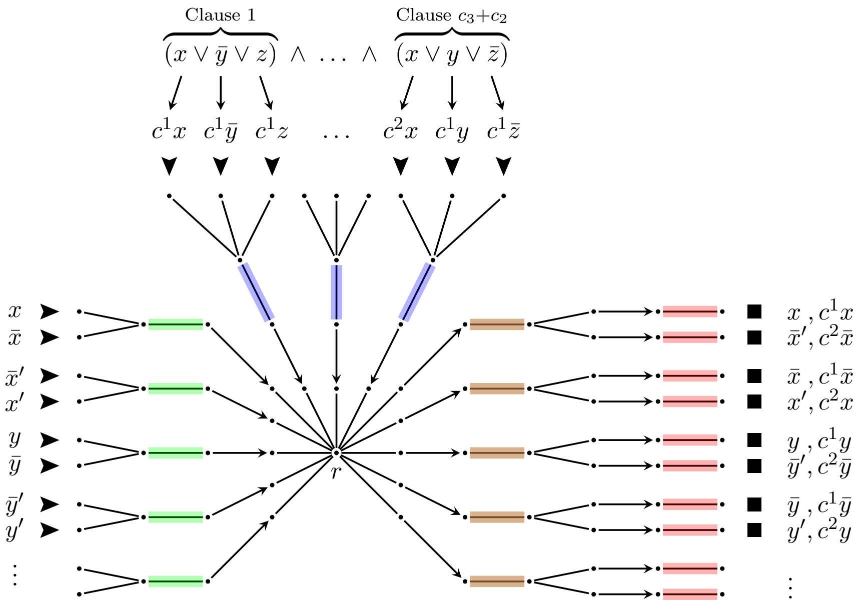

Figure 2.3: Illustration of the tree network used for the hardness reduction. The origins and destinations of the vehicles are indicated by darts and squares, along with the respective vehicle labels. The traffic direction is indicated by arrows on some of the arcs. The colored edges represent the so-called gates and are open only for certain timesteps.

For the ease of explication, we split the makespan into phases I - IV. Their lengths and blockings are indicated in Table 2.2.

We will show in the following that for a 3B3S instance, there exists a feasible routing plan with makespan $T = 4n + 3c_3 + 2c_2 + 36$ to the corresponding CFVRP instance, if and only if the 3B3S instance has a satisfying assignment.

If direction: Consider a fixed satisfying assignment of the 3B3S instance. It has at least one satisfied literal per clause. We call the corresponding clause vehicle the satisfying vehicle (pick any in case there are several candidates). For the variable vehicles, we call the vehicles x and x' (resp. $\bar{x}$ and $\bar{x}'$) true if the variable x is set true (false) in the satisfying assignment, the others we call false vehicles.

We construct the following routing plan. In phase I, one non-satisfying vehicle per clause is sent from origin to destination. More precisely, one vehicle per

Table 2.2: The table shows when the colored edges (blue, green, brown and red) are blocked. The makespan is split into four phases I-IV, each of the length indicated. In each phase, the edges of one color are either open, blocked or blocked except for one particular time slot, called *gate*. Gate[1] means that the open slot is the time slot $[1,2]$ of the respective phase. Similarly, gate[2] refers to slot $[2n+5, 2n+6]$ and gate[3] to slot $[2n+c_3+c_2+8, 2n+c_3+c_2+9]$ within the phase. Gate[4] is open for slot $[1,2]$ only in case the blue edge corresponds to a clause with three variables and closed otherwise.

		edge color (according to Figure 2.3)			
phase	length	blue	green	brown	red
I	$c_3 + c_2 + 9$	gate[1]	closed	open	open
II	$c_3 + 9$	gate[4]	closed	open	open
III	$2n + 9$	closed	gate[1]	gate[2]	open
IV	$2n + c_3 + c_2 + 9$	open	open	open	gate[3]

clause is sent directly to the pre-root node until time 3. Note that the blue gate is open at the right timestep. Next, the vehicles cross the root node one after another and proceed to their destinations (the brown and red edges are open in phase I). For crossing the root node and proceeding to the destinations, the $c_3 + c_2$ vehicles need another $c_3 + c_2 + 6$ timesteps.

Phase II is a repetition of phase I, except that the blue gate is only open for the 3-literal-clause vehicles. In this phase, the remaining c_3 non-satisfying clause vehicles are sent to their destinations in $c_3 + 9$ timesteps.

In phase III, the true variable vehicles are sent to their destinations. There are $2n$ such vehicles, and their paths are independent except for the root node. They pass the green gate, the root-node and the brown gate before proceeding to their destinations in total time of $2n + 9$.

At this point, the satisfying clause vehicles (one per clause) and the false variable vehicles (two per variable) remain to be routed. These vehicles all have different destinations. If a satisfying clause vehicle had the same destination as a false variable vehicle that would be a contradiction. The green and blue edges open at the beginning of phase IV. The $2n + c_3 + c_2$ vehicles pass the root, wait two nodes away from their individual destinations before passing the red gate to arrive simultaneously at time $2n + c_3 + c_2 + 9$ of phase IV. Appending the phases I-IV results in a feasible routing plan with total makespan $T = 4n + 3c_3 + 2c_2 + 36$.

Only if direction: After completion of phases I-III, at least one vehicle per clause has not yet passed the blue gate. Similarly, for the variable vehicles, at least two vehicles per variable remain to pass the brown gate, and the remaining variables contain one of the pairs $\{x, x'\}$ or $\{\bar{x}, \bar{x}'\}$ (note that, for example, if x passes the brown gate in phase III, it blocks the green gate for $\bar{x}$ and the brown gate for $\bar{x}'$).

During phase IV, the red gate must in consequence be passed by at least one clause vehicle per clause and by one pair of variable vehicles per variable. If pair $\{x, x'\}$ passes the red gate in phase IV, the gate slot is taken and c^1x (and c^2x) cannot pass the gate in that phase. However, $c^1\bar{x}$ (and $c^2\bar{x}$) can pass as they have different destinations than $\{x, x'\}$. On the other hand, $\{\bar{x}, \bar{x}'\}$ can pass together with c^1x and c^2x but not with $c^1\bar{x}$ and $c^2\bar{x}$. We can construct a 3B3S solution out of a routing by setting the variable x to false in the first case and to true in the latter. As stated before, at least one clause vehicle per clause needs to pass the red gate. This corresponds to at least one satisfied literal in the 3B3S instance.

$\square$

We have presented two hardness results in this chapter, the first for the bidirectional case, the standard setting of CFVRP, and the second for the directed case. We learn that CFVRP cannot be solved to optimality in polynomial time unless P=NP. While the first result shows that this remains true even for instances without intersections, the second does the same for instances without opposing traffic.

The combination of both restrictions is the CFVRP on directed paths. For this, the computational complexity remains open. The greedy farthest-destination-first algorithm [Leu04], which is optimal for the directed path case in packet routing, does not necessarily lead to optimal solutions in the CFVRP. The case is interesting as a polynomial-time algorithm would directly lead to a 2-approximation for the CFVRP on undirected paths.

Chapter 3

Approximation Algorithms

In this chapter, we present approximation algorithms with guarantees on the quality of the solutions and with polynomial runtime bounds. The algorithms are variants of sequential routing with particular priority sequences and route choices. We start by presenting a routing algorithm for trees. Then we show two ways of extending the tree algorithm to general graphs, resulting in the first routing algorithms for CFVRP with approximation guarantees sublinear in k.

3.1 Tree Approximation

Throughout this section, we assume that the given graph $G = (V, E)$ is a tree and we fix an arbitrary root node $r \in V$. The nodes are numbered as follows. We perform a depth-first search (DFS) on G starting at r, and number the nodes in the order in which they are first visited during the DFS.

The vehicles are partitioned into *increasing vehicles* and *decreasing vehicles*. A vehicle is increasing if the label of its destination is larger than the one of its origin, and decreasing otherwise. We use k^+ and k^- to denote the number of increasing and decreasing vehicles, respectively.

Vehicles will be routed on the unique path from origin to destination. On this path, the node which is closest to the root is called the *bending node* of the vehicle. The labels of the last node before and the first node after the bending node are referred to as *in-label* and *out-label*, respectively. Notice, that the bending node can coincide with the origin or destination node. In this case the in-label or out-label, respectively, is not defined. Increasing vehicles are always guaranteed to have out-labels while decreasing vehicles certainly have in-labels. See Figure 3.1 for an illustration.

The tree routing algorithm, TreeRouting, is a special case of sequential

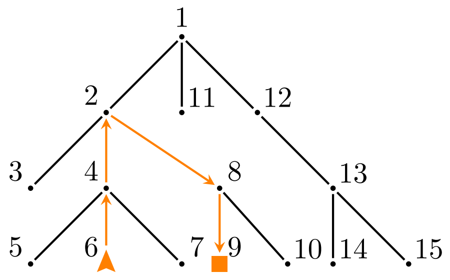

Figure 3.1: Tree with nodes labeled in DFS order, starting at the root node with label 1. The route for an increasing vehicle with origin at label 6 and destination at label 9 is indicated. The bending node of the vehicle has label 2, the in-label is 4 and the out-label 8. Note that all increasing vehicles have direction left-to-right or top-to-bottom in this illustration, and the decreasing right-to-left or bottom-to-top, respectively.

routing, where vehicles are routed in order of decreasing priorities, for any priority list satisfying the following rules:

i) increasing vehicles have priority over decreasing vehicles,
ii) among two increasing vehicles, the one with higher out-label has priority,
iii) among two decreasing vehicles, the one with lower in-label has priority,
iv) ties are broken using an arbitrary fixed vehicle ordering.

Theorem 3.1. *The makespan obtained by* TREEROUTING *is bounded by* $4L + k$*.*

Recall that L is the maximum origin-destination distance over all vehicles. For the proof of the theorem, we start by showing that a particular direct routing exists with the desired makespan, and then deduce Theorem 3.1 from this result. For the rest of this section, we assume that priorities satisfying the priority rules of TREEROUTING are assigned to the vehicles.

Lemma 3.2. *There exists a direct routing along the unique origin-destination paths with a makespan of at most* $4L+k$*, and such that for every node* $v \in V$*, the vehicles that visit* v*, do this in order of decreasing priorities.*

Proof. We will show how to construct a direct routing only containing the increasing vehicles with a makespan of $2L+k^+$, such that vehicles visit nodes in order of decreasing priorities. Analogously, a routing with a makespan of

```
Algorithm TreeRouting
  Sort the vehicles in order of decreasing priorities.
  Apply sequential routing along the unique paths using this ordering.
```

$2L + k^-$ can be obtained for the decreasing vehicles.[1] The result then follows by serially applying those two routings.

Notice that direct routing along given paths is fully specified by fixing for each vehicle the passage time at one node on its route. Consider the direct routing that is obtained by fixing for each vehicle the passage time at its bending node as follows: the passage time at the bending node of the highest priority vehicle is set to L, the one with second-highest priority is $L + 1$, and so on, hence leading to a passage time at the bending node of $L + k^+ - 1$ for the increasing vehicle with the lowest priority. Thus, if this routing is conflict-free, then all (increasing) vehicles arrive latest at timestep $2L + k^+ - 1$, thus respecting the desired makespan.

We finish the proof by showing that if two vehicles visit the same node v, then the one with lower priority occupies v strictly later than the one with higher priority. This implies as well that the routing is conflict-free. Notice that in particular no edge conflicts are then possible because having an edge conflict between two vehicles would violate the priority ordering at one of its endpoints. Hence, consider two vehicles π and ψ, and assume that ψ has a higher priority than π. We distinguish four cases.

Case 1: π and ψ do not share any node. Here, the claim follows trivially.

Case 2: π and ψ share exactly one common node v. Note that v must be the bending node of at least one of the two vehicles. Let τ_v^ψ and τ_v^π be the passage times of ψ and π at node v, and let τ^ψ and τ^π be the bending times of ψ and π. We show that the higher priority vehicle ψ passes first at v, i.e. $\tau_v^\psi < \tau_v^\pi$, by proving the following chain of inequalities.

$$\tau_v^\psi \leq \tau^\psi < \tau^\pi \leq \tau_v^\pi$$

To check the first inequality assume by sake of contradiction that $\tau_v^\psi > \tau^\psi$, i.e., ψ reaches v after having bent. Hence, v must be the bending node of π, and thus the bending node of π is a descendant of the out-node of ψ (i.e., the

[1] To reduce this case to the increasing case, one can for example swap for every decreasing vehicle the origin and destination, thus turning them into increasing ones. A routing obtained using the procedure for increasing vehicles can then be transformed into a legal routing for the decreasing vehicles by reverting time.

out-node of ψ lies on the path between r and the bending node of π). This contradicts that ψ has a higher priority than π. The second, strict inequality holds because of the assignment of smaller bending node passage times to higher priority vehicles. The third inequality holds by a reasoning analogous to the one used for the first inequality.

It remains to discuss cases where π and ψ share more than one node. As G is a tree and routes are paths, these common nodes necessarily form a connected path. It remains to distinguish in which direction this common subpath is traversed by the vehicles.

Case 3: π and ψ use a common subpath in the same direction. Let v denote the common node with the smallest label. It corresponds to the one of the two bending nodes which is closer to the root. The same analysis as in the second case shows that ψ passes v first. As we apply direct routing and ψ and π use the common subpath in the same direction, the lead of ψ over π is the same on all common nodes, and the claim hence follows.

Case 4: π and ψ use a common subpath in opposite directions. The vehicles cannot bend on the common subpath because that would contradict with both vehicles being increasing. One vehicle hence approaches the root while the other one goes away from it. Observe that the approaching vehicle has the larger out-label, and hence must be the higher-priority vehicle ψ. Let v again denote the common node with the smallest label. Using the same reasoning as in the second case, we again obtain that ψ passes at node v first. It follows that ψ leaves the common path before π enters it, proving the claim. □

Theorem 3.1 can now be derived by showing that no vehicle occupies any node later in the TreeRouting algorithm than in the routing suggested by Lemma 3.2.

Proof of Theorem 3.1. Consider a fixed routing according to Lemma 3.2, using the same vehicle priorities as TreeRouting. We will prove the following claim which implies the result: no vehicle occupies any node later in the routing obtained by TreeRouting than the routing according to Lemma 3.2. Consider a step of TreeRouting, where some vehicle π is to be routed, and assume that all vehicles with higher priorities than π are already routed by TreeRouting such that the claim holds. We show that TreeRouting will route π such that π also satisfies the claim.

Notice that π could be routed according to the same schedule as in Lemma 3.2, because of the following. In the routing according to Lemma 3.2, there is no vehicle with higher priority than π that passes any vertex v after π. By assumption, any vehicle ψ that passes through v and has higher priority than π, occupies v in TreeRouting at the same time or earlier than in the routing according to Lemma 3.2. Hence, no conflict at v between ψ and π is possible when π is routed as in the routing according to Lemma 3.2. The arrival time of π in TreeRouting is hence no later than in the routing according to Lemma 3.2. The same is true for the passage times at all other nodes, as the sequential schedule needs at least as much time to bring π to the destination as the direct schedule of Lemma 3.2. □

3.2 Hot Spot Routing

After having presented a routing algorithm for trees, we show now how it can be extended to yield a sequential routing scheme with $O(\sqrt{k})$ approximation guarantee for general graphs. It proceeds along the following three steps.

i) Selection of routes for the vehicles with guaranteed upper bounds on both route length and node congestion.
ii) Identification of busy nodes (*hot spots*) and routing of the vehicles going through hot spots with the tree approximation algorithm.
iii) Routing of the remaining vehicles by exploiting the fact that the congestion and hence the conflict potential is limited (*low congestion routing*).

Selection of routes. We will determine for each vehicle $\pi \in \Pi$ an s_π-t_π path $P_\pi \subset E$, satisfying the following properties, where we denote by $\Pi_v \subseteq \Pi$ for $v \in V$, the vehicles whose paths contain v:

i) the *congestion* $C = \max_{v \in V}\{|\Pi_v|\}$ is bounded by $O(\mathrm{OPT})$,
ii) the *dilation* D of the chosen paths, which is the length of the longest path P_π, is bounded by $O(\mathrm{OPT})$.

Notice that both, the congestion C and the dilation D are lower bounds on the minimum makespan that can be achieved with the chosen paths. The problem of finding routes with small congestion and dilation is well-known in packet routing. Using an algorithm of Srinivasan and Teo [ST97] or a recently improved version presented in [KPSW09], a collection of paths with the above properties can be found in polynomial time. We use such an algorithm as a subroutine in our routing approach.

The algorithm. We start by computing origin-destination paths $\{P_\pi\}$ with short congestion and dilation as discussed above. In a first phase, the algorithm goes through the vertices $v \in V$ in any order and checks whether there are more than $\sqrt{k}$ vehicles not routed so far whose origin-destination paths contain v. If this is the case, all those vehicles are routed on a shortest path tree rooted at v using TreeRouting. Notice that these vehicles are hence not necessarily routed along the paths $\{P_\pi\}$. In a second phase, direct sequential routing (with an arbitrary order) is applied to all vehicles not routed this far. The paths used here are the ones determined at the beginning, i.e., $\{P_\pi\}$. The algorithm is summarized below.

Algorithm HotSpot

```
Generate origin-destination paths $\{P_\pi\}$ with low congestion and dilation
Initialize $\Pi' \leftarrow \Pi$
while There exists a node $v$ with $|\Pi_v \cap \Pi'| > \sqrt{k}$ do
    Route the vehicles $\Pi_v \cap \Pi'$ on a shortest path tree with root $v$, using
    TreeRouting
    $\Pi' \leftarrow \Pi' \setminus \Pi_v$
end while
Route the remaining vehicles $\Pi'$ in an arbitrary order, applying direct se-
quential routing using the paths $\{P_\pi\}$.
```

Theorem 3.3. HotSpot *has an approximation quality of* $O(\sqrt{k} \cdot \mathrm{OPT})$.

For the proof, we need some more notation. For any graph $H = (W, F)$ with given edge lengths (variable edge lengths will be used later in Section 3.3) and two vertices $v, w \in W$, we denote by $l_H(v, w)$ the distance of a shortest path between v and w in H. For $U \subseteq F$, we denote by $H[U]$ the graph (W, U), where U inherits the edge lengths from H. $l_{H[U]}(v, w)$ therefore stands for the shortest path length from v to w in the subgraph of H induced by edges U.

Proof. The while-loop gets iterated at most $\sqrt{k}$ times, since at each iteration at least $\sqrt{k}$ of the k vehicles are routed. Consider the routing of a group of at least $\sqrt{k}$ vehicles $\Pi_v \cap \Pi'$ during the while-loop. Since each vehicle $\pi \in \Pi_v \cap \Pi'$ is routed along a shortest path tree T rooted at r, and P_π contains v, we have $l_{G[T]}(s_\pi, t_\pi) \leq |P_\pi| \leq D$. Furthermore, the number of vehicles in $\Pi_v \cap \Pi'$ is bounded by C. Thus, the algorithm TreeRouting routes all vehicles in $\Pi_v \cap \Pi'$ in time $O(C + D)$, and hence, all vehicles routed during the while loop will reach their destination in $O(\sqrt{k}(C + D)) = O(\sqrt{k} \cdot OPT)$ timesteps.

Let π be any of the remaining vehicles that are routed during the second phase of the algorithm. Consider all potential departure times for π from its origin, starting after the last vehicle of the first phase has arrived. We want to bound the total number of departure times for π that lead to conflicts due to previously scheduled vehicles during the second phase. If some departure time is not possible, then this must be due to either a node conflict or an edge conflict with another vehicle previously scheduled during the second phase. However, for every node v on the path P_π, at most $\sqrt{k}$ vehicles have previously been routed over v during the second phase. Hence, the occupation of these nodes by other vehicles blocks $O(D\sqrt{k})$ possible departure times. Furthermore, if some departure time t for π is not possible due to some edge conflict with another vehicle ψ routed during the second phase, then either there is also a node conflict for the same departure time (if π and ψ traverse the edge in the same direction), or the departure time $t+1$ corresponds to a node conflict between π and ψ (if π and ψ traverse the edge in opposite directions). Hence, the total number of departure times that are blocked by edge conflicts is bounded by the total number of departure times blocked by node conflicts which is $O(D\sqrt{k})$. We conclude that π waits at most $O(D\sqrt{k})$ timesteps at its origin before directly traveling to its destination in at most D steps. Hence, this second phase is completed in at most $O(D\sqrt{k})$ timesteps, thus leading to a total makespan bounded by $O(\sqrt{k}(C+D))$ and proving the claim. $\square$

3.3 Low-Stretch Routing

In this section we present a second approach how to extend the tree routing scheme from Section 3.1 to general graphs. It is an an efficient randomized method with makespan guarantee $O(\log^3 k)\mathrm{OPT} + k$. This guarantee is improved in the multiplicative factor compared to HotSpot at the cost of the additional additive term $+k$.

The approach uses tree embeddings to extend the routing algorithm designed for tree topologies to arbitrary graph topologies. A direct application of tree embedding would here only lead to an approximation guarantee that is polylogarithmic in the number of vertices, whereas we are interested in approximation guarantees independent of the size of the graph. To achieve this goal we will determine a routing by applying tree embedding techniques to a compacted version of the graph G with size of order $O(k^2)$.

The high level idea is to find a collection of $O(\text{polylog}(k))$ trees in G such that for each vehicle $\pi \in \Pi$, there exists a tree T in the collection such that the distance of the s_π-t_π path in T is at most an $O(\text{polylog}(k))$-factor larger than the distance between s_π and t_π in G. Every vehicle π is then assigned to a tree with a short s_π-t_π distance. We then go through the collection of trees in any order and sequentially route first all vehicles assigned to the first tree, then all that are assigned to the second tree and so on. Each group of vehicles that is assigned to the same tree is routed using the tree routing algorithm TreeRouting.

We will apply the following results about low-stretch trees of Abraham et al. [ABN08] to a compacted version of G to find a good collection of spanning trees.[2] Recall that $l_{H[U]}(v, w)$ denotes the shortest path length from v to w in the subgraph of H induced by edge set U.

Theorem 3.4 ([ABN08])**.** *For any edge-weighted graph $H = (W, F)$, one can draw in polynomial time a spanning tree T of H out of a distribution such that for any $v, w \in W$, the expected stretch is bounded by $O(\log^2 |W|)$, i.e.,*

$$\mathbf{E}\left[l_{H[T]}(v, w)/l_H(v, w)\right] = O\left(\log^2 |W|\right).$$

We transform the unit length network $G = (V, E)$ into a graph $H = (W, F)$ of size $O(k^2)$ with non-negative edge lengths, such that both graphs have the same origin-destination distances for each vehicle. For this purpose, we first compute for each vehicle $\pi \in \Pi$ a shortest path $P_\pi \subseteq E$. To do this, we temporarily perturb the unit edge lengths slightly such that the shortest paths are unique.[3]

Let $W \subseteq V$ be the set of all vertices that are either terminals, or have at least three adjacent edges in $\cup_{\pi\in\Pi} P_\pi$. The graph $H = (W, F)$ is obtained from G by applying the following operations. See Figure 3.2 for an illustration.

i) Delete all edges and nodes which are not part of any path P_π.

ii) Every path $P \subseteq G$ between two nodes $v, w \in W$, without any other nodes of W on the path, is replaced by an edge between v and w of length $|P|$.

Notice that H can be interpreted as a compact graph version of $(V, \cup_{\pi\in\Pi} P_\pi)$. Every edge of H corresponds to a path in G (possibly of length one). More generally, any subset of edges $U \subseteq F$ can be mapped to corresponding edges in G. H has the following properties.

[2] In [SZ11] we included a reference to the earlier result of Dhamdhere et al. in [DGR06]. This result, however, was withdrawn according to a footnote in [ABN08]. In the latter paper, Abraham et al. subsequently presented an even stronger result.

[3] Such a perturbation can for example be performed by fixing an arbitrary ordering of the edges $E = \{e_1, \ldots, e_m\}$, and assigning the lengths $\ell(e_i) = 1 + \epsilon^i$ for $i \in \{1, \ldots, m\}$, for any $\epsilon \leq \frac{1}{2}$. This perturbation can also be performed purely symbolically.

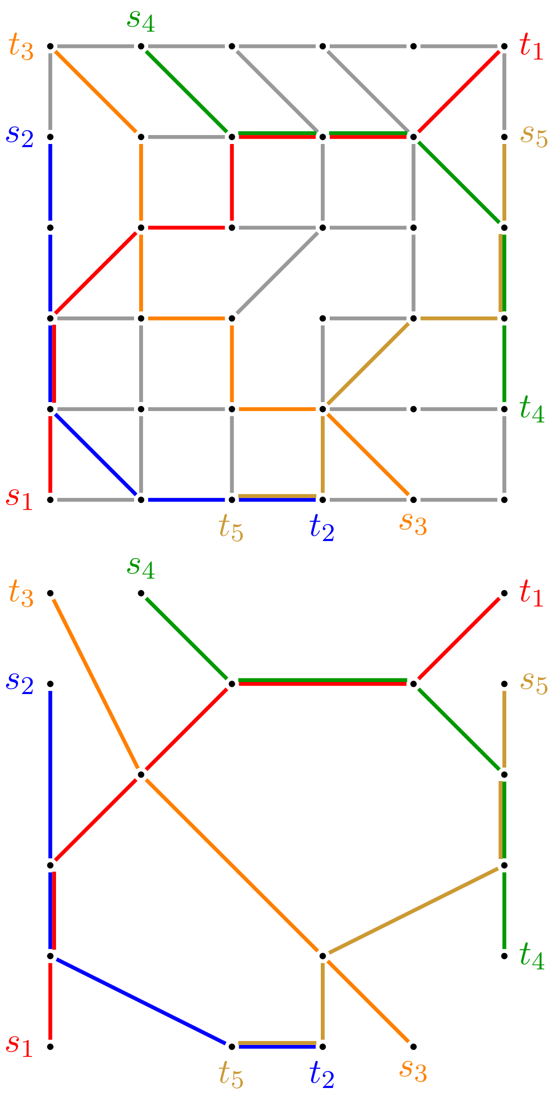

Figure 3.2: Transformation of original graph G (*above*) into compact graph H (*below*). The paths $\{P_\pi\}_{\pi \in \Pi}$ are drawn as colored lines. The compact graph is obtained by removal of all nodes which do not correspond to a terminal or a junction of paths.

Lemma 3.5. *The size of $H = (W, F)$ is bounded by $|W| = O(k^2)$, and for each vehicle, the origin-destination distance in H is the same as in G.*

Proof. The claim about origin-destination distances holds since H contains a compacted version of the path P_π for every vehicle $\pi \in \Pi$.

To bound the size of H, consider the graph $G' = (V, \cup_{\pi\in\Pi} P_\pi)$. Since H is obtained from G' by eliminating all degree two vertices that are not terminals, each non-terminal vertex in H is a vertex of degree at least three in G'. To prove the claim, it hence suffices to show that G' has at most $O(k^2)$ vertices of degree ≥ 3. If a non-terminal vertex $v \in V$ is of degree at least three in G', then there exist at least two vehicles $\pi, \psi \in \Pi$ such that $|\delta_G(v) \cap (P_\pi \cup P_\psi)| \geq 3$, where $\delta_G(v)$ are the edges adjacent to v in G. We call such a node a *junction* of the two vehicles π, ψ. Observe that for any two vehicles $\pi, \psi \in \Pi$, there are at most two junctions of π and ψ: the paths P_π and P_ψ are unique shortest paths in G w.r.t. the perturbed unit lengths; hence $P_\pi \cap P_\psi$ is a path in G, implying that only the two endpoints of P can be junctions of π and ψ. Since each of the $\binom{k}{2}$ unordered pairs of vehicles leads to at most two junctions, the total number of junctions is bounded by $k(k-1)$, which implies that G' has at most $O(k^2)$ vertices of degree ≥ 3. $\square$

The following lemma now follows by standard techniques.

Lemma 3.6. *Let $p = 2\log(k)$, and let $U_1, \ldots, U_p \subseteq F$ be random spanning trees of H obtained by applying Theorem 3.4. Let $T_1, \ldots, T_p$ be the spanning trees in G that correspond to $U_1, \ldots, U_p$. Then, with probability at least $1 - 1/k$, we have that for every vehicle $\pi \in \Pi$, there exists a tree $T \in \{T_1, \ldots, T_p\}$ such that*

$$l_{G[T]}(s_\pi, t_\pi)/l_G(s_\pi, t_\pi) = O(\log^2 k).$$

Proof. By Theorem 3.4 and using that the size of H is bounded by $O(k^2)$ (Lemma 3.5), we have that for any $i \in \{1, \ldots, p\}$ and $\pi \in \Pi$,

$$\mathbf{E}\left[\frac{l_{H[U_i]}(s_\pi, t_\pi)}{l_H(s_\pi, t_\pi)}\right] \leq c \log^2 k,$$

for some constant $c > 0$. By construction of H we have $l_H(s_\pi, t_\pi) = l_G(s_\pi, t_\pi)$ and $l_{H[U_i]}(s_\pi, t_\pi) = l_{G[T_i]}(s_\pi, t_\pi)$. Using Markov's inequality, we get

$$\Pr\left[\frac{l_{G[T_i]}(s_\pi, t_\pi)}{l_G(s_\pi, t_\pi)} \geq ec \log^2 k\right] \leq \frac{1}{e}.$$

Since the random trees $T_1, \ldots, T_p$ are drawn independently, we obtain that the probability that there is at least one tree T_i in the collection such that $\frac{l_{G[T_i]}(s_\pi, t_\pi)}{l_G(s_\pi, t_\pi)} \leq ec \log^2 k = O(\log^2 k)$, is at least $1 - (\frac{1}{e})^p \geq 1 - \frac{1}{k^2}$. Using a union bound over all vehicles, the result is obtained. $\square$

The algorithm. The algorithm works as follows. Using Lemma 3.6 (repeatedly if necessary), we obtain in expected polynomial time a collection of spanning trees $T_1, \ldots, T_p$ with $p = O(\log(k))$ such that we can assign each vehicle π to a tree $T_{i(\pi)}$ satisfying $l_{G[T_{i(\pi)}]}(s_\pi, t_\pi) = O(\log^2 k \cdot l_G(s_\pi, t_\pi))$. Let k_j denote the number of vehicles assigned to T_j. For each tree T_j in the collection, the TreeRouting algorithm can be used to route all vehicles that are assigned to T_j in time bounded by

$$4 \max\{l_{G[T_j]}(s_\pi, t_\pi) \mid \pi \in \Pi, i(\pi) = j\} + k_j = O(\log^2 k)L + k_j,$$

which follows by the guarantee on the stretch provided by Lemma 3.6. Finally, routing first all vehicles assigned to T_1, then, as soon as all those vehicles arrived at their destinations, route all vehicles assigned to T_2 and so on, thus leads to an algorithm with expected polynomial running time and the claimed bound on the makespan given by

$$O(\log^2 k)pL + \sum_{i=1}^{p} k_j = O(\log^3 k)L + k.$$

Chapter 4

Conclusion

We have investigated conflict-free routing on bidirectional networks, a problem with relevance for various applications in the transportation sector.

It was observed earlier that the sequential routing scheme often performs well when there is a number of alternative route choices of almost shortest-path-length, such as it is the case for grid graphs [Ste08]. Large delays caused by opposing traffic can often be avoided by choosing the route with least delays out of the alternatives. More challenging are networks with a lack of alternative routes. For the special case of tree networks, only a single path exists between origin an destination and the delays can be influenced only by the choice of the priority sequence. We now have presented a method for choosing the sequence with a makespan guarantee of $O(\mathrm{OPT}) + k$.

Combining this result with ideas from packet routing, we were able to extend the tree algorithm to obtain an $O(\sqrt{k})$-approximation for general graphs. It is the first efficient algorithm with sublinear approximation guarantee in this generality. Additionally, a randomized algorithm leading to a makespan of $O(\mathrm{polylog}(k)) \cdot \mathrm{OPT} + k$ was presented that relies on tree embedding techniques. Applied to a compacted version of the graph one obtains an approximation guarantee independent of the graph size.

Further, we have presented new hardness results for the problem. We have shown that the conflict-free vehicle routing with minimal makespan is hard to find even on a path. For a problem variant with directed edges, we have shown that the variant is hard to solve on tree networks.

Summarizing, we have presented a number of new theoretical results in a field which has found only limited attention in the past from the combinatorial optimization community. This is somewhat surprising in view of the large application area that it relates to and also given the considerable attention that the neighboring field of packet routing could attract. We hope that our results contribute to an improved theoretical understanding of conflict-free vehicle routing.

There is potential for further improvement on the quality guarantees for routing algorithms. A large gap remains between the approximation guarantees and the hardness results we could prove. For example, it remains open whether a constant factor approximation exists, such as it is known for packet routing [LMR94]. Also there remains a large gap between the theoretical bounds and the performance desirable for applications. While the results in this study make a statement on how approximation bounds and computation times of the algorithms scale with the size of the instances, it would be interesting to understand how this quality versus computation trade-off looks for concrete realistic-scale instances.

Part II

Conflict-free Vehicle Routing in Personal Rapid Transit

Chapter 5

Introduction

5.1 Personal Rapid Transit

Personal Rapid Transit (PRT) is a rather novel mode of public transportation, where small automated vehicles, called *pods*, transport passengers on demand on a unidirectional track network. Passengers can choose the destination of the ride when entering the pod at a station. Then a central automatic system guides the pod through the network to the desired destination. See Figure 5.1 for visualizations of the PRT concept.

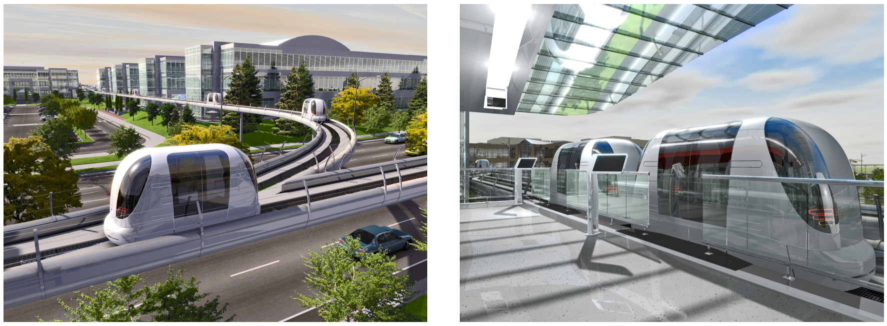

Figure 5.1: Visualizations of Personal Rapid Transit. *Left:* Track junction with pods. *Right:* PRT station. Both renderings are reproduced here by courtesy of Ultra [ULT11].

The expected benefit from a PRT system for passengers is convenient non-stop mobility that is predictable and without some of the drawbacks of mass transportation. A passenger can travel alone if desired, can use the time of the ride for working, does not depend on a timetable and does not need to change lines to reach his destination. PRT is intended to complement public transportation systems, such as subways and trains, by offering a quick means of transportation for short distances and for accessing the mass transportation hubs, thus addressing the well-known *last mile problem.* PRT reduces the time spent for walking, waiting and transfers which are often high when using conventional public transit systems for short trips [ATR03, Low03]. The goal of PRT is hence to improve public transportation and increase its share, also by providing an alternative to the automobile.

In addition, PRT is expected to provide considerable benefits to the public. The electricity powered system is claimed to be highly energy-efficient as it runs on demand, is light-weight and can profit from the high efficiency factor of electric propulsion [ULT11]. It has the potential to reduce greenhouse gas emissions, depending on the source of the electric energy. Citizens with access to PRT do not only profit from fast and on-demand transportation, they also enjoy a high quality of life as PRT operation emits little noise and pollutants. A PRT system can contribute to the vision of a *walkable city* with a high comfort of living, by reducing or banning car traffic from the city and thereby gaining space for pedestrians and green parks [KKH08]. Besides, a non-congested and hence predictable transportation system can be an important economic factor and increase accessibility and the prosperity of certain areas. PRT has been found to be cost-effective in a number of case studies [BT05, TA03]. An important factor here is that the tracks are light-weight (compared to railway or subway tracks) and cause low installation and infrastructure costs [CH07]. Track layout is adaptive to existing infrastructure, can be built at ground level or elevated, and stations can be integrated into buildings to guarantee direct access.

Even though the general idea of PRT is several decades old, only recently first projects were deployed and many more entered planning and decision stages. For early sources on PRT, we refer to the books of Fichter [Fic64] and Irving et al. [IBOB77]. Already at the time, governmental institutions have shown interest in PRT. In 1968, PRT appears in a study of the United States Department of Housing and Urban Development as a fast, safe and non-polluting alternative for city transportation [HUD68].

PRT has then disappeared from most agendas for several decades. Since recently, alternative transportation modes have been reevaluated in view of the increasing challenges of congested roads, greenhouse gas (GHG) emissions and the limited supply of oil. To emphasize the size of the challenge, we quote the White Paper of the European Commission for Mobility and Transport [EU11].

> Commission analysis shows that while deeper cuts can be achieved in other sectors of the economy, a reduction of at least 60% of

> GHGs by 2050 with respect to 1990 is required from the transport sector, which is a significant and still growing source of GHGs. By 2030, the goal for transport will be to reduce GHG emissions to around 20% below their 2008 level.

A second recent study commissioned by the European Commission Directorate [WH10] lists PRT as an energy-efficient mode for local transportation, particularly well suited as a local circulator at airports, shopping malls, university campuses, tourist attractions and business parks. The Swedish Ministry of Enterprise Energy and Communications has commissioned a study on the potential of introducing PRT in Swedish cities. The following quote is from their 2009 report [SWE09].

> Podcars are a possible solution to certain transport problems that are difficult to manage with existing traffic solutions. They can be a sustainable alternative to private car transport. However, podcars have only been tested on a small scale and there are no commercial systems currently in operation. Studies do show, however, that this type of transport system can be profitable both from a societal and firm perspective. An appropriate way of determining how well podcars could function would be to test one or several pioneer systems under real-life operation. Full-scale pioneer tracks would give decision makers, planners and suppliers the necessary experience to further develop podcar technology.

It accounts for the frequent opinion that PRT is on one hand a promising transportation alternative for solving the increasingly urgent congestion and pollution issues. On the other hand, decision makers are cautious as PRT has not yet shown whether it will be able to keep its promises in terms of attractivity, performance, reliability and cost-effectiveness. The first larger PRT systems will have to prove that they can fulfill the high expectations that have been raised. The further development of the transportation mode will depend heavily on the experiences made with pioneering systems.

A comprehensive report [CH07] commissioned by the state of New Jersey, United States, lists the following potential applications of PRT.

- Areas with high demand for local circulation
- Areas with the potential to extend the reach of nearby conventional public transportation
- Areas with congested local circulation and constrained parking

Furthermore, they see the opportunity for PRT networks to grow. We cite from [CH07].

> Initial PRT system implementations could potentially be viable in the areas previously described such as regional centers, campuses, congested locations and as extensions to conventional public transportation system station. PRT could also be expected to be viable as a connector of these initial systems, providing an integrated public transportation network across a region, eliminating the need to transfer between modes or within the mode. As a scalable network system, PRT could initially be deployed to support the locations with the highest need and then expand to connect these initial deployments as demand and economic conditions allow.

Currently, as of the end of year 2011, there are two PRT systems in operation, both of small size albeit with option for future expansion. One is located in the zero-emission city of Masdar, United Arab Emirates, connecting two stations [2ge11]. The second is built at London's Heathrow Airport, replacing a bus line between Terminal 5 and a car parking. It currently consists of 3 stations and 21 pods on 3.8 kilometers of track [ULT11].

A number of projects are in planning or decision phases. Amritsar, India, has approved a PRT system for transferring visitors between bus and railway terminals and the Golden Temple. The planned network consists of 3.3 km track and 200 pods transferring up to 100'000 passengers a day [ULT11]. In San Jose, USA, the plan is to connect the airport to a light rail station, office parks, a university and sports facilities [SAN09]. In Ithaca, USA, a feasibility study for a PRT network connecting downtown with the college campuses has been commissioned [ITH10]. The latter proposed system includes 26 stations, 9 miles of track and 350 pods. In Daventry, UK, PRT is the leading candidate technology for a new transportation system enabling the cities planned expansion [SKM08]. Notable is also a project for the city of Gurgaon, India, where a city-wide PRT system consisting of 143 stations, 105 kilometers of track and 3000 pods is being discussed [Suk11, Cha11].

PRT systems are promoted and produced by a number of distributors, each having its unique features. A consensus of properties common to all PRT systems was established by the Advanced Transit Association (ATRA) [ATR89]. We reproduce the recently republished definition from [ATR11]:

1. Direct origin-to-destination service with no need to transfer or stop at intermediate stations.

2. Small vehicles available for the exclusive use of an individual or small group traveling together by choice.
3. Service available on demand by the user rather than on fixed schedules.
4. Fully automated vehicles (no human drivers) which can be available for use 24 hours a day, 7 days a week.
5. Vehicles captive to a guideway that is reserved for their exclusive use.
6. Small (narrow and light) guideways are usually elevated but also can be at or near ground level or underground.
7. Vehicles able to use all guideways and stations on a fully connected PRT network.

5.2 Control Challenges in PRT

In the prospect of larger PRT systems, the question naturally arises on how much capacity such a system can have, how the available capacity is best utilized, and finally, how a system needs to be designed such that capacity is sufficient for making PRT an efficient and attractive transportation scheme.

The fact that all operations are controlled centrally is a major advantage, e.g. compared to road traffic, and opens up exciting possibilities for optimized resource utilization. However, optimized central control comes with two major challenges. First, it involves combinatorial optimization problems that are often computationally hard. Second, control decisions for PRT must be taken in real-time, to avoid that the system stands idle while waiting for instructions.

We list in the following a number of control tasks in PRT together with references to research publications for each. Each of the tasks also represents a potential capacity bottleneck of a PRT system, that can lead to waiting passengers and hence a decreased level of service in case the system layout does not contain enough resources and/or the available resources are not used to full potential.

Routing Central control guides the pods through the track network to the respective destinations. As throughput of each line segment and junction is limited, the challenge is to find routes and schedules for each of the pods such that the network resources are used efficiently and passengers reach their destinations as early as possible. There may be several alternative routes leading to the destination. The best route choice is dependent on the traffic on the network. In particular, the shortest route may not always be the optimal choice, e.g., when a longer route is less congested and hence available at an earlier time. For a review on PRT routing literature we refer the reader to the next section.

Empty vehicle redistribution As transit demand between stations is usually not balanced, idle pods need to be redistributed. The objective is to minimize the time passengers spend waiting for pods. Empty vehicle redistribution can be done in a reactive way (idle pods are moved only on request) or in a proactive way (idle pods are moved in anticipation of future requests). Depending on the transit demand, additional pods need to be taken out of the vehicle depot or excess pods need to be placed into storage. For literature on empty vehicle redistribution, we refer to [LMHW10, LM11, And03, ZJM09].

Station handling PRT stations consist of several berths for boarding and unboarding of passengers. A parking queue with idle pods allows to respond quickly to new transit requests. A challenge is to manage station capacity such that arriving pods have space for docking but also enough empty pods are ready for new demand.

Ride sharing Some sources suggest systems for grouping passengers with the same destination and to promote ride sharing (comparable to advanced elevator systems), and thereby increasing the overall passenger throughput of the system [Joh05, LMHD09, And09].

Battery recharging and maintenance Maintenance of the pods and recharging the batteries (for battery-driven systems) is done at a special maintenance station. Periodical stops need to be planned for each of the pods, after a number of kilometers traveled or when a certain battery level is reached. Maintenance management ensures that enough functional vehicles are ready at every point in time. For a discussion of maintenance topics, we refer to [MS11].

While each of the mentioned problems is in itself challenging, an additional difficulty arises from the fact that demand is not known in advance and that all decisions need to be made in real-time. Furthermore, the solutions of each of the problems has effect on the others. A number of simulation studies ([MS11, CG11, THA$^+$07] and references therein) have been performed to investigate PRT systems as a whole.

Other research focuses on one of the challenges in particular. Such research helps to understand one of the components in depth by developing specialized algorithms and by deriving theoretical capacity limits. Such research, even if making simplifying assumptions about the interaction with the remaining system, can in the end contribute to improved overall PRT solutions. Such an approach was followed by Lees-Miller [LM11] who investigated empty vehicle redistribution in depth. Also the present work focuses on one of the mentioned challenges in particular, which is the congestion-free routing in a PRT system.

5.3 Routing Literature

PRT routing literature differentiates between synchronous and asynchronous routing approaches. In *synchronous* approaches, the entire route is planned ahead before pods leave the origin station. After a routing for a pod is planned, the necessary track resources get reserved, thereby guaranteeing a conflict-free operation. Such approaches are sometimes also called *clear-path control*. The routings are computed in a central unit which also maintains the reservation tables for the track resources.

On the other hand, *asynchronous* approaches take their routing decisions based on local information. The name stems from the idea that the pods do not need to have a synchronized clock. There is no coordinated plan and each pod takes its own routing decisions. Pods only communicate with a local controller responsible for a junction or a track region. Routing decisions hence need to be taken based on local information such as the distance to the vehicle ahead. Such approaches have been promoted in [KM75, IBOB77, And98].

The synchronous routing algorithms suggested within the PRT community range back to Wade [Wad73]. Their approach finds the earliest conflict-free departure time using a shortest-path algorithm, assuming that the pods travel at nominal speed along some (shortest) route. Somewhat later, Rubin [Rub75] suggested an algorithm also taking alternative routes into account. Irving et al. [IBOB77] introduced a load balancing algorithm which chooses routes according to shortest paths with a penalty for congested roads. After these early papers, little has been published on synchronous PRT routing, as it was considered inflexible, suboptimal and computationally demanding. The reservations against synchronous control could be refuted in the meantime, see e.g. [Xit08]. In particular, the data transfer is manageable with current communication technology. The provider of the pioneering PRT system at Heathrow airport uses synchronous routing with a central controller [ULT09]. The advantage of synchronous control is that global information can be used to employ sophisticated algorithms for coordinated routing.

Related routing literature also comes from neighboring research fields such as *Automated Guided Vehicle (AGV)* routing or packet routing. The arguably most common routing paradigm for online routing problems of similar flavors is a sequential scheme, which we call *sequential routing*. Here, whenever a new request appears, a quickest way to fulfill this request is determined, without creating conflicts with already fixed routes for earlier requests. Once a routing for a particular request is fixed, it will not be changed later. All synchronous routing schemes for PRT discussed above are of this type. Sequential routing schemes proposed for AGV systems can be found in [KT91, MKGS05]. They allow for waiting in transit, which means that vehicles can be delayed (slowed down from the nominal speed) in case this leads to earlier arrival time. Also, they allow for non-shortest routes, again if the detour delay can be compensated by less departure and/or transit delays. Finding the best route and schedule for a pod in these schemes corresponds to performing a variant of a shortest-path algorithm. Sequential routings can hence be computed efficiently and perform well in a variety of settings [Ste08, RVVN90].

There has been some work on AGV routing approaches deciding on all requests concurrently. This is in contrast to the sequential schemes, where one request is routed after the other. Stenzel [Ste08] suggests a concurrent approach for finding routes such that the congestion is balanced. Oellrich [Oel08] uses an integer multicommodity flow formulation on a time-expanded graph to compute concurrent routes and schedules. In the latter work, it is assumed that all requests are known from the beginning.

5.4 Outline

In this thesis, we focus on comparing and improving routing approaches as well as on understanding the capacity of a PRT track network. We look at routing in PRT from a viewpoint in combinatorial optimization and investigate the benefit that can be obtained by applying methods from that field. Central to this analysis will be the trade-off of performance gain versus computational effort. The question is whether more sophisticated routing schemes can outperform the known schemes significantly. Furthermore, we investigate what capacity limitations PRT systems can have with respect to the throughput of the track network.

We introduce an *adaptive* routing approach that reoptimizes the routing for all open requests at each timestep. This is in contrast to the known approaches where the routing for the pods is fixed at time of appearance. The policy of never changing a fixed route in the future is a restriction that has its price. Pods routed later might have to take large detours, even though the slight rerouting of previously routed pods could free the desired tracks for new requests. In this thesis, we are interested in understanding and addressing this potential drawback. The presented adaptive approach is based on a fractional multicommodity flow formulation and takes all open requests into account concurrently. It assigns routes and priorities with the objective of minimizing the average delay. This is in contrast to the sequential approaches in which each request chooses its routing in a greedy way. As the focus is on advanced coordination of the pods, we assume the existence of a central dispatching unit with knowledge of and influence to all pods on the system.

In Chapter 6 of this thesis we introduce the model framework. It specifies and explains the view on PRT systems underlying the following chapters. The focus will be clearly on the routing aspects of PRT while simplifying other aspects thereby keeping the model concise.

In Chapter 7 we discuss a measure for network capacity, the maximal throughput that can be achieved on a given track network. This capacity depends on a number of parameters such as the travel velocity and the safety distance between pods. However, these parameters are system-specific and their discussion is outside the scope of this thesis. We will therefore suggest a parameter-independent measure for network capacity which is relative to the capacity of a single track line.

Chapter 8 introduces a number of tools and preliminary observations helpful for the further discussions of routing algorithms. In Section 8.1, routing feasibility and deadlocks are discussed. In Section 8.2, time-expanded graphs are introduced as a way to account for the time dimension in routing. A path in the time-expanded graph contains at the same time information about route choice as well as about the schedule followed along the route. On the basis of the time expansions, PRT routing is reformulated as an integer multicommodity flow over time problem in Section 8.3.

The natural question arises why one does not solve the integer multicommodity flow problem from Section 8.3, which is a standard problem in combinatorial optimization, to optimality? The answer is given in Chapter 9, where it is shown that the optimal solution of the routing problem is NP-hard to find.

In Chapter 10 we discuss the algorithms which will be compared in the computational analysis chapter. The first algorithm (Section 10.1) is the standard sequential algorithm as it is common in AGV routing. To our knowledge, this algorithm or a variant thereof is also implemented by PRT providers and several of the available PRT system simulators. The second algorithm (Section 10.2) is a representative of the class of asynchronous schemes. It uses no planning ahead and coordination between pods is reduced to a minimum. In Section 10.3 we then present the new concurrent and adaptive algorithm, which is based on rounding of the multicommodity flow relaxation.

Chapter 11 presents the computational analysis. We use two base scenarios, one artificial and one received from a PRT system provider. The chapter contains a series of computational experiments which help to understand the strengths and weaknesses of each of the routing schemes.

In Chapter 12 we will conclude with a summary and with lessons that can be learned from this project for the design of new PRT systems.

Chapter 6

Routing Model

We derive in this chapter the formal model framework which will be the basis for the considerations on PRT systems in the following chapters. Aspect by aspect, we discuss the modelling of the PRT network (Section 6.1) and the pods and travel requests (6.2), the time discretization in our model (6.3) and finally the modelling aspects related to the online nature of PRT (Sections 6.4 - 6.6). At the end of the chapter, we give a compact version of the problem formulation in Section 6.7.

6.1 Network

A PRT track network typically consists of a number of interconnected directed loops. Directed means that the tracks are one-way, each having a designated driving direction. The pods run on the track network with a certain nominal speed corresponding to the maximum allowed velocity. This velocity can vary for different sections of the track. For safety, the pods need to keep a minimal temporal separation, the *headway time*, such that breaking is possible even in unexpected emergency situations. We assume the headway time to be given and is the same everywhere on the network. The minimal safety distance, the *headway distance*, then depends on the respective speed.

We use a discretized representation of the track network. For this purpose, the tracks are divided into segments where the length of one segment corresponds to the headway distance. The partitioning is done such that the speed limit is constant inside each segment and such that junctions are located on the border between segments. We assume that a partitioning of the tracks with the above properties, which is a good enough approximation to the reality, can be found. Note that, in such a partitioning, the time needed to traverse one segment at nominal speed equals one headway time.

The discretized network can be represented by a directed graph $G = (V, A)$

with node set V and arc set A. The nodes represent track segments and the arcs define their neighboring relations. The orientation of the arcs defines the traffic direction. We assume that G is strongly connected, i.e. that any node can be reached from any other by a directed path. Furthermore, we require that G is simple, i.e. there is at most one arc between any pair of nodes (no parallel arcs and no cycles of length two). We use $n = |V|$ and $m = |A|$ to refer to the sizes of the node and arc sets in the network graph.

Stations are located along the tracks, allowing passengers to access the transit system. PRT stations are designed such that parked pods do not disturb the pods in transit. Pods which are in the station for loading or unloading passengers, or waiting for new demand, are located on parking spaces inside the station. Pods in transit pass the station on a bypass track. Nodes corresponding to track segments with a station are called *terminal nodes.*

6.2 Requests and Pods

PRT requests are of one of the following two types.

Passenger requests are initiated by passengers arriving at a station with the intention of using PRT for transit. They choose a destination before boarding.

Redistribution requests These requests are initiated by the vehicle redistribution unit, which sends empty pods to stations with a (expected) shortage.

As soon as a request is received, a central automated routing controller integrates the request into the routing plan. Such a request can be issued either by a passenger or by the vehicle redistribution unit. For the routing, we do not make a difference of which of the two types the request is. Furthermore, it is assumed that all pods are identical and that no pod-specific information is needed for planning the trips. Pods are identified by the request they are executing and requests correspond to the pods they are served by. We will therefore use the notions of pods and requests interchangeably.

We make a rather strong assumption about the availability of empty pods. Whenever a new request appears, we assume that there is an empty pod immediately available. Differently stated, we assume that the vehicle redistribution unit manages to avoid shortages. The main reason for this assumption is that we are primarily interested in the problem of finding routes between origin-destination pairs. The above assumption allows us to isolate this question from other challenges interesting in their own right, such as the handling of a possible shortage of pods, or the navigation of spare pods to places with excess demand.

The necessary information for the controller are the origin and destination of the requests. This is cast in the following notation. Let us denote the set of all requests by Π. Each request $\pi \in \Pi$ has an origin node $s_\pi \in V$ and a destination node $t_\pi \in V \setminus \{s_\pi\}$.

6.3 Discrete Dynamics and Conflict Notion

Let time be subdivided into discrete time units, such that each time unit has the length of the given headway time. This choice is convenient as one time unit is needed to cross one track segment or, stated in terms of the graph representation of the network, to jump from one node to the next one. Each pod is located on a node at each timestep. Between the timesteps, the pods either move along an arc (in arc direction) to a neighboring node or wait at the current node.

We introduce the following notions. A *route* for a pod π is a directed path through G from s_π to t_π. Routes can be non-simple paths (in case of driving loops). A *schedule* defines the timetable by which a pod proceeds along its route. We will later introduce *dynamic paths* to encode both route and schedule simultaneously by specifying paths through a time-expanded network. The notion of a *routing* is used for the assignment of both route and schedule to one or several requests in Π.

A routing is said to be *conflict-free* if no two requests are in conflict. A conflict occurs if two pods occupy the same node at the same time. For directed graphs this implies that the simultaneous passage through an arc is also forbidden. As a relaxation of the conflict definition at terminal nodes, we assume that pods can only be conflicting while in transit, i.e., a pod cannot cause a conflict before its departure time and after its arrival time. We call this the *parking assumption.* The motivation for the parking assumption stems from the fact that in typical PRT settings, the stations are on separate side tracks, and thus do not interfere with the remaining network. As mentioned before, we consider capacity handling inside each station to be a separate traffic management problem.

We would like to emphasize that the model for the dynamics is not restrictive despite its discretisation and its simplicity. For this purpose, we remark that vehicle trajectories stemming from a model with continuous time and space, under reasonable assumptions, can be converted into a routing corresponding to our model. To underline this, we sketch the conversion in Figure 6.1.

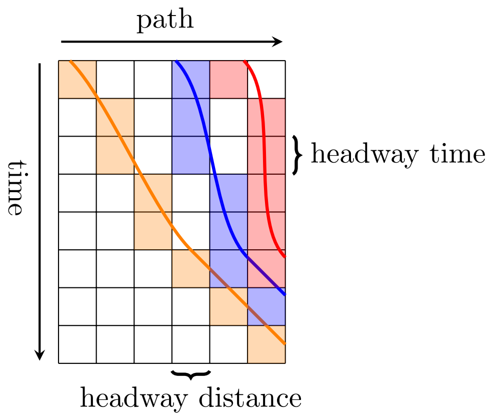

Figure 6.1: The figure shows how continuous trajectories can be transformed into a discrete routing. The colored lines represent continuous trajectories in time-space along a track line. They are separated horizontally by at least the required headway distance. The grid represents the discretisation of both space and time. For each cell row, the first cell touched by a trajectory is colored with the respective color. The sequence of cells with the same color now corresponds to a discrete and conflict-free routing according to our model.

In the other direction, a routing resulting from the discretized model can in principle also be converted into trajectories in continuous time and space with the same travel time. An important issue here is to find smooth trajectories with accelerations adequate for the transport of persons. When a pod is scheduled to wait for a number of time units, this preferably translates into a reduced velocity over several track segments rather than into a full stop. The concrete choice of trajectories, however, depends on system-specific parameters.

6.4 Online Routing

Routing in PRT is an online optimization problem. Decisions need to be taken in real-time and without knowledge of future events. We describe in this section what routing decision need to be taken at which time, and what information is available.

In order to capture the different time points in which passengers would like to use PRT, we attribute a release time $\tau_\pi \in \{0, 1, \dots\}$ to each request $\pi \in \Pi$. It reflects the time when the pod is loaded with passengers and ready to start the journey through the network. Pod π can leave its origin node only after timestep τ_π. This is also the time when the existence of pod π gets revealed to the controller, together with the information about its origin and destination.

Whenever new requests get released, the controller needs to integrate the new demand into the routing plan and hence take new routing decisions. When doing this, the online algorithm has the following information available.

Network graph. The network topology is available in the form of graph $G = (V, A)$.

Current positions and destinations of all open requests. A request is *open* if it has been released but has not yet arrived at the destination. The current positions and destinations are necessary information for routing. In addition, for pods that are currently located at the origin, the information is needed whether the pod is still parked or in transit. The status is technically relevant as only pods in transit are relevant for conflict-avoidance because of the parking assumption.

***(Optional)* Last computed routing.** This is an optional input and will be useful for some of the algorithms. In case a (partial) routing has already been computed at an earlier execution of the algorithm, it can be used as a warm start, reducing the computation load for current execution.

A second optional input which can in principle be useful to enhance routing algorithms is information on expected future traffic. In case a probability distribution of the expected future request occurrences is available, this forecast information can be used to take into account expected future effects of current decisions. The routing algorithms presented in this work do not make use of such information. Nonetheless will we discuss an extension for forecast information to one of the algorithms (see Section 10.3.3).

The output of the online algorithm are routing decisions for the next timestep for all open requests: the decision can be to either wait on the current node or to move to a neighboring node. This is the minimal information needed by the pods to continue their journeys. It is assumed in this setting that the algorithms are real-time, i.e. that the computations can be performed almost instantaneously. We will discuss the validity of this assumption in Section 11.1.3 of the computational analysis.

Optional additional output is useful if the routing algorithm plans the routing for more than one timestep into the future. The additional information can then be used at the next round by using the old solution as a starting point and hence reducing the workload. Our model allows for routing plans to be adapted until they are implemented. In other words, only the part of the routing in the past is fixed.

PRT operation is intended to be non-stop, 24 hours a day and throughout the year. This corresponds in principle to an infinite horizon problem, with infinitely many requests and infinitely many executions of the online algorithm. Here, we restrict the time horizon in order to keep things finite. A natural choice is a horizon corresponding to an operation time of 24 hours. Setting start and end of the operation period into the late night hours, when one expects little traffic in the system and the effects introduced by the horizon boundaries are small, is a natural choice.

Technically, two different time horizons are needed. T is called the *release horizon*. It is distinguished from the *operation horizon* T'. New requests are accepted up to the release horizon. The operation horizon $T' > T$ needs to be large enough such that all requests released within $[0, T]$ can be finished until time T' (The shortcut notation $[\alpha, \beta]$ represents the integer interval $\{\alpha, \alpha + 1, \dots, \beta\}$ and will be used throughout the thesis).

6.5 Objective Function

In order to optimize routings and to compare the performance of routing algorithms, we need to evaluate the quality of a solution. We first introduce appropriate quality measures before discussing the online optimization in the next section. For this purpose, let Π be the set of all requests that are released at timesteps $[0, T]$.

Arrival time The arrival time arr_π naturally refers to the timestep in which request π reaches its destination node.

Travel time The travel time tra_π denotes the length of the time span between release and arrival time, i.e.

$$\mathrm{tra}_\pi = \mathrm{arr}_\pi - \tau_\pi.$$

The travel time hence includes driving times and waiting both at the origin and during transit.

Delay Finally, the delay del_π refers to the difference between the arrival time in the given routing and the arrival time in the best case, i.e. when request π departs immediately at τ_π and can travel the shortest path of length l_π without any waiting. The delay is derived by the following formula.

$$\mathrm{del}_\pi = \mathrm{arr}_\pi - (\tau_\pi + l_\pi) = \mathrm{tra}_\pi - l_\pi$$

Delays can be divided into the three categories *departure delay*, *transit delay* and *detour delay*. Departure delays occur when a pod cannot depart from the origin immediately at the release time. Transit delays occur when a pod needs to wait on its route between origin and destination. Detour delays are caused by driving a route which is longer than the shortest possible route.

The quantities above clearly depend on the routing decisions on which they are evaluated. For readability, we refrain from adding an additional index and will make sure that the underlying routing solution is clear from the context.

In principle, the goal is to find a routing in which the passengers experience as little delay as possible. However, there exist trade-offs between the delays of the pods; prioritizing one can lead to extra delays for others. In order to define a system optimum, we measure the overall quality of a routing by simply averaging over the delays of all requests. The objective reads as $\min 1/|\Pi| \cdot \sum_{\pi\in\Pi} \mathrm{del}_\pi$ where the minimization is over the set of all routings fulfilling the requests in Π. An equivalent objective, up to the non-influenceable factor $|\Pi|$, is to minimize the sum of the delays.

$$\min \sum_{\pi\in\Pi} \mathrm{del}_\pi \tag{6.1}$$

Again equivalent, up to an additive constant, is to minimize the sum of the arrival times. This is shown in the following formula, where the last two terms cannot be influenced by routing decisions.

$$\sum_{\pi \in \Pi} \mathrm{arr}_\pi = \sum_{\pi \in \Pi} \mathrm{del}_\pi + \sum_{\pi \in \Pi} \tau_\pi + \sum_{\pi \in \Pi} l_\pi$$

The average-delay objective has the drawback that it returns solutions that are good in average but may be unfortunate for some requests which might have particularly long delays even in the optimal solution. We use the average-delay objective as the primary objective for this thesis as a natural and simple choice.

6.6 Online Optimization

How can the objective 6.1 be used by the online controller in order to make good routing decision? It can certainly be used at the end of the day (or operation horizon) to measure the performance of the routing solution in retrospective. It can also be used to compute an *a-posteriori* optimal solution in retrospective, when all requests are known. It is certainly a lower bound to any solution achievable by online algorithms, as it represents the system optimum in case all demands were known from the beginning.

However, at the time the online controller takes the routing decisions, the requests to appear in the future are yet unknown. As it is not clear what other requests will later interfere with the currently known requests, there is no means of deciding at any time before the end of the release horizon which routing is optimal. The terms of objective function 6.1 are only partly revealed and the routing with minimal average delay is at that time not well defined. This problem can be overcome by replacing the original *offline objective* function with alternative *online objectives.*

One choice for an online objective is to use objective 6.1 without the unknown terms. We refer to this approach as *snapshot optimization*, as it uses the information available at the current moment and makes no assumptions about the future. Decomposing the sum in objective 6.1 into sums for the past requests already arrived at the destination Π^p, the open requests Π^o and the future requests Π^f yields

$$\sum_{\pi \in \Pi} \mathrm{del}_\pi = \sum_{\pi \in \Pi^p} \mathrm{del}_\pi + \sum_{\pi \in \Pi^o} \mathrm{del}_\pi + \sum_{\pi \in \Pi^f} \mathrm{del}_\pi \ .$$

As the requests Π^f are unknown to the online controller and the requests Π^p

cannot be influenced anymore, the snapshot objective reduces to

$$\min \sum_{\pi \in \Pi^o} \mathrm{del}_\pi \tag{6.2}$$

over the set of all routings which transport the open requests from the current positions to the destinations.

A way of also taking future requests into account is to add a term for the expected future delays. The additional term incorporates expected effects of the current decisions onto future delays into the objective. It can be beneficial, for example, to route an open request with some extra delay if it is expected that several future requests will save delays in turn. In case the expected future demand is known, one can add a term for the expected future delays to the objective function. However, it is not straight-forward how to compute the effects of current decisions onto delays of future requests. As an extension to the flow routing algorithm, we will in Section 10.3.3 discuss one possibility how the effects of expected future traffic can be taken into account.

6.7 Model Statement

The presented model for PRT routing will be the basis for all following sections. We refer to the problem as to the *Personal Rapid Transit Routing Problem (PRTRP)*. It is summarized here in compact form. Recall that the shortcut notation $[\alpha, \beta]$ is used to represent the integer interval $\{\alpha, \alpha + 1, \ldots, \beta\}$.

Personal Rapid Transit Routing Problem (PRTRP).

- Given is a directed graph $G = (V, A)$ and a release horizon T.
- A set of requests (pods) Π needs to be routed, each request $\pi \in \Pi$ having an origin-destination pair $(s_\pi, t_\pi) \in \binom{V}{2}$ and a release time $\tau_\pi \in [0, T]$. In an online manner, the existence of the requests gets revealed at the respective release time.
- A discretized time setting is considered with pods moving from vertex to vertex. At each timestep, every pod can either stay on its current position or move to a neighboring vertex (in arc direction).
- The departure time dep_π is the time before the first move of pod π and the arrival time arr_π is the time right after the last move. Pod π is said to be in transit in the time span $[\mathrm{dep}_\pi, \mathrm{arr}_\pi]$. A conflict occurs if two

pods are located on the same node at the same time and both are in transit.

- The goal is to find a conflict-free routing minimizing the sum of the arrival times.

The PRTRP is closely connected to the CFVRP presented in Part I of the dissertation. The main differences are the directed graph, the online nature and the average-delay objective of the PRTRP. Also strongly related is the packet routing problem, where nodes represent network routers, arcs are transmission channels and requests correspond to data communications. In contrast to PRTRP and CFVRP, packet routing research defines edge conflicts, due to limited transmission bandwidth, whereas nodes are often assumed to have infinite capacity. The packet routing setting is hence a relaxation of our setting where node conflicts (and arc conflicts) are forbidden. Further, the PRTRP can be interpreted as an integer variant of the multicommodity flow over time problem with limited storage.

Chapter 7

Network Capacity

The track network is a potential bottleneck of a PRT system. In case transit demand exceeds the amount of traffic that can be handled by the network, some of the requests need to be backlogged and waiting times occur. Therefore, the achievable throughput is an important quantity for the design of PRT systems.

In the first section of this this chapter, we introduce a notion for the amount of traffic that a network can cope with and call it the *network capacity*. In the second section we introduce a *relaxed network capacity*, which can be computed efficiently and which is a theoretical upper bound on the network capacity. It will be useful for benchmarking the routing algorithms in the computational study (Chapter 11).

7.1 Definition

The notion of a *line capacity* has been discussed in detail within the PRT community. It refers to the maximum throughput over a track segment. It is proportional to the inverse of the headway time and therefore dependent on several design parameters such as safety policy, nominal speed and the brake actuation time [SM07]. Typical PRT system proposals use nominal speeds of 24–48 km/h with headway times of 1–5 seconds [And09, Con12]. The resulting line capacities vary between 720 and 3600 pods per hour. In our model, these system-specific parameters are hidden in the length of one timestep (which equals one headway time); line capacity always equals one request per timestep, as at any node, the maximum throughput is one pod per timestep.

We extend the notion of capacity from a single track line to a track network. It now refers to the maximum throughput of an entire network topology. For this, we assume to know how the traffic is distributed over the origin-destination combinations. We call such a distribution a *demand pattern*. Then, the question is how much traffic, for the given demand pattern, can be supported by the network?

More formally, we assume that the demand pattern is given in form of a demand matrix $D = \{d_{u,v}\}_{u,v \in V'}$, where $V' \subset V$ is the set of terminal nodes. The value of $d_{u,v} \geq 0$ represents the demand from terminal node u to terminal node v. The demand matrix is normalized in the sense that $\sum_{u,v \in V'} d_{u,v} = 1$, the values $d_{u,v}$ are hence relative to the total demand. We assume zero round trip demands, $d_{v,v} = 0$ for all $v \in V'$. The demands contain all trips on the network, including passenger transit and empty vehicle redistribution.

The network capacity depends on the demand pattern, as illustrated in Figure 7.1. It is measured in pods per timestep and the quantity can equivalently be understood to be given in multiples of the line capacity.

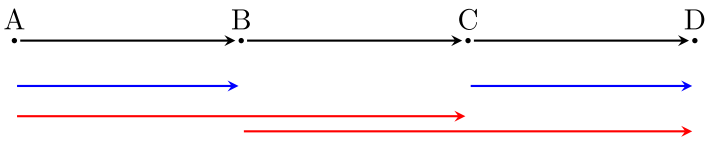

Figure 7.1: Maximum throughput on a line graph for two different demand patterns given in blue and red. *Blue:* $d_{A,B} = d_{C,D} = 0.5$, other demands are zero. In this case, two requests can be served per timestep, one from A to B and one from C to D. The network capacity is 2, corresponding to double line capacity. *Red:* $d_{A,C} = d_{B,D} = 0.5$, other demands are zero. There is a bottleneck between B and C where only one pod can pass per timestep. The network capacity is hence 1, corresponding to single line capacity.

The capacity is the largest demand the network can cope with. If this demand is exceeded, more new requests enter the system than what it is able to process. In such a case, the number of open requests increases over time. The capacity hence corresponds to the largest release rate, for a given network and a demand matrix, such that the arrival rate can match the release rate and the number of open requests does not necessarily increase over time. We now formally define the capacity notion.

Definition 7.1. *Given is a graph G and a demand matrix D. Assume that an inexhaustible reserve of requests is ready for each origin-destination pair. Let*

$[0, T]$ be an arbitrarily large time span. For a conflict-free routing solution, let $a_{u,v}$ denote the average number of arrivals per timestep in the time span $[0, T]$ for pods traveling from u to v. The network capacity $\varphi_{\max}(G, D)$ is the largest φ for which there exists a routing with arrival rate $a_{u,v} \geq \varphi \cdot d_{u,v}$ for all $u, v \in V'$ and for some horizon T.

For demands below capacity, there hence exists a routing such that the arrival rate matches the release rate and the number of open requests is stable or decreases. On the other hand, for demands above capacity, it is clear that a part of the released requests needs to be backlogged. If the situation persists, the number of such backlogged request increases together with the delays.

7.2 Relaxed Network Capacity

We discuss here a relaxation of the capacity $\varphi_{\max}(G, D)$. The relaxation is a theoretical upper bound to the network capacity and has the advantage that it can be computed efficiently.

The relaxation is based on a network flow formulation. One aims to send as much flow as possible through graph G, such that the flow is distributed among origin-destination pairs as given by demand matrix D. The flows thereby represent the average traffic rates in pods per timestep.

The flow formulation is given in (7.1). It is an LP of maximum concurrent flow type. Let $P_{u,v}$ denote the set of all directed paths from u to v and let P be the union of all paths for all origin-destination pairs, $P = \cup_{u,v \in V'} P_{u,v}$. Furthermore, let x_p be the variables for the flow amount sent along path $p \in P$. Then, constraints 7.1b assert that the flows are distributed between the origin-destination pairs such as given by demand matrix D. Constraints 7.1c enforce that no more than a total of one unit of flow passes through a node.

$$\text{Maximize } \varphi, \text{ subject to} \tag{7.1a}$$

$$\sum_{p \in P_{u,v}} x_p \geq \varphi \cdot d_{u,v} \qquad \text{for all } u, v \in V' \tag{7.1b}$$

$$\sum_{p \in P : w \in p} x_p \leq 1 \qquad \text{for all } w \in V \tag{7.1c}$$

$$x_p \geq 0 \qquad \text{for all } p \in P \tag{7.1d}$$

Theorem 7.2 proves that the optimal solution to LP 7.1 indeed provides an upper bound on capacity.

Theorem 7.2. *The optimal objective value $\overline{\varphi}(G, D)$ of LP 7.1 is an upper bound on the capacity $\varphi_{\max}(G, D)$.*

Proof. If one assumed, for the sake of contradiction, that there existed a routing with arrival rate $a_{u,v} > \overline{\varphi}(G, D)\ d_{u,v}$ for all $u, v \in V'$ over some time span $[0, T]$, then one could construct a solution to LP 7.1 out of the average traffic rates from the routing, with an objective value larger than $\overline{\varphi}$. The existence of such a routing would hence contradict the optimality of $\overline{\varphi}(G, D)$. □

The LP in formulation 7.1 contains a possibly exponentially large set of path-flow variables. This problem can be overcome by transforming it into en equivalent arc-flow formulation containing one variable per arc and origin-destination pair. The transformed LP has polynomial size and can be solved efficiently using LP algorithms. Alternatively, the use of a column generation approach (see e.g. [AMO93]) or an approximate solution scheme (e.g. [GK98]) may be beneficial for large problems.

In the computational studies in Chapter 11 we will observe that the relaxed capacity is rather close to the achievable throughput in all tested scenarios. These observations give evidence that the relaxed capacity is a quantity which is not only useful as an upper bound to the network capacity but also as an estimate for the operational throughput.

Chapter 8

Routing Preliminaries

8.1 Feasibility

An important issue for transportation systems on guideways is deadlock avoidance. A *deadlock* is a constellation of pods and corresponding requests such that it is not anymore possible to route all pods to their destinations since pods on the tracks block each other.

Due to the parking assumption there always exists a feasible strategy to route the pods to their destinations: one could simply serve the requests one-by-one, thus only having a single pod on the tracks at any time, while the pods for all other open requests are waiting at the corresponding start points. However, the parking assumption (at the destinations) implies something much stronger, namely that it is impossible to create deadlocks, no matter how one routes the pods.

Theorem 8.1. *No deadlocks are possible in the suggested PRT model, i.e., for any time τ, any set of open requests and position of pods serving the open requests, it is possible to fulfill all requests.*

Proof. To show that all current and future requests can be served, it is sufficient to show that all requests in transit can be served, since one can wait with serving non-started and future requests until the ones in transit are completed, and then use e.g. a simple serial one-by-one routing strategy as mentioned above. Hence, let Π' be the set of requests in transit, and for $\pi \in \Pi'$ let $v_\pi \in V$ be the current position of pod π. We prove the theorem by showing that during the next timestep, the pods can be routed such that they reach positions $v'_\pi \in V$ for $\pi \in \Pi'$ satisfying

$$\sum_{\pi \in \Pi'} l(v'_\pi, t_\pi) < \sum_{\pi \in \Pi'} l(v_\pi, t_\pi), \tag{8.1}$$

where $l(v_a, v_b)$ stands for the shortest path length from v_a to v_b. In other words, the inequality requires a strict decrease in the total distance of the pods to their destinations to hold. For each $\pi \in \Pi'$ we fix an arbitrary arc a_π outgoing of v_π that would bring pod π closer to its destination. Recall that G is strongly connected by model assumption and that such an arc therefore exists. Let $A' = \{a_\pi \mid \pi \in \Pi'\}$. Consider the subgraph (V, A') of G. In (V, A') each vertex has at most one outgoing arc, since each vertex contains at most one pod. If A' contains a directed cycle C, then we can move each pod π located on a vertex of C along its arc a_π. All other pods stay on their current positions. This clearly does not create a conflict and fulfills (8.1). Otherwise, if (V, A') does not contain any directed cycle, then there exists at least one arc $a_\pi \in A'$ whose head vertex is not occupied by a pod. The corresponding pod π can be moved along a_π without conflict, bringing it closer to its destination. □

This result strongly relies on the parking assumption. Without this assumption, it is much more involved to decide whether it is possible to bring all vehicles on the network to their destinations. For results on the feasibility problem without the parking assumption, we refer to the work of Werren [Wer10].

8.2 Time Expansion

The time expanded graph is an inflated version of the network graph with encoded time information. Such networks have been suggested already by Ford and Fulkerson [FF58, FF62] to solve dynamic flow problems. We introduce the time expansion largely following the notation in [FS06] with some adaptations for our model.

The time expansion of $G = (V, A)$ is denoted by $\overline{G} = (\overline{V}, \overline{A})$. The node set $\overline{V}$ is obtained by creating $T + 1$ copies of V, where the ith copy is labeled V_i for $i \in [0, T]$. The copy of node v in V_i is denoted by v_i. Each such node encodes a time-space combination. The arc set of the time expansion is created by introducing arcs corresponding to the allowed transitions in time-space. *Waiting arcs* correspond to the possibility of waiting on a node. Waiting arcs (v_i, v_{i+1}) are introduced for each $v \in G$ and $i \in [0, T-1]$. *Transit arcs* correspond to driving through the network. For every arc $(u, v) \in A$ and every $i \in [0, T-1]$ there is an arc (u_i, v_{i+1}) in $\overline{A}$. As the time horizon T, we use here the operation horizon needed to route all requests to their destinations. The construction is illustrated in Figure 8.1.

Additionally, we introduce an origin and a destination node for each request. These *terminal nodes* are needed to model the parking assumption. Before leaving the origin station, pod π is parked on its origin node $\overline{s}_\pi$ and after arrival it parks on the destination node $\overline{t}_\pi$. The origin node is connected to the expanded graph by *departure arcs* $(\overline{s}_\pi, s_{\pi,i})$ for every $\pi \in \Pi$ and $i \in [\tau_\pi, T]$, where $s_{\pi,i}$ is the ith copy of the departure node of π. Similarly, the *arrival arcs* $(t_{\pi,i}, \overline{t}_\pi)$ connect to the destination node.

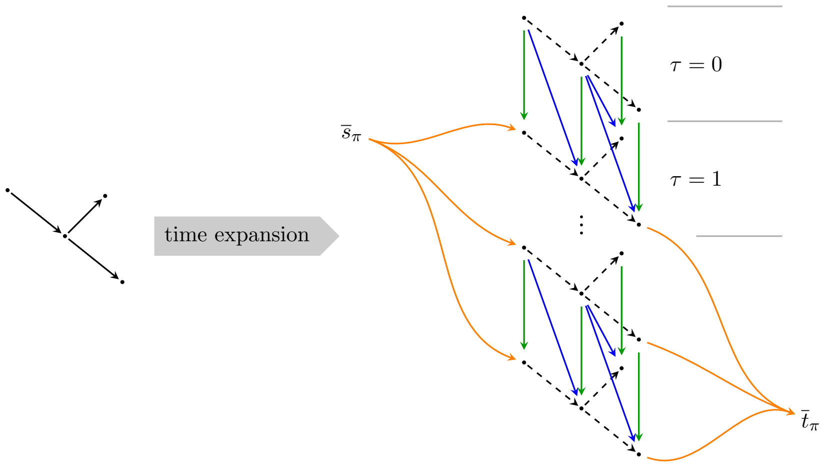

Figure 8.1: A small network piece and its time expansion. The transit arcs (blue), waiting arcs (green), departure and arrival arcs (orange) are shown together with the external node pair for a request π with release time 1.

Remark 8.2. *Particularly for large horizons T, the expanded graph can be very large. For practical implementations, we suggest the following improvements.*

- *Trips with same origins or destinations can use the same terminal nodes. For this purpose, introduce a terminal node pair per terminal node and not per request. The separate terminal nodes for each request were chosen here for simplicity of presentation.*

- *For online purposes, not the entire time expansion needs to be physically stored. Usually only the near future is relevant for planning. The time expansion can hence be replaced by a much smaller version with less time layers. In a rolling horizon regime, at each timestep, the last past* *time layer can be deleted and a new time layer is added at the end of the expansion. This way the total size remains constant.*

8.3 Flow Formulation

A routing for a request π corresponds to a path through the time expansion from $\overline{s}_\pi$ to $\overline{t}_\pi$. We call a such a path a *dynamic path.* It encodes at the same time the chosen route and schedule. We can also identify conflicts on the basis of dynamic paths.

Observation 8.3. *Two routings are in conflict if and only if the corresponding dynamic paths share a node.*

Note that also the parking assumption is correctly represented in this conflict notion. A dynamic path visits exactly two terminal nodes, its source and sink. Other terminals cannot be part of the dynamic path, as they are dead ends. As each request has its own pair of terminal nodes, there can consequently be no conflicts on terminal nodes.

Based on Observation 8.3 we introduce formulations for the PRTRP, first for the offline and then for the online problem.

8.3.1 Offline Formulation

Recall that the offline version of the PRTRP is the variation of the routing problem where all requests are known from the beginning. It is equivalent to optimizing the routing a-posteriori, at the end of the day when all requests are known.

As a consequence of Observation 8.3, the offline version of the PRTRP can be formulated as a node-disjoint path problem or equivalently as a binary multicommodity flow problem (MCF) on the time expanded network. Both are well studied problems in combinatorial optimization (see, e.g. [Sch03]).

The multicommodity flow consists of a number of k commodities where each commodity is associated with a request. For each commodity, one unit of flow needs to be sent through the time expansion from the origin to the destination node of the associated request. The total flow capacity at each node is limited to one unit. Let the flow of the commodity (associated with request) π on arc $a \in \overline{A}$ be x_a^π. Requiring the flow variables to be 0 or 1 leads to the integer linear program (ILP) stated in (8.2). The short notations δ_v^+ (and δ_v^-) stand for the sets of outgoing (resp. incoming) arcs of node v. The indicator variable b_v^π has value 1 in case $v = \overline{s}_\pi$, -1 in case $v = \overline{t}_\pi$ and 0 otherwise.

$$\sum_{a\in\delta_v^+} x_a^\pi - \sum_{a\in\delta_v^-} x_a^\pi = b_v^\pi \qquad \text{for all } v \in \overline{V}, \pi \in \Pi \tag{8.2a}$$

$$\sum_{\pi\in\Pi}\sum_{a\in\delta_v^+} x_a^\pi \le 1 \qquad \text{for all } v \in \overline{V} \tag{8.2b}$$

$$x_a^\pi \in \{0,1\} \qquad \text{for all } a \in \overline{A}, \pi \in \Pi \tag{8.2c}$$

ILP 8.2 consists of flow conservation constraints 8.2a, node capacity constraints 8.2b and integrality constraints 8.2c. The capacity constraints are slightly different than in standard MCF settings, as we set the capacity limit on nodes instead of arcs. However, the formulation can in principle be transformed into a MCF with arc capacities. One can interpret (8.2) as a multicommodity flow over time problem, such as it is studied in [FS02, FS06]. More precisely, it is a variant of flow over time which can be described as integral min-cost multicommodity flow problem over time with limited node storage.

A solution to ILP 8.2 is a set of node-disjoint dynamic paths, one for each commodity. This leads to the following observation.

Observation 8.4. *A solution to ILP 8.2 corresponds to a conflict-free routing for the PRTRP and vice versa. As a consequence of the feasibility of the PRTRP, ILP 8.2 is always feasible for a large enough time horizon.*

To evaluate the travel time of a request from its dynamic path, one can introduce weights w_a for the arcs $a \in \overline{A}$. The weights are chosen such that the sum of the weights along a dynamic path corresponds to the travel time.

$$w_a = \begin{cases} 1 & \text{if } a \text{ is a transit or waiting arc.} \\ \tau - \tau_\pi & \text{if } a \text{ is a departure arc with } a = (s_\pi, v),\ v \in V_\tau \ . \\ 0 & \text{if } a \text{ is an arrival arc.} \end{cases}$$

With this weight function, the ILP 8.2 can be turned from a feasibility problem into an optimization problem, by adding the following objective function. It aims to minimize the total (resp. average) delay over all requests.

$$\text{Minimize} \quad \sum_{\pi\in\Pi}(\sum_{a\in\overline{A}} w_a x_a^\pi - l_\pi) \tag{8.3}$$

A second helpful weight function is given by the following commodity-dependent arc weights $h^\pi_{(u,v)}$. It is beneficial when computing shortest paths repeatedly, such as we will discuss for the sequential routing schemes in Section 10.1.

$$h^\pi_{(u,v)} = w_{(u,v)} + l^\pi_v - l^\pi_u$$

In this relation, $l^\pi_v = l(v, t_\pi)$ is a short notation for the shortest path distance from node v to the destination of commodity π. All weights $h^\pi_{(u,v)}$ are non-negative as $w_{(u,v)} + l^\pi_v \geq l^\pi_u$ holds. The path length of a path $P \in \mathcal{P}^\pi$ measured with weights h differs from the path lengths with respect to weights w by l_π

$$\sum_{(u,v)\in P} h^\pi_{(u,v)} = \sum_{(u,v)\in P} (w_{(u,v)} + l^\pi_v - l^\pi_u) = \sum_{(u,v)\in P} w_{(u,v)} - l_\pi$$

and objective 8.3 can be rewritten as follows.

$$\text{Minimize} \quad \sum_{\pi\in\Pi}\sum_{a\in\overline{A}} h^\pi_a x^\pi_a \tag{8.4}$$

The weight function h assigns to each arc the delay that vehicle π experiences when it uses this arc. The weights for arcs approaching the destination are decreased compared to weights w. This type of weight function is known from goal oriented search.

8.3.2 Online Formulation

Recall that the term snapshot optimization (defined in Section 6.6) refers to the approach to online optimization in which the current situation (snapshot) is optimized without taking into account the possible occurrences of future events. Differently stated, one optimizes for the scenario in which no new demand will interfere with the currently known demand. The goal is to find the routing minimizing the sum of delays of all known open requests.

Only requests are known with release times no later than the current time τ. Furthermore, the dynamic paths for open requests already in transit are fixed up to time layer τ. In the flow formulation, one can take account of this fact by choosing the current position on the current time layer to be the origin of the flow. The current positions v^π of the pods on the network graph can be mapped to a position $\overline{v}_\pi$ on the time expansion. For a pod π still parked at the origin node, $\overline{v}_\pi$ corresponds to the source terminal node $\overline{s}_\pi$. For a pod in transit $\overline{v}_\pi$ corresponds to node $v_{\pi,\tau}$, which is the copy of node v^π on time layer τ. Conflict-free routing corresponds to finding a dynamic path P_π from $\overline{v}_\pi$ to $\overline{t}_\pi$ such that the paths P_π are node-disjoint for all open requests $\pi \in \Pi^o$.

The formulation of the snapshot optimization problem reads as follows.

$$\text{Minimize} \quad \sum_{\pi\in\Pi^o}\sum_{a\in\overline{A}} h_a^\pi x_a^\pi \qquad \text{subject to} \tag{8.5a}$$

$$\sum_{a\in\delta_v^+} x_a^\pi - \sum_{a\in\delta_v^-} x_a^\pi = b_v^\pi \qquad \text{for all } v\in\overline{V}, \pi\in\Pi^o \tag{8.5b}$$

$$\sum_{\pi\in\Pi^o}\sum_{a\in\delta_v^+} x_a^\pi \le 1 \qquad \text{for all } v\in\overline{V} \tag{8.5c}$$

$$x_a^\pi \in \{0,1\} \qquad \text{for all } a\in\overline{A}, \pi\in\Pi^o \tag{8.5d}$$

In this ILP, b_v^π now refers to the current sources.

$$b_v^\pi = \begin{cases} 1 & \text{if } v = \overline{v}_\pi \text{ or } v = \overline{v}_\pi \\ -1 & \text{if } v = \overline{t}_\pi \\ 0 & \text{otherwise} \end{cases}$$

The existence of a solution is guaranteed by the feasibility result in Theorem 8.1. The time layers relevant for snapshot optimization can be restricted to $[\tau, \tau + T']$, where T' is chosen large enough to accommodate the optimal solution. We will give in Section 10.2 an algorithm for which there exists a theoretical bound on the total delay of $\sum_{\pi\in\Pi^o} \mathrm{del}_\pi \le k^2 n$ (see Observation 10.1), where $k = |\Pi^o|$. This is clearly also true for the optimal snapshot solution and the choice of $T' = k^2 n$ is certainly sufficiently large.

The optimal solution to this ILP can in principle be computed using an ILP solver. However, these problems can be large and will in most cases take far too much time to solve to optimality. For exact approaches in general there is little hope for fast algorithms as we will discuss in the next chapter.

Chapter 9

Computational Complexity

Applying methods from algorithmic complexity theory, we will show that PRTRP is hard to solve to optimality by showing that it belongs to the class of NP-hard problems, similarly as it was done in Chapter 2 for the CFVRP. The result in this chapter implies that the existence of an efficient algorithm for PRTRP would imply P=NP, which is widely believed to be wrong. Furthermore the question of whether P equals NP is considered one of the most important unsolved problems in mathematics (see Millennium Problems of the Clay Institute of Mathematics, [Coo92]). There is hence little hope for finding optimal solutions to general PRTRP instances in reasonable computation time. Even if NP-hardness results are negative results, they give incentive to shoot for a weaker goal than finding an optimal solution, like the development of efficient approximation algorithms.

The hardness is proven by relating the PRTRP to a sumcoloring problem (see [NSS99]), a problem with application in very large scale integrated (VLSI) circuit design that was proven to be NP-hard by Szkaliczki [Szk99].

We consider the case where all requests have the same release time. It is a special case of both the snapshot and the a-posteriori optimization problems.

Theorem 9.1. *The PRTRP is NP-hard on paths.*

Proof. We prove NP-hardness by reduction from the *saturated interval sum-coloring problem (SISP)*, stated in the following. Given is a set of q intervals with integral endpoints $S = \{[a_1, b_1], \ \dots \ , [a_q, b_q]\}$ and $\beta \in \mathbb{N}$ colors. The task is to assign a color $c_s \in [0, \beta - 1]$ to every interval $s \in S$ such that no two intersecting intervals have the same color and such that the sum over the assigned colors $\sum_{s \in S} c_s$ is minimized. We restrict the considerations to the so-called *saturated case*, in which each interval is a subset of $[0, \alpha]$ for a given α and every integer in $[0, \alpha]$ appears in exactly β intervals. Furthermore, we assume without loss of generality that $b_i - a_i \geq 4$ for all $i \in [1, q]$.

The NP-hardness of the SISP follows directly from the NP-hardness of the interval placement problem shown in [Szk99, Mar05]. The interval placement problem is a continuous version of the SISP and the hardness proof is analogous.

In order to prove NP-hardness of PRTRP, we will show that it is at least as hard as SISP. For this purpose, we show how to create from any given instance of SISP a corresponding instance of PRTRP, such that the optimal value of the PRTRP instance allows for deducing the optimal value of the SISP instance.

The instance of PRTRP consists of a directed path of length 2α and a set of requests Π. The fixed velocity of the vehicles and the integral departure times define time-space diagonals along which the vehicles can reach their destinations (see Figure 9.1). If we neglect the possibility for the vehicles to wait during transit for a moment (we will give a justification for this later in the proof), the task corresponds to packing the request onto diagonals, without overlaps and such that the sum of the delays is minimized. Let us label the diagonals with increasing integers starting from -2α, such as shown in the figure. We call the diagonals with negative labels partial diagonals. A request $\pi \in \Pi$ that gets assigned to the diagonal with label c_π experiences a waiting-at-origin delay of $a_\pi + c_\pi$ and has arrival time $b_\pi + c_\pi$. The minimization of the sum of delays hence corresponds to minimizing $\sum_{\pi\in\Pi} d_\pi = \sum_{\pi\in\Pi} a_\pi + \sum_{\pi\in\Pi} c_\pi$. As the first term depends on given data only, it can be dropped and the minimization of $\sum_{\pi\in\Pi} c_\pi$ is an equivalent objective.

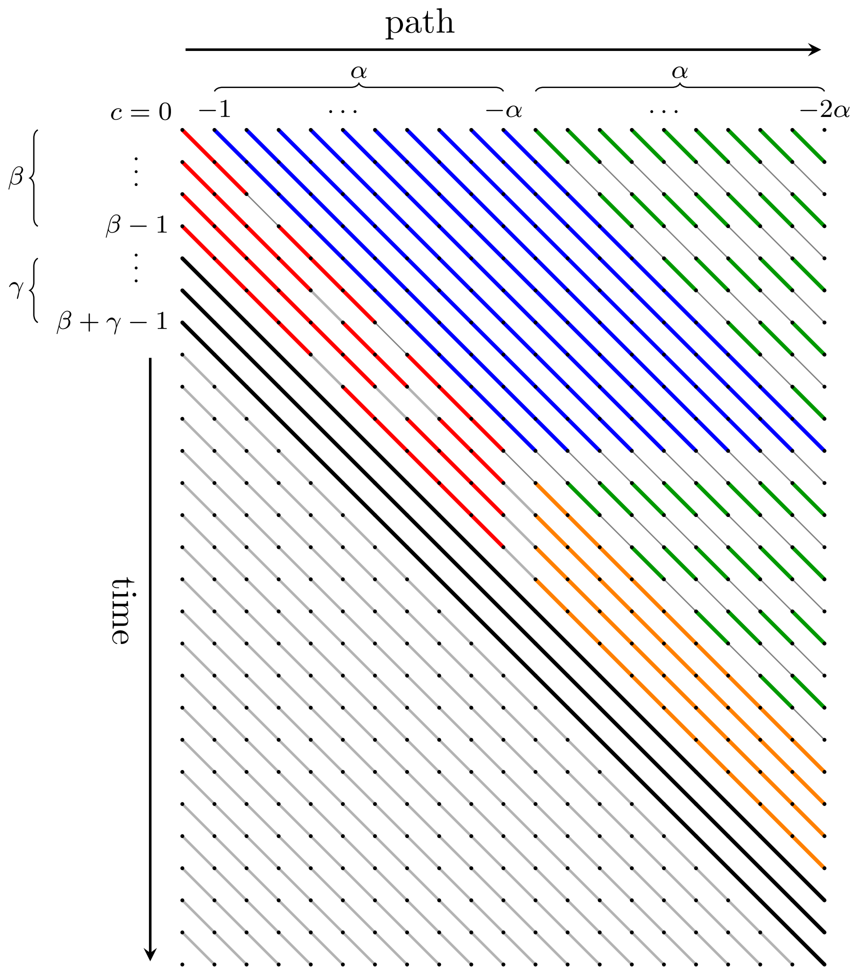

Figure 9.1: The figure schematically shows a PRTRP instance illustrated by the time-space diagram with the optimal PRTRP solution. The interval requests (red), 2α-requests (black), α-requests (blue), $(\alpha - 1)$-requests (orange) and the 1-requests (green) are distinguished by color.

We complete the instance description by listing the requests in Π with the help of Figure 9.1. All requests have release time zero.

- *Interval requests* are introduced according to the intervals in the SISP instance. There are q such requests with route lengths between 4 and α. Because of the saturation of the SISP instance, we know that these requests fit onto β full diagonals. We denote the set of interval requests by Π_{interval}.
- More requests are introduced as in Figure 9.1. We refer to the requests by the length of their routes.

  2α-requests. These requests travel the entire path. The number of such requests is denoted by γ and will be specified later in the proof.

  α-requests. α requests of length α are introduced such as shown in the figure.

  $(\alpha - 1)$-requests. β requests of length $\alpha - 1$ are introduced, again such as shown in the figure.

  1-requests. We introduce requests of length 1 in order to fill up the capacity of the partial diagonals, except for some time-space nodes which remain unoccupied at the end of the path. The number of 1-requests is not larger than $\alpha^2/2$.

A reference solution to the PRTRP instance is shown in Figure 9.1. The interval and the $(\alpha - 1)$-requests are assigned to diagonals $[0, \beta - 1]$, the 2α-requests to diagonals $[\beta, \beta + \gamma - 1]$, the α-requests to diagonals $[-\alpha, -1]$ and the 1-requests fill up the partial diagonals $[-2\beta, -1]$. Apart from the interval requests, this assignment is unique up to swaps of identical requests. Such swaps are without effect on the objective. However, the placement of the interval requests to diagonals $[0, \beta - 1]$ is not trivial and has effect on the objective function.

The claim is that the reference solution is optimal if and only if the assignment of the interval requests to diagonals $[0, \beta - 1]$ minimizes $\sum_{\pi \in \Pi_{\text{interval}}} c_\pi$. This corresponds to solving the underlying SISP instance to optimality which will complete the reduction.

How can we guarantee that there is no better solution to the PRTRP instance than in the reference solution? And why is it not beneficial to use waiting in transit? We start by showing that there exists an optimal solution in which the 2α-requests have lower priority than all other requests (are at the bottom of Figure 9.1) and do not wait during transit. For this, we introduce a swapping operation that allows us to generate from an arbitrary optimal solution one which fulfills the above property. Let π be a 2α-request which has some shorter request underneath it.

1. Removing π creates an empty slot in the schedule.
2. Shift up everything below the empty slot by one diagonal.
3. Insert π at the bottom, assigning the earliest conflict-free schedule without waiting in transit.

Applying this operation repeatedly moves the 2α-requests to the bottom and removes waiting of 2α-requests without increasing the objective value. If one swapping operation increases the delay of π by ψ (moving π down by ψ diagonals), the consequence is that the delay of at least ψ requests (at least one per diagonal, as there cannot be empty diagonals in the optimal solution) gets in turn reduced by one each.

We proceed by showing that, in an optimal solution, all requests are assigned to diagonals with labels $[(-2\alpha),(\beta+\gamma-1)]$. It follows that waiting in transit is suboptimal, as waiting requests would occupy extra time-space nodes which are not available. Only at the path end there is some spare capacity, but waiting there is not beneficial. Let us assume, for the sake of contradiction, that there is a request which occupies (parts of) diagonal $\beta+\gamma$ (or higher). We know from before that there exists an optimal schedule in which the 2α-requests are at the bottom of the schedule. They will hence occupy diagonals $[(\beta+1),(\beta+\gamma)]$ instead of $[\beta,(\beta+\gamma-1)]$ as in the reference solution. The contribution to the objective function of the 2α-requests hence increases by at least γ. This needs to be compensated by a better allocation of the shorter requests. There are in total not more than $q+\alpha+\alpha^2/2+\beta$ such requests. Their maximum delay is bounded by $2\alpha+\beta$. We choose γ to be larger than $(q+\alpha+\alpha^2/2+\beta)(2\alpha+\beta)$ such that it is impossible that a solution scheduling requests outside $[(-2\alpha),(\beta+\gamma-1)]$ could be optimal.

It remains to show that the assignment of the requests to the diagonals in the reference solution is optimal. The optimality of the assignment of the 2α-requests to diagonals $[\beta,\beta+\gamma-1]$ has already been proven. Diagonals $[-2\alpha,-(\alpha+1)]$ can only be filled by 1-requests. Diagonals $[-\alpha,-1]$ are each filled by one α-request plus a number of 1-requests, while diagonals $[0,\beta-1]$ are each filled by interval requests plus one $(\alpha-1)$-request. This must be the case in an optimal solution because of the following. Suppose for the sake of contradiction that some interval request π was located on a partial diagonal with label $\psi<0$. Consequently, the α-request π' which is located on ψ in the reference solution must now be located on some other diagonal ϕ with larger label number. One can, in this case, generate an improved solution by exchanging the requests on ψ with the corresponding requests on ϕ, i.e. π' and the ones allocated on the same diagonal on the lower right. The interval request π moves to ϕ together with an $(\alpha-1)$-request and possibly some more interval requests. The α-request π' moves to ψ together with a number of 1-requests. At least two 1-requests are needed to replace each interval request. The exchange operation is improving, as more requests move up (reduction of delay by $\phi - \psi$) than down (increase by the same amount). A solution with an interval request on a partial diagonal can therefore not be optimal and the interval requests must hence be assigned to diagonals $[0, \beta - 1]$. The assignment of the remaining request categories follows directly. □

We have seen that even the simple setting of a directed path is NP-hard for the objective of minimizing the average delay. This is different for the case of makespan optimization, for which the case of a directed path is open, as it was pointed out in Chapter 2 of the first part of this dissertation.

Chapter 10

Routing Algorithms

This chapter includes the description of three algorithms, as we will use them for the computational study.

The first is the well-established sequential routing. Its characteristic is that it routes the requests sequentially, one after the other, and that the routes will not be changed later. Routings are planned into the future and a pod only leaves the origin station after a dynamic path is reserved all the way to the destination. Such approaches are greedy in the sense that routes and schedules are chosen without taking into account the effect the choice has on other requests. It belongs to the class of synchronous routings, as network-wide coordination and data-transfer is necessary. Sequential routing has been studied in depth in the AGV routing literature, we refer to [KT91, MKGS05, Ste08, SZ11] for further reference.

The second is what we call a push algorithm, as the pods simply push forward towards their destination. No planning into the future is made. This approach is greedy timestep-wise, as pods push forward in each timestep, without taking into account the future. Push routing is a representative of asynchronous routing, as it can be implemented using local communication only.

As a third approach we present a new concurrent and adaptive flow routing algorithm. It is based on a multicommodity flow formulation on the time-expanded graph, where the commodities represent requests. The formulation is solved for all open requests simultaneously with the objective of minimizing the average delay. The algorithm plans ahead into the future, however only fixes the routing for the next upcoming timestep. All further plans can later be adapted if it turns out to be beneficial.

10.1 Sequential Routing

In sequential routing, once a request π appears, a routing is fixed for π that will not be changed later, no matter what other requests are revealed in the future. If at timestep τ a set of new requests Π_τ appears, we determine routings for those requests as follows. We go through the requests Π_τ in any order, determining routings one-by-one. The routing of a pod π is chosen to be the quickest possible without creating conflicts with previously fixed routings, i.e., this includes requests with reveal time earlier than τ and those considered before π at timestep τ.

Such a quickest route can be obtained by computing a shortest $\overline{v}_\pi - \overline{t}_\pi$ path in $\overline{G}$ with respect to weights w_a and over all nodes not used by any previously fixed route. Here, Π^o is the set of open requests at current time τ and $\overline{v}_\pi$ denotes the current position of a pod $\pi \in \Pi^o$ on the time expansion. Further, the arc weights w_a for all $a \in \overline{A}$ represent the travel times such as introduced in 8.3.1. The procedure is illustrated in Figure 10.1.

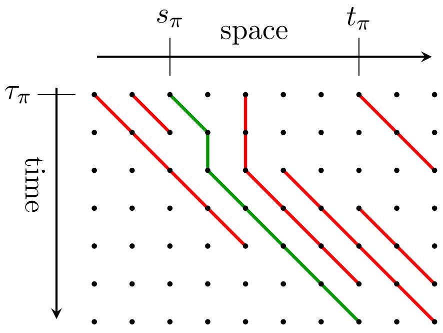

Figure 10.1: Sequential routing on a path network. The dynamic paths of previously routed requests are given in red. A new request with origin s_π and destination t_π is released at time τ_π. Sequential routing assigns the dynamic path indicated in green to the new request, as it leads to the earliest arrival time.

The relevant time horizon for the search is bounded theoretically by $\tau + \sum_{\pi\in\Pi^o}(l_\pi + 1)$. The (very conservative) bound stems from comparing sequential routing to the serial strategy of scheduling the departure of the next request only after the last pod in transit has reached its destination. This yields a schedule with latest arrival time $\tau + \sum_{\pi\in\Pi^o}(l_\pi + 1)$, in the worst case scenario when all requests in Π^o are released at the same time τ. As the sequential scheme chooses dynamic paths with shortest travel time, the resulting arrival times will be no later than in the serial strategy. The size of the relevant part of the time expansion is hence bounded by a polynomial in $|\Pi^o|$ and the network graph size n.

In consequence, the sequential routing can be computed in polynomial time using Dijkstra's shortest path algorithm. In practice, the computation times can be improved when using delays h_a^π instead of travel times w_a as arc weights for the search, while the result is the same (as discussed already in Section 8.3.1). This improvement is known as goal-oriented search (the goal-oriented version of Dijkstra is sometimes called A^* algorithm), and is illustrated in Figure 10.2. Note that the number of nodes to be checked is smaller when using goal-oriented search. For examples with larger networks and travel distances, the effect is even larger.

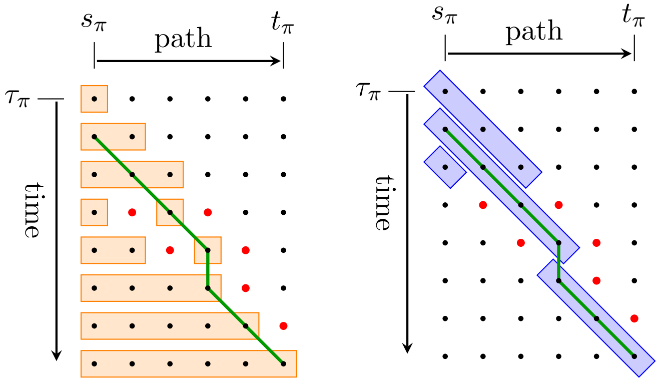

Figure 10.2: Sequential routing of a request (s_π, t_π) with release time τ_π. The nodes occupied by earlier requests are marked in red and a routing with earliest arrival time is given in green. *Left:* Dijkstra algorithm, using arc travel times w_a as weights, proceeds by checking all reachable nodes time layer by time layer until the destination is reached. The boxes indicate the nodes that need to be checked in each time layer. *Right:* Applying Dijkstra to the delays h_a^π yields that the nodes are checked with increasing delays. The top box contains the nodes reachable with delay 0, the second box with delay 1 and so on.

Two notable variants to sequential routing are the *direct sequential* and *shortest-path* sequential routings. In direct sequential, no waiting in transit is allowed. Finding such a routing corresponds to a shortest path search through a time expansion without waiting arcs. The second variant, shortest-path sequential, only allows routing along shortest paths and prohibits detours. The idea behind both versions is that both waiting and detours lead to increased resource occupation and can therefore reduce the system throughput. This effect will discussed in Section 11.1.5 of the computational analysis considering concrete instances.

In terms of approximation guarantees, Stenzel shows in [Ste08] that sequential routing can be a factor $\Theta(k)$ worse than the offline optimum on directed graphs. They also give an example on a path network where this bound is tight. Additionally, they show that for directed paths the delay for a request π is bounded by $del_\pi \leq |\Pi'|$, where Π' is the set of open requests routed before π. The same is true for arbitrary directed graphs when the combined variant of direct and shortest-path sequential routing is employed. For direct routes along shortest routes, a request π can only get delayed by at most one timestep from each pod in $|\Pi'|$.

One could think of an adaptive version of sequential routing in which routings are recomputed when new requests appear. Such an approach could, for example, compute sequential routings using several priority sequences and choose the best routing out of these. However, for situations in which a part of the pods is already in transit, it can happen that some of the priority sequences will not lead to feasible routings. This is as the parking assumption does not hold on the current positions. It is not even clear whether there always exists a feasible priority sequence when starting from an arbitrary online situation.

10.2 Push Routing

In push routing, each pod simply moves forward whenever possible. In every timestep, the pods push forward along a given shortest path if the next position is either free or if the vehicle at the next position can also be pushed forward along its path. The procedure is simple, needs little coordination and no planning into the future.

We formalize the procedure by using a notation similar to the one in the proof of Theorem 8.1. At a time τ, let the open requests in transit be $\Pi' \subset \Pi^o$ and let their current positions be v_π for each $\pi \in \Pi'$. We assume further that for each π a shortest path P_π is given. Let a_π be the arc outgoing of $v_\pi \in V$ which is part of P_π. Let $A' = \{a_\pi \mid \pi \in \Pi'\}$ and let us consider the subgraph (V, A') of G. In (V, A') each vertex has at most one outgoing arc, since each vertex contains at most one pod. In case (V, A') is not connected, each of its connected components can be treated separately. We can therefore assume in the following that (V, A') is connected.

We distinguish two cases (illustrated in Figure 10.3). Note that there can be at most one cycle in a connected component, because each node has at most one outgoing arc. In case A' contains no cycle then the component is an in-tree. Only the requests belonging to one path in the tree can be moved, all other pods need to wait. The path is chosen using some priority rule, such as giving priority to the longest path or to the path containing the request with the earliest release time. Otherwise, in case A' contains a cycle C, then we can move each pod π located on a vertex of C along its arc a_π. The pods corresponding to arcs in $A' \setminus C$ must stay at their current positions. Requests that are still parked at the origin get departed whenever their departure node is currently not occupied. Pods departing are inserted into the set Π' for the next timestep, whereas pods that have arrived at their destination get removed from Π'.

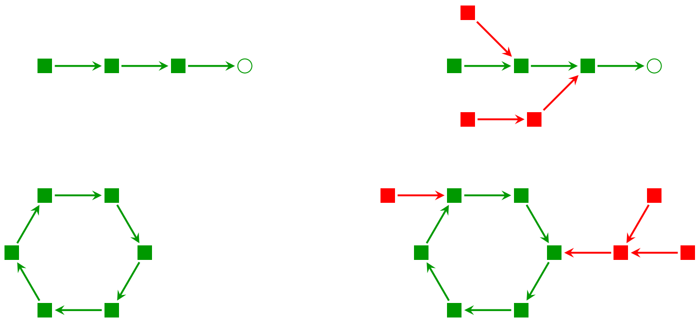

Figure 10.3: Distinction of cases. The pod locations are drawn as rectangles, each with an arrow pointing to the next location along its route. Unoccupied locations are drawn as circles. *Upper Left:* In case (V, A') (resp. a connected component in (V, A')) is a path, all pods can move forward simultaneously. *Upper Right:* In case of a tree, one path is chosen along which pods can move (green), all other pods wait (red). *Lower Left:* In case of a cycle, all pods can move forward simultaneously. *Lower Right:* If the graph contains a cycle plus incoming arcs, only the pods along the cycle can move.

As it was already argued in Theorem 8.1, at least one pod can move forward in each timestep. A pathological example in which only exactly one pod can move forward is when all pods want to move into the same node. The sum of the remaining route lengths $\sum_{\pi\in\Pi^o} l(v_\pi, t_\pi)$ decreases by at least one in each timestep. If no new requests appear, the sum will after at most $\sum_{\pi\in\Pi^o} l(v_\pi, t_\pi)$ timesteps reach zero and all pods will have arrived.

Observation 10.1. *The push algorithm routes the open requests Π^o to their destinations in at most $\sum_{\pi\in\Pi^o} l(v_\pi, t_\pi) \leq kn$ timesteps, where $k = |\Pi^o|$. The total delay is hence bounded by $\sum_{\pi\in\Pi^o} \mathrm{del}_\pi \leq k^2 n$.*

Push routing requires coordination only for detecting whether a pod belongs to a path, tree or cycle structure. This minimal amount of coordination is necessary to decide who can move and who needs to wait (e.g. in a situation as shown on the lower right in Figure 10.3). The case detection can occur by communication between neighboring pods and does not require a central computer. The Push algorithm is therefore a representative of the class of asynchronous routing strategies.

10.3 Flow Routing

The equivalence between the PRTRP and a network flow problem has been established already in Section 8.3. Now we want to take advantage of this close relationship by designing algorithms that compute flows and transform these into routings. For this purpose we solve the fractional relaxation of the snapshot multicommodity flow formulation 8.5. Even if the resulting fractional solution does not directly correspond to a routing, it can be the basis for creating such. We will explain how to solve the relaxation quickly (Section 10.3.1) and how to create a routing from this solution (Section 10.3.2).

This approach, which we call *flow routing*, has the properties that it plans into the future and takes into account all requests simultaneously. Only the first timestep of the routing plan is really implemented, the further steps are computed to estimate the future effects of current actions. The procedure is repeated each timestep, always including the latest requests into the computations adaptively.

This scheme is only possible in case one has a fast algorithm at hand for finding good solutions to the snapshot problem in real-time. The direct way of solving formulation 8.5 is not promising because of the computational effort involved. We therefore compute the fractional solution and use it as a basis for generating an integral solution by rounding. The integral optimum is often much more difficult to find than the optimum of the continuous relaxation.

This is also the case here, where the integral problem is NP-hard while the fractional relaxation can be solved in polynomial time. However, the lower computation times come with the drawback of loosing the optimality guarantee. Flow routing will have to prove its performance for concrete problem instances in the computational analysis.

Using flow models is common in transportation planning. The *static traffic assignment problem (STAP)* aims to route traffic through a road network without creating congestion and such that the overall travel times are minimized. For an account on the related literature of STAP and for fast solution methods, we refer to [DSE09]. One important difference between the PRTRP and the STAP is that the latter routes traffic rates rather than particular vehicles. Traffic rates in the STAP can be fractional, while a routing in the PRTRP requires an integral flow solution. A second important difference is that there is no time dimension in STAP. It is assumed that the traffic rates are stationary over time. Therefore, the routing problem can be solved directly on the network graph without using a time expansion.

10.3.1 Solving the Flow Relaxation

We state the continuous relaxation of the snapshot optimization problem. It corresponds to formulation 8.5 without the integrality constraints. Again, b_v^π refers to the sources on the time expansion corresponding to the current positions $\overline{v}_\tau$ of the pods at current time τ.

$$\text{Minimize} \quad \sum_{\pi \in \Pi^o} \sum_{a \in \overline{A}} h_a^\pi x_a^\pi \qquad \text{subject to} \tag{10.1a}$$

$$\sum_{a \in \delta_v^+} x_a^\pi - \sum_{a \in \delta_v^-} x_a^\pi = b_v^\pi \qquad \text{for all } v \in \overline{V}, \pi \in \Pi^o \tag{10.1b}$$

$$\sum_{\pi \in \Pi^o} \sum_{a \in \delta_v^+} x_a^\pi \leq 1 \qquad \text{for all } v \in \overline{V} \tag{10.1c}$$

$$x_a^\pi \geq 0 \qquad \text{for all } a \in \overline{A}, \pi \in \Pi^o \tag{10.1d}$$

The existence of a solution is guaranteed by the feasibility result in Theorem 8.1. As already observed in Section 8.3.2, it is sufficient to use the time expansion with time layers $[\tau, \tau + k^2 \cdot n]$, where $k = |\Pi^o|$ corresponds to the number of commodities. The LP formulation is therefore of polynomial size in k and in the graph size n. It can be solved to optimality in polynomial time using standard LP solution techniques. Due to the potentially large problem size and the need for real-time implementation, we review in the following two approaches which are particularly suitable for the fast solution of large-scale MCF.

Formulation 10.1 is closely related to the multicommodity flow over time problem, such as studied by Fleischer and Skutella [FS02, FS06]. The differences are that they minimize the makespan and allow for unlimited storage of flow in nodes. Furthermore, they allow for arbitrary arc travel times. A consequence of the latter difference is that a time expansion of exponential (pseudopolynomial) size may become necessary. Simply solving the LP over the time expansion as we suggest it here does therefore not lead to a polynomial-time algorithm. It is in general not possible to solve the multicommodity flow over time problem in polynomial time, unless P=NP, as was shown by Hall et al. [HHS07].

In the following, we discuss two approaches suitable for solving formulation 10.1.

Column Generation

As column generation is a standard approach for solving MCF problems, we give only a short account of the method here. We refer to [AMO93] for more information.

In order to discuss column generation, we introduce a second formulation of LP 10.1 which changes from arc flow to path flow variables. Let $\mathcal{P}^\pi$ denote for each $\pi \in \Pi^o$ the set of the potential dynamic paths for request π and let $\mathcal{P}$ be the union of these path sets for all open requests, $\mathcal{P} = \cup_{\pi\in\Pi^o}\mathcal{P}^\pi$. Further, let $h_P = \sum_{a\in P} h_a^\pi$ be the total delay that commodity π experiences along dynamic path $P \in \mathcal{P}^\pi$. The variable x_P then denotes the amount of flow of commodity π along a path $P \in \mathcal{P}^\pi$. Note that the formulation can contain exponentially many variables, one for each path which may potentially carry flow.

$$\text{Minimize} \quad \sum_{\pi \in \Pi^o} h_P x_P \qquad \text{subject to} \tag{10.2a}$$

$$\sum_{P \in \mathcal{P}^\pi} x_P = 1 \qquad \text{for all } \pi \in \Pi^o \tag{10.2b}$$

$$\sum_{P \in \mathcal{P} : v \in P} x_P \leq 1 \qquad \text{for all } v \in \overline{V} \tag{10.2c}$$

$$x_P \geq 0 \qquad \text{for all } P \in \mathcal{P} \tag{10.2d}$$

The dual of LP 10.2 is given below. It contains a dual variable z_π for each commodity and a dual variable y_v for each node.

$$\text{Maximize} \quad \sum_{\pi \in \Pi^o} z_\pi - \sum_{v \in \overline{V}} y_v \qquad \text{subject to} \tag{10.3a}$$

$$\sum_{v \in P} y_v + h_P \geq z_\pi \qquad \text{for all } P \in \mathcal{P}^\pi, \pi \in \Pi^o \tag{10.3b}$$

$$y_v \geq 0 \qquad \text{for all } v \in \overline{V} \tag{10.3c}$$

The concept behind column generation is to solve LP 10.2 introducing variables only for a subset of the paths $\mathcal{P}' \subset \mathcal{P}$. Starting with few candidate paths, one can augment the set $\mathcal{P}'$ with new paths until it contains all relevant path variables and an optimal solution is found. The number of required paths is typically much smaller than $|\mathcal{P}|$.

Solving the LP with the path variables from $\mathcal{P}'$ returns primal and dual solutions, optimal with respect to the current path set $\mathcal{P}'$.[1] According to constraints 10.3b, z_π attains the value of the shortest path for commodity π in $\mathcal{P}'$, measured in travel time plus the sum of the dual node costs along the path. One can check whether the dual solution fulfills the inequality for all paths in $\mathcal{P}$ by computing the shortest path length between $\overline{v}_\pi$ and $\overline{t}_\pi$ and comparing its length to z_π. If z_π exceeds the shortest path length of commodity π, one adds the shortest path to $\mathcal{P}'$ and solves the LP again. This process is repeated until no shorter paths than z_π can be found for any of the commodities. At that point, one has a guaranteed optimal solution to LP 10.2 at hand.

[1] In order to have a feasible initial set $\mathcal{P}'$ for the column generation procedure, we introduce additional arcs from origin to destination of each commodity and use these arcs as initial start set $\mathcal{P}'$. With these paths we associate very high delays, such that they are not interesting for the optimal solution.

Primal-dual Approximation Schemes

We also give a short account of an approximation scheme for MCF. It is based on work by Garg and Könemann [GK98], Fleischer [Fle00] and Albrecht [Alb01]. Such schemes are particularly fast and well-suited, also for large-scale problems. They yield approximate solutions with objective value guaranteed to be not larger than $1 + \epsilon \cdot \mathrm{OPT}$, where the precision $\epsilon > 0$ can be chosen arbitrarily. The convergence rate is $O(\epsilon^{-2})$ in all of the mentioned approaches. The *Excessive Gap* method, as discussed in [DSE09], has a even better convergence rate of $O(\epsilon^{-1})$ but appears to be impractical because of the costly subproblems that need to be solved.

The algorithm approximates the following LP 10.4, which is slightly different from the relaxation stated above. A delay budget H is introduced as an additional problem parameter. Constraints 10.4c and 10.4d bound the node congestions to λ and the sum of the delays to λH. The objective is to find a flow which fulfills the delay budget and the node congestion constraints with the minimal possible λ. A solution with $\lambda \leq 1$ corresponds to a solution to (10.1) with total delay of at most H. The solution with minimal total delay can be determined by finding the minimal H for which a solution with $\lambda \leq 1$ exists, e.g. by using binary search.

$$\text{Minimize} \quad \lambda \qquad \text{subject to} \tag{10.4a}$$

$$\sum_{P \in \mathcal{P}^\pi} x_P = 1 \qquad \text{for all } \pi \in \Pi \tag{10.4b}$$

$$\sum_{P \in \mathcal{P}: v \in P} x_P \leq \lambda \qquad \text{for all } v \in \overline{V} \tag{10.4c}$$

$$\sum_{P \in \mathcal{P}} h_P x_P \leq \lambda H \tag{10.4d}$$

$$x_P \geq 0 \qquad \text{for all } P \in \mathcal{P} \tag{10.4e}$$

The corresponding dual LP reads as follows. The dual variables are z_π for the demand constraints 10.4b, y_v for the node congestion constraints 10.4c and y_H for the delay budget constraint 10.4d.

$$\text{Maximize} \quad \sum_{\pi \in \Pi^o} z_\pi \qquad \text{subject to} \tag{10.5a}$$

$$\sum_{v \in \overline{V}} y_v + H\ y_H = 1 \tag{10.5b}$$

$$\sum_{v \in P} y_v + h_P\ y_H \geq z_\pi \qquad \text{for all } P \in \mathcal{P} \tag{10.5c}$$

$$y_v \geq 0 \qquad \text{for all } v \in \overline{V} \tag{10.5d}$$

$$y_H \geq 0 \tag{10.5e}$$

The following primal-dual algorithm approximates the LP formulation (Algorithm PrimalDual). It maintains primal and dual solutions and improves these in rounds. The algorithm can be interpreted as a repeated game between a primal and a dual player. The primal player has the objective of sending flow as cheaply as possible. In each round, he sends one unit of flow for one commodity after another. The dual player can charge tolls for each unit of flow going through a node and also for the total delay incurred by the flows. He has a total budget of tolls he can raise and aims at making the task of the primal player as costly as possible. His strategy is to increase the tolls on the nodes where the opponent sends the flows.

Algorithm PrimalDual

```
Input: Time expansion with commodities with sources and sinks
Output: Flow $(x_P)_{P \in \mathcal{P}}$ and dual costs $(y_v)_{v \in \overline{V}}, y_H$
  set $x_P = 0$ for all $P \in \mathcal{P}$
  set $h = 0$, $c_v = 0$ for all $v \in \overline{V}$
  set $y_H = 1$, $y_v = 1$ for all $v \in \overline{V}$
  while (termination criterion) do
    for $\pi \in \Pi^o$ do
      find shortest path $P^* \in \mathcal{P}^\pi$ with respect to cost $\sum_{v \in P} y_v + h_P \cdot y_H$
      send one flow unit along $P^*$, $x_{P^*} = x_{P^*} + 1$
      update $c_v = c_v + 1$ for all $v \in P^*$
      update $h = h + h_{P^*}$
      update $y_v = e^{\epsilon c_v}$ and $y_H = e^{\epsilon h}$
    end for
  end while
```

The algorithms in [GK98, Fle00, Alb01] differ in termination criterion and dual cost update. Also, Fleischer [Fle00] and Albrecht [Alb01] suggest modifications such that the shortest paths do not need to be recomputed in every round, thereby saving computational effort. The output of the algorithm can be transformed into primal and dual solutions to the LP as follows.

Primal solution A primal feasible solution can be obtained dividing the flow resulting from the algorithm by the number of rounds performed. This solution satisfies constraint 10.4b. For λ one chooses the smallest value such that 10.4c and 10.4d are satisfied.

Dual solution A dual feasible solution is generated by normalization of the values of $(y_v)_{v\in\overline{V}}$ and y_H according to constraint 10.5b. For each commodity, the corresponding z_π then corresponds to the shortest path length with respect to $\sum_{v\in P} y_v + h_P \cdot y_H$.

The dual solution can be used to compute a lower bound to the optimal λ which in turn can be used as a termination criterion.

10.3.2 Rounding

The fractional solution obtained from solving relaxation 10.1 now is to be turned into a routing. Figure 10.4 gives an example of a fractional solution which does not yet correspond to a conflict-free routing. The task is to round the fractional flow values such that a consistent routing results.

We use a rounding scheme proceeding timestep by timestep. At time τ, the flows between time layers τ and $\tau + 1$ get rounded. Rounding of further timesteps is not necessary, as the online scheme only requires to know the routing decisions for the current time. Rounding further into the future also is not helpful as the underlying flow solution will change due to the new positions the pods will have one timestep later.

Let v_π^τ be the node in the time expansion occupied by request $\pi \in \Pi^o$ at current time τ and let $V^\tau = \{v_\pi^\tau \mid \pi \in \Pi^o\}$. It contains nodes in time layer τ as well as source terminal nodes of requests that are still located at the origin. Similarly, let $V^{\tau+1}$ be the set of nodes in the time expansion where the pods can be at time $\tau+1$. It includes nodes on time layer $\tau+1$ plus terminal nodes for requests that will still be parked at the origin or will have arrived at the destination. For $\pi \in \Pi^o$ and $v \in V^{\tau+1}\setminus\{\overline{s}_\pi \mid \pi \in \Pi^o\}$, let $x_\pi(v) \in [0,1]$ be the flow value of commodity π through arc (v_π^τ, v) in the fractional solution 10.1. As a special case, $x_\pi(\overline{s}_\pi) = 1 - \sum_{v\in V^{\tau+1}} x_\pi(v)$ represents the part of the flow going to later time layers for pods parked at the origin.

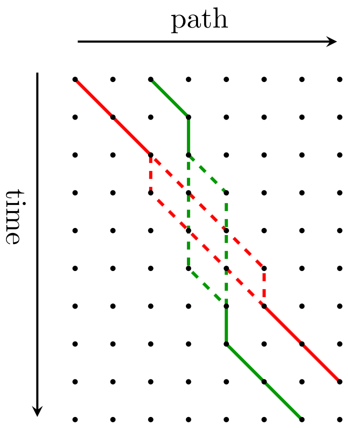

Figure 10.4: This example shows what fractional solutions to (10.1) can do more than integral ones. The flows are given for two commodities (red and green), where the thick lines correspond to flows of 1 and the dashed lines to flows of 1/2. Using fractionality, commodities can overtake on a path without violating the node capacity constraints.

Let $\mathbf{x}_\pi \in [0,1]^{V^{\tau+1}}$ for a $\pi \in \Pi^o$ be the vector containing the scalars $x_\pi(v)$ for all $v \in V^{\tau+1}$. Note that the entries of each such vector are in $[0,1]$ and sum to one. We interpret the vectors as probability distributions over the positions of the pods at the next timestep. Concretely, we look for a method routing pod π to node v with probability $x_\pi(v)$. Further, the routing decisions need to be coordinated such that conflicts are avoided.

More precisely, we want to determine to which vertex $v_\pi^{\tau+1} \in \overline{V}$ pod π will be sent in timestep $\tau+1$, such that $v_\pi^{\tau+1} \neq v_{\pi'}^{\tau+1}$ for any $\pi, \pi' \in \Pi^o$ with $\pi \neq \pi'$ and such that the marginal probabilities are preserved, i.e., $\Pr[v_\pi^{\tau+1} = v] = x_\pi(v)$ for all $v \in V^{\tau+1}$. The problem can easily be translated into the well-known problem of rounding fractional matchings in bipartite graphs. More precisely, the vectors $\mathbf{x}_\pi$ for $\pi \in \Pi^o$ can be represented as a fractional matching in a (complete) bipartite graph $(V^\tau, V^{\tau+1})$, where the weight of the edge $\{v_\pi^\tau, v_\pi^{\tau+1}\}$ is equal to $x_\pi(v_\pi^{\tau+1})$. See Figure 10.5 for an example. This well-studied setting is discussed e.g. in [GKPS06, CVZ10].

We use the specialized technique as presented in [GKPS06], since it provides an efficient way for rounding the fractional matching by doing local rounding steps on fractional cycles. The procedure performs a depth-first search along

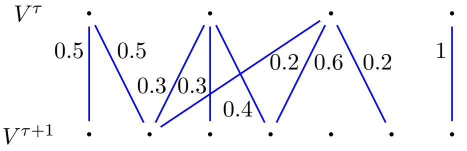

Figure 10.5: The fractional flows between V^τ (the positions of the requests at time τ) and $V^{\tau+1}$ (the positions of the requests at time $\tau+1$) corresponds to a fractional matching in a bipartite graph. The task is to round this matching to integrality without introducing conflicts.

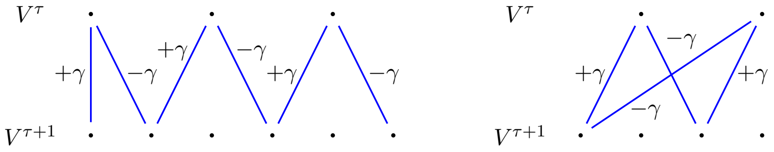

Figure 10.6: The rounding procedure detects maximal paths (*left*) or cycles (*right*) consisting of edges with fractional flow values. The edge flows alternatingly are increased respectively decreased by the same amount γ such that the outflows at the nodes in V^τ remain unchanged. The inflows at the nodes in $V^{\tau+1}$ can change only at the two end nodes in the case of the maximal path (*left*). Note that these end nodes necessarily are in $V^{\tau+1}$ and have slack capacity.

edges with fractional flow until a cycle or a maximal path is found (illustrated in Figure 10.6). These edges get alternatingly assigned to one of the two sets E_1 and E_2. Then, one determines the maximal interval $[-\alpha, \beta]$ such that adding a flow amount of $\gamma \in [-\alpha, \beta]$ to the edges in E_1 and reducing the same flow amount γ from the edges in E_2 does not violate the node capacity constraints $\sum_{\pi \in \Pi^o} x'_\pi(v) \leq 1$ for all $v \in V^{\tau+1}$ and the non-negativity constraints $x'_\pi(v) \geq 0$ for all $\pi \in \Pi^o$ and $v \in V^{\tau+1}$. With probability $p_\alpha = \beta/(\alpha+\beta)$, one then decreases the flows for edges in E_1 and increases the flows for edges in E_2 by an amount α. With the remaining probability $p_\beta = 1 - p_\alpha$, the flows in E_1 get increased by β and the flows in E_2 get decreased by β. Such a local rounding step has the following properties.

i) The resulting vector $\mathbf{x}'_\pi$ again corresponds to a fractional matching.
ii) Vector $\mathbf{x}'_\pi$ has at least one integral entry more than $\mathbf{x}_\pi$.
iii) The marginal probabilities are preserved as

$$\mathbf{E}[x'_\pi(v)] = x_\pi(v) + p_\alpha \alpha - p_\beta \beta = x_\pi(v) \ .$$

Repeating the procedure yields an integral solution after at most $|V^\tau||V^{\tau+1}|$ rounding steps.

After having obtained an integral assignment through the rounding, we send pod π from its current position v_π^τ to the new position $v_\pi^{\tau+1}$ with $x_\pi(v_\pi^{\tau+1}) = 1$. Next timestep, the new MCF problem is again solved and the rounding procedure is repeated to determine how to send the pods between $\tau + 1$ and $\tau + 2$, and so on.

The preservation of the marginal probabilities has the consequence that also expected values dependent on probabilities $x_\pi(v)$ are preserved. One can, for example, compute the expected number of pods moving closer to their destination between timestep τ and $\tau + 1$. Let $\Lambda(\pi)$ the set of nodes which are closer to the destination of π than its current position. Then the desired quantity reads as follows, by the linearity of expectations.

$$\mathbf{E}[\sum_{\pi\in\Pi^o}\sum_{v\in\Lambda(\pi)} x'_\pi(v)] = \sum_{\pi\in\Pi^o}\sum_{v\in\Lambda(\pi)} x_\pi(v)$$

10.3.3 Extensions

We discuss two variations of the presented flow routing. The first has the purpose of reducing the computation effort by limiting the planning horizon. The second is an extension for taking into account forecasts on future demand.

Limited Delay Horizon

The running time of the column generation method (and also the primal-dual scheme) for solving LP 10.2 heavily depends on the congestion on the network, as we will see in the computational analysis chapter. This is because heavy traffic results in large delays which in turn translates into a large number of paths which need to be integrated into the formulation. As only one new path per commodity is added per column generation step, this also means that many column generation iterations may be necessary. Furthermore, the solution time of solving the LP in each iteration increases.

If the LP cannot be solved in the available time, one has the option of reducing the planning horizon. One simple way of doing this is to restrict the path set to dynamic paths with a delay of at most a given threshold, here called *delay horizon*. A limited delay horizon is particularly helpful when more requests are open than the network is able to cope with. Instead of solving the MCF relaxation with paths far into the future, the delay horizon gives an option of backlogging demands which can not be served in the near future. Spending much effort for planning into the future is also not desirable as the situation probably changes before the plans can be implemented.

Integrating Future Demand Forecasts

In case a forecast for future demands is available, it can potentially be helpful for estimating and taking into account the effect of current decisions on future demand. Corresponding to the interpretation of fractional flows as probabilities used in section 10.3.2, one can introduce the uncertain future requests with a demand corresponding to their occurrence probability.

Let $p_{u,v,\tau'} \in [0,1]$ be the probability for a release of a request from u to v at future time $\tau' > \tau$. One can introduce an additional commodity $\psi_{u,v,\tau'}$ for each non-zero $p_{u,v,\tau'}$ that is known. Let the set of these commodities be represented by Ψ. Each such commodity can be inserted into the MCF formulation 10.1 with a demand $p_{u,v,\tau'}$ and with the restriction that its flows are prohibited for all departure arcs corresponding to departure times prior to the release time. Otherwise, the additional commodities Ψ are treated the same way as the ones corresponding actually known open requests. The objective turns into minimizing $\sum_{\pi\in\Pi^o}\sum_{a\in\overline{A}} h_a^\pi x_a^\pi + \sum_{\psi\in\Psi}\sum_{a\in\overline{A}} h_a^\psi x_a^\psi$, where the second term relates to minimizing the expectation of the delays for future requests. Note that this extension may heavily increase the number of commodities in the MCF which may result in large computation times. Further ideas for approximating such large systems, in particular for the effects of the commodities in Ψ, may be required.

Chapter 11

Computational Analysis

This chapter contains a computational study in which we compare the performance of the presented routing algorithms experimentally. We use two scenarios for this purpose, the first is an artificial test network while the second is a scenario we received by courtesy of the PRT provider Ultra.

We will in this chapter analyze the presented algorithms with respect to a series of criteria. Delay is the main quality criterion used in this thesis. We use average delay as the core measure but also look at the delay distribution and at different types of delay. Closely related is the routing throughput. We will see that each of the algorithms has a maximal throughput above which the system gets overloaded and the number of open requests increases rapidly together with the delays. A third important measure will be the computation time as a measure for the applicability of the approaches in real-time settings.

11.1 Grid Scenario

As a first network for the computational study we use a simple grid topology. More precisely, we consider a grid of size 8×8, where each edge is subdivided by introducing three additional vertices. The edges are oriented in an alternating way as shown in Figure 11.1. We call the original 64 vertices of the grid the *grid nodes*, and the other ones *subdivision nodes*. The network consists of a total of 400 nodes and 448 arcs. Only the grid nodes are terminal nodes, and hence used as origins and destinations for requests.

Requests are randomly generated as follows. Over 1000 timesteps, the number of new requests at each timestep is an independent Poisson random variable with some parameter λ, which we call *release rate*. Hence, a release rate of e.g. $\lambda = 3$ means that three new requests appear at each of the first 1000 timesteps in average. For each request, origin and destination are chosen uniformly at random among all pairs of two different terminal nodes.

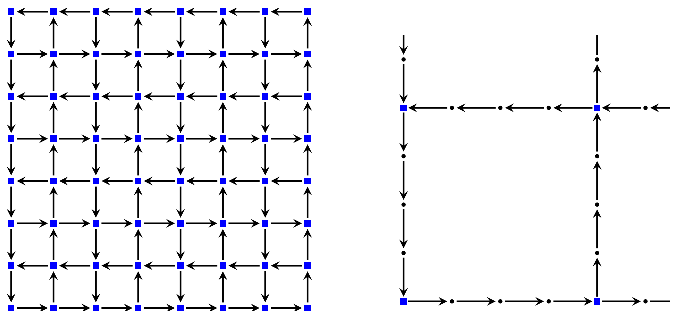

Figure 11.1: *Left:* Topology of the grid instance used for the computational study. The blue squares indicate terminal nodes (stations). *Right:* Zoom into the lower left corner.

11.1.1 Comparison of Algorithms

The plot in Figure 11.2 shows the average delay for sequential routing (*Seq*), push routing (*Push*) and the adaptive flow routing approach (*Flow*), in dependence of the release rate.

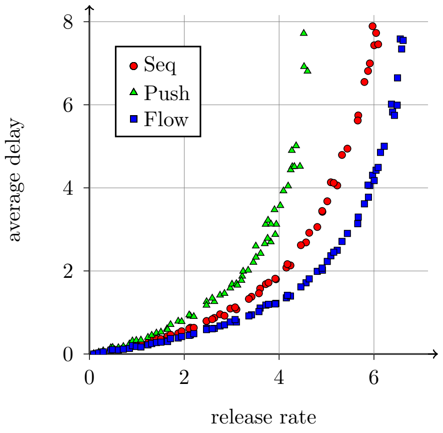

Figure 11.2: Average delay per request for different release rates for sequential routing (*Seq*), push routing (*Push*) and the adaptive flow routing approach (*Flow*).

For flow routing we use here a delay horizon (such as introduced in 10.3.3) of 5 (we will argue in Section 11.1.4 why this choice is reasonable). The underlying multicommodity flow problem is optimized using the column generation technique until an optimality gap of 5% is reached. We call CPLEX 12.3 for solving the LP formulation in each column generation iteration.

As one would expect, the three approaches perform similarly for low release rates. Due to low congestion, the potential for optimization is limited. With increasing traffic intensity, the performance differences of the three approaches become evident.

We first compare the sequential and the push routing schemes. The sequential algorithm outperforms push routing with its ability to plan resource occupations ahead into the future. Recall that Push routing sends pods that are in transit forward along a shortest path to the destination whenever there is a empty slot ahead. Similarly for departing pods, which leave the origin station whenever the first node of the route is not occupied. The push approach sends more and more pods onto the network, even in case of congestion. This increases congestion even more, and many of the pods experience large delays while queuing up for passing the next intersection ahead, similar to traffic jams in road traffic. As a consequence, large demand leads to a decrease in system throughput for the push algorithm. This is in contrast to the sequential scheme, which plans trips to the end and makes reservations for all required resources prior to departure. When Seq faces high demand, it will leave the excess pods at the origin station until a slot can be found. This way they do not occupy network resources, further congestion is avoided and no traffic jams can occur.

Comparing Seq and the adaptive Flow approaches, we observe that Flow leads to less delays in average. Again, for low release rates, the differences are small, as there are many good routes to handle new requests, even if the previously routed pods keep their reservations fixed. However, for higher release rates, adaptivity turns out to be valuable and the adaptive approach reduces average delays by roughly one third.

These observations are confirmed by Figure 11.3 where the arrival rate, i.e. the average number of pods arriving at their destinations over time, is compared to the release rate. The arrival rate is computed after cutting off the start-up (first 100 timesteps) and cool-down phases (after the release of the last request). A system is stable when the arrival rate equals the release rate. Otherwise, if the arrival rate does not match the release rate over a long time span, the number of open requests increases over time and also the delays increase rapidly. We call a release rate *stable* with respect to a network, a demand pattern and a routing algorithm if the arrival rate matches the release rate over a large number of timesteps. We call the largest stable release rate the *critical* release rate. Recall that in Chapter 7, we have derived the relaxed capacity $\overline{\varphi}$ dependent on network and demand pattern and have argued that release rates above $\overline{\varphi}$ cannot be stable, independent of which routing algorithm is applied.

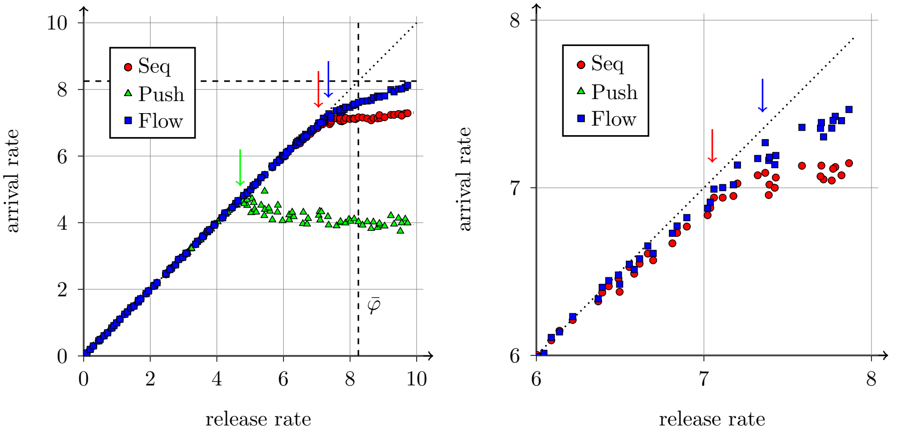

Figure 11.3: *Left:* Arrival rate against release rate for the three approaches. The system is stable if the arrival rate matches the release rate and the data points lie on the dotted diagonal line. The critical rate, where the stable operation ends, is indicated for each approach by an arrow. The horizontal and vertical dashed lines indicate the relaxed capacity $\overline{\varphi}$. *Right:* Zoom into the critical rates for Seq and Flow.

While Seq and Flow have roughly the same critical release rate, Push can handle one third less traffic demand. The arrival rate of Push even reduces with additional demand for the reasons mentioned before. Sequential routing reaches roughly 85% of the relaxed capacity while Flow reaches 89%. Above the critical rates, Seq keeps its arrival rate roughly constant while Flow can further increase its output. This can be explained with the ability of the latter to choose which part of demand it serves and which demand is backlogged. In case of excess demand, flow routing will try to serve as many requests as possible with the available resources. This results in a slower increase of the number of open requests, as can be observed in Figure 11.4.

The flow algorithm (respectively the underlying flow formulation) aims at

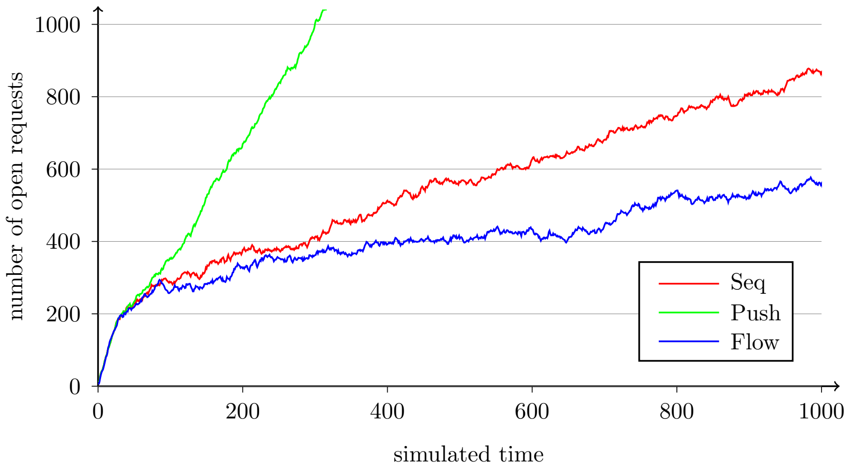

Figure 11.4: Number of open requests over simulation time. In the plotted instance, 7695 requests ($\lambda = 7.8$) are released over 1000 timesteps. The release rate is slightly above critical for Seq and Flow, and clearly above for Push. Flow uses the available network capacity more efficiently and the number of open request grows only slowly. For Push, the number of open requests increases rapidly and leave the plotted area.

routing as many requests as quickly as possible also in case of stable operation. This can be observed in the histogram of Figure 11.5. Comparing the distributions of Seq and Flow, one observes that Flow routes considerably more requests with low delays but also has a longer tail distribution and a maximum delay which is a multiple of Seq. While Seq routes the requests with priority according to the release order, Flow does not know such a prioritization. It chooses the next requests to depart such to optimize the objective of minimizing the sum of delays, resulting in a 45% decrease of the average delay for this instance compared to Seq. However, this scheme has the danger of large delays for a small part of the requests, as they might repeatedly not be part of the set of departing requests. This is in particular the case for long trips crossing highly demanded network regions.

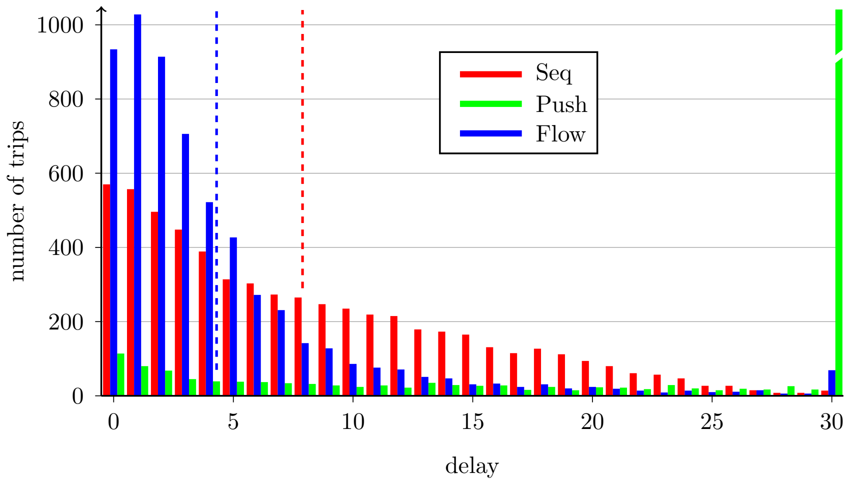

Figure 11.5: Distribution of delays for an instance with 5971 requests released over 1000 timesteps ($\lambda = 6.0$). The release rate of 6.0 is stable for Seq and Flow but unstable for Push. All request with delays ≥ 30 are cumulated in the rightmost bar. The vertical dashed lines indicate the average delays. The maximum delays are 31 for Seq, 68 for Flow and 519 for Push.

11.1.2 Delay Types

Further insights can be gained by comparing the different kinds of delays. We distinguish the three delay types *departure delay*, *waiting delays* and *detour delays*. The first refers to delays stemming from late departure, the second to delays through waiting while in transit and the third category to delays from driving a route which is not shortest. The delays split into the three categories are shown in Figure 11.6. For the departure delay, the picture is basically the same as for total delay, except that Push can roughly keep up with Seq in this category until the curve for Push explodes at its critical release rate. Flow produces less departure delays.

For the waiting delays, one observes that Seq and Flow stabilize after their critical rates. If additional delays cannot be avoided, they get assigned to pods which have not departed yet. This has the advantage that the network does not get further congested. Flow routing can again use its ability for concurrent and adaptive planning to keep the delays lower than Seq. Also push routing has a plateau in the waiting delays, as can be seen in the zoom-

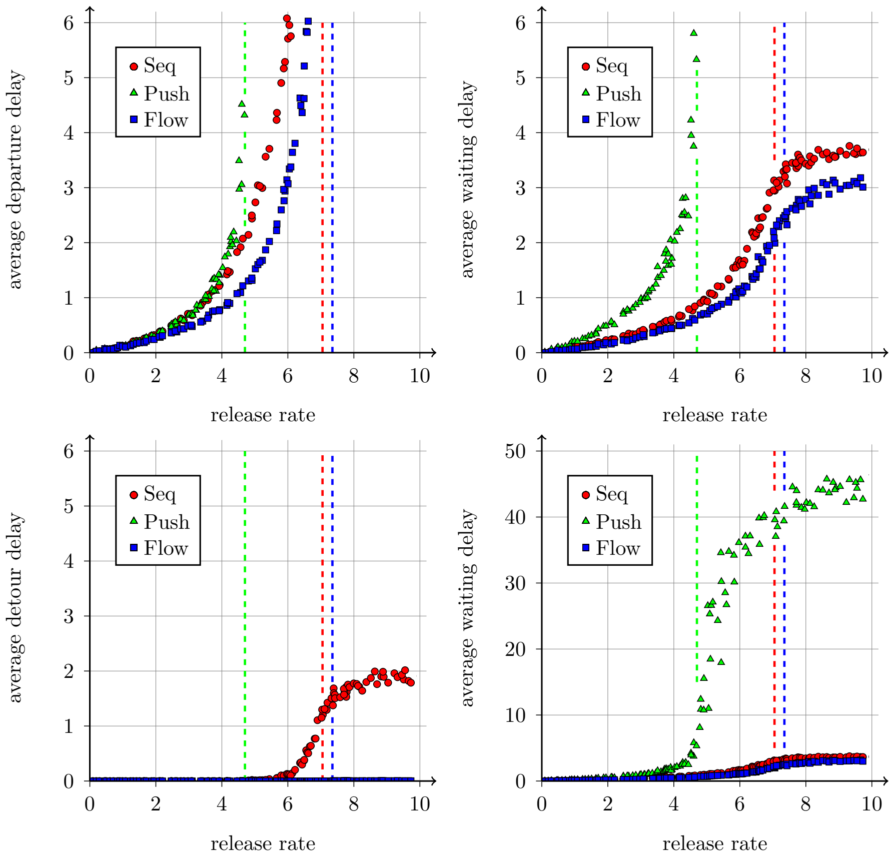

Figure 11.6: Average delays split up into the three categories departure delays (*upper left*), waiting delays (*upper right*) and detour delays (*lower left*). The vertical lines indicate the critical release rates for the three algorithms. *Lower right:* Zoom-out of the waiting delay plot, to capture the plateau of the Push curve.

out of Figure 11.6 (lower right). Here, the plateau stems from the fact that at some point the maximum congestion level of the network is reached.

Detour delays here only occur for Seq. Push has none as it uses shortest routes by design. Flow in principle can choose routes which are not shortest. However, for the present parameter choice and network topology, no detours are possible within the delay horizon. Note that the route choice is still free among all routes of shortest path length (whereof many can exist in grid networks) and that flow routing makes the assignment of pods to these in a coordinated way. Different choices for the delay horizon parameter are discussed in the next section.

11.1.3 Computation Times

The plot in Figure 11.7 shows the time needed to compute the routings. The simulations were performed by a standard laptop computer with Intel Core i7 2.7GHz dual-core processor and 4GB RAM. The computation times are to be considered with reservations, as there is potential for more efficient implementation of the algorithms, more extensive parameter tuning and more powerful hardware. Nonetheless, Figure 11.7 gives an indication of the orders of magnitude and of the dependency between traffic demand and computation time.

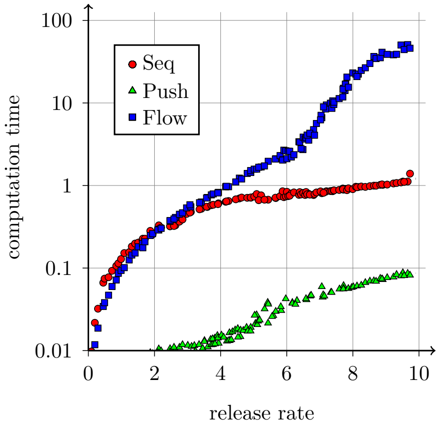

Figure 11.7: Computation time in seconds per simulated timestep for the three approaches (logarithmic scale).

Computation time of the flow algorithm increases rapidly with increasing release rate. This is due to the underlying multicommodity flow LP, which grows with increasing number of commodities (open requests) and the increasing number of paths that need to be taken into account when congestion increases. Nonetheless, the computation time for stable release rates are promising for a real-time capable implementation. For release rates outside the stability region, the computation times for Flow increase quickly.

11.1.4 Delay Horizon Trade-off in Flow Routing

Recall that the delay horizon decides in the flow routing approach on how many path alternatives are generated for each request. For pods at the origin, path alternatives which induce a larger delay than the delay horizon are not taken into account. Pods which cannot be assigned to such a path remain at the origin and are backlogged.

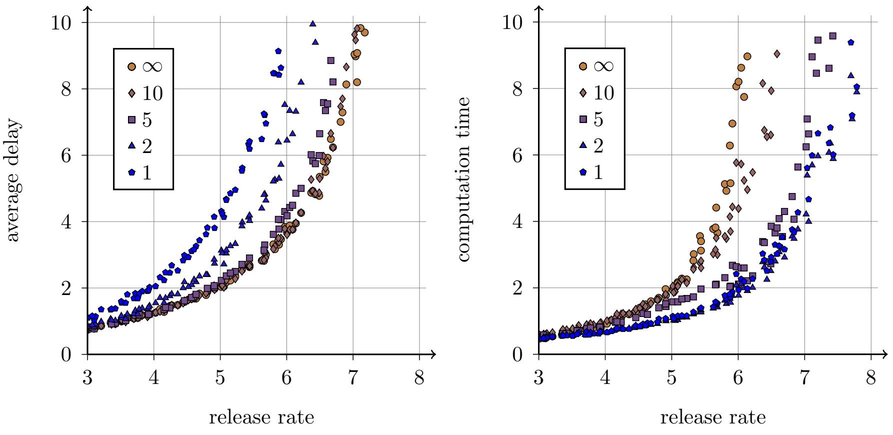

Figure 11.8: *Left:* Average delay for flow routing with different delay horizons $1, 2, 5, 10$ and ∞. *Right:* Corresponding computation times, in seconds per simulated timestep.

The delay horizon importantly influences the number of path alternatives in the multicommodity flow problem which in turn dominates the computation time. Furthermore, one can argue that planning too far into the future is not relevant, as the situation will change due to new request releases. On the other hand, a short delay horizon can diminish the coordination potential and restrict from taking into account alternative routes. The trade-off between accuracy and computation time is shown in Figure 11.8. For this, we again use column generation with a termination gap of 5%.

One observes that a delay horizon of 5 is a good compromise to the trade-off, as it provides almost the same results as delay horizon ∞ but using

delay horizon	average number of CG iterations per timestep	average number of variables in MCF LP
∞	5.28	1828.6
10	4.93	1687.2
5	3.51	1254.3
2	2.52	919.4
1	2.42	785.8

Table 11.1: Numbers of column generation (CG) iterations and numbers of path variables in the MCF problem for different delay horizons. The numbers are given for an instance with 5971 requests released over 1000 timesteps.

significantly less computation effort. The effect of the delay horizon on the computation effort is emphasized by the numbers in Table 11.1. It shows that both the numbers of column generation iterations per timestep and the path variables decrease when restricting the delay horizon. Both effects bring a reduction in solution time for the MCF relaxation.

11.1.5 Variants of Sequential Routing

In Section 10.1 we discussed the following versions of sequential routing. Direct sequential routing is the variant of Seq where no waiting is allowed for pods after departure. Shortest-path sequential in turn does not allow driving detours. A third variant results if both restrictions are combined. In Figure 11.9 we compare the average delays for Seq and its variants.

The data shows that the unrestricted sequential routing achieves the lowest delays, together with shortest-path sequential. The variants with prohibited waiting in transit loose as their set of routing alternatives is restricted. On the other hand, one can also observe that for high release rates, the variant with prohibited waiting yields the best throughput. It appears that reduced resource occupation pays off for high transit demands.

11.1.6 Optimality Gap in the Offline Case

The performance of the routing algorithms and in particular the potential for further improvement can also be studied by comparing the resulting delays to a theoretical lower bound. As a lower bound, we use here the optimal average

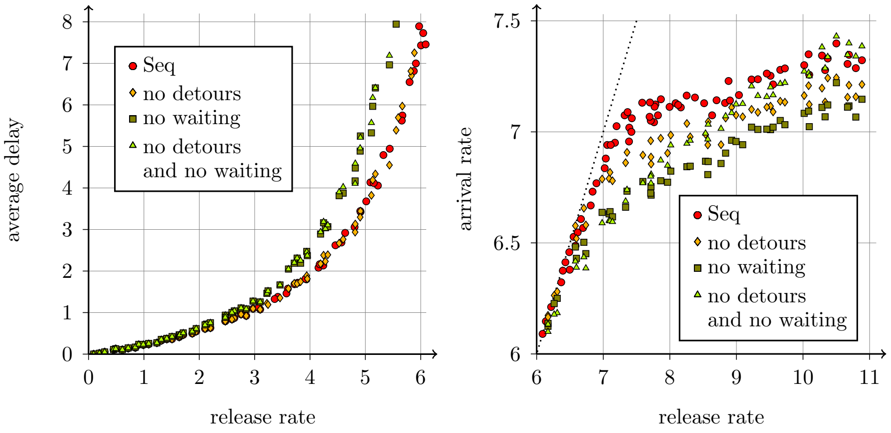

Figure 11.9: Average delays (*left*) and arrival rates (*right*) for sequential routing variants. In the second picture, the dotted line corresponds to stable arrival rates.

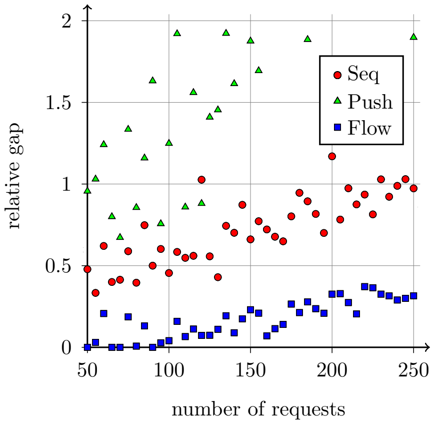

Figure 11.10: Relative gap between total delay for the three routing approaches and the lower bound from the fractional optimal MCF solution.

delay that a router could find if it knew all requests from the beginning and if it were allowed to split requests into fractions. This quantity certainly is a lower bound to what a router without these privileges can achieve and can be computed efficiently by solving the corresponding multicommodity LP.

In order to give the hypothetical router not too much of an unfair advantage we use instances in which all requests have release time zero. In this offline case all requests are known from the beginning. The scenario parameter is now the total number of requests. Except this, we use the same network and demand distribution as before. The delay horizon for Flow is again set to 5 and column generation is used until an optimality gap of 5% is reached.

The relative gap of a routing is defined as follows, where LB corresponds to the value of the lower bound.

$$\frac{\sum_{\pi\in\Pi} \mathrm{del}_\pi - \mathrm{LB}}{\mathrm{LB}}$$

The results are shown in Figure 11.10. The gap of the flow routing stems from the rounding procedure. Additionally, a part of the gap is contributed by inaccuracies from column generation termination criterion and delay horizon. One observes that the coordinated planning in Flow manages to close a large part of the optimality gap compared to sequential routing.

11.1.7 Variable Demand

This far, we have assumed that demand is stationary over time. In this section we compare the properties of routing paradigms under demand changes.

For this we introduce a peak traffic phase with double demand. We run simulations over three phases of 400 timesteps each, where the phase has release rate 3.5, the second has release rate of 7.0 and the third is again back to 3.5. Figure 11.11 shows the evolution of the number of open requests over the three phases. In the first phase, all three algorithms are stable and the number of open requests is similar at this low demand intensity. In the second phase, the number of open requests increases. For Push, the demand level is clearly instable, whereas the open requests settle at constant levels for Seq and Flow. This is expected, as the peak release rate of 7.0 is stable for Seq and Flow but not for Push, according to the observations from Figure 11.3.

The third phase is again at low intensity. Seq and Flow both recover quickly from the high demand and achieve the same low level of open requests as in the first phase. Push in contrast recovers only slowly, which is a result of the decreased throughput in congested operation observed in Figure 11.3.

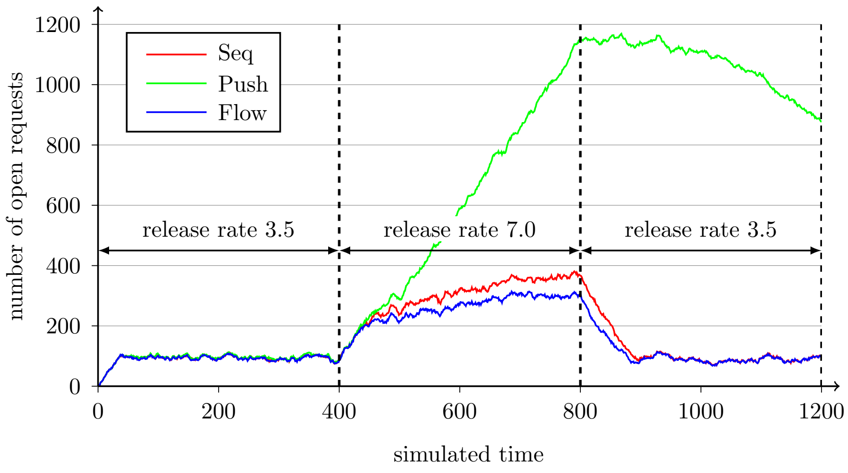

Figure 11.11: Number of open requests over time. The simulation of 1200 timesteps is split into three parts, where the second represents a peak phase with double release rate.

11.2 Case Study

The second network considered is from a case study for a new PRT system. We received the data by courtesy of Ultra, the company which developed and now operates the Heathrow PRT system. The network is shown in Figure 11.12, it consist of 60 stations and a total track length of 39.4 km. The nominal speed on the track segments is 10 m/s (7.5 m/s for track segments with high slope or curvature). Such to make the network fit into the model, we have subdivided the tracks into segments of headway length. For this, we have assumed a headway time of 5 seconds. The resulting graph consists of 702 nodes and 819 arcs.

We compare simulation results for two demand patterns, visualized in Figure 11.13. The first is part of the Ultra test scenario and represents a peak load case.

We can use the relaxed capacity derived in Chapter 7 to verify the suitability of the system layout for the given demand. The demand scenario consists of 4170 passenger trips per hour. Assuming a pod sharing rate of 1.5 passengers per pod, this corresponds to 2780 pods per hour. On the other hand, the

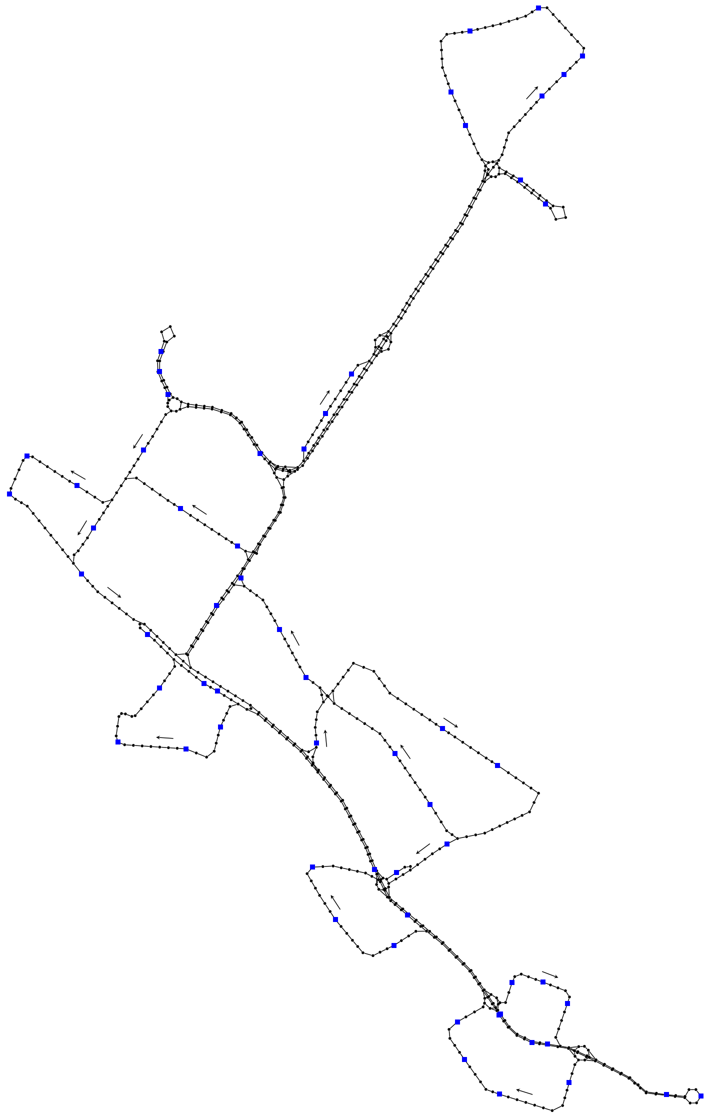

Figure 11.12: Second network topology used for the computational study. The black dots represent nodes, stations are shown in blue.

relaxed capacity for the given network and demand pattern results to be $\bar{\varphi} = 3.16$ requests per timestep. The achievable throughput for the given network and demand pattern is hence at most 3.16 times the capacity of a single line. With a headway of 5 seconds (single line capacity of 720 pods per hour), a capacity limit of 2275 pods per hour results for the network. It turns out that this capacity is not enough to match the required demand of 2780 pods per hour. A line capacity of at least 880 pods per hour is required, corresponding to a headway of less than 4.1 seconds. This example emphasizes the usefulness of the relaxed capacity for the design of PRT systems.

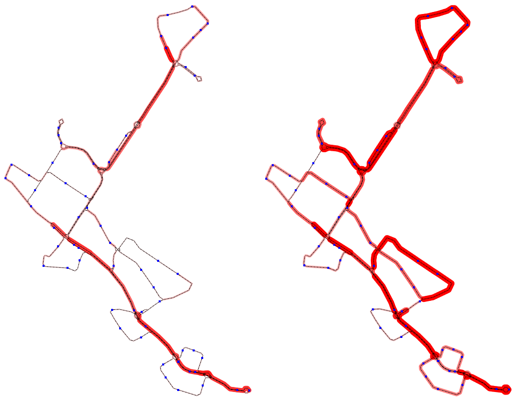
Figure 11.13: Congestion distribution. Arcs are highlighted according to the amount of flow they carry in the optimal solution of the concurrent flow problem 7.1. *Left:* For the peak demand scenario. *Right:* For uniformly distributed demands.

Figure 11.14 shows the simulation results for this scenario. Here, the performances of Seq, Push and Flow are very comparable. The reason is that the demand is concentrated in the scenario and that alternative routes are rare. The scenario contains two independent capacity bottlenecks. The best throughput is achieved by pushing pods with the highest possible rate through this bottleneck. All three investigated algorithms do this, and all are stable basically up to the theoretical capacity bound.

As a second demand pattern, we use as earlier the uniformly distributed demands with parametrized intensity. For this demand pattern the congestion is shown in Figure 11.13, right side, and the relaxed capacity is at 2.95 requests per timestep.

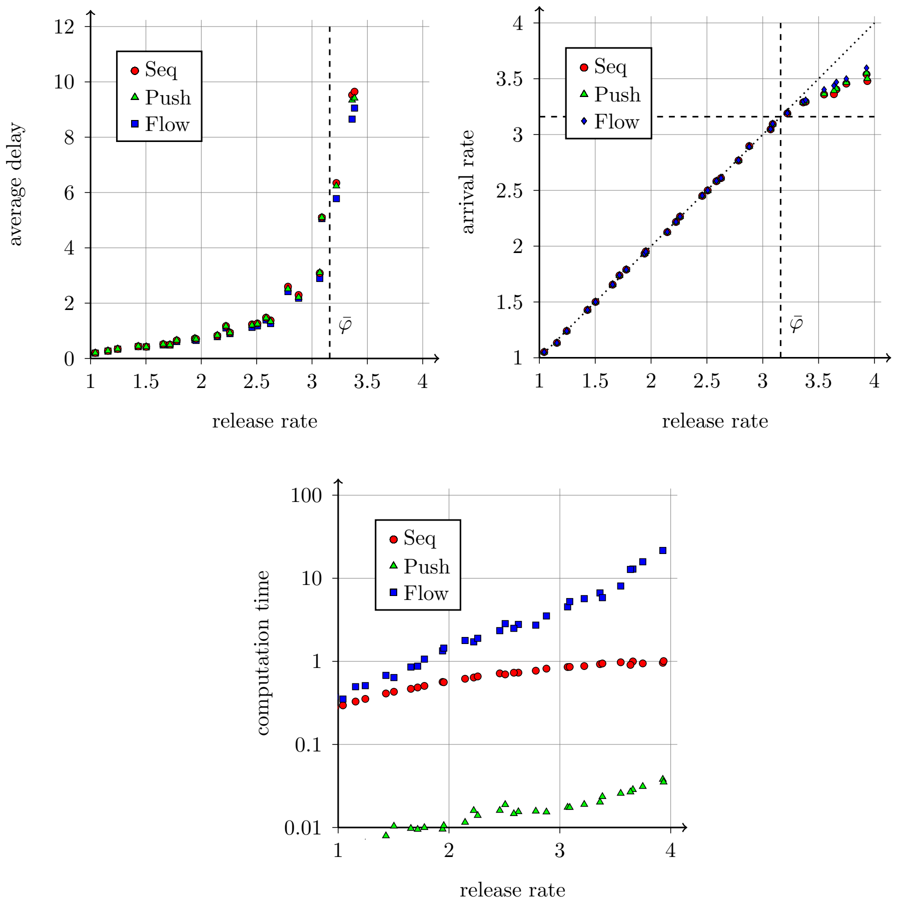

Figure 11.14: Results for the test scenario comparing Seq, Push and Flow in dependency of the release rate. *Upper left:* Average delay. *Upper right:* Arrival rate. *Below:* Computation time in seconds per timestep (logarithmic scale).

The analysis for this case is shown in Figure 11.15. Here, the differences between the algorithms are more pronounced. At release rates of about 80 to 90 percent of the theoretical capacity bound, flow routing achieves average delays which are up to one third better than for the other two algorithms. At release rates above the capacity bound, we observe the same effects as for the grid scenarios. Seq keeps the arrival rate constant, Flow routes as many requests as possible and Push gets congested and the arrival rate decreases. Also in this case we would like to emphasize that the relaxed capacity seems to predict the achievable performance well.

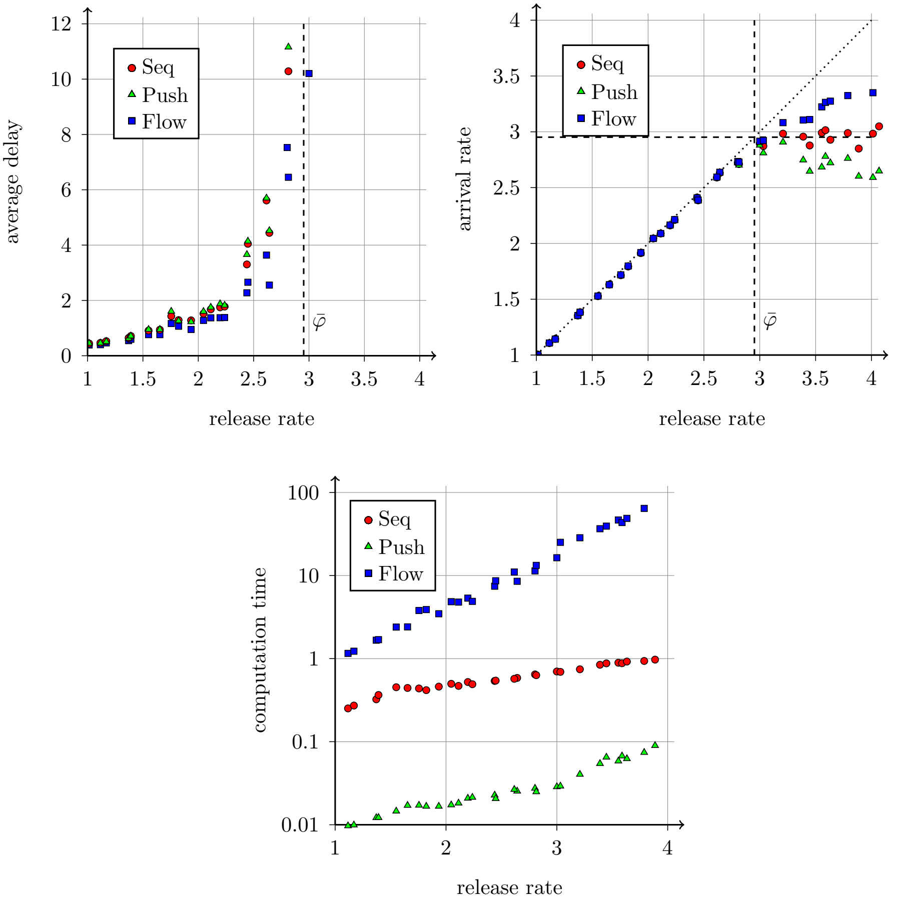

Figure 11.15: Results for the test scenario with uniform demand pattern. *Upper left:* Average delay. *Upper right:* Arrival rate. *Below:* Computation time in seconds per timestep (logarithmic scale).

Chapter 12

Conclusion

In this part of the thesis, we have presented a model suitable for the investigation of online routing algorithms. The model has been designed in particular for routing in Personal Rapid Transit. We have in this research also tried to establish the link between the research fields of PRT routing, AGV routing and combinatorial optimization in general.

The first contribution of this project is a better understanding of the capacity of a PRT network. Much literature is available on the capacity of a single track line. Our contribution is to suggest a method for computing the relaxed capacity as an upper bound on the capacity of a track network. The relaxed capacity is measured in multiples of the line capacity and is independent of design and system-specific parameters. We have for this purpose used a standard flow formulation. The results show that the relaxed capacity is close to the achievable throughput; in the test scenarios we observed a gap of around 10% or smaller. We therefore believe that this quantity is helpful for estimating network capacity in the design stage of new PRT systems. For the case study received from Ultra, for example, we could conclude that a headway time of less than 4.1 seconds is necessary in order for the network to cope with the demand.

The second main contribution is the evaluation of routing schemes and in particular of the benefit of adaptivity and coordination in routing algorithms. For this purpose, we have introduced a new adaptive algorithm based on a flow formulation through a time-expanded network. Also, we have reviewed the well-known sequential routing scheme and the push routing algorithm as a representative of the asynchronous routing schemes. With respect to the computational complexity of PRT routing, we could show that it is NP-hard to find the optimal solution. There is hence little hope for finding methods computing an exact solution in useful time and the focus needs to be shifted to approximation algorithms and heuristics.

The computational results allow several conclusions.

i) Sequential routing is a comparably simple and fast scheme, and its results are solid. It performed well in all computations and has almost no limits in the problem size it can handle. Its maximum throughput is close to the achievable capacity and remains so in case of excess demand.

ii) The flow routing scheme is able to outcompete sequential routing in some cases. The adaptive and concurrent planning leads to considerably less delays. Also the throughput is higher in situations with excess demand. In other cases we have observed that the benefit of concurrent planning is limited and that the results of flow and sequential routing are comparable. Flow routing is a relatively heavy approach as it requires a large amount of computations for taking concurrent routing decisions repeatedly. With advanced multicommodity flow solvers and more rigorous tuning, we expect that the computational efficiency can be further increased.

iii) The advantage of push routing is that it is conceptually simple, very fast and can be implemented in a distributed way such that no central control is necessary. However, the results show that push routing is not competitive with the two other approaches as it quickly gets overcongested and real traffic jams can build up. We believe that this is an inherent problem of asynchronous (distributed) schemes, as the lack of a central controller makes planning and efficient use of the available resources difficult.

In summary, one can conclude that both the adaptive flow algorithm and sequential routing have their strengths and a choice between them needs careful analysis of the requirements.

This project has shown that adaptivity can lead to considerable reductions in passenger delays. We have presented an adaptive algorithm exploiting this potential and yielding results close to theoretical bounds. This result can be the incentive for developing other, possibly simpler and faster, adaptive algorithms. An idea into this direction would be to combine sequential routing with adaptivity and to benefit from the advantages of both.

Another direction for future improvements is the choice of the objective function. A concurrent approach requires a measure for the concurrent quality of a solution. It needs to be able to trade-off additional delays for some pods against the earlier arrival of others. We have used here the unweighted sum of delays as a measure treating each request with equal priority. However, it turns out that some requests can experience very large delays in this setting. Requests which are for some reason difficult to route may get deferred more and more into the future. Even if consistent with the objective function, this behavior is not desirable. There is need for more investigation in this area, ideas go towards penalizing large delays in the objective or measuring delays relatively to the trip length. In the end, there is also an almost ethical question that one needs to answer: how much delay may be charged to one group of passengers if a second group can benefit in turn?

Concluding, we hope that this dissertation can support the current exciting developments towards large-scale implementations of PRT with advanced methods from operations research. Our focus in this work was clearly limited to the routing aspects in PRT. More work is necessary to integrate other aspects such as empty vehicle routing, station handling and maintenance management. All these challenges have in common that the possibility for central control allows for coordinated and optimized operation. The success of larger-scale systems will depend on the question whether system design and operation lead to a level of service which is able to keep the promises and is competitive with other transportation modes.

Bibliography

[2ge11] 2getthere. Personal Rapid Transit. URL: `http://www.2getthere.eu/?page_id=58`, accessed December 11, 2011.

[ABN08] I. Abraham, Y. Bartal, and O. Neiman. Nearly tight low stretch spanning trees. In *Foundations of Computer Science, FOCS '08*, pages 781–790, 2008.

[ABR10] J.A.D. Atkin, E.K. Burke, and S. Ravizza. The airport ground movement problem: Past and current research and future directions. In *4th International Conference on Research in Air Transportation, ICRAT*, 2010.

[adHS95] F. Meyer auf der Heide and C. Scheideler. Routing with bounded buffers and hot-potato routing in vertex-symmetric networks. In *Proceedings of the Third Annual European Symposium on Algorithms, ESA*, pages 341–354, 1995.

[Alb01] C. Albrecht. Global routing by new approximation algorithms for multicommodity flow. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20:622 –632, 2001.

[AMO93] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows.* Prentice Hall, 1993.

[And98] J.E. Anderson. Control of Personal Rapid Transit systems. *Journal of Advanced Transportation*, 32:57–74, 1998.

[And03] I.J. Andréasson. Reallocation of empty Personal Rapid Transit vehicles en route. *Transportation Research Record*, 1838:36–41, 2003.

[And09] I.J. Andréasson. Extending PRT capabilities. In *Proceedings of the 12th APM conference ASCE*, 2009.

[ATR89] Personal Rapid Transit (PRT); Another option for urban transit? Technical report, Advanced Transit Association Inc. (ATRA), 1989.

[ATR03] Status and potential of Personal Rapid Transit, ridership analysis. Technical report, Advanced Transit Association Inc. (ATRA), 2003.

[ATR11] Personal Rapid Transit definition. Available online: `http://www.prtconsulting.com/definition.html`, accessed December 11, 2011.

[BMIMS04] C. Busch, M. Magdon-Ismail, M. Mavronicolas, and P. Spirakis. Direct routing: Algorithms and complexity. In *Proceedings of the 12th Annual European Symposium on Algorithms, ESA*, pages 134–145, 2004.

[BT05] P.H. Bly and P. Teychenne. Three financial and socio-economic assessments of a Personal Rapid Transit system. In *Proceedings of the Tenth International Conference on Automated People Movers*, pages 39+, 2005.

[CG11] M. Castangia and L. Guala. Modelling and simulation of a PRT network. *17th International Conference on Urban Transport and the Environment*, pages 459–472, 2011.

[CH07] J.A. Carnegie and P.S. Hoffman. Viability of Personal Rapid Transit in New Jersey. Technical report, presented to New Jersey State Legislature, 2007.

[Cha11] S. Chaturvedi. PRT promises to improve city commutation in Gurgaon, Amritsar. URL: `http://www.governancenow.com/news/regular-story/prt-promises-improve-city-commutation-gurgaon-amritsar`, Accessed Dec 08, 2011.

[Con12] PRT Consulting. Personal Rapid Transit capacity. Available online: `http://www.prtconsulting.com/docs/MullerPersonalRapidTransitCapacity.pdf`, accessed January 04, 2012.

[Coo92] S. Cook. The P versus NP problem. Available online: `www.claymath.org/prizeproblems/pvsnp.htm`, 1992.

[CVZ10] C. Chekuri, J. Vondrák, and R. Zenklusen. Dependent randomized rounding via exchange properties of combinatorial structures. In *Proceedings of the 51st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 575–584, 2010.

[DGR06] K. Dhamdhere, A. Gupta, and H. Räcke. Improved embeddings of graph metrics into random trees. In *Proceedings of the seventeenth annual symposium on discrete algorithms, SODA*, pages 61–69, 2006.

[DSE09] V.L. Dos Santos Eleutério. *Finding Approximate Solutions for Large Scale Linear Programs.* PhD thesis, ETH Zurich, 2009.

[EU11] White paper - roadmap to a single european transport area - towards a competitive and resource efficient transport system. Technical report, European Commission, 2011.

[FF58] L.R. Ford, Jr. and D.R. Fulkerson. Constructing maximal dynamic flows from static flows. *Operations Research*, 6:419–433, 1958.

[FF62] L.R. Ford, Jr. and D.R. Fulkerson. *Flows in Networks.* Princeton University Press, 1962.

[Fic64] D. Fichter. *Individualized automatic transit and the city.* published by B.H. Sikes, 1964.

[Fle00] L.K. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM Journal on Discrete Mathematics*, 13:505–520, 2000.

[FLKH05] R. Freling, R.M. Lentink, L.G. Kroon, and D. Huisman. Shunting of passenger train units in a railway station. *Transportation Science*, 39:261–272, 2005.

[FS02] L.K. Fleischer and M. Skutella. The quickest multicommodity flow problem. In *6th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 36–53, 2002.

[FS06] L.K. Fleischer and M. Skutella. The quickest multicommodity flow problem. In *Integer Programming and Combinatorial Optimization*, volume 2337 of *Lecture Notes in Computer Science*, pages 36–53. Springer-Verlag, 2006.

[GBM$^+$02] J. Garcia, A. Berlanga, J.M. Molina, J.A. Besada, and J.R. Casar. Planning techniques for airport ground operations. In *Proceedings of 21st Digital Avionics Systems Conference*, 2002.

[GHS98] T. Ganesharajah, N.G. Hall, and C. Sriskandarajah. Design and operational issues in AGV-served manufacturing systems. *Annals of Operations Research*, 76:109–154, 1998.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1979.

[GK98] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proceedings of the 39th Symposium on Foundations of Computer Science (FOCS)*, pages 300–309, 1998.

[GKPS06] R. Gandhi, S. Khuller, S. Parthasarathy, and A. Srinivasan. Dependent rounding and its applications to approximation algorithms. *ACM*, 53:324–360, 2006.

[HHS07] A. Hall, S. Hippler, and M. Skutella. Multicommodity flows over time: Efficient algorithms and complexity. *Theoretical Computer Science*, 379:387 – 404, 2007.

[HUD68] Tomorrow's transportation; new systems for the urban future. Technical report, Ministry of Housing and Urban Development (HUD) and Urban Mass Transit Association (UMTA), 1968.

[IBOB77] J.H. Irving, H. Bernstein, C.L. Olson, and J. Buyan. *Fundamentals of Personal Rapid Transit.* Lexington Books, 1977.

[ITH10] Feasibility of PRT in Ithaca, New York - executive summary. Technical report, New York State Department of Transportation, 2010.

[Joh05] R.E. Johnson. Doubling Personal Rapid Transit capacity with ridesharing. *Transportation Research Record: Journal of the Transportation Research Board*, pages 107–112, 2005.

[KBK93] N.N. Krishnamurthy, R. Batta, and M.H. Karwan. Developing conflict-free routes for automated guided vehicles. *Operations Research*, 41:1077–1090, 1993.

[KJR07] K. Kim, S. Jeon, and K. Ryu. Deadlock prevention for automated guided vehicles in automated container terminals. In *Container Terminals and Cargo Systems*, pages 243–263. Springer Berlin Heidelberg, 2007.

[KKH08] Beyond oil: Shanghai. Technical report, School of Architecture, Royal University College of Fine Arts, Stockholm, 2008.

[KM75] A.L. Kornhauser and P. McEvaddy. A quantitative analysis of synchronous vs. quasi-synchronous network operations of automated transit systems. *Transportation Research*, 9:241–248, 1975.

[KPSW09] R. Koch, B. Peis, M. Skutella, and A. Wiese. Real-time message routing and scheduling. In *APPROX-RANDOM*, pages 217–230, 2009.

[KT91] Chang W. Kim and J.M.A. Tanchoco. Conflict-free shortest-time bidirectional AGV routeing. *International Journal of Production Research*, 1991.

[Leu04] J.Y-T. Leung. *Handbook of scheduling: algorithms, models, and performance analysis.* Chapman Hall, 2004.

[LM11] J.D. Lees-Miller. *Empty Vehicle Redistribution for Personal Rapid Transit.* PhD thesis, University of Bristol, 2011.

[LMHD09] J.D. Lees-Miller, J.C. Hammersley, and N. Davenport. Ride sharing in Personal Rapid Transit capacity planning. In *Automated People Movers 2009*, pages 321–332, 2009.

[LMHW10] J.D. Lees-Miller, J.C. Hammersley, and R.E. Wilson. Theoretical maximum capacity as benchmark for empty vehicle redistribution in Personal Rapid Transit. *Transportation Research Record*, 2010.

[LMR94] T. Leighton, B. Maggs, and S. Rao. Packet routing and job-shop scheduling in O(congestion + dilation) steps. In *Combinatorica*, volume 14, pages 167–180, 1994.

[Low03] M. Lowson. Service effectiveness of PRT vs collective-corridor transport. *Journal of Advanced Transportation*, 37:231–241, 2003.

[Mar05] D. Marx. A short proof of the NP-completeness of minimum sum interval coloring. *Operations Research Letters*, 33:382–384, 2005.

[MKGS05] R.H. Möhring, E. Köhler, E. Gawrilow, and B. Stenzel. Conflict-free real-time AGV routing. In *Proceedings of Operations Research*, pages 18–24, 2005.

[MS11] K. Müller and S.P. Sgouridis. Simulation-based analysis of Personal Rapid Transit systems: service and energy performance assessment of the Masdar City PRT case. *Journal of Advanced Transportation*, 45:252–270, 2011.

[NSS99] S. Nicoloso, M. Sarrafzadeh, and X. Song. On the sum coloring problem on interval graphs. *Algorithmica*, 23:109–126, 1999.

[Oel08] M. Oellrich. *Minimum-Cost Disjoint Paths Under Arc Dependences - Algorithms for Practice.* PhD thesis, Technische Universität Berlin, 2008.

[PSW09] B. Peis, M. Skutella, and A. Wiese. Packet routing: Complexity and algorithms. In *Proceedings of the 7th Workshop on Approximation and Online Algorithms, WAOA*, 2009.

[PT88] E.R. Petersen and A.J. Taylor. An optimal scheduling system for the Welland canal. *Transportation Science*, 22:173, 1988.

[Rub75] F. Rubin. Routing algorithms for urban rapid transit. *Transportation Research*, 9:215–223, 1975.

[RVVN90] P.I. Rivera-Vega, R. Varadarajan, and S.B. Navathe. Scheduling data redistribution in distributed databases. In *Proceedings of the Sixth International Conference on Data Engineering*, pages 166–173, 1990.

[SAN09] Request for proposal for San José automated transit network FFRDC development services. Technical report, Department of Transportation of the City of San José, 2009.

[Sch98] C. Scheideler. *Universal Routing Strategies for Interconnection Networks.* Springer-Verlag, 1998.

[Sch03] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency.* Springer-Verlag, 2003.

[SKM08] Daventry PRT scoping study, phase 2 report. Technical report, Sinclair Knight Merz Pty Ltd., 2008.

[SM07] J. Schweizer and L. Mantecchini. Performance analysis of large scale PRT networks: Theoretical capacity and microsimulations. Technical report, Bologna University, 2007.

[Spe06] I. Spenke. *Complexity and Approximation of Static k-splittable Flows and Dynamic Grid Flows.* PhD thesis, Technische Universität Berlin, 2006.

[ST97] A. Srinivasan and C. Teo. A constant-factor approximation algorithm for packet routing, and balancing local vs. global criteria. In *Proceedings of the ACM Symposium on the Theory of Computing, STOC*, pages 636–643, 1997.

[Ste08] B. Stenzel. *Online Disjoint Vehicle Routing with Application to AGV Routing.* PhD thesis, Technische Universität Berlin, 2008.

[Suk11] S. Sukayna. Gurgaon eyes elevated pods. *Times of India*, 2011.

[SV08] R. Stahlbock and S. Voß. Operations research at container terminals: a literature update. *OR Spectrum*, 30:1–52, 2008.

[SWE09] Podcars - new travel on track. Technical report, Swedish Ministry of Enterprise, Energy and Communications and Svensk Information, 2009.

[SZ11] K. Schüpbach and R. Zenklusen. Approximation algorithms for conflict-free vehicle routing. In *Proceedings of the 18th Annual European Symposium on Algorithms (ESA)*, pages 640–651, 2011.

[SZ12] K. Schüpbach and R. Zenklusen. An adaptive routing approach for personal rapid transit. Technical report, ETH Zürich, 2012.

[Szk99] T. Szkaliczki. Routing with minimum wire length in the dogleg-free manhattan model is NP-complete. *SIAM Journal on Computing*, 29:274–287, 1999.

[TA03] G. Tegnér and I.J. Andréasson. Personal automated transit for Kungens Kurva, Sweden - a PRT system evaluation within the EDICT project. In *9th APM Conference*, 2003.

[THA$^+$07] G. Tegnér, M. Hunhammar, I.J. Andréasson, J.E. Nowack, and K. Dahlström. PRT in Sweden: From feasibility studies to public awareness. In *Proceedings of 11th International Conference on Automated People Movers*, 2007.

[ULT09] Advanced Transport Systems ultra PRT. Available online: `http://www.ultraprt.com/uploads/Documents/ULTraDescriptionOct09.pdf`, accessed May 1st, 2009.

[ULT11] Ultra PRT. ULT: `http://www.ultraglobalprt.com`, accessed December 11, 2011.

[Vis06] I.F.A. Vis. Survey of research in the design and control of automated guided vehicle systems. *European Journal of Operational Research*, 170:677–709, 2006.

[Wad73] R.M. Wade. The Manhattan project: a cost oriented control system for a large Personal Rapid Transit network. *IBM Corp., Technical Report*, 1973.

[Wer10] S. Werren. Feasibility of the conflict-free routing problem. Technical report, Semester thesis at ETH Zürich, 2010.

[WH10] D. Wynn and N. Hill. EU Transport GHG: Routes to 2050 - review of potential radical future transport technologies and concepts. Technical report, prepared under a contract between the European Commission and AEA Thechnology, 2010.

[Xit08] C. Xithalis. Synchronous control method for Personal Rapid Transit systems. In *10th International Conference on Application of Advanced Technologies in Transportation*, 2008.

[ZJM09] P. Zheng, D. Jeffery, and M. McDonald. Development and evaluation of traffic management strategies for Personal Rapid Transit. In *Industrial Simulation Conference*, 2009.

Short Curriculum Vitae

Kaspar Schüpbach

born on June 22, 1981

from Landiswil BE, Switzerland

since 05/2008	**Doctoral studies** ETH Zurich Doctoral student and teaching assistant at the Institute for Operations Research
2006 – 2007	**Master Studies** ETH Zurich MSc in Computational Science and Engineering
2001 – 2005	**Bachelor Studies** ETH Zurich BSc in Computational Science and Engineering
1996 – 2001	**High School** Kantonsschule Büelrain, Winterthur