Robust Routing in Urban Public Transportation Networks

A thesis submitted to attain the degree of
DOCTOR OF SCIENCES of ETH ZÜRICH
(Dr. sc. ETH Zurich)

presented by
Tobias Pröger
Dipl.-Inform., Technische Universität Dortmund
born on September 5, 1987
citizen of Germany

accepted on the recommendation of
Prof. Dr. Peter Widmayer, examiner
Prof. Dr. Joachim M. Buhmann, co-examiner
Prof. Dr. Anita Schöbel, co-examiner

2016
Abstract

In public transportation, it is a well-studied problem to compute journeys that leave a given departure stop \(d\) at a given departure time \(t_D\), and that are planned to reach a given target stop \(t\) as early as possible. A related problem asks, given a departure stop \(d\), a target stop \(t\) and a latest allowed arrival time \(t_A\), to compute a journey that is planned to leave \(d\) as late as possible and reaches \(t\) on time (i.e., not later than \(t_A\)). If the real arrival and departure times of each vehicle correspond to the planned ones then it is certainly sufficient to use the planned timetable for computing such journeys.

In reality, however, delays in public transportation are omnipresent due to traffic congestions, road or track work, passengers boarding and leaving vehicles, etc. Hence, the real vehicle movements often deviate substantially from the scheduled ones. This is problematic because not only does the travel time of single vehicles increase, but different arrival times may also lead to failed transfers between vehicles of different lines, increasing further the overall travel time. The first part of this thesis therefore studies the problem of finding journeys that are \textit{robust} against delays. More concretely, we assume that a departure stop \(d\), a target stop \(t\) and a latest allowed arrival time \(t_A\) are given, and our goal is compute journeys that likely reach \(t\) on time.

Instead of modeling delays explicitly by artificial probability distributions, we will use historic delay data and look for journeys that performed well w.r.t. the given past data. For this purpose we develop a solution concept that separates the topological structure (i.e., the lines and the transfers used) from concrete travel times, and propose a three-stage approach for finding robust journeys. As a first step, we generate all \(dt\)-routes (i.e., feasible sequences of lines) whose number of transfers do not exceed a given threshold. After that, we compute one or more robust route(s) and corresponding departure times. Finally, appropriate transfers are computed using the planned timetable.

We will develop various methods for computing robust routes. Simple methods ensure robustness only by adding additional buffer time at each transfer. A more sophisticated method, originally proposed by Buhmann et al. (2013) for combinatorial optimization, will be adapted to public transportation. Since this method does only take historic data of two different days into account where in reality often more data is available, we study possible generalizations of this method to incorporate data of multiple past days. We will also adapt a mean-risk model originally proposed by Lim et al. (2012) for private transportation to public transportation. This method chooses routes that minimize a weighted sum of the mean and the standard deviation of the minimal times.
one had to leave in advance, where times are measured w.r.t. the data of the past days. We will then experimentally evaluate these methods on real-world data of the public transportation network of Zürich.

In the second part of the thesis, we will further investigate the aforementioned method by Buhmann et al. (2013) for robust optimization. In this method it is assumed that a problem generator generates similar problem instances that differ slightly due to noise. Assuming that nothing is known about the type of the noise or the problem generator itself except for exactly two instances generated by it, the goal is to identify solutions that are likely to perform well for future, yet unknown instances generated by the same generator. For this purpose, Buhmann et al. (2013) introduced the concept of a $\gamma$-approximation set which contains all $\gamma$-optimal solutions. They then compute the $\gamma$ that maximizes the similarity of the given two instances, which relates the actual size of the intersection of the $\gamma$-approximation sets to the expected number of solutions in the intersection. In this thesis we will propose and experimentally evaluate models in which this method outperforms other methods. As a first step towards a formal investigation of the method, we will give an analytic expression to estimate the similarity of two instances, and experimentally evaluate the quality of the estimation. Furthermore, we will discuss possible generalizations to incorporate multiple input instances.
Ein oft untersuchtes Problem im öffentlichen Verkehr ist die Berechnung von Reisen, die einen gegebenen Abfahrtsort $d$ zu einer gegebenen Abfahrtszeit $t_D$ verlassen, und die einen gegebenen Zielort $t$ geplant so früh wie möglich erreichen. In einem verwandten Problem soll für einen gegebenen Abfahrtsort $d$, einen gegebenen Zielort $t$ und eine Zielankunftszeit $t_A$ eine Reise berechnet werden, die $d$ so spät wie möglich verlässt und $t$ dennoch pünktlich (d.h., nicht später als $t_A$) erreicht. Stimmen die realen Ankunfts- und Abfahrtszeiten jedes Fahrzeugs mit den geplanten überein, dann ist es sicherlich ausreichend, den Fahrplan als Berechnungsgrundlage für solche Reisen zu benutzen.

In der Realität sind Verspätungen allerdings aufgrund von Verkehrsunfallen, Straßen- oder Gleisbauarbeiten, ein- oder aussteigenden Passagieren, usw., allgegenwärtig. Daher weichen die realen Bewegungen einzelner Fahrzeuge von den geplanten deutlich ab. Dies ist problematisch, da sich nicht nur die Reisezeit einzelner Fahrzeuge erhöht, sondern geänderte Ankunftszeiten auch zu verpassten Anschlüssen zu Fahrzeugen anderer Linien führt, was die Gesamtreisezeit weiter erhöht. Der erste Teil der vorliegenden Arbeit untersucht daher das Problem der Berechnung von Reisen, die robust gegenüber Verspätungen sind. Konkret nehmen wir an, dass ein Abfahrtsort $d$, ein Zielort $t$ und eine Zielankunftszeit $t_A$ gegeben sind, und unsere Aufgabe besteht in der Berechnung von Reisen die $t$ wahrscheinlich pünktlich erreichen.

Anstatt Verspätungen explizit durch künstliche Wahrscheinlichkeitsverteilungen zu modellieren, werden wir historische Verspätungsdaten benutzen, und Reisen suchen die gute Eigenschaften in Bezug auf die gegebenen Daten aufweisen. Zu diesem Zweck entwickeln wir ein Lösungskonzept das die topologische Struktur (d.h., benutzte Linien und Umstiege) von konkreten Reisezeiten trennt, und schlagen einen dreistufigen Ansatz zur Berechnung robuster Reisen vor. In einem ersten Schritt generieren wir alle $dt$-Routen (d.h., zulässige Liniensequenzen), deren Anzahl von Umstiegen einen vorgegebenen Grenzwert nicht übersteigt. Danach berechnen wir eine oder mehrere Route(n) und entsprechende Abfahrtszeiten. Abschliessend benutzen den Fahrplan zur Berechnung geeigneter Umstiege.

sind, kann die genannte Methode lediglich die Daten zweier vergangener Tage einbeziehen. Wir werden daher mögliche Verallgemeinerungen der Methode untersuchen, sodass die Daten mehrerer vergangener Tage einbezogen werden können. Weiterhin werden wir ein Risikomodell, ursprünglich von Lim et al. (2012) für den privaten Verkehr entwickelt, in unser Szenario übertragen. Diese Methode wählt Routen aus, die eine gewichtete Summe des Mittelwerts und der Standardabweichung der minimalen Zeiten, um die man in Bezug auf die Daten der vergangenen Tage im Voraus hätte abfahren müssen, minimieren. Wir werden die Methoden dann experimentell auf echten Verspätungsdaten des ÖPNV-Netzes von Zürich vergleichen.

Acknowledgments

Many people accompanied me during the last years while working on this PhD thesis. First, I would like to thank Peter Widmayer for giving me the opportunity to work in his group and obtain a PhD here. I am deeply thankful for his support and motivation, in particular that he often spontaneously had time for helpful discussions, and in general for the great working atmosphere.

I am also grateful to my co-examiners Joachim M. Buhmann and Anita Schöbel who agreed to read the thesis and examine me, and who also gave me valuable feedback.

Research lives from the collaboration with other people. Therefore I wish to thank my co-workers Kateřina Böhmová, Joachim M. Buhmann, Alexey Gronskiy, Matúš Mihalák, Gustavo Sacomoto, Marie-France Sagot, Rastislav Šrámek, and Peter Widmayer for many fruitful discussions and the results we finally obtained. Testing how algorithms behave on real-world data is always interesting and important. Unfortunately usually it is hard to obtain such data. Therefore I am particularly thankful to the Verkehrsbetriebe Zürich, in particular to Peggy Neubert, for providing and explaining real-world tram and bus data from Zürich without hesitation.

Large parts of this thesis were proof-read by Alexey Gronskiy, Sandro Montanari, Sascha Pampus and Pavel Stoimenov. I am grateful to them for their willingness to spend their time, and for the critical feedback they gave me.

I also would like to thank the members of my group and of the Institute of Theoretical Computer Science in general for supporting the friendly working atmosphere. In particular I would like to thank Kateřina Böhmová, Alexey Gronskiy, Sandro Montanari and Paolo Penna.

It is not obvious that people recognize the beauty of mathematics. Therefore I wish to thank my former math teacher Mr. Bergkemper for showing this to me. Last, I wish to say thanks to my friends and family, in particular to Eva, Isabell, Lisa and $\sqrt{Tobias^2}$ for their great support, in particular during the last years.
# Contents

1 Introduction 1  
1.1 Motivation 1  
1.2 Mathematical Preliminaries 3  
1.3 Outline of the Thesis 4  

2 Modeling Urban Public Transportation Networks 5  
2.1 Related Work 5  
2.2 Stops, Lines, and Transportation Networks 9  
2.3 Routes 10  
2.4 Trips and Timetables 11  
2.5 Solution Concept, Approach and Outlook 13  
2.6 Modeling Issues 17  
2.7 Implementation Details 20  
2.8 Conclusion 20  

3 Listing all Solutions 23  
3.1 Related Work 23  
3.2 Computing the Shortest $dt$-Routes and $L$-distances 25  
3.3 A Straightforward Solution for Listing Routes 29  
3.4 A Polynomial Delay Algorithm for Listing Routes 32  
3.5 A Faster Solution with Preprocessing 34  
3.6 Listing all Paths 35  
3.7 Conclusion 42  

4 Measuring the Robustness of Routes 45  
4.1 Related Work 45  
4.2 Computing Journeys from Routes 48  
4.3 Transfer Buffers 54  
4.4 A Similarity-Based Approach 55  
4.5 Function-Based Approaches 59  
4.6 Conclusion 61  

5 Experimental Evaluation 63  
5.1 Input Data 63  
5.2 Situation in Zürich 66  
5.3 Experimental Setup 67  
5.4 Arrival Rate, Departure Time, Standard Deviation 69
1. Motivation

Finding reasonable journeys in public transportation networks is a natural and long-studied problem, and information systems to compute such journeys for users are available since a long time. More concretely, when traveling from a departure stop \( d \) to a target stop \( t \), one often seeks for journeys that reach \( t \) as early as possible. Other important criteria to minimize are, for example, the number of transfers between vehicles or the overall cost of a journey.

Sometimes, however, the reliability of a journey is the most important objective to a user. For example, if there is some really important event (e.g., a meeting or the departure of a plane) taking place at \( t \) at a given time, then one usually is interested in a reliable journey from \( d \) to \( t \) that reaches \( t \) no later than at time \( t_A \). Now, determining the right moment to leave \( d \) is nontrivial: We want to reach \( t \) at time \( t_A \) at the latest, but we don’t want to leave \( d \) much too early. In an ideal situation, every bus and every tram is on time, and it is sufficient to compute a journey that is planned to leave \( d \) as late as possible but still reaches \( t \) at the latest at \( t_A \). However, in reality, traffic can be congested and we should expect delays. Thus, we are looking for a robust journey from \( d \) to \( t \) that arrives before time \( t_A \), but still leaves \( d \) at a “reasonable” time.

Firmani et al. [32] observed in an experimental study on the transportation network of Rome that the timetable information and the real movement of the vehicles (based on GPS data) are “only mildly correlated”. They conclude that an “important issue to investigate is how to compute robust routes” that are “less vulnerable to unexpected events”. Our goal is to provide methods for finding such robust routes.

The authors of the aforementioned paper propose to use real-time information already in the planning phase, and a recent case study indicates that the usage of real-time information indeed improves the predicted travel times and quality of the proposed journeys [1]. However, for the situation described above, integrating real-time information is certainly not sufficient as journeys are usually planned some time in advance. For example, a trip to the airport is usually planned a few hours earlier, and the right departure time needs to be
Introduction

computed before the start of the journey. Moreover in reality it often happens that delays occur suddenly and cannot be foreseen in advance, especially not at the time when the journey is planned. For example, consider a $dt$-journey that consists of two lines $l_1$ and $l_2$, and imagine that the transfer time between the lines is 2 minutes. Even if $l_1$ leaves $d$ on time, every upcoming delay of more than 1 minute (which might always occur) leads to a late arrival at $t$. Thus, considering just real-time data might not always be the best solution.

Methods for planning robust journeys already exist. Simple strategies simply add some additional buffer time at each transfer or at the end of the journey (i.e., they compute a journey that is planned to arrive few minutes earlier). More sophisticated methods often make certain stochastic assumptions about arrival or travel times of vehicles (such as the assumption that the bus arrival times are exponentially distributed where the mean is the inverse frequency of the corresponding line, or that every vehicle suffers between every two consecutive stops some additional delay that follows some specified distribution). Some methods also use historic data to estimate typical travel times of vehicles. Existing methods often have two drawbacks: 1) they model delays explicitly, and 2) they assume that the random variables modeling arrival or travel times of vehicles are independent. From a traveller’s perspective, the behavior of single vehicles is of least interest. One instead is interested whether the travel plan as a whole works and leads to an on-time arrival or not.

The standard approach to describe a travel plan from $d$ to $t$ is to specify, according to a scheduled timetable, the concrete sequence of vehicles together with appropriate transfer stops and departure/arrival times for each transfer stop. Such a travel plan may look like this: Take Bus 31 at 8:04 from stop $d$ and leave it at 8:07 at transfer stop $s$; from there take Tram 10 at 8:08 and leave it at 8:15 at $t$. In case of delays, such a travel plan may easily become infeasible: Imagine a situation where Bus 31 left $d$ at 8:04, but arrived to $s$ only at 8:09, and Tram 10 left $s$ at 8:07 on time. Then, the described travel plan would bring the passenger to stop $s$ but it does not specify how to proceed further in order to arrive at $t$. In urban public transportation networks, however, there is an easy way to proceed: one simply takes the next vehicle of the same line (unless one knows a better option by heart).

We observe that standard solution concepts (such as paths in a time-expanded graph) are not suitable in such a setting. Therefore, we will compute recommendations to travelers in form of one or more journeys which specify an initial time $t_D$, a sequence of lines such as $(l_1(Tram), l_2(Bus), \ldots, l_k(Tram))$ and $k - 1$ appropriate transfer stops $(\theta_1, \ldots, \theta_{k-1})$. Such a journey can be interpreted as follows: start waiting at $d$ at time $t_D$, take the first vehicle of line $l_1$ that arrives and travel to stop $\theta_1$, then change to the first arriving ve-
hicle of line $l_2$, etc. Observe that such a plan is always feasible, even if delays occur. Hence, given multiple recorded timetables (i.e., concrete realizations of a timetable on a concrete day in the past), we look for journeys that performed well w.r.t. the given recorded timetables. Assuming that both the delays in the recorded timetables as well as future delays are typical, we can expect these journeys to work well also in the future.

Instead of measuring the robustness of journeys, we will measure the robustness of routes which are simply feasible sequences of lines. Hence, we ignore the transfers and the departure time of a journey. Notice that the computation of a route and a departure time are sufficient: once these are known, appropriate transfer stops for a journey can easily be computed using the planned timetable. Our solution approach now works as follows. As a first step, we list all routes whose number of transfers do not exceed a given threshold. After that, we will compute one or more robust route(s) and corresponding departure times, and reconstruct the journey(s) along these route(s) from the planned timetable.

1.2 Mathematical Preliminaries

Sets The set of all natural numbers is the set of all strictly positive integers $\mathbb{N} = \{1, 2, \ldots\}$, and by $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ we denote the set of all non-negative integers. Analogously, the set of all real-valued numbers is $\mathbb{R}$, the set of strictly positive real-valued numbers is $\mathbb{R}^+$, and by $\mathbb{R}^+_0 = \mathbb{R}^+ \cup \{0\}$ we denote the set of all non-negative real-valued numbers. For two integers $i, j$, the Kronecker delta $\delta_{ij}$ is defined as $1$ if $i = j$, and as $0$ otherwise.

Graph Theory Let $G = (V, A)$ be a directed graph. A walk in $G$ is a sequence of vertices $\langle v_0, \ldots, v_k \rangle$ such that $(v_{i-1}, v_i) \in A$ for all $i \in \{1, \ldots, k\}$. For a walk $w = \langle v_0, \ldots, v_k \rangle$ and a vertex $v \in V$, we write $v \in w$ if and only if there exists an index $i \in \{0, \ldots, k\}$ such that $v = v_i$. Analogously, for a walk $w = \langle v_0, \ldots, v_k \rangle$ and an arc $a = (u, v) \in A$, we write $a \in w$ if and only if there exists an index $i \in \{1, \ldots, k\}$ such that $u = v_{i-1}$ and $v = v_i$. The length of a walk $w = \langle v_0, \ldots, v_k \rangle$ is $k$, the number of arcs in the walk, and is denoted by $|w|$. A walk $w$ of length $|w| = 0$ is called degenerate, and non-degenerate otherwise. For two walks $w_1 = \langle u_0, \ldots, u_k \rangle$ and $w_2 = \langle v_0, \ldots, v_l \rangle$ with $u_k = v_0$, $w_1 \cdot w_2$ denotes the concatenation $\langle u_0, \ldots, u_k = v_0, \ldots, v_l \rangle$ of $w_1$ and $w_2$. A path is a walk $\pi = \langle v_0, \ldots, v_k \rangle$ such that $v_i \neq v_j$ for all $i \neq j$ in $\{0, \ldots, k\}$, i.e. a path is a walk without crossings. Given a path $\pi = \langle v_0, \ldots, v_k \rangle$, every contiguous subsequence $\pi' = \langle v_i, \ldots, v_j \rangle$ is called a subpath of $\pi$. A path $\pi = \langle s = v_0, v_1, \ldots, v_{k-1}, v_k = t \rangle$ is called an st-path. For a vertex $v \in V$, let $N^-_G(v)$ denote the out-neighborhood of $v$. 

1.3 Mathematical Preliminaries
1.3 Outline of the Thesis

The thesis is structured as follows. Chapter 2 discusses how we model transportation networks to support the computation of robust journeys. Chapter 3 discusses how, given two stops $d$ and $t$, the set of all feasible length-bounded $dt$-routes (i.e., sequences of lines) can be efficiently computed. We also study the problem of listing all paths in the transportation network that are induced by some length-bounded route. Chapter 4 proposes methods to compute robust routes and suitable departure times. Simple methods simply add additional buffer time at each transfer. We will show how an approximation set-based approach, originally developed by Buhmann et al. [18] for combinatorial optimization, can be adapted to public transportation. Since this approach by construction can only handle two recorded timetables, we discuss possible generalizations to incorporate multiple recorded timetables. Moreover, we also study a mean-risk based model, originally proposed by Lim et al. [43] for private transportation, which balances travel time and standard deviation of the departure times. In Chapter 5 we will evaluate our model, algorithms and robustness measures on real-world data describing the tram and bus network of Zürich, Switzerland. Finally, Chapter 6 investigates the aforementioned approximation set-based method by Buhmann et al. [18]. We will experimentally identify situations where this method outperforms other approaches, develop and investigate an analytic expression for the similarity score, and study possible generalizations of the approach.
In this chapter we describe how urban public transportation networks can be modeled in a way that they support the efficient computation of robust routes. Remember that our goal is to learn from historic data instead of modeling delays explicitly. We therefore need to develop a solution concept that allows routes to be comparable among data collected in different past time periods, and that also allows to compute intuitive and meaningful recommendations to a user. For this purpose, we separate the topological structure of a journey from the concrete travel times, and propose a solution approach in which we first generate some meaningful structural alternative routes, and only then integrate travel times and evaluate the robustness of each alternative using historic data.

2.1 Related Work

Problems arising in public transportation systems have been studied for a long time, hence there is a huge amount of literature. We will now give a brief explanation on the most prominent ideas for computing journeys between a given departure stop \( d \) and a given target stop \( t \) that either leave \( d \) or reach \( t \) at a specified time. For an excellent survey on the topic (discussing many of the subsequent articles), see [3].

**Early approaches** Tulp and Siklóssy [63, 64] model train networks by two graphs: A so-called static graph contains a vertex for every stop and an arc between two stops \( v \) and \( w \) if there is some train (line) which visits \( v \) directly before \( w \), and the length of an arc is the geographic distance between \( v \) and \( w \). Furthermore there is also a so-called dynamic multigraph with the same vertices which contains an arc \((v, w)\) for every concrete train (i.e., vehicle) that visits \( v \) directly before \( w \). With these arcs, the departure time at \( v \) and the arrival time at \( w \) are associated [65]. Now, their algorithm works in two phases. In the first phase it identifies, for a well-chosen value \( p \), the set of all vertices \( V' \) in the static network that lie on some \( dt \)-path whose length approximates the length
of a shortest $dt$-path by a factor of at most $(1 + p)$ \cite{61}. In the second phase, the concrete trains to be used and transfers are computed. For this, their algorithm considers the subgraph of the dynamic graph induced by $V'$, and uses a modified version of Dijkstra’s algorithm \cite{28} to compute, for a specified departure time, an earliest arrival time at the target stop $t$. It then performs a backwards search from $t$ to find a possibly better journey that leaves even later but still reaches $t$ on time \cite{65}. The authors also discussed some speed-up heuristics such as the elimination of stops with at most two neighbors in the static network, using the well-known A* algorithm \cite{36}.

Müller \cite{46} developed a similar but different approach that was used by the former Deutsche Bundesbahn (German Federal Railway) \cite{6}. Not all details are publicly available. The train network is modeled as a directed hierarchical graph which contains a vertex for every train (line), and an arc between two train (lines) $v$ and $w$ if they have at least one stop in common. The vertex set is partitioned into $k$ levels corresponding to different train classes, where a higher level indicates a higher (i.e., faster) train class. Arcs between vertices on the same level can be considered undirected (i.e., for every arc $(v, w)$ between two vertices on the same level the graph also contains $(w, v)$), arcs between vertices on different levels are always directed from the lower to the higher level. Now, given two vertices $v_d$ and $v_t$ (the first and the last line of a journey which visit the departure stop $d$ and the target stop $t$, respectively), Müller studied the problem of computing a vertex $v_p$, a path from $v_d$ to $v_p$ and a path from $v_t$ to $v_p$ in the aforementioned graph such that the sum of their lengths is minimal. The problem is solved using a bidirectional version of the A* algorithm. It starts the graph exploration at $v_d$ and $v_t$, respectively, and in every step the corresponding two components are extended until they meet. For that, one has to compute a lower bound on the length of a path between a vertex $v$ and some vertex in the other component. Once a path was found, concrete trains and transfers between these lines have to be computed, but it is not explained how this is done. Baumann and Schmidt \cite{6} claim that the system “in principle” can “determine all possible ways [paths]”, but since there are too many they only consider the “reasonable” ones. Also here the details remain unclear.

The similarity between these two aforementioned systems is that they first restrict the solution space (either by using geographic information or by bounding the number of transfers), and only then concrete trains and transfers are computed. Preuss and Syrbe \cite{54} sketched a different, distributed, system for finding journeys between different means of transport and different operators. Every operator specifies a graph in which vertices correspond to stops, street crossings or airports, and a directed arc $(v, w)$ indicates that there exists some
direct connection from $v$ to $w$. Arcs store some additional information such as the travel time or a timetable with the connections along this arc. The graphs of different operators may have vertices in common with other graphs that allow to change the mode of transport or the operator. For finding journeys, the authors propose two distributed variants of the A* algorithm. The hierarchical variant first looks for long-distance connections (by car, train or plane) between close vertices, then for medium-distance connections to these vertices, and it then incorporates local transportation and walking. In the cooperative variant, every time a vertex is found which allows to continue the search on the graph of a different operator, corresponding queries are sent to estimate a lower bound of the cost to the destination, and the search is continued at the most promising vertex. It is important to understand that there is not one superordinate server, but the queries are redirected and answered by the servers of the corresponding operators. Multiple criteria such as the overall price or the travel time are incorporated into one joint cost function.

**Time-expanded and time-dependent graphs** Instead of heuristically generating a set of solutions and finding among these the best one (e.g., with respect to the earliest arrival time, cost, etc.), one can also transform the instance of the transportation problem into a directed graph in which paths correspond to journeys, and compute a shortest path using Dijkstra’s algorithm. Essentially, there are two well-known models that gained a lot of attention in the literature: the *time-expanded* and the *time-dependent* models. In the original version [53, 59] of a time-expanded graph (also called *space-time network*), each vertex corresponds to an *event*, i.e., the arrival or the departure of one concrete train at one stop at one concrete time. We therefore can associate (at least implicitly) with every vertex its corresponding stop and its time. For every train, each departure and the succeeding arrival vertex (if exists) are connected by an arc. Moreover, for every stop, every vertex has an arc to its succeeding (w.r.t. time) vertex. The cost of an arc $(v, w)$ is the difference between the time at $w$ and the time at $v$.

In the original version of the time-dependent graph [16, 50], every vertex corresponds to a stop, and there is an arc between two stops $v$ and $w$ if there exists a train that visits $w$ directly after $v$. Every arc $a$ stores a cost function $f_a(t_v)$ that indicates the arrival time at $w$ when $v$ is reached at time $t_v$. For finding a shortest path between two stops, a straightforward generalization of Dijkstra’s algorithm can be used [16].

Both models have been refined to incorporate more realistic scenarios such as minimum transfer times, walking and traffic days [48, 55]. Especially for the time-expanded model, many speed-up heuristics have been proposed (e.g., [23, 55, 59, 60]). Pyrga et al. [55] performed an extensive experimental evaluation
on real-world networks to compare the time-expanded and the time-dependent model. They concluded that the time-dependent model allows to answer queries much faster than the time-expanded model, but the latter one allows to model special situations more easily. More information about both models, extended scenarios, models and speed-up techniques can be found in the aforementioned survey [3].

Also, the problem of computing Pareto-optimal journeys has been studied for both models [29, 49, 55]. For a set of criteria (such as arrival time, number of transfers), a journey $j$ is dominated by a journey $j'$ if $j'$ is not worse than $j$ in any criterion, and if $j'$ is strictly better than $j$ in at least on criterion. A journey $j$ is Pareto-optimal if it is not dominated by any other journey $j'$.

**Recent Developments** Both the time-dependent and especially the time-expanded graph are huge, hence newer solution approaches for journey computation use them only for preprocessing or avoid using them at all. Bast et al. [2] precompute for every two stops $d$ and $t$ a set of so-called transfer patterns from $d$ to $t$. Each transfer pattern is a sequence of stops that correspond to the transfer (including departure and target) stops in some Pareto-optimal journey. For every stop $d$, the transfer patterns to all other stops are stored in a directed graph (later in the paper, only transfer patterns to larger hub stops are computed, and only hub stops store the patterns to every other stop). To compute all Pareto-optimal $dt$-journeys that leave $d$ after a given departure time, the corresponding transfer patterns are extracted from the aforementioned graph, and from them a time-dependent query graph is built on which a multi-criteria version of Dijkstra’s algorithm is run.

A related idea is guidebook routing [5] which uses a different solution concept. The goal is to propose journeys of the form “Depart at $d$, use Bus 31 or Tram 3 to stop $s$, from there use Tram 5 to $t$, every 7.5 minutes”. For this purpose, one starts with a set of transfer patterns from $d$ to $t$ and transforms them into a suitable network in which arc capacities correspond to the number of trips (concrete realizations of lines) that contain this direct connection. Now, for a given $k$, one computes the paths with the $k$ largest flows using a Dijkstra-like algorithm, and uses them for recommendation. The authors observe that even a relatively small number of such paths approximate the set of optimal journeys well.

Both aforementioned approaches allow queries to be answered fast, but require a large preprocessing time. RAPTOR [24] computes all Pareto-optimal journeys and works directly on the set of given trips. It proceeds in rounds, where in the $k$-th round all optimal journeys with $k - 1$ transfers are computed. For that one considers all lines for which some trip can be reached by a journey with $k - 2$ transfers (or that visit the departure stop $d$ if $k = 1$).
The arrival times at every stop $s$ on that line with $k-2$ transfers were already computed in the previous round, and one now checks whether boarding the currently considered line at that stop helps to catch an earlier trip for that line (and hence improve the arrival times for the remaining stops on the line). A related but different idea was recently proposed by Witt [67]. Instead of revisiting lines and computing earliest trips in every round, he precomputes a set of transfers between trips, and dismisses transfers that are not reasonable (e.g., because they lead to a line traversing the opposite direction or don’t improve the arrival time). In every round, these precomputed transfers are then used to find trips that improve the arrival time. This approach requires only little time for preprocessing (the author states that few minutes suffice even for large networks), and achieves fast query times. Dibbelt et al. [26] proposed the Connection Scan Algorithm which stores all elementary connections (trip segments between two consecutive stops) in one array that is sorted by departure time. Now the idea is, in a way, to simulate a run of Dijkstra’s algorithm on a time-expanded graph. For every stop, an upper bound on the arrival time is maintained. The algorithm then considers all connections in increasing time-wise order and checks whether using this connection an earlier arrival can be achieved.

### 2.2 Stops, Lines, and Transportation Networks

Let $\mathcal{S}$ be a set of stops. A line is a sequence $l = (s_1, \ldots, s_k)$ of stops from $\mathcal{S}$ where the stop $s_i$ is visited directly before the stop $s_{i+1}$. Two lines serving the same stops but in a different order are explicitly distinguished. We especially distinguish between two lines serving the same stops but in opposite directions, despite the fact that they might be operated under the same identifier in reality. Let $\mathcal{L}$ denote the set of all lines. The transportation network induced by $\mathcal{S}$ and $\mathcal{L}$ is the directed graph $\mathcal{N}_{\mathcal{S}, \mathcal{L}} = (\mathcal{S}, A_{\mathcal{S}, \mathcal{L}})$ where $A_{\mathcal{S}, \mathcal{L}}$ contains an arc $(s, s')$ if and only if $\mathcal{L}$ contains a line $l$ which visits $s$ directly before $s'$. Such graphs were used previously, e.g., under the name station graph in [60]. Observe that every line in $\mathcal{L}$ corresponds to a walk in $\mathcal{N}_{\mathcal{S}, \mathcal{L}}$. For simplicity we assume that 1) the stops $\mathcal{S}$ and the lines $\mathcal{L}$ are chosen such that $\mathcal{N}_{\mathcal{S}, \mathcal{L}}$ is strongly connected (which especially implies that every stop in $\mathcal{S}$ is served by at least one line), that 2) every line consists of at least two stops, and that 3) every line contains a stop $s$ at most once (i.e., a line does not cross itself and hence can be interpreted as a path). These assumptions are not very restrictive, but they allow both a simpler implementation as well as a simpler running time analysis of the algorithms developed in the next chapters. We define the size of a transportation network $\mathcal{N}_{\mathcal{S}, \mathcal{L}}$ as $|\mathcal{N}_{\mathcal{S}, \mathcal{L}}| = |\mathcal{S}| + \sum_{l \in \mathcal{L}} |l|$, i.e., as the sum
of the number of stops and the sizes of the lines. Notice that this definition measures rather the input size than the size of the graph \( N \).

For an arc \( a = (s, s') \in A_{S, L} \), let \( L_a \subseteq L \) be the set of all lines that visit the stop \( s \) directly before the stop \( s' \) (i.e., the set of all lines whose corresponding path in \( N_{S, L} \) contains \( a \)). Analogously, for a stop \( s \in S \), let \( L_s = \bigcup_{(s, s') \in A_{S, L}} L(s, s') \) be the set of all lines that visit \( s \) and some other stop \( s' \) directly after \( s \) (i.e., the set of all lines \( l \) whose path in \( N \) contains an outgoing arc from \( s \)). Notice that for a line \( l = \langle s_1, \ldots, s_k \rangle \), the set \( L_{s_k} \) does not contain \( l \), because \( s_k \) is the last stop on \( l \). This may at first glance sound like an artificial restriction; however, observe that it is never reasonable to board a line at the last stop. Therefore the set \( L_s \) can be interpreted as the set of all lines \( l \) which are reasonable to be boarded at the stop \( s \).

For two lines \( l_1, l_2 \in L \), we define \( l_1 \cap l_2 \) to be the unordered set of all stops \( s \in S \) that are served both by \( l_1 \) and \( l_2 \).

### 2.3 Routes

Let \( \beta \in \mathbb{N} \) be an integer and \( d, t \in S \) be two stops. A sequence \( \langle l_1, \ldots, l_\beta \rangle \in L^\beta \) of \( \beta \) lines forms a so-called feasible route from \( d \) to \( t \) (or shorter, a \( dt \)-route or simply route if \( d \) and \( t \) are clear from the context) if there exist \( \beta + 1 \) stops \( \theta_0 := d, \theta_1, \ldots, \theta_{\beta - 1}, \theta_\beta := t \) such that for every \( i \in \{1, \ldots, \beta\} \), both \( \theta_{i-1} \) and \( \theta_i \) are stops on the line \( l_i \), and \( l_i \) visits \( \theta_{i-1} \) (not necessarily directly) before \( \theta_i \). We say that a transfer between the lines \( l_i \) and \( l_{i+1} \) occurs at the stop \( \theta_i \). Notice that two lines might have multiple possible transfers. The length of a route \( r = \langle l_1, \ldots, l_\beta \rangle \) is \(|r| = \beta\), i.e. the number of transfers plus one. Let

\[
\mathcal{R}^\beta_{dt} = \{ r \in L \cup L^2 \cup \cdots \cup L^\beta \mid r \text{ is a feasible route from } d \text{ to } t \} \tag{1}
\]

be the set of all feasible routes from \( d \) to \( t \) of length at most \( \beta \). Consider Figure 1 as an illustration.

The definition of a route \( r \) allows a line \( l \) to occur multiple times in \( r \). This is reasonable because there might be two transfer stops \( s, s' \) on \( l \) and one or more intermediate lines that travel faster from \( s \) to \( s' \) than \( l \) does. Additionally, observe that a route does not contain any time information.

Given two stops \( d, t \in S \), a minimum \( dt \)-route \( r \) has smallest length among all \( dt \)-routes in \( \mathcal{R}^\infty_{dt} \) (i.e., a \( dt \)-route \( r \) is minimum if and only if \( \mathcal{R}^{|r|-1}_{dt} = \emptyset \)). We define the \( L \)-distance \( d_L(d, t) \) from \( d \) to \( t \) as the length of a minimum \( dt \)-route (i.e., \( d_L(d, t) = \min \{ \beta \in \mathbb{N} \mid \mathcal{R}^\beta_{dt} \neq \emptyset \} \)). Moreover, for every two stops \( d, t \in S \) and every line \( l \in L_d \), let \( d^L_N(d, t, l) \) be the \( L \)-distance from \( d \) to \( t \) such that \( l \) is the first line used (i.e., \( d^L_N(d, t, l) \) denotes the minimum length among all \( dt \)-routes in \( \mathcal{R}^\infty_{dt} \) that start with the line \( l \)). Notice that since we assumed \( N \) to
Figure 1  An example where two lines share multiple transfer stops. For \( d = s_2 \) and \( t = s_7 \), the line sequence \( \langle l_1, l_2 \rangle \) is a feasible \( dt \)-route, and we can either transfer at \( \theta_1 = s_3 \) or at \( \theta'_1 = s_6 \). A transfer at \( s_9 \) is not possible because \( l_2 \) does not visit \( s_9 \) before \( s_7 \). Notice that although there are two possible transfers, there is only one route! Also, observe that for \( d = s_2 \) and \( t = s_1 \), \( \langle l_1, l_2 \rangle \) is not a feasible \( dt \)-route, because a line has a unique direction and hence no appropriate transfer exists.

be strongly connected, both \( d_L(d, t) \) as well as \( d_L^N(d, t, l) \) are well-defined; they especially are bounded integers. Section 3.2 describes how these values can be efficiently computed.

2.4 Trips and Timetables

Trips  While the only information associated with a line itself are its consecutive stops, it usually is operated multiple times per day. Each of these concrete realizations that starts at a certain time is called a trip. With every trip \( \tau \) we associate a line \( L(\tau) \in L \). By \( L^{-1}(l) \) we denote the set of all trips associated with a line \( l \in L \). A trip \( \tau \) can visit all or only a proper subset of the stops that \( L(\tau) \) visits, but no other stops. This definition is motivated from practice:

1) When buses are heavily delayed, the operator may decide to turn around the vehicle in advance in order to prevent a further accumulation of delays, hence some stops will not be served.

2) Stops may be blocked due to road or rail work.

3) Roads or tracks may be blocked due to accidents, demonstrations, fire-fighting jobs etc. In such a case vehicles may be forced to skip certain stops and to take a detour instead where they might visit stops that the associated line does not visit. One may argue that these additional stops should be incorporated into the trip as well. However, special situations as described above occur rather infrequently, and travelers are usually not aware of these special stops. We would also have to create new lines, one for each detour, and would therefore have many routes that are hardly ever feasible. Hence we decided to not integrate the additional stops.
Arrival and Departure Times  For a trip $\tau$ and a stop $s \in S$, let $A(\tau, s) \in \mathbb{N}_0$ be the arrival time of $\tau$ at stop $s$, if $\tau$ stops at $s$, and $A(\tau, s) = +\infty$ otherwise. Analogously, let $D(\tau, s) \in \mathbb{N}_0$ be the departure time of $\tau$ at $s$, if $\tau$ stops at $s$, and $D(\tau, s) = -\infty$ otherwise. For every trip $\tau$ and every stop $s \in S$ we require

\[
\left( (A(\tau, s) \neq +\infty) \land (D(\tau, s) \neq -\infty) \right) \Rightarrow A(\tau, s) \leq D(\tau, s),
\]

i.e., a trip reaches a stop $s$ before leaving it. Moreover, for every two stops $s_1, s_2 \in S$ where $L(\tau)$ visits $s_1$ prior to $s_2$, we require

\[
D(\tau, s_1) \leq A(\tau, s_2),
\]

i.e., a trip reaches $s_2$ only after leaving at $s_1$. Notice that inequality (3) stays valid even if $\tau$ does not visit $s_1$ or $s_2$.

Timetables  A set of trips for a specified time horizon (e.g., a day or a week) is called a \textit{timetable}. We distinguish between

1) The \textit{planned} timetable $T_{\text{planned}}$. We assume it to be periodic, i.e., every line realized by some trip $\tau$ will be realized by a later trip $\tau'$ again (probably not on the same day).

2) \textit{Recorded} timetables $T_i$ that describe how various lines were operated during a given time horizon. These recorded timetables are concrete executions of the planned timetable, and can be seen as the planned timetable with some additional randomness. When a vehicle on a trip $\tau$ turns around in advance at a stop $s$, we set $D(\tau, s) = -\infty$, and $A(\tau, s') = +\infty$ and $D(\tau, s') = -\infty$ for every stop $s'$ that $L(\tau)$ visits after $s$. Analogously, when a vehicle on a trip skips certain stops, we set $A(\tau, s) = +\infty$ and $D(\tau, s) = -\infty$ for every stop $s$ that is left out.

In the following, \textit{timetable} refers both to the planned as well as to a recorded timetable. The set of recorded timetables is denoted by $T$. We require that every (planned and recorded) timetable $T$ respects the FIFO property, i.e. vehicles of the same line do not overtake each other. Thus, for any two trips $\tau$ and $\tau'$ in $T$, either $\tau$ visits every stop not later than $\tau'$ does, or $\tau'$ visits every stop not later than $\tau$ does. More formally, we require that

$$
\forall \tau, \tau' \in T \text{ where } L(\tau) = L(\tau') : \left( \forall s \in L(\tau) : (A(\tau, s) \leq A(\tau', s) \lor A(\tau, s) = +\infty) \land \\
(D(\tau', s) = -\infty \lor D(\tau, s) \leq D(\tau', s)) \right) \lor \\
\left( \forall s \in L(\tau') : (A(\tau', s) \leq A(\tau, s) \lor A(\tau', s) = +\infty) \land \\
(D(\tau, s) = -\infty \lor D(\tau', s) \leq D(\tau, s)) \right).$$

(4)
The requirement that trips of the same line satisfy the FIFO property is commonly used in route planning algorithms, e.g., in [24, 26, 67]. It allows us to design simpler algorithms. Moreover, in urban areas, it happens rarely that buses of the same line overtake each other; for trams overtaking is even impossible due to the shared track.

On the Time Horizon of a Timetable  Obviously, the planned timetable and all recorded timetables in \( T \) must have a similar time horizon. One possible choice is a planning horizon of one (operational) day. Arrival and departure times have to be modeled in a way that the inequalities (2) and (3) remain valid. Since vehicles often also operate after midnight, it is not reasonable to consider days from 12:00 a.m. to 11:59 p.m., because trips starting before and ending at or after midnight violate the inequalities. Instead, one often defines an operational day that starts, e.g., at 4:00 a.m. and that ends on 3:59 a.m. on the next day. Hence, arrival and departure times can be measured in seconds elapsed since the start of the operational day. Of course, a similar problem as before appears for trips that start before 4:00 a.m. and end at 4:00 a.m. or later. One possible way out is to consider weekly instead of daily timetables; however, this does not solve the problem completely and also reduces the running time of our algorithms (if they are not modified). Since this problem is negligible in the context of robust routing in urban areas, it is not further explored in this thesis. We note that the problem of finding (not necessarily robust) schedules using overnight connections is a research topic for its own, see, e.g., the aforementioned survey [3].

2.5 Solution Concept, Approach and Outlook

Journeys  In the following, let \( d, t \in S \) be two stops, \( \beta \in \mathbb{N} \) be an upper bound on the number of lines to be used (i.e., the maximum allowed number of transfers plus one), and \( \varepsilon(s,l,l') \) be the minimum required time to change from line \( l \) to line \( l' \) at the stop \( s \). A journey consists of a departure time \( t_D \), a \( dt \)-route \( r = (l_1, \ldots, l_k) \in \mathcal{R}^\beta_{dt} \) (where \( k \leq \beta \)) and \( k - 1 \) appropriate transfer stops \( \theta_1, \ldots, \theta_{k-1} \) (where \( l_1 \) visits \( d \) before \( \theta_1 \), \( l_i \) visits \( \theta_{i-1} \) before \( \theta_i \) for every \( i \in \{2, \ldots, k-1\} \), and \( l_k \) visits \( \theta_{k-1} \) before \( t \)). The intuitive interpretation of such a journey is to

1) start at the stop \( d \) at time \( t_D \), take the first arriving vehicle of line \( l_1 \) (i.e., the first trip of line \( l_1 \) that reaches \( d \) at time \( t_D \) or later),

2) for every \( i \in \{1, \ldots, k-1\} \), leave \( l_i \) at the stop \( \theta_i \) and take the next arriving vehicle of line \( l_{i+1} \) immediately (i.e., the first trip \( \tau_{i+1} \) of line
Modeling Urban Public Transportation Networks

$l_{i+1}$ with $D(\tau_{i+1}, \theta_i) \geq A(\tau_i, \theta_i) + \varepsilon(\theta_i, l_i, l_{i+1})$ where $A(\tau_i, \theta_i)$ denotes the arrival time of the trip $\tau_i$ of line $l_i$ used to travel to $\theta_i$, and finally to

3) leave $l_k$ at the stop $t$.

The concept of a journey, and especially the decision to use a route (i.e., a line sequence) instead of a trip sequence is motivated from the perspective of a traveler: in urban areas, one usually just takes the first arriving vehicle of the corresponding line because trips of the same line are indistinguishable for a traveler; travelers usually do not care whether the arriving vehicle is the 13th or the 14th realization of this line on the day, and they certainly would not wait until the “right”/recommended vehicle arrives. As a thought experiment, imagine that all lines in our network operate six times per hour, and now imagine that on some day some lines always have a delay of 10 minutes while others are always on time. Although the network is delayed, a traveler would not notice unless he traveled very early in the morning. In classical models such as the time-expanded model, many solutions from the planned timetable would be infeasible on that day; our solution concept of a journey is always feasible (although the arrival time may vary, of course).

**Goal** A user specifies a query in the form $(d, t, t_A)$ where $d \in \mathcal{S}$ is the departure stop, $t \in \mathcal{S}$ is the target stop and $t_A$ is the latest allowed arrival time at $t_A$. We then use the set of stops $\mathcal{S}$, the set of lines $\mathcal{L}$, the planned timetable $T_{\text{planned}}$ and the set of recorded timetables $\mathcal{T}$ to compute a recommendation to the user in form of one or more (robust) journeys from $d$ to $t$ that will likely arrive on time (i.e., before time $t_A$) on a day for which the concrete travel times are not known yet. The notion of robustness is formalized in Chapter 4.

One may wonder why a user has to specify the latest allowed arrival time and not, e.g., the earliest departure time. This is motivated from a practical point of view: one really needs to be on time for an important meeting, a flight, etc. Moreover, we have a clear indicator whether our recommended journey was a good choice: either we arrive at time $t_A$ or earlier, or not. However, if one is interested in specifying the earliest departure time, all algorithms and concepts in this thesis can easily be adapted to it.

**A Small Example** Consider a transportation network induced by a set of stops $\mathcal{S} = \{s_1, \ldots, s_9\}$ and a set of lines $\mathcal{L} = \{l_1, \ldots, l_4\}$ where $l_1 = \langle s_1, \ldots, s_7 \rangle$, $l_2 = \langle s_7, \ldots, s_1 \rangle$, $l_3 = \langle s_8, s_3 \rangle$ and $l_4 = \langle s_2, s_9, s_6 \rangle$ (see Figure 2). Remember that $l_1$ and $l_2$ are not treated as one line because the order of their stops is different. The planned timetable $T_{\text{planned}}$ consists of ten trips that are also shown in Figure 2. Since the purpose of the example is to demonstrate the concepts and highlight certain situations that may appear in practice, we assume for simplicity that every trip has zero waiting time at each stop (i.e., for
2.5 Solution Concept, Approach and Outlook

Figure 2 An exemplary network with the stops $S = \{s_1, \ldots, s_9\}$ and the lines $L = \{l_1, \ldots, l_4\}$ where $l_1 = \langle s_1, \ldots, s_7 \rangle$, $l_2 = \langle s_7, \ldots, s_1 \rangle$, $l_3 = \langle s_8, s_3 \rangle$ and $l_4 = \langle s_2, s_9, s_6 \rangle$. The tables below describe the available trips. For each table, a time in row $s_i$ and column $\tau_j$ denotes the arrival and the departure time of the trip $\tau_j$ at the stop $s_i$ (for simplicity, we assume zero waiting time at each stop). At any stop $s \in S$, the transfer time between any two different lines $l, l' \in L_s$ is set to $\varepsilon(s, l, l') = 1$ minute.

<table>
<thead>
<tr>
<th>$l_1$</th>
<th>$\tau_1$</th>
<th>$\tau_2$</th>
<th>$\tau_3$</th>
<th>$l_2$</th>
<th>$\tau_4$</th>
<th>$\tau_5$</th>
<th>$\tau_6$</th>
<th>$l_3$</th>
<th>$\tau_7$</th>
<th>$\tau_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>706</td>
<td>716</td>
<td>726</td>
<td>$s_7$</td>
<td>7107</td>
<td>7117</td>
<td>727</td>
<td>$s_8$</td>
<td>720</td>
<td>728</td>
</tr>
<tr>
<td>$s_2$</td>
<td>708</td>
<td>718</td>
<td>728</td>
<td>$s_6$</td>
<td>7112</td>
<td>722</td>
<td>732</td>
<td>$s_3$</td>
<td>722</td>
<td>730</td>
</tr>
<tr>
<td>$s_3$</td>
<td>711</td>
<td>721</td>
<td>731</td>
<td>$s_5$</td>
<td>716</td>
<td>726</td>
<td>736</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$s_4$</td>
<td>715</td>
<td>725</td>
<td>735</td>
<td>$s_4$</td>
<td>723</td>
<td>733</td>
<td>743</td>
<td>$l_4$</td>
<td>$\tau_9$</td>
<td>$\tau_{10}$</td>
</tr>
<tr>
<td>$s_5$</td>
<td>722</td>
<td>732</td>
<td>742</td>
<td>$s_3$</td>
<td>727</td>
<td>737</td>
<td>747</td>
<td>$s_2$</td>
<td>731</td>
<td>742</td>
</tr>
<tr>
<td>$s_6$</td>
<td>726</td>
<td>736</td>
<td>746</td>
<td>$s_2$</td>
<td>730</td>
<td>740</td>
<td>750</td>
<td>$s_9$</td>
<td>733</td>
<td>744</td>
</tr>
<tr>
<td>$s_7$</td>
<td>731</td>
<td>741</td>
<td>751</td>
<td>$s_1$</td>
<td>732</td>
<td>742</td>
<td>752</td>
<td>$s_6$</td>
<td>735</td>
<td>746</td>
</tr>
</tbody>
</table>

Every trip $\tau$ and every $s \in L(\tau)$ we have $A(\tau, s) = D(\tau, s)$, and that at any stop $s \in S$, the transfer time between any two different lines $l, l' \in L_s$ is set to $\varepsilon(s, l, l') = 1$ minute. Consider the following queries:

Query 1: $d = s_1$, $t = s_7$, $t_A = 800$. There exist two obvious $dt$-routes: First, there is a direct connection $r_1 = \langle l_1 \rangle$, and also the route $r_2 = \langle l_1, l_4, l_1 \rangle$ (using the transfer stops $\theta_1 = s_2$, $\theta_2 = s_6$). By definition $R^3_{dt}$ also contains the route $r_3 = \langle l_1, l_2, l_1 \rangle$ which we ignore here. Using $r_1$, we can leave $d$ at 726 and reach $t$ at time 751. Using $r_2$, we can leave $d$ at 726, arrive at $s_2$ at time 728, from there use $\tau_9$ to travel to $s_6$, and from there use $\tau_2$ to travel to $t$. The arrival time is 741, hence $r_2$ is faster than $r_1$. This shows why it might be reasonable that a line appears multiple times in a route. Observe that when the target arrival time was set to $t_A = 745$, $r_2$ seems to be a better choice than $r_1$, because the latter one would have to leave $d$ at 716 to reach $t$ on time. However, $r_1$ might be more reliable.
than \( r_2 \) because it doesn’t contain any transfers while \( r_2 \) has two. This example already shows that the selection of an appropriate route is non-trivial. Methods to compute robust routes will be discussed in detail in Chapter 4.

Query 2: \( d = s_8, t = s_7, t_A = 00 \). Again, there are two obvious \( dt \)-routes: We can use \( r_1 = \langle l_3, l_1 \rangle \) (using the transfer stop \( \theta_1 = s_3 \)), or \( r_2 = \langle l_3, l_2, l_4, l_1 \rangle \) (using the transfer stops \( \theta_1 = s_3, \theta_2 = s_2 \) and \( \theta_3 = s_6 \)). Notice that the set \( R^4_{dt} \) again contains a trivial alternative, namely \( \langle l_3, l_1, l_2, l_1 \rangle \) which we ignore. Using \( r_1 \), we can leave \( d \) at \( 7^{28} \), reach \( s_3 \) at \( 7^{30} \), from there use \( \tau_3 \) to reach \( t \) at \( 7^{51} \). Using \( r_2 \), we would have to leave earlier: from \( \tau_8 \) we may transfer to \( \tau_5 \) and from there to \( \tau_{10} \); however since \( \tau_{10} \) reaches \( s_6 \) at \( 7^{46} \) and we assumed a transfer time of one minute, \( \tau_3 \) cannot be reached any more (it could only be reached if it left \( s_6 \) at \( 7^{47} \) or later). Therefore \( r_1 \) seems to be a better option than \( r_2 \). As one can easily see, this is no longer true when the target arrival time is set to \( t_A = 7^{45} \): using \( r_2 \) it is sufficient to leave \( d \) at \( 7^{20} \) while for \( r_1 \) we have to leave earlier.

This small and rather artificial example showed that the “optimal” route does not only depend on the departure and the target stop, but may be also influenced by the target arrival time. Moreover, we saw why it is reasonable that a route may contain a line multiple times.

**Solution Approach**  Assume that a user specified a query \((d, t, t_A)\). We use the following abstract framework to compute robust journeys:

1) Compute an appropriate value for \( \beta \), the maximum length of a route.

2) Compute the set \( R^\beta_{dt} \).

3) Compute a subset \( R \subseteq R^\beta_{dt} \) containing the most robust routes.

4) For every route \( r \in R \), find an appropriate journey along \( r \).

Apart from the \( d \), \( t \) and \( t_A \), the framework can also access the set \( S \) of stops, the set \( L \) of lines, the planned timetable \( T_{\text{planned}} \) and the set of recorded timetables \( T \). Chapter 3 contains algorithms to compute the set of all feasible length-bounded \( dt \)-routes. Chapter 4 discusses how the robustness of individual routes can be evaluated, and how robust routes and corresponding journeys can be found. Notice that we really evaluate the robustness of routes (i.e., line sequences) instead of the robustness of journeys. Concrete travel times and transfers are ignored to some extent, and are incorporated only at a later stage. The reason for this is that the number of routes is considerably smaller than the number of journeys. Thus, the whole approach can be used in
2.6 Modeling Issues

In the following we discuss some consequences of the way how we modeled transportation networks. In particular we consider situations that arise when modeling the transportation network of Zürich, because this network will be used later as a basis for the experimental evaluation of our algorithms. The most critical point in our model is our definition of a line as an (ordered) sequence of stops. Although no two lines have the same stop sequence, they may have the same identifier (such as “Bus 31”) in reality and might therefore be considered equivalent from a traveler’s perspective. Figure 3 gives an example for various lines that might be operated under the same identifier.

**Standard realization of a line**  In most cases, a line \( l = \langle v_1, \ldots, v_k \rangle \) is realized throughout the whole day with high frequency. This line corresponds to \( l_{\text{standard}} \) in Figure 3. Often there exists a similar line (a backward line) in the opposite direction \( l' = \langle v_k, \ldots, v_1 \rangle \) which contains the same stops as \( l \) but in reverse order. In some rare cases, the stop sequence of the backward line slightly differs. This mostly happens when buses travel along one-way streets.

**Standard realization changes over the day**  In most cities there exist observable patterns of how people use public transportation. During the work days, there are clearly visible peak hours at which people commute to or from work. The planned schedule of public transportation services usually tries to react to increased or decreased demand. As a consequence, the planned frequency of a line may change during the day. Sometimes, however, also the realization of

![Figure 3](image-url)  Different lines with the same identifier. The line \( l_{\text{standard}} \) describes the most frequent realization. The line \( l_{\text{evening}} \) leaves a stop out, but this happens only in the evening. Towards the end of the operational day, the line \( l_{\text{depot}} \) goes back to the depot. Sometimes early in the morning, buses by plan turn around in advance such that there is a fast and good coverage of vehicles. An example for such a line is \( l_{\text{turnaround}} \).
the line itself changes, e.g. because in the evening certain stops are completely left out and some other stops are visited instead. Hence we have to introduce a new line to model such a situation. In Figure 3, $l_{evening}$ is an example for such a line.

Such lines clearly introduce difficulties in the context of finding robust routes: if, at some point, the realization of a line changes from $l_{standard}$ to $l_{evening}$ and a traveler misses the last trip of $l_{standard}$, then following the earlier computed route does not lead to an arrival at the target stop on that day. In reality, however, if one does not want to travel to one of the stops that is left out, one could have just used $l_{evening}$ instead.

**A vehicle comes from or goes to the depot** At the beginning of an operational day (and shortly before the beginning of the rush hours), vehicles come from the depot to start the service. Analogously, at the end of the operational day (and after the rush hours) vehicles travel back to the depot. Often these lines can also be used for traveling, but their line sequence may differ considerably from the standard realization. The low frequency of such lines makes an inclusion of them into a robust route highly undesirable. For example, consider Figure 3 and assume that we want to travel from $d$ to $t$. It might be tempting to use $l_2$ until $s_1$, and from there use $l_{depot}$. Now, if the planned trip of $l_2$ is slightly delayed and the transfer to $l_{depot}$ fails, one cannot continue the recommended journey until the next trip of $l_{depot}$ arrives, which might be only a few hours later (or even only on the next day).

**Cyclic lines** In some rare cases, a line $l$ does not have a corresponding backward line going in the opposite direction. Instead, after visiting the last stop on $l$ the realizing vehicle travels back to the first stop of $l$, hence $l$ has a cyclic topology. Such a situation is rather difficult to capture, because in a journey it might be necessary to take two consecutive trips of the same line. Although the trip is changed, the transfer should not be counted because it is the same physical vehicle. One way out might be to concatenate $l$ with itself, but then we violate our previously made assumption that every line visits every stop of the network at most once.

**A vehicle turns back in advance** Shortly after the beginning of an operational day, one especially tries to spread vehicles as fast as possible such that many stops are covered. Therefore, vehicles sometimes are planned to turn around in advance such that the most important stops of that line are covered already early in the morning. Since this behavior can be found in the planned timetable, we introduce an additional line (e.g., $l_{turnaround}$ in Figure 3).

In reality, vehicles can always be delayed. If the delays are huge, it sometimes happens that while waiting for a vehicle of line $l$, no vehicle arrives for a long time and then suddenly many vehicles of the same line arrive in short
2.7 Modeling Issues

time intervals. Once such a situation is established, it is even intensified: since the times to pick up and drop off passengers increase (due a larger amount of waiting passengers), the delay of the first and possibly also of some of the succeeding vehicles increase. Therefore the network operator may decide that a vehicle should turn around in advance before reaching the final destination. This results in a line that is realized just partially. Although this situation is planned by the network operator, it is not reflected in the planned timetable. As previously mentioned, if some trip leaves some stops out, then we just set the arrival times of the corresponding stops to $+\infty$, and the departure times to $-\infty$. We do not introduce a new line here, because the planned timetable usually does not contain any corresponding line. This modeling allows to use such “broken” trips as long as we do not visit stops that are not served any more. Otherwise, we can simply take the succeeding trip.

A vehicle overtakes another In some rare cases, a vehicle is greatly delayed and the succeeding vehicle of the same line $l$ overtakes it. This usually happens only with buses. When the FIFO property is no longer satisfied, some of the algorithms of the next chapters may not give a “correct” answer, because there might be a connection that departs later but arrives earlier. However, this is not a problem from the traveler’s perspective: in reality passengers have no knowledge that such a situation will occur and will therefore simply board the first arriving vehicle (instead of waiting for the succeeding one). Hence, although it might be better to wait for the next vehicle, such a solution will never be realized in reality.

Repeated use of a single line We allowed routes to use a line $l$ more than once. The reason is that between two stops $s$ and $s'$, some other line might have a much shorter travel time than $l$ (e.g., if $l$ was a bus, and there also was some suburban train between $s$ and $s'$). See the example discussed in the previous section, and especially Figure 2.

Nearby stops Some of the larger stops that many lines visit are sometimes split into several stops $s_1, \ldots, s_k$ that are very close to each other, but have slightly different names. Although a traveler might consider them to be equivalent, we explicitly distinguish them. This especially implies that if two lines $l_1$ and $l_2$ serve completely different stops and $l_1$ visits some stop $s_i$ (from the ones above) and $l_2$ visits some other stop $s_j \neq s_i$ (also from the ones above), then our model does not allow to transfer from $l_1$ to $l_2$, although in reality a transfer exists (with walking).
2.7 Implementation Details

In an implementation, we first can assume that $S \subseteq \mathbb{N}$ or even $S = \{1, \ldots, |S|\}$, i.e. every stop is represented by a unique integer. An array, indexed from 1 to $\max\{s \in S\}$, stores at position $s$ information associated with the stop $s$ such as the name of $s$, its coordinates and the set $L_s$ containing pointers to all lines that visit $s$.

A line $l$ can be represented by an array of length $k$ where the $i$-th position stores the ID of the $i$-th stop on $l$. Additionally, for each line $l$ we maintain a function $p_l : S \to \{1, \ldots, |l|\} \cup \{\infty\}$ that maps a stop $s$ to the position of $s$ on $l$, or to $\infty$ if $l$ does not visit $s$. If these functions are represented using Cuckoo Hashing [52], then the test whether a line visits a given stop can be performed in time $O(1)$. Also, constant time is sufficient to compute the successor of a given stop $s$ on a line $l$ (or to determine that $s$ is the last stop on $l$), and to check whether a line visits some stop $s$ before some other stop $s'$ (or not). Given two lines $l_1$ and $l_2$, their intersection can easily be computed by iterating over all stops in $l_1$ and checking whether $l_2$ also visits them (or vice versa if $|l_2| < |l_1|$), giving a running time of $O(\min\{|l_1|, |l_2|\})$.

A trip $\tau$ can be represented as a triple $(p_\tau, A_\tau, D_\tau)$ consisting of a pointer $p_\tau$ to the line $L(\tau)$ associated with $\tau$, and two arrays $A_\tau$ and $D_\tau$ indexed from 1 to $|L(\tau)|$ where the $i$-th position denotes the arrival and the departure time of $\tau$ at the $i$-th stop on $L(\tau)$. The corresponding stops can be reconstructed by following the pointer to $L(\tau)$, hence for a given trip $\tau$ and a given stop $s$, both $A(\tau, s)$ and $D(\tau, s)$ can be evaluated in time $O(1)$.

A timetable $T = \{\tau_1, \ldots, \tau_k\}$ can be represented using the following structure. For every stop $s$, we have an array $T_s$ that contains pointers to all trips $\tau_i$ that visit $s$ (i.e., for which $A(\tau_i, s) \neq \infty$ and $D(\tau_i, s) \neq \infty$). The elements of $T_s$ are ordered increasingly by the departure time of the corresponding trips at $s$ (breaking ties arbitrarily). The timetable $T$ is an array indexed from 1 to $\max\{s \in S\}$ where the $s$-th position contains a pointer to $T_s$ (or $\text{null}$ if $s \notin S$). Hence, given a stop $s$, a timetable $T$ and a time $t$, we can easily compute a trip that leaves $s$ at time $t$ or earlier by performing a binary search on $T_s$. Of course, the same method works for computing a trip that leaves $s$ at time $t$ or later. The running time of this operation is bounded by $O(\log |T_s|)$.

2.8 Conclusion

We developed a model for urban public transportation networks and outlined an approach for computing robust journeys. To make historic data from different time periods comparable we introduced the solution concept of a route which
is simply a sequence of lines without transfer stops or concrete travel times. To compute robust journeys, we first generate all feasible length-bounded routes, evaluate the robustness for each of them and then derive appropriate journeys from the most robust routes. We also discussed the consequences of our modeling.

The way how we modeled transportation networks differs considerably from classical models such as the time-expanded or the time-dependent graph. These graphs integrate both the structure of the model as well as timetables into one common graph in which journeys correspond to paths. This, however, makes it less suited when incorporating historic delay data, as we would have to compare paths among different given graphs. Instead, we separate routes from concrete travel times. Such ideas were already used in early models [46, 63], although there not all but only a heuristically restricted subset of possible solutions were considered. The idea of considering sequences of lines was already used by Müller [46]. However, a direct consequence of searching for shortest paths in directed hierarchical graphs implies that all lines are always undirected; i.e., one cannot distinguish between the special realizations that a line can have. Also, only a solution for computing a route with a minimum number of transfers was presented. Baumann and Schmidt [6] mentioned that all possibilities could be generated, but don’t give more details. However, we will see in the next chapter that in general undirected lines are much easier to handle than directed ones. The reason is simply that undirected lines can be traveled in both directions. Having undirected lines is clearly reasonable in train networks, but is unacceptable for urban transportation networks involving buses because buses might pass through one-way streets.

Also transfer pattern routing [2] distinguishes between the topology of a solution (i.e., the transfer pattern) and concrete travel times (that are incorporated later using the query graph). In some way, our solution concept is also similar to guidebook routing [5] which was presented very shortly after our initial publication [12]. However, guidebook routing enables to join different lines (“Use Bus 31 or Tram 3 to stop s”) which our concept of a route does not support, and guidebook routing incorporates transfers. Having these alternatives clearly helps a traveler in reality and might even lead to more robust journeys. Unfortunately it seems that all algorithms and concepts in this thesis cannot trivially be generalized to such a solution concept, so joining different lines definitely is an interesting topic for future research.
In this chapter we describe an algorithm that lists all feasible routes in a systematic way. More concretely, we assume that we are given a transportation network $N = N_{S,L}$ induced by a set of stops $S$ and by a set of lines $L$, a departure stop $d \in S$, a target stop $t \in S$ and an upper bound $\beta \in \mathbb{N}$ on the length of a route. We first develop an algorithm that lists all routes in $R = R_{dt}^{\beta}$, show how its running time can be bounded by a polynomial in the sum of the input and the output size (i.e., in $|N| + |R|$), and show how the running time can be further improved using preprocessing. Moreover we study the problem of listing paths in $N$ that are induced by some length-bounded route, and again present an algorithm whose running time is polynomial in the sum of the input and the output size.

3.1 Related Work

**Listing algorithms** For a given combinatorial optimization problem, a listing algorithm is an algorithm that systematically lists all or a specified subset of the possible solutions. Listing combinatorial objects (such as paths, cycles, spanning trees, etc.) in graphs is a widely studied field in computer science (see, e.g., [7]). As in [38, 57], the delay of a listing algorithm is the maximum of the time elapsed until the first solution is output and the times elapsed between any two consecutive solutions are output. In this chapter, we are especially interested in listing algorithms with a polynomial delay (w.r.t. the input size) [38].

**Listing all paths** The currently fastest algorithm for listing all $dt$-paths in directed graphs was given by Johnson [39] and runs in time $O((n + m)(\kappa + 1))$ where $n$ and $m$ are the number of vertices and arcs, respectively, and $\kappa$ is the number of all $dt$-paths (i.e., the size of the output). For undirected graphs, an optimal algorithm was presented by Birmele et al. [9].

**Listing the $K$-shortest paths** A related problem is the computation of the $K$-shortest $dt$-paths for a given constant $K$. Yen [69] and Lawler [42] proposed an algorithm that can be implemented to achieve a running time of $O(K(nm +
n^2 \log n). For undirected graphs, Katoh et al. [40] proposed an algorithm with running time $O(K(m + n \log n))$. Eppstein [31] gave an $O(K + m + n \log n)$ algorithm for listing the first $K$ shortest $dt$-walks, i.e., paths in which vertices are allowed to appear more than once.

**Listing $\alpha$-bounded paths** Recently, Rizzi et al. [56] studied a different parameterization of the $K$ shortest paths problem where they ask to list all $dt$-paths with length at most $\alpha$ for a given $\alpha$. The difference to the classical $K$ shortest paths problem is that the lengths (instead of the overall number) of the paths output is bounded. Thus, depending on the value of $\alpha$, $K$ might be exponential in the input size.

For directed graphs, they propose a simple recursive algorithm that works as follows. Every recursive call of the algorithm essentially obtains as parameters the partial $du$-path $\pi_{du}$ (initially, $\pi_{du} = \langle d \rangle$), the remaining path length $\alpha'$ (initially, $\alpha' = \alpha$) and the remaining graph $G' = (V, A')$ (initially, $G' = G$). If the last vertex of $\pi_{du}$ is $t$, the corresponding path is output. Otherwise, all arcs $(u, v) \in A'$ leaving $u$, the last vertex on $\pi_{du}$, are considered. A trivial algorithm would simply invoke a recursive call for every neighboring vertex $v$ (where the partial path is extended by $v$, $\alpha'$ is decreased by the cost of the edge $(u, v)$ and $u$ is removed from $G'$ to avoid the generation of walks that contain a vertex more than once). This, however, might lead to an exponential running time even if no solution is output. Rizzi et al. [56] invoke such a recursive call on $v$ only if it is guaranteed to output at least one solution. This is exactly the case if $G' - \{u\}$ contains a $vt$-path with cost at most $\alpha' - c((u, v))$ (where $c((u, v))$ denotes the cost of $(u, v)$). Determining these vertices $v$ is easy using a standard trick: at every recursive call, one computes $G' - \{u\}$, replaces every edge $(v, w)$ by its reversed edge $(w, v)$, and runs Dijkstra’s algorithm from $t$ in this graph to determine for every $v \in G' - \{u\}$ the distance to $t$.

They also propose an improved algorithm for undirected graphs. Both algorithms have polynomial delay, and their running time coincides with the running time of the algorithm of Yen and Lawler for directed graphs, and with the running time of the algorithm of Katoh et al. for undirected graphs. However, the above algorithm needs only $O(n + m)$ space.

Neither these, nor the aforementioned algorithms can directly be used for the listing problems discussed in this chapter, because we have the additional constraint of bounded transfers, and also multiple lines may induce the same edge in the transportation network. A more detailed explanation is given in Section 3.6. However, the algorithms in this chapter use a similar idea as the one by Rizzi et al. [56] (and by Sacomoto [57]). We use a similar presentation and notation so that the similarities and also the differences between their algorithm and the algorithms here can be clearly seen.
Listing in Transportation Networks As already mentioned in Chapter 2, computing Pareto-optimal journeys has been studied in many cases [29, 49, 55]. Xu et al. [68] studied the problem of listing all paths in a transportation network that have at most two transfers. They encode the various possible situations that may occur directly into their algorithm. This algorithm is then used as a subroutine to list the \( k \) earliest-arriving paths. The same problem was addressed very recently by Vo et al. [66]. They propose a time-dependent version of Yen’s algorithm [69]. To avoid the generation of similar paths, they propose a similarity measure between two paths \( P_1 \) and \( P_2 \) which essentially relates the number of common lines in both paths to the minimum number of lines that give rise to \( P_1 \) and to \( P_2 \), respectively. Nguyen et al. [51] studied an implicit listing problem in hypergraphs. Every hyperarc has exactly one tail \( t \) and multiple heads \( H \), and the authors assume that for any \( H' \subseteq H \), the hyperarc from \( t \) to \( H' \) is also implicitly contained. Essentially, the tail represents a stop \( s \), and every vertex in \( H \) represents one line that visits \( s \). Vehicles of the lines are assumed to arrive according to a given distribution. Now, for a given departure stop \( d \) and a given target stop \( t \), one aims to identify so-called efficient hyperarcs \((i, H')\) such that for every \( j \in H' \) the expected travel time from \( j \) to \( t \) is smaller than the expected travel time from \( i \) to \( t \). Hence, they do not explicitly list all possible paths, but encode the reasonable paths implicitly using efficient hyperedges.

3.2 Computing the Shortest \( dt \)-Routes and \( L \)-distances

Computing a Shortest \( dt \)-Route Given a transportation network \( N \), computing a shortest \( dt \)-route (i.e., a route with a minimum number of transfers) is a well-studied problem [48]. The idea is to transform \( N \) into an auxiliary directed graph \( \Gamma[N] \) and then solve a shortest path problem on \( \Gamma[N] \). The graph \( \Gamma[N] = (V[\Gamma], A[\Gamma]) \) has the properties that 1) \( S \subseteq V[\Gamma] \), and 2) for any two stops \( d, t \in S \) the cost of a shortest \( dt \)-path in \( \Gamma[N] \) is exactly \( d_L(d, t) \). For its construction, we first add every stop \( s \in S \) to \( V[\Gamma] \). Additionally, for every stop \( s \in S \) and every line \( l \in L_s \), we create a new vertex \( s_l \) and add it to \( V[\Gamma] \). The set \( A[\Gamma] \) contains three different types of arcs:

1) If \( s \in S \) is the direct successor of \( u \in S \) on a line \( l \), we create a traveling arc \((s_l, u_l)\) with cost 0. These arcs are used for traveling along a line \( l \).

2) For every stop \( s \in S \) and every line \( l \in L_s \), we create a boarding arc \((s, s_l)\) with cost 1. These arcs are used to board the line \( l \) at the stop \( s \).

3) For every stop \( s \in S \) and every line \( l \in L_s \), we create a leaving arc \((s_l, s)\) with cost 0. These arcs are used to leave the line \( l \) at the stop \( s \).
Let \( N \) be a transportation network induced by the stops \( S = \{ a, \ldots, g \} \) and the lines \( L = \{ l_1, l_2, l_3 \} \) with \( l_1 = \langle a, d, e \rangle \), \( l_2 = \langle b, e, g \rangle \) and \( l_3 = \langle c, d, e, f \rangle \). The figure shows the graph \( \Gamma[N] \) for this transportation network. Dotted lines represent arcs of cost 0, solid lines represent arcs of cost 1. The dotted circles represent meta-vertices of the corresponding stations. For simplicity the vertices \( s_{l_k} \) are labeled \( s_k \).

Figure 4 shows the construction. It is similar to the train-route graph for the Minimum Number of Transfers Problem described by Müller-Hannemann et al. [48] except that we penalize boarding arcs instead of leaving arcs. The graph is described explicitly here because it can be used to efficiently compute \( L \)-distances which are needed for the listing algorithms described in the Sections 3.4–3.6.

**Theorem 1.** Given a transportation network \( N = N_{S,L} \) and two stops \( d, t \in S \), a shortest \( dt \)-route can be computed in time \( \Theta(|N|) \), where \( |N| = |S| + \sum_{l \in L} |l| \) is the input size.

**Proof.** We construct the graph \( \Gamma[N] \) and run Dial’s (implementation of Dijkstra’s) algorithm [25] on the vertex \( d \). Let \( \pi_{dt} \) be a shortest \( dt \)-path in \( \Gamma[N] \). It is easy to see that the cost of \( \pi_{dt} \) is exactly \( d_L(d, t) \) [48]. Furthermore, \( \pi_{dt} \) induces a \( dt \)-path in \( N \) by replacing every traveling arc \( (s_l, u_l) \) by \( (s, u) \), and ignoring the arcs of the other two types [60]. Analogously the route can be extracted from \( \pi_{dt} \) by considering the lines \( l \) of all boarding arcs \( (s, s_l) \) in \( \pi_{dt} \) (or, alternatively, by considering the lines \( l \) of all leaving arcs \( (s_l, s) \) in \( \pi_{dt} \)).
3.2 Computing the Shortest $dt$-Routes and $L$-distances

![Figure 5](image_url)

A transportation network induced by the lines $l_1 = \langle d, s_3 \rangle$, $l_2 = \langle s_3, s_4 \rangle$, $l_3 = \langle s_4, t \rangle$, $l_4 = \langle s_1, s_2, d, s_5 \rangle$ and $l_5 = \langle s_2, t, s_5, s_1 \rangle$. The route $r_1 = \langle l_1, l_2, l_3 \rangle$ is a shortest one. It uses three different lines and two transfers. However, the route $r_2 = \langle l_4, l_5, l_4, l_5 \rangle$ is an optimal solution when the goal is to minimize the number of different lines: it uses only two different lines (but three transfers).

For every stop $s$ served by a line $l$, $\Gamma[\mathcal{N}]$ contains at most two vertices (namely, $s_l$ and $s$), thus we have $|V[\Gamma]| \in O(|\mathcal{N}|)$. Furthermore, $A[\Gamma]$ contains every arc $a$ of every line, and exactly two additional arcs for every vertex $s_l$. Thus we obtain $|A[\Gamma]| \in O(|\mathcal{N}|)$. Since the largest arc weight is $C = 1$ and Dial’s algorithm runs in time $O(|V[\Gamma]|C + |A[\Gamma]|)$, a shortest $dt$-route can be found in time $O(|\mathcal{N}|)$.

**Computing the values** $d^c_{\mathcal{N}}(s, t, l)$

Given a stop $t \in \mathcal{S}$, we can use the solution from the previous paragraph to compute the values $d^c_{\mathcal{N}}(s, t, l)$ for every $s \in \mathcal{S}$ and every $l \in \mathcal{L}_s$. For this purpose, we first compute the transportation network $\mathcal{N}_R$ which has the same vertices as $\mathcal{N}$ and contains an arc $(v, w)$ if and only if $\mathcal{N}$ has an arc $(w, v)$ (i.e., we reverse every arc in $\mathcal{N}$). Now, for every $s \in \mathcal{S}$ and every $l \in \mathcal{L}_s$, the length of a shortest path in $\Gamma[\mathcal{N}_R]$ from $t$ to $s_l$ is exactly $d^c_{\mathcal{N}}(s, t, l)$. We then run Dial’s algorithm on $\Gamma[\mathcal{N}_R]$ from the vertex $t$ to compute the lengths of all these shortest paths. By Theorem 1 we know that the overall running time to compute all values $d^c_{\mathcal{N}}(\cdot, t, \cdot)$ is bounded by $O(|\mathcal{N}_R|) = O(|\mathcal{N}|)$.

A similar idea (using a shortest-path algorithm from $t$ in $\Gamma[\mathcal{N}_R]$) was used by Disser et al. [29] to estimate the minimum required number of transfers between any stop to a given target. Here we also take the first line of each route into account.

**Minimizing the Number of Different Lines**

We saw that, given two stops $d, t \in \mathcal{S}$, a $dt$-route with a minimum number of transfers can be efficiently computed. A related, but different problem is the following: Given two stops $d, t \in \mathcal{S}$, compute a $dt$-route that uses as few different lines as possible. The difference between these two problems is that in the original formulation, every transfer is penalized while the new variant penalizes only the transfers to lines that were never used before. Figure 5 gives an example when these problems have different optimal solutions. Surprisingly, Böhmová proved that computing a route with a minimum number of different lines is hard [10].
**Theorem 2** (Böhmová (2016)). Given a transportation network $\mathcal{N} = \mathcal{N}_{S, L}$, two stops $d, t \in S$ and $\varepsilon > 0$, the problem to compute a $dt$-route with a minimum number of different lines cannot be approximated within $(1 - \varepsilon) \ln n$ unless $P = NP$.

For the proof, an arbitrary instance of the SET-COVER problem is transformed into a suitable transportation network, and it is shown that this reduction is approximation-preserving [10]. Arguably, neither the graph used in the proof nor the problem itself seem to be exceptionally relevant from a practical point of view. Real transportation networks (especially train networks) often have the property that lines can be traveled in both directions, i.e. $L$ contains a line $l = (s_1, \ldots, s_k)$ if and only if also $l_R = (s_k, \ldots, s_1)$ is contained. In such a case, both lines can be considered equivalent (in real network, such lines are even often operated under the same identifier such as Bus 31 or Tram 6), and can be interpreted as an undirected graph $G_l = G_{l_R}$ with the vertex set $\{s_1, \ldots, s_k\}$ and the edge set $\{(s_i, s_{i+1}) \mid i \in \{1, \ldots, k - 1\}\}$. Then $L$ is a set of undirected graphs in which exactly two vertices have degree 1 and all other vertices have degree 2, and the induced transportation network is an undirected graph. As the following theorem shows, the problem of computing a route with a minimum number of different lines becomes easy.

**Theorem 3.** Let $L$ be a set of undirected lines (i.e., undirected graphs in which exactly two vertices have degree 1 and all other vertices have degree 2), and let $\mathcal{N} = \bigcup_{G_i \in L} G_i$ be the induced (undirected) transportation network. Every shortest $dt$-route uses a minimum number of different lines, and such a route can be computed in time $\mathcal{O}(|\mathcal{N}|)$ where $|\mathcal{N}| = |S| + \sum_{l \in L} |l|$ is the input size.

**Proof.** Let $r = (l_1, \ldots, l_h)$ be a $dt$-route that uses a minimum number of different lines, and let $\theta_0 := d, \theta_1, \ldots, \theta_{h-1}, \theta_h := t$ be some suitable transfer stops of $r$. We first show that there always exists a $dt$-route $\tilde{r}$ that uses every line at most once and in total does not use more different lines than $r$ does. Suppose that some line $l$ occurred multiple times in $r$. Let $i$ be the smallest index such that $l_i = l$, and let $j$ be the largest index such that $l_j = l$. The line $l_i$ is boarded at $\theta_{i-1}$, the line $l_j$ is left at $\theta_j$. Now, $r' = (l_1, \ldots, l_i, l_{j+1}, \ldots, l_h)$ is still a feasible route with the transfers stops $\theta_0, \ldots, \theta_{i-1}, \theta_j, \ldots, \theta_h$: for $k < i$ and $k > j$, the line $l_k$ visits $\theta_{k-1}$ before $\theta_k$ (because $r$ is a route), and the line $l_i$ visits $\theta_{i-1}$ before $\theta_j$ (because both stops appear on $l$, and $l$ can be traveled in both directions). In comparison to $r$, $r'$ uses the line $l$ only once. Also, $r'$ obviously does not use more different lines than $r$ does (because we possibly only left lines out). Repeating the above argument for every line $l$ that occurs multiple times in $r$, we obtain a route $\tilde{r}$ in which no line appears more than once, and that still uses a minimum number of different lines.
Now, observe that every line in $L$ appears in a shortest $dt$-route $r_S$ at most once: Suppose that there was some route $r_D$ that used less different lines than $r_S$ does. We could then simply consider $\bar{r}_D$, which cannot have less transfers than $r_S$ has (since $r_S$ by definition has the minimum number of transfers). Hence, every shortest $dt$-route also uses a minimum number of different lines.

Given the undirected transportation network $\mathcal{N}$, a shortest $dt$-route can easily be computed in time $O(|\mathcal{N}|)$ using the solution described in the previous paragraph (where $\Gamma[\mathcal{N}]$ is undirected and Dial’s algorithm is replaced by a breadth-first search that prefers arcs of weight 0 over those with weight 1).

### 3.3 A Straightforward Solution for Listing Routes

In this section, we describe a straightforward algorithm to list all routes whose worst-case running time might be exponential. The next sections will discuss techniques to bound the running time.

**Graph of Line Incidences** Given the set of all lines $\mathcal{L}$, consider the undirected line incidence graph $G_I = (\mathcal{L}, A_I)$ where $A_I$ contains an arc between two lines $l_i$ and $l_j$ if and only if $l_i \cap l_j \neq \emptyset$, i.e. if $l_i$ and $l_j$ have at least one common stop. This graph is similar to the directed hierarchical graph by Müller [46] (where $G_I$ only has one level) and the conflict graph by Delling et al. [24]. Observe that every $dt$-route $r = \langle l_1, \ldots, l_k \rangle \in \mathcal{L}^k$ with $k \in \{1, \ldots, \beta\}$ has the following properties.

1) $l_1$ visits $d$ and $l_k$ visits $t$

2) The vertices $l_1, \ldots, l_k$ form a path in $G_I$ (i.e., $l_i \cap l_{i+1} \neq \emptyset$ for every $i = 1, \ldots, k-1$).

3) There exists a sequence of stops $d = \theta_0, \theta_1, \ldots, \theta_{k-1}, \theta_k = t$ such that for every $i \in \{1, \ldots, k\}$, the line $l_i$ visits $\theta_{i-1}$ (not necessarily directly) before $\theta_i$.

**Algorithmic Idea** The observations from the previous paragraph lead to the following algorithmic idea to list all routes. First, we determine the set $\mathcal{L}_d$ of all lines passing through $d$. Then, for each line $l_1 \in \mathcal{L}_d$, we perform a kind of depth-first search in $G_I$ starting in the vertex $l_1$. The difference to the usual depth-first search is that we do not stop when finding a vertex that has already been found earlier. Each step of the algorithm is characterized by a search state: a partial route $\rho_k = \langle l_1, \ldots, l_k \rangle$ explored so far (initially, $\rho_1 = \langle l_1 \rangle$), and the last transfer stop $\theta_{k-1}$ used that allowed the transfer to the line $l_k$ when the journey starts at $d$ and follows the route $\rho_k$ (initially, $\theta_0 = d$). In each step of the depth-first search, when visiting a vertex $l_k$, the algorithm tries to
Figure 6  The lines $l_k$ and $l_{k+1}$ have the stops $l_k \cap l_{k+1} = \{s_3, s_6, s_{11}, s_{14}, s_{15}\}$ in common. If the current transfer stop $\theta_{k-1}$ for some partial route $\rho_k = \langle l_1, \ldots, l_k \rangle$ was $s_2$, then the stop $s_3$ is the current transfer stop $\theta_k$ for the partial route $\rho_{k+1} = \langle l_1, \ldots, l_k, l_{k+1} \rangle$. If $\theta_{k-1}$ was $s_{12}$, then $\theta_k$ would be $s_6$.

extend the partial route $\rho_k = \langle l_1, \ldots, l_k \rangle$ to a neighbor $l_{k+1}$ of $l_k$ in $G_I$. After that an appropriate transfer stop $\theta_k$ has to be computed. More specifically, to process a search state with the partial route $\rho_k = \langle l_1, \ldots, l_k \rangle$, and the current transfer stop $\theta_{k-1}$, we perform the following tasks:

1) We check whether the line $l_k$ contains the stop $t$ and whether $l_k$ visits $t$ after $\theta_{k-1}$. If this is the case, then the partial route $\rho$ is feasible, and hence it is output as one of the solutions in $\mathcal{R}$.

2) If the partial route $\rho$ has length at most $\beta - 1$, we try to extend it in the following way: for every neighbor $l_{k+1}$ of $l_k$ in $G_I$, we compute the transfer $\theta_k$ that reaches $l_{k+1}$ as early as possible (with respect to $l_{k+1}$; see Figure 6 for an example). This can be achieved by considering all stops in $l_k \cap l_{k+1}$ that $l_k$ visits after $\theta_{k-1}$, and selecting among these the vertex $\theta_k$ whose position on $l_{k+1}$ is as small as possible. If such a vertex does not exist (because $l_k \cap l_{k+1}$ does not contain any stops that $l_k$ visits after $\theta_{k-1}$), we are done. Otherwise, we continue with the new search state consisting of the new partial route $\rho_{k+1} = \langle l_1, \ldots, l_k, l_{k+1} \rangle$ and the new transfer vertex $\theta_k$.

Notice that we always try to extend $\rho_k$, even if $\rho_k$ was already output in the first step. Also, notice that a line may occur multiple times in the partial route $\rho_k$. Figure 7 shows an example why both these assumptions are reasonable from a practical point of view.

Algorithm  We now give a more technical description of the algorithm described in the previous paragraph. First, notice that we do not need to explicitly construct the line incidence graph $G_I$; it was just an auxiliary construction to explain the basic idea of the algorithm. Also, observe that it is not reasonable to consider every stop in $l_k \cap l_{k+1}$ for every line $l_{k+1}$. Instead, we only consider all successors of $\theta_{k-1}$ on $l_k$ with increasing distance from $\theta_{k-1}$. We
A transportation network with one-way streets induced by a line $l_1 = \langle s_1, \ldots, s_{12} \rangle$ (solid) and a line $l_2 = \langle s_{13}, s_2, s_{11}, s_{14} \rangle$ (dotted). To travel from $d = s_1$ to $t = s_{12}$, it is reasonable to use $l_1$ until $s_2$, after that use $s_2$ from $s_2$ to $s_{11}$ and from there finally use $l_1$ again.

keep track which other lines can be reached from $l_k$, and for each such line $l$ which is the transfer stop $\theta^l_k$ by which $l$ is reached as early as possible (i.e., there is no stop $\zeta^l_k$ visited by $l_k$ after $\theta^l_{k-1}$ that $l$ visits earlier than $\theta^l_k$). Algorithm 1 shows the details. To list all routes we simply invoke $\text{ListRoutes}(\langle l_1 \rangle, d)$ for every $l_1 \in L_d$.

**Algorithm 1: ListRoutes($\langle l_1, \ldots, l_k \rangle, \theta^l_{k-1}$)**

```
1 for $l \in L$ do $\theta^l_k \leftarrow \infty$
2 for each successor $v$ of $\theta^l_{k-1}$ on $l_k$ in increasing order do
3     if $v = t$ then OUTPUT($\langle l_1, \ldots, l_k \rangle$)
4     for $l \in L_v$ do
5         if $\theta^l_k = \infty$ or $l$ visits $v$ earlier than $\theta^l_k$ then $\theta^l_k \leftarrow v$
6 for $l \in L$ do
7     if $\theta^l_k \neq \infty$ and $k < \beta$ then
8         ListRoutes($\langle l_1, \ldots, l_k, l \rangle, \theta^l_k$)
```

Running time  The drawback of this algorithm is that it may need $\Omega(\Delta^\beta)$ many steps in the worst case, where $\Delta$ is the maximum degree of a vertex in $G_I$. On the other hand, there might be very few or even no routes that are output. The following sections discuss how the algorithm can be slightly modified such that the theoretical worst-case running time can be bound by a polynomial in the size of the input and the number of listed routes. Nevertheless, already this algorithm is reasonably fast for medium-sized transportation networks such as the one in Zürich; see Chapter 5 for details.

Practical Considerations  Algorithm 1 generates all routes that comply with the definition given in Section 2.3. However, especially if $\beta$ is much larger than
Figure 8 An extract of the transportation network in Zürich when traveling from $d = \text{Vogelsangstrasse}$ to $t = \text{Hinterbergstrasse}$. The lines are $l_1 = (d, s_1, s_2, s_3, s_4, s_5)$ and $l_2 = (s_5, s_4, t, s_1, d)$. Obviously, the only $dt$-route is $(l_1, l_2)$. In reality, $l_1$ and $l_2$ have the same identifier and hence can be considered equivalent; the lines split only because they pass through one-way streets.

the $d_L(d, t)$ (i.e., the minimum length of any $dt$-route), then many routes are undesirable from a traveler’s perspective. For example, one could take some line $l$, take the equivalent line in the opposite direction and then use $l$ again. Although this is certainly not forbidden, such a route is obviously not relevant in practice. Algorithm 1 can be slightly modified to avoid the generation of at least some non-relevant routes: First, in step 3 we immediately terminate the loop once a route is output. Hence we only extend our search to some line $l$ if $l_k$ and $l$ have a common stop $\theta_k^l$ that $l_k$ visits after $\theta_{k-1}$ and before $t$. Second, in step 8, we branch only if $l$ and $l_k$ differ in at least one stop. In practice one could for example check whether $l$ and $l_k$ have a different identifier (such as Bus 31 and Bus 32). Of course, these modifications still allow the generation of some non-relevant routes. Moreover, the second modification is problematic when the network contains a situation similar to the one shown in Figure 8, because some stops could not be reached any more, or only via a detour.

A solution might be to introduce some dominance concept where a route $r$ dominates some other route $r'$ if $r'$ is a “slight modification” of $r$ (e.g., $r$ with a “detour”). Nevertheless, the aforementioned problems are only relevant from a practical point of view, not from a conceptional one: if some route $r$ dominates some other route $r'$, then $r'$ will have a larger travel time and more transfers, and thus will never be more robust than $r$. Also it will likely be disregarded in a later phase when the most robust routes are being computed. The only problem with the generation of dominated routes is that it increases the running time. On the other hand, if we set $\beta = d_L(d, t)$ or $\beta = d_L(d, t) + 1$, then this is still feasible in practice (again, see Chapter 5).

### 3.4 A Polynomial Delay Algorithm for Listing Routes

To improve the listing algorithm described in the previous section, we use a strategy similar to the one by Rizzi et al. [56] where a recursive call is invoked only if it is guaranteed to output at least one solution (see Section 3.1). As a preliminary step we compute the values $d_L^s(s, t, l)$ for every stop $s$ and every line
l ∈ L_s (using the solution discussed in Section 3.2). After that, we again keep track of the current partial route ρ_k = ⟨l_1, ..., l_k⟩ and the last used transfer θ_{k-1}. However, while Algorithm 1 tries to extend ρ_k to every possible line, we now only extend ρ_k to a line l_{k+1} if from that line t can be reached using at most β − k many lines (including l_{k+1}). This can easily be checked: when considering a partial route ρ_k = ⟨l_1, ..., l_k⟩ and a line l with θ_{l_k} ≠ ∞, t can be reached from l with at most β − k many lines if and only if \( d_{l_{k+1}}^L(θ_{l_k}, t, l) ≤ β − k \). Algorithm 2 shows the details. As before, one can simply list all routes by invoking ListRoutes(⟨l_1⟩, d) for every \( l_1 ∈ L_d \) where \( d_{l_1}^L(d, t, l_1) ≤ β \). As the following theorem shows, extending ρ_k only if this definitely leads to a solution that is output guarantees a polynomial delay.

**Algorithm 2: ListRoutes(⟨l_1, ..., l_k⟩, θ_{k-1})**

1. Compute \( d_{l_1}^L(s, t, l) \) for each \( s ∈ S \) and \( l ∈ L_s \)
2. for \( l ∈ L \) do \( θ_{l_k}^l ← ∞ \);
3. for each successor v of θ_{k-1} on l_k in increasing order do
   4. if \( v = t \) then OUTPUT(⟨l_1, ..., l_k⟩) ;
   5. for \( l ∈ L_v \) do
     6. if \( θ_{l_k}^l = ∞ \) or l visits v earlier than \( θ_{l_k}^l \) then \( θ_{l_k}^l ← v \);
7. for \( l ∈ L \) do
   8. if \( θ_{l_k}^l ≠ ∞ \) and \( d_{l_{k+1}}^L(θ_{l_k}^l, t, l) ≤ β − k \) then
   9. ListRoutes(⟨l_1, ..., l_{k+1}⟩, θ_{l_k}^l)

**Theorem 4.** Algorithm 2 has delay \( O(β|N|) \), and its total time complexity is \( O(β|N| · K) \), where where \( |N| = |S| + \sum_{l ∈ L} |l| \) is the input size and \( K \) is the number of returned solutions. Moreover, the space complexity is \( O(|N|) \).

**Proof.** Remember that testing whether a given stop s is visited by a line l can be performed in constant time (Chapter 2). The same is true for testing whether a line visits a stop s earlier than some other stop s'.

Step 1 can be computed using \( O(|N|) \) operations and requires \( O(|N|) \) space (Section 3.2). Steps 3–6 can also be implemented to run in time \( O(|N|) \), as every step takes only constant time and is performed at most once for every stop s and every line l containing s. Since the recursive calls in step 9 are only performed if it is guaranteed to output a solution, we observe (similar to [56]) that the height of the recursion tree is bounded by \( O(β) \), hence the delay of the algorithm is \( O(β|N|) \). Hence the time complexity is in \( O(β|N| · K) \).
The earliest transfer from \( l_3 \) to \( l_1 \) is \( s_3 \). However, the earliest transfer using the route \( \langle l_4, l_3, l_1 \rangle \) is \( s_4 \). We have \( \nu(l_1, l_2, s_1) = s_2 \) and \( \nu(l_1, l_2, s_k) = \infty \) for every \( k \neq 1 \). We have \( \nu(l_1, l_3, s_5) = \nu(l_1, l_3, s_6) = \infty \), \( \nu(l_1, l_3, s_3) = \nu(l_1, l_3, s_4) = s_5 \) and \( \nu(l_1, l_3, s_1) = \nu(l_1, l_3, s_2) = s_3 \).

### 3.5 A Faster Solution with Preprocessing

Although we already have a polynomial-delay algorithm for listing all routes, an overall running time of \( O(\beta |N| \cdot K) \) is undesirable from a practical point of view. Algorithm 2 has delay \( O(\beta |N|) \) for two reasons: 1) initially we compute the values \( d_{L,N}(\cdot, t, \cdot) \), and 2) for every partial route \( p_k = \langle l_1, \ldots, l_k \rangle \) we investigate all possible transfer stops from \( l_k \) to other lines to find the optimal one for every line. Issue 1) can easily be solved by computing the values \( d_{L,N}(s, t, l) \) for every \( s, t \in S \) and every line \( l \in L \) in advance and then storing them. Since for every \( t \) there are at most \( O(|N|) \) many values \( d_{L,N}(\cdot, t, \cdot) \) and all of them can be computed in time \( O(|N||S|) \), we need overall time \( O(|N||S|) \). To solve issue 2), we precompute for every line \( l \), every stop \( s \) on \( l \) and every line \( l' \neq l \) the stop \( \nu(l, l', s) \) which is visited by \( l \) after \( s \) and by which \( l' \) is reached as early as possible (on \( l' \), see Section 3.3). If no such stop exists, we set \( \nu(l, l', s) = \infty \). For every line \( l = \langle s_1, \ldots, s_k \rangle \in L \), the values \( \nu(l, \cdot, \cdot) \) can be computed as follows. We consider the stops \( s_k, \ldots, s_1 \) in this order. We set \( \nu(l, l', s_k) = \infty \) for every \( l' \in L \). After that, considering a stop \( s_i \), we set

\[
\nu(l, l', s_i) = \begin{cases} 
  s_{i+1} & \text{if } l' \in L_{s_{i+1}} \text{ and } \nu(l, l', s_{i+1}) = \infty \\
  s_{i+1} & \text{if } l' \in L_{s_{i+1}} \text{ and } l' \text{ visits } s_{i+1} \text{ before } \nu(l, l', s_{i+1}) \\
  \nu(l, l', s_{i+1}) & \text{otherwise}
\end{cases}
\]

Since the computation of each entry requires only constant time, the values \( \nu(l, l', s) \) can be computed using time and space \( O(|N||L|) \). Hence, for preprocessing time and space \( O(|N|(|S| + |L|)) \) suffice. Now, however, \( dt \)-route listing queries can be performed much faster using the following algorithm. Figure 9 gives an example for a transportation network and the corresponding values \( \nu(l, l', s_i) \).

**Theorem 5.** The values \( d_{L,N}(s, t, l) \) and \( \nu(l, l', s) \) can be precomputed using time
3.6 Listing all Paths

Algorithm 3: \textsc{ListRoutes}(⟨l_1, \ldots, l_k⟩, \theta_{k-1})

1 \textbf{if} \ l_k \textit{ visits} \ t \ \textbf{after} \ \theta_{k-1} \ \textbf{then} \ \textbf{OUTPUT}(⟨l_1, \ldots, l_k⟩) ;
2 \textbf{for} \ l \in \mathcal{L} \ \textbf{do}
3 \quad \theta^t_k \leftarrow \nu(l_k, l, \theta_{k-1})
4 \quad \textbf{if} \ \theta^t_k \neq \infty \ \textbf{and} \ d_{\mathcal{N}}^t(\theta^t_k, t, l) \leq \beta - k \ \textbf{then}
5 \quad \textbf{ListRoutes}(⟨l_1, \ldots, l_k, l⟩, \theta^t_k)

and space $\mathcal{O}(|\mathcal{N}|(|\mathcal{S}| + |\mathcal{L}|))$. Assuming that these values have been precomputed, Algorithm 3 has delay $\mathcal{O}(\beta |\mathcal{L}|)$, and its total time complexity is $\mathcal{O}(\beta |\mathcal{L}| \cdot K)$, where $K$ is the number of returned solutions.

\textbf{Proof.} The straightforward proof is similar to the proof of Theorem 4. \hfill \Box

To see the speedup, remember that Algorithm 2 has a delay of $\mathcal{O}(\beta |\mathcal{N}|)$ while Algorithm 3 has a delay of only $\mathcal{O}(\beta |\mathcal{L}|)$. In real networks, the input size $|\mathcal{N}| = |\mathcal{S}| + \sum_{l \in \mathcal{L}} |l|$ is usually way larger than $|\mathcal{L}|$ is.

3.6 Listing all Paths

\textbf{Motivation} In the previous sections we studied how all bounded feasible routes can efficiently be listed. However, from the perspective of a traveler who (at least briefly) knows the structure of the transportation network $\mathcal{N}$, one could argue that not every route is a good alternative, because many routes may share a similar or even the same path in $\mathcal{N}$ (see Figure 10).

In this section we therefore study the problem of listing suitable $dt$-paths in $\mathcal{N}$. We already saw in Section 3.1 that listing all $dt$-paths in $\mathcal{N}$ is a well-studied problem. However, it is not reasonable to list all paths, because some of them may require a large number of transfers. Chapter 2 explained that every line corresponds to a path in $\mathcal{N}$. We therefore formulate the listing problem

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{Figure10.png}
\caption{An extract of the transportation network in Zürich when traveling from $d = \text{Sihlpost}$ to $t = \text{ETH/Universitätsspital}$. The lines are $l_{PB} = ⟨s_3, t⟩$, $l_3 = l_{31} = ⟨d, s_1, s_2, s_3, s_5, s_6⟩$, $l_6 = l_{10} = ⟨s_2, s_3, s_4, t⟩$, $l_9 = ⟨s_6, s_7, t⟩$ and $l_{14} = ⟨d, s_1, s_2⟩$. The set $\mathcal{R}_{dt}^1$ is empty, and $\mathcal{R}_{dt}^2$ contains already 10 routes. However, observe that only three routes differ substantially (i.e., have different paths in the network).
} \end{figure}
Listing all Solutions

Figure 11 A transportation network induced by the stops $S = \{d, s_1, \ldots, s_8, t\}$ and the lines $L = \{l_1, \ldots, l_5\}$ where $l_1 = \langle d, s_1, s_2, s_4, s_5 \rangle$, $l_2 = \langle s_1, s_3, s_4, s_6, s_8, t \rangle$, $l_3 = \langle s_5, s_4, s_6, s_8, t \rangle$, $l_4 = \langle s_5, s_7 \rangle$, $l_5 = \langle s_7, t \rangle$. The set $R^2_{dt}$ contains the routes $\langle l_1, l_2 \rangle$ and $\langle l_1, l_3 \rangle$. Hence, $P^2_{dt}$ contains the paths $\langle d, s_1, s_2, s_4, s_6, s_8, t \rangle$ (induced by the routes $\langle l_1, l_2 \rangle$ and $\langle l_1, l_3 \rangle$) and $\langle d, s_1, s_3, s_4, s_6, s_8, t \rangle$ (induced by the route $\langle l_1, l_2 \rangle$). The path $\langle d, s_1, s_2, s_4, s_5, s_7, t \rangle$ is not included in $P^2_{dt}$, because it is only induced by the route $\langle l_1, l_4, l_5 \rangle$ which has length 3. Also, $\langle d, s_1, s_2, s_4, s_5, s_4, s_6, s_8, t \rangle$ (induced by $\langle l_1, l_3 \rangle$) is not included because it visits $s_4$ twice and hence is not a simple walk.

Figure 12 An extract of the transportation network in Zürich when traveling from $d =$ Bezirksgebäude to $t = $ ETH/Universitätsspital. The lines are $l_3 = \langle d, s_2, s_8, s_9, \ldots, s_{13}, s_{15} \rangle$ (solid) and $l_9 = \langle s_1, s_2, \ldots, s_7, s_{13}, s_{14}, t \rangle$ (dotted). The set $R^2_{dt}$ contains only the route $\langle l_3, l_9 \rangle$, but this route induces two paths in $N$.

As follows: Given a set of stops $S$, a set of lines $L$, the induced transportation network $N = N_{S, L}$, two stops $d, t \in S$ and an integer $\beta \in \mathbb{N}$, compute the set $P^\beta_{dt}$ of all $dt$-paths (simple $dt$-walks) $\pi$ in $N$ for which $R^\beta_{dt}$ contains a route $r = \langle l_1, \ldots, l_k \rangle$ such that $\pi$ is the concatenation of suitable subpaths of $l_1, \ldots, l_k$ (see Figure 11 for an example).

For solving this problem, one could for example use the algorithm in Section 3.4 to generate the set $R = R^\beta_{dt}$ explicitly, and then for each route in $R$ compute the corresponding paths. However, this approach may take too much time, because there is no one-to-one correspondence between routes and paths: many routes can correspond to the same path, and one route can give rise to multiple paths (see Figure 12). To some extent, such an approach was used by Xu et al. [68] whose algorithm outputs each path as often as some route
3.6 Listing all Paths

realizes this path (instead of just once), and also only lists paths with no more than two transfers.

In the worst case, Algorithm 2 may generate exponentially many routes that correspond to the same path: Imagine that we have the transfer stops \( \theta_0 = d, \theta_1, \ldots, \theta_{\beta-1}, \theta_\beta = t \) and that for every \( i \in \{1, \ldots, \beta\} \) there exist two lines \( l_i^1 \) and \( l_i^2 \) that both visit \( \theta_{i-1} \) before \( \theta_i \), and that both have the same subpath between these two stops in \( N \). Then, for every \( (b_1, \ldots, b_\beta) \in \{1,2\}^\beta \), \( \langle l_i^{b_i}, \ldots, l_i^{b_\beta} \rangle \) is a feasible route in \( R_{dt}^\beta \), hence \( R_{dt}^\beta \) contains \( 2^\beta \) many routes that induce the same path in \( N \). Moreover, since every path \( \pi \in P_{dt}^\beta \) should be output only once, we would need to store every computed path to be able to check at a later point whether it has already been output or not.

Another straightforward solution for the above problem might be to construct an auxiliary graph from \( N \) on which one of the well-known algorithms for listing paths (e.g., Yen’s algorithm) is run. For example, one could create a directed graph \( G_X = (V,A_X) \) where \( A_X \) contains an arc between \( v \) and \( w \) if there exists a line that visits \( v \) (not necessarily directly) before \( w \). Every \( dt \)-path in \( G_X \) of length at most \( \beta \) corresponds to some \( dt \)-path in \( N \): A path \( \langle d = \theta_0, \theta_1, \ldots, \theta_{k-1}, \theta_k = t \rangle \) in \( G_X \) defines a sequence of \( k+1 \) transfer stops which define a feasible route in \( R_{dt}^k \), and from which a corresponding path in \( N \) can easily be reconstructed. However, as in the previous idea, exponentially many paths in \( G_X \) might correspond to only one path in \( N \). The argument is similar as before: Consider the stops \( S = \{d = s_0^1, s_0^2, s_1^1, s_1^2, \ldots, s_\beta^1, s_\beta^2 = t\} \) and the lines \( L = \{l_1, \ldots, l_\beta\} \) where for every \( i \in \{1, \ldots, \beta\} \), \( l_i = \langle s_{i-1}^1, s_{i-1}^2, s_{i}^1, s_{i}^2 \rangle \). Now, \( P_{dt}^\beta \) contains exactly one path (namely, \( \langle d, s_0^2, s_1^1, s_1^2, \ldots, s_\beta^1, t \rangle \)), but for every \( (b_1, \ldots, b_{\beta-1}) \in \{1,2\}^{\beta-1} \), \( G_X \) contains the path \( \langle d, s_1^{b_1}, \ldots, s_{\beta-1}^{b_{\beta-1}}, t \rangle \), hence there might be an exponential gap between the running time and the sum of the input and the output size.

In the following, we develop a polynomial delay algorithm that uses only \( O(|N|) \) space. It is again based on the idea of Rizzi et al. [56] (see Section 3.1). Since the algorithm removes certain vertices from the lines and the induced transportation network, we first have to extend some of our previous definitions.

**Deleting Stops** For a path \( \pi \) and a set of stops \( S \), let \( S - \pi \) be the set \( S \) from which every stop \( s \in \pi \) is removed. For a path \( \pi \) and a line \( l \in L \), let \( l - \pi \) be the union of (possibly degenerate) paths that we obtain after removing every stop \( s \in \pi \) and all arcs adjacent to \( s \) from \( l \) (see Figure 13 for an example). For simplicity, we also call each of these unions of paths a line, although they might be disconnected, degenerated (i.e., consist of only one stop), or even empty (i.e., they might not contain any stop). However, we note that the algorithms in Section 3.2 also work when we allow disconnected, degenerated and empty...
Let \( l = \langle s_1, s_2, s_3, \ldots, s_8 \rangle \) be a line (solid) and \( \pi = \langle d, s_2, s_3, s_4, s_6, t \rangle \) be a path (dotted). Then, \( l - \pi = \langle s_1, s_5, s_7, s_8 \rangle \) is the disjoint union of the degenerate lines (\( s_1 \)) and (\( s_5 \)), and the non-generated line (\( s_7, s_8 \)).

A transportation network with the lines \( l_1 = \langle d, s_1, s_2, s_3, u, v \rangle \) and \( l_2 = \langle s_4, u, v \rangle \). Let \( \pi_{du} = \langle d, s_1, s_2, s_3, u \rangle \) be a path and \( a = (u, v) \) be an arc. Now \( d_N(\pi_{du}, a, l_1) = 1 \) and \( d_N(\pi_{du}, a, l_2) = 2 \) (because \( l_1 \) has to be used to travel from \( d \) to \( u \), and \( l_2 \) can be used to travel from \( u \) to \( v \)).

Distance Measures

For two vertices \( d, u \in S \), a \( du \)-path \( \pi_{du} \) in \( N \), an arc \( a = (u, v) \) and a line \( l \in L_a \), let \( d_N(\pi_{du}, a, l) \) be the minimum length of a route that ends with the line \( l \) and that gives rise to the path \( \pi_{dv} \) in \( N \) (see Figure 14 for an example). Notice that this minimum is well defined, because a route with the aforementioned properties always exists: the line \( l \) contains \( a \), and for every arc \( (s, s') \) in \( \pi_{du} \), there exists at least one line in \( L \) that visits \( s \) directly before \( s' \) (by definition of \( N \)). For a transportation network \( N' = N_{S',L'} \) induced by a set of stops \( S' \subseteq S \) and a set of (possibly disconnected) lines \( L' \) that use only stops in \( S' \), two stops \( v, t \in S' \) and a line \( l \in L_v \), let \( d_{N'}^{L'}(v, t, l) \) be the \( L' \)-distance from \( v \) to \( t \) in \( N' \) such that \( l \) is the first line used (see Figure 15). More formally, if \( R_{vt}^\infty \) is the set of all \( vt \)-routes of arbitrary length that start with \( l \) and use only lines from \( L' \), then \( d_{N'}^{L'}(v, t, l) \) is the minimum length of a route in \( R_{vt}^\infty \). Notice that for \( N' = N \) and \( L' = L \), the new definition of \( d_{N'}^{L'}(v, t, l) \) is compatible to the former definition of \( d_N^L(v, t, l) \) given in Chapter 2.

Algorithmic Idea

We now formulate an algorithm that systematically lists the paths in \( P_{dt}^{\beta} \), i.e. the \( dt \)-paths \( \pi \) in \( N \) for which \( R_{dt}^\beta \) contains a route \( r = \)
Listing all Paths

Figure 15 A transportation network $N'$ with the lines $L' = \{l_1, \ldots, l_4\}$ where $l_1 = \langle v, s_1, s_2 \rangle \cup \langle s_3, s_4, t \rangle$ (disconnected), $l_2 = \langle s_2, s_3 \rangle$, $l_3 = \langle v, s_5 \rangle$ and $l_4 = \langle s_5, t \rangle$. We have $d_{N'}(v, t, l_1) = 3$ (because $l_1$ can only be used until $s_2$, from there $l_2$ has to be used until $s_3$, and from there $l_1$ again can be used to travel to $t$), and $d_{N'}(v, t, l_3) = 2$ (following the route $\langle l_3, l_4 \rangle$).

$\langle l_1, \ldots, l_k \rangle$ such that $\pi$ is the concatenation of suitable subpaths of $l_1, \ldots, l_k$. To obtain a polynomial-delay algorithm that uses only $O(|N|)$ space, our algorithm works similar to the one by Rizzi et al. [56]: the transportation network is traversed in a depth-first search fashion starting from $d$, and the solution space $P_{dt}^\beta$ is recursively partitioned at every call until it contains exactly one solution (i.e., one path).

When the algorithm considers a $du$-path $\pi_{du}$, we first check whether $u = t$. In that case, $\pi_{du}$ is output. Otherwise, we compute the graph $N$ corresponding to the transportation network $N$ from which all vertices (and all adjacent edges) in $\pi_{du}$ are removed. To bound the running time of the algorithm we maintain the invariant (I) that the current partition (i.e., the paths in $P_{dt}^\beta$ with prefix $\pi_{du}$) contains at least one solution. More concretely, we require that $N$ contains at least one $ut$-path $\pi_{ut}$ that extends $\pi_{du}$ so that $\pi_{du} \cdot \pi_{ut} \in P_{dt}^\beta$. Here, the algorithm of Rizzi et al. [56] can no longer be applied: new ideas to maintain the invariant are necessary because our objective is to list only paths $\pi$ which are induced by a suitable length-bounded route.

Checking whether to recurse or not Let $\pi_{du}$ be the $du$-path that the algorithm currently considers, $S' = S - \pi_{du}$, $L' = L - \pi_{du}$, $N' = N - \pi_{du} = NS', L'$ and $v \in N_{N'}(u) \cap N'$, i.e., $v$ is a neighbor of $u$ that is not contained in $\pi_{du}$. We recursively continue on $\pi_{du} \cdot (u, v)$ only if the invariant (I) is satisfied, i.e. if $P_{dt}^\beta$ contains a path with prefix $\pi_{du} \cdot (u, v)$. This is the case if and only if

$$\min \{d_N(\pi_{du}, a, l_i) - \delta_{ij} + d_{N'}^L(v, t, l_j) \mid l_i \in L_a \text{ and } l_j \in L_v\} \leq \beta. \quad (6)$$

Basically, $\min\{d_N(\pi_{du}, a, l_i) - \delta_{ij} + d_{N'}^L(v, t, l_j) \mid l_i \in L_a \text{ and } l_j \in L_v\}$ is the minimum length of a route that gives rise to a path in $N$ with prefix $\pi_{du} \cdot a$.

Computing $d_N(\pi_{du}, a, l_i)$ and $d_{N'}^L(v, t, l_j)$ Given a transportation network $N'$ induced by the stops $S'$ and the lines $L'$, and a stop $t \in S'$, we can compute
the values $d'_{N'}(v,t,l_j)$ for every $v \in S'$ and every $l_j \in L'$ using the solution presented in Section 3.2 for computing the values $d_{\tilde{N}'}(v,t,l)$: we consider the $L'$-distances from $t$ in the reverse graph $N'R$ (with all the arcs in $N'$ reversed). Considering $N'$ instead of $N$ ensures that lines do not use vertices that have been deleted in previous recursive calls of the algorithm. Thus we compute $\Gamma[N'R]$ and run Dial’s algorithm on the vertex $t$. Then, the length of a shortest path in $\Gamma[N'R]$ from $t$ to $v_l_j$ is exactly $d'_{N'}(v,t,l_j)$.

Given the transportation network $\tilde{N}$ and a path $\pi_{du}$ in $\tilde{N}$, the values $d_{\tilde{N}}(\pi_{du},a,l_i)$ need to be computed only for arcs $a = (u,v) \in N$ with $v \neq \pi_{du}$ (i.e., only for arcs from $u$ to a vertex $v \in N_-(u) \cap N'$), and only for lines $l_i \in L_a$. Consider the graph $N''$ that contains every arc from $\pi_{du}$ and every arc $(u,v) \in N$ with $v \neq \pi_{du}$, and that contains exactly the vertices incident to these arcs. Now we compute $H = \Gamma[N'']$ (see Section 3.2) and run Dial’s algorithm on the vertex $d$. For every $v \in N_-(u) \cap N'$ and every line $l_i \in L(u,v)$, the length of a shortest path in $H$ from $d$ to $v_l_i$ is exactly $d_{\tilde{N}}(\pi_{du},u,v,l_i)$.

As previously mentioned, a similar idea (traversing the graph from the target stop $t$ backwards to compute for every stop $s$ the minimum number of transfers of an $st$-journey) was used by Disser et al. [29]. However, they did not use it in the context of listing all length-bounded paths, but as a speedup heuristic in a multi-criteria version of Dijkstra’s algorithm to avoid the generation of unnecessary labels. Here, we also go one step further and take the first line of the corresponding route into account.

Algorithm Algorithm 4 shows the details of the aforementioned approach. To limit the space consumption of the algorithm, we do not pass the transportation network $N'$ as a parameter to the recursive calls, but compute it at the beginning of each recursive call from the current prefix $\pi_{du}$. For the same reason, we do not perform the recursive calls immediately in step 8, but first create a list $V_R \subseteq V$ of vertices for which the invariant (I) is satisfied, and only then recurse on $(v,\pi_{du},v)$ for every $v \in V_R$. To list all paths in $P_{dt}^\beta$, we invoke $\text{ListPaths}(d,\langle d \rangle)$.

**Theorem 6.** Algorithm 4 has delay $O(|S||\tilde{N}|)$. The total time complexity is $O(|S||\tilde{N}| \cdot K)$, where is $K$ the number of returned solutions. Moreover, the space complexity is $O(|\tilde{N}|)$.

**Proof.** As in [56] we first analyze the cost of a given call to the algorithm without including the cost of the recursive calls performed inside. Theorem 1 states that steps 3 and 4 can be performed in time $O(|\tilde{N}|)$. We will now show that the steps 6–8 can be implemented in time $O(|\tilde{N}|)$. Notice that for a fixed prefix $\pi_{du}$ and a fixed vertex $v \in N_-(u) \cap N'$, for computing the minimum in step 7, we need to consider only the values $d_{\tilde{N}}(\pi_{du},u,v,l_i)$ that
3.6 Listing all Paths

Algorithm 4: ListPaths(u, π_{du})

1. if u = t then OUTPUT(π_{du}); return;
2. \( L' \leftarrow \mathcal{L} - \pi_{du}; \ N' \leftarrow \mathcal{N} - \pi_{du} \)
3. Compute \( d_N'(\pi_{du}, (u, v), l_i) \) for each \( v \in N_N(u) \cap N' \) and \( l_i \in \mathcal{L}_{(u,v)} \)
4. Compute \( d_{N'}^l(v, t, l_j) \) for each \( v \in N_N(u) \cap N' \) and \( l_j \in \mathcal{L}_v \)
5. \( V_R \leftarrow \emptyset \)
6. for \( v \in N_N(u) \cap N' \) do
   7. \( d \leftarrow \min\{d_N(\pi_{du}, (u, v), l_i) + d_{N'}^l(v, t, l_j) - \delta_{ij} \mid l_i \in \mathcal{L}_{(u,v)} \) and \( l_j \in \mathcal{L}_v \} \)
   8. if \( d \leq \beta \) then \( V_R \leftarrow V_R \cup \{v\} ; \)
9. for \( v \in V_R \) do
10. \( \text{ListPaths}(v, \pi_{du} \cdot (u, v)) \)

are minimum among all \( d_N'(\pi_{du}, (u, v), \cdot) \), and only the values \( d_{N'}^l(v, t, l_j) \) that are minimum among all \( d_{N'}^l(v, t, \cdot) \). Let \( \Lambda_v \subseteq \mathcal{L}_{(u,v)} \) be the list of all lines \( l_i \) for which \( d_N(\pi_{du}, (u, v), l_i) \) is minimum among all \( d_N(\pi_{du}, (u, v), \cdot) \). Analogously, let \( \Lambda_v' \subseteq \mathcal{L}_v \) be the list of all lines \( l_j \) for which \( d_{N'}^l(v, t, l_j) \) is minimum among all \( d_{N'}^l(v, t, \cdot) \). Let

\[
\mu_v = \min \left\{ d_N(\pi_{du}, (u, v), l_i) \mid l_i \in \Lambda_v \right\},
\]

\[
\mu'_v = \min \left\{ d_{N'}^l(v, t, l_j) \mid l_j \in \Lambda_v' \right\}
\]

be the minimum values of \( d_N(\pi_{du}, (u, v), \cdot) \) and \( d_{N'}^l(v, t, \cdot) \), respectively. Both values as well as the lists \( \Lambda_v \) and \( \Lambda_v' \) can be computed in steps 3 and 4, and their computation only takes overall time \( O(|\mathcal{N}|) \). Now the expression in step 7 evaluates to \( \mu_v + \mu'_v \) if \( \Lambda_v \cap \Lambda_v' = \emptyset \), and to \( \mu_v + \mu'_v - 1 \) otherwise. Assuming that \( \Lambda_v \) and \( \Lambda_v' \) are ordered ascendingly by the index of the contained lines, it can easily be checked with \( |\Lambda_v| + |\Lambda_v'| \leq |\mathcal{L}_{(u,v)}| + |\mathcal{L}_v| \) many comparisons if their intersection is empty or not. Using this method, each of the values \( d_N(\pi_{du}, \cdot, \cdot) \) and \( d_{N'}^l(\cdot, t, \cdot) \) is accessed exactly once (when computing \( \Lambda_v \) and \( \Lambda_v' \)), and since each of these values has a unique corresponding vertex in the graphs \( H \) and \( \Gamma[\mathcal{N'}'] \), there exist at most \( O(|\mathcal{N}|) \) many such values. Thus, the running time of the steps 6–8 is bounded by \( O(|\mathcal{N}|) \) which is also an upper bound on the running time of Algorithm 4 (ignoring the recursive calls).

Notice that \( \Lambda_v \) and \( \Lambda_v' \) can be sorted without using a comparison-based sorting method like Mergesort. We instead use an auxiliary array \( \alpha \) indexed from 1 to \( |\mathcal{L}| \), store a pointer to \( l_i \) at position \( i \) for every \( l_i \in \Lambda_v \), and then collect the pointers in ascending order by considering \( \alpha[i] \) for \( i = 1, \ldots, |\mathcal{L}| \).
Listing all Solutions

The set \( \Lambda^i_v \) can be ordered analogously. The overall time required is \( O(\|L\|) \subseteq O(\|N\|) \), hence we save an additional factor of \( \Theta(\log |L|) \) in comparison to comparison-based sorting methods.

To analyze the delay of \( O(\|S\|\|N\|) \), we again use an argument similar to one in [56]: Since every recursive call appends one vertex to the current partial path \( \pi_{du} \) and the length of \( \pi_{du} \) is bounded by \( |S| \), the height of the recursion tree is bounded by \( |S| \). Hence, after a path is output, at most \( |S| \) many previous recursive calls terminate and at most \( |S| \) many new recursive calls are invoked until the next path is output.

For analyzing the space complexity, observe that \( L', G' \) and the values \( d_N(\pi_{du}, (u, v), l_i) \) and \( d_{L'}(v, t, l_j) \) can be removed from the memory after step 8 since they are not needed any more. Thus, we only need to store the lists \( V_R \) between the recursive calls. Consider a path in the recursion tree, and for each recursive call \( i \), let \( u^i \) be the vertex \( u \) and \( V_R^i \) be the list \( V_R \) of the \( i \)-th recursive call. Since \( V_R^i \) contains only vertices adjacent to \( u^i \) and \( u^i \) is never being considered again in any succeeding recursive call \( j > i \), we have

\[
\sum_i |V_R^i| \leq |A_{S, L}| \leq |N|,
\]

where \( A_{S, L} \) is the number of edges in \( N \). This completes the proof of the space complexity being bounded by \( O(\|N\|) \).

3.7 Conclusion

We proposed a simple algorithm that systematically lists the set of feasible \( dt \)-routes with at most \( \beta \) lines, and also gave some first ideas how the generation of irrelevant routes can be avoided. Since the running time of this algorithm might be exponential in \( \beta \), we showed how the techniques proposed by Rizzi et al. [56] can be used to bound the running time by a polynomial in the sum of the input and the output size. Still, our algorithm generates some routes that are feasible but undesirable in practice (such as routes going back and forth), especially if \( \beta \) is too large. It remains an interesting open problem to study how the generation of these undesirable routes can be avoided.

Moreover, we argued that many routes might correspond to the same path in the transportation network, and might therefore be considered equivalent from a traveller’s perspective. Therefore we studied how to compute the set of all paths in the transportation network for which a suitable route of length at most \( \beta \) exists. To make the algorithm usable in practical situations, it would certainly be worthwhile to further study whether heuristical speedup techniques (such as pruning based on the geography of the stops) helps to decrease the
running time, and how such techniques influence the quality of the solutions.
Listing all Solutions
Measuring the Robustness of Routes

In this chapter we discuss how the robustness of routes can be measured, and how a set of robust routes can be computed. For this purpose we assume that the set of stops \( S \) and the set of lines \( L \) are given, and that the set of all length-bounded dt-routes \( R = R_{\beta}^d \) has already been computed. As a first result we show, given a dt-route \( r \) and an initial departure time \( t_D \), how to compute a dt-journey along \( r \) that leaves \( d \) at time \( t_D \) or later, and that reaches \( t \) as early as possible. We also provide a simple algorithm that uses the aforementioned result to compute a latest departing journey when a target arrival time \( t_A \) instead of a departure time is given. Afterwards we show how a similarity-based approach by Buhmann et al. [18] can be adapted to find robust routes, and also discuss some generalizations of this method.

4.1 Related Work

Extensions to Existing Methods

Disser et al. [29] studied the computation of Pareto-optimal journeys in a time-dependent graph, and introduced the “reliability” of a journey as an additional optimization criterion. The reliability of a transfer is a function mapping the time between the arrival and the departure of the succeeding vehicle to a number between 0 and 1, and the reliability of a journey is the product of the reliabilities of all transfers. It can be interpreted as a probability that the journey can be used as planned. The same measure was later used in the earlier-mentioned RAPTOR algorithm [24] (see also Chapter 2 for a brief explanation of the algorithm).

RAPTOR was later again modified to support the computation of robust journeys [22]. Instead of evaluating the reliability of each transfer using the aforementioned formula, one stores the earliest time at which a stop can be reached if the previous transfer failed. This quantity is then used as an additional criterion to be minimized. Their experimental evaluation reveals that this modification indeed computes more robust journeys than the much simpler reliability model.
To investigate the robustness of transfer patterns, Bast et al. [4] performed a series of experiments in which trips were randomly delayed. Transfer patterns were precomputed on the network without delays, and queries were answered using these transfer patterns (but with updated travel times in the query graph). Then the same queries were performed on a time-expanded graph in which the delays were incorporated, and the results were compared. They observed that, even for a huge number of delayed trips, there is nearly always a transfer pattern that includes the optimal path, and it rarely happens that the algorithm proposes a path that is much worse than the optimum.

**Online Information** When a delay occurs, a real-time traffic information system has to update the arrival and the departure times of the corresponding trips. Especially, some transfers between different trips may become infeasible. Updating the time-dependent and especially the time-expanded graph is discussed in [19, 21]. In reality, however, it hardly suffices to update only the delayed trip, because often trains wait for each other, causing additional delays. Instead of updating each trip manually and identifying all trips that wait for this trip, Müller-Hannemann and Schnee [47] use an additional graph, the so-called dependency graph, to model these delays. The dependency graph contains a vertex for every event (i.e., for every arrival and every departure of a concrete train at a concrete station), and additionally so-called forecast and schedule vertices. Now directed arcs model dependencies between actions. More concretely, every arc $(v, w)$ is directed towards an event $w$, and every arc stores a number indicating a lower bound on the time at which $w$ takes place. Every time when an event takes place or a delay happens, the dependency graph is updated. However, queries are not answered by the dependency graph, but using a regular time-expanded graph that is regularly updated using the information maintained by the dependency graph.

In an experimental case study on the public transportation network of Rome, Allulli et al. [1] observed that using real-time information indeed does not only give better travel time estimates, but also helps to improve the quality of the journeys itself.

**Recoverable Robustness** When delays occur and some trip is missed at some stop $s$, one could simply compute a new journey from $s$. However, keeping in mind that delays can always emerge it might be better to compute already in advance journeys with backup options. This problem, the computation of recoverable robust journeys, was studied by Goerigk et al. [33]. They model the network as an event-activity network which basically is a time-expanded graph where transfers are modeled differently: from every arrival event there is an arc to every departure event of every later train at the same station (instead of just one transfer arc to the succeeding (w.r.t. time) vertex). Uncertainty is modeled
using a set of delay scenarios, where in every scenario a delay may happen on every arc that is not a transfer arc, hence some transfer arcs may no longer be feasible (if the delay exceeds the maximum waiting time of the connecting succeeding vehicle). Initially, no delays occur and future delays are unknown, but there exists a point in time at which all delays are immediately disclosed to the traveller. Given a finite set of delay scenarios, the goal is to compute a path such that for every delay scenario \( d \), the corresponding event-activity network (reflecting \( d \)) contains a path to the destination. The authors propose a bicriteria label-setting algorithm to compute paths that minimize the travel time in case of no delay, and the maximal travel time of the overall journey (i.e., of the path and the corresponding best recovery path) that may occur in any scenario.

**Strict and Light Robustness**  
Goerigk et al. [34] also applied two other methods from robust optimization to compute robust journeys. Given an event-activity network as defined in the previous paragraph and an infinite set of delay scenarios, a natural problem is the computation of a strictly robust path that is feasible in any scenario. This problem can be reduced to the problem of computing all transfer arcs which are maintained in any delay scenario. The authors proved that this problem is \( \text{NP} \)-hard. However, using a stronger condition on transfer arcs, at least there exists a polynomial-time algorithm to compute a subset of these arcs. In an experimental evaluation of this algorithm, the authors observed that strict robustness leads to a large increase of travel time. Hence they propose to compute, for a given \( B \), light robust paths whose travel times do not exceed the sum of the length of a fastest path and \( B \), and that minimize the number of non-robust transfer arcs. Their experiments indicate that these paths are indeed better suited for real-life applications.

**Stochastic Methods**  
A common method to model uncertainty is to assume that vehicle arrivals and travel times are random variables, whose distribution is either assumed to be known explicitly [15, 51], or is estimated using historic delay data [14, 37]. An often formulated goal is the computation of one or more journeys that minimize the expected arrival time at the target stop.

For example, Boyan and Mitzenmacher [15] assumed that the distribution of the bus arrival times has increasing failure rate, and showed that an optimal travel plan has the form “take [bus] \( B_1 \) whenever it arrives; take [bus] \( B_2 \) if it arrives before time \( t^*_2 \); take [bus] \( B_3 \) if it arrives before time \( t^*_3 \)” where the \( t^*_i \) are monotonically decreasing. They also showed that such a travel plan can be computed with a dynamic programming approach. Hickman [37] suggests to estimate the distributions of the arrival and travel times from historic data, and to generate statistically non-dominated paths. Botea et al. [14] proposed a strategy for computing a tree which encodes various travel possibilities. Nodes
correspond to places, edges correspond to actions that can either succeed (such as taking a certain bus) or fail. Both outcomes lead to subsequent travel possibilities. They also propose rules to avoid the generation of unreasonable nodes: for example, one does not have to visit earlier nodes again.

Dibbelt et al. [26, 27] assumed that every elementary connection (i.e., every trip segment between two consecutive stops) departs on time, but may arrive too late. The arrival time is drawn from some specified distribution. Now they extended the Connection Scan Algorithm (see Chapter 2) to compute a journey with minimum expected arrival time. The algorithm is also augmented to compute a decision graph in which vertices represent stops and edges represent travel possibilities between these stops. Hence, this graph encodes various journeys with backup options.

4.2 Computing Journeys from Routes

Motivation The previous chapter already described how to compute the set \( R \) of all routes between a departure stop \( d \) and a target stop \( t \) that have at most \( \beta - 1 \) transfers. In this chapter, we will provide methods that select robust routes, and for every selected route \( r \) compute a suitable departure time \( t_D \) such that leaving the departure stop \( d \) at time \( t_D \) and following \( r \) likely leads to an on-time arrival at the target stop \( t \).

However, as stated in the introduction and in Chapter 2, our goal is to recommend journeys (i.e., routes together with appropriate transfer stops and an initial departure time) to users, and not just routes with an initial departure time. Given a \( dt \)-route \( r \in R \), a departure time \( t_D \) and a timetable \( T \), in this section we develop an algorithm that computes a \( dt \)-journey along \( r \) that leaves \( d \) at time \( t_D \) and reaches \( t \) as early as possible. Hence, for computing such a journey, we only need to identify appropriate transfer stops \( \theta_1, \ldots, \theta_{|r|-1} \).

Notice that, once such an algorithm is established, for computing robust journeys it is sufficient to compute a robust route \( r \) and a suitable departure time \( t_D \). We can then search for a scheduled journey in the planned timetable along \( r \) that leaves \( d \) no later than at time \( t_D \), and recommend this to the user.

Computing an Earliest Arriving Journey: Idea In the following considerations, we assume the underlying timetable \( T \) (either the planned or a recorded timetable) to be fixed, and that the arrival and departure times relate to \( T \). Given \( d, t \in S \), an initial departure time \( t_D \in \mathbb{N} \), and a route \( r = \langle l_1, \ldots, l_k \rangle \in R \), we can compute a journey along \( r \) that arrives as early as possible as follows. We start at \( d \) at time \( t_D \) and take the first trip of line \( l_1 \) that arrives. Then we compute an appropriate transfer stop \( \theta \in l_1 \cap l_2 \) and the arrival time \( t_1 \) of \( l_1 \) at \( \theta \), leave \( l_1 \) there and compute recursively the earliest arrival time
Figure 16 A situation with the set of stops \( S = \{d, t, s_1, \ldots, s_9\} \) and four lines \( l_1 = (d, s_1, s_3, s_4, s_6, s_8) \), \( l_2 = (s_1, s_2, s_4, s_7, s_8) \), \( l_3 = (s_2, s_5, s_7, s_9) \) and \( l_4 = (s_5, s_9, t) \). The only \( dt \)-route is \( r = (l_1, l_2, l_3, l_4) \). We can transfer from \( l_1 \) to \( l_2 \) at \( s_1 \), at \( s_4 \) and at \( s_8 \). However, \( s_8 \) is no appropriate transfer when traveling from \( d \) to \( t \), because \( l_3 \) cannot be reached any more. Also, choosing \( s_4 \) as a transfer fixes the next transfers \( s_7 \) and \( s_9 \) because otherwise \( t \) is not reachable.

when departing from \( \theta \) at time \( t_1 + \varepsilon(\theta, l_1, l_2) \) or later, following the route \( \langle l_2, \ldots, l_k \rangle \). Remember that \( \varepsilon(\theta, l_1, l_2) \) denotes the time to transfer from \( l_1 \) to \( l_2 \) at the stop \( \theta \). The only non-trivial part of this algorithm is the selection of an appropriate transfer stop \( \theta \), mainly due to three reasons:

1) The lines \( l_1 \) and \( l_2 \) may operate with different speeds (e.g., because \( l_1 \) is a fast tram while \( l_2 \) is a slow bus), or \( l_1 \) and \( l_2 \) separate at a stop \( s_1 \) and join later again at a stop \( s_2 \) but the overall travel times of \( l_1 \) and \( l_2 \) differ between \( s_1 \) and \( s_2 \). Depending on the situation, it may be better to leave \( l_1 \) as soon or as late as possible, or anywhere in-between.

2) The lines \( l_1 \) and \( l_2 \) may separate at a stop \( s_1 \) and join later again at a stop \( s_2 \). If all transfer stops in \( l_2 \cap l_3 \) are served by \( l_2 \) before \( s_2 \), then leaving \( l_1 \) at \( s_2 \) is not an option since \( l_3 \) is not reachable anymore. See Figure 16 for a visualization.

3) The value \( \varepsilon(\theta, l_1, l_2) \) may differ for different \( T_i \).

The idea now is to find the earliest trip of line \( l_1 \) that departs from \( d \) at time \( t_D \) or later, iterate over all stops \( s \in l_1 \cap l_2 \), and compute recursively the earliest arrival time when continuing the journey from \( s \) having a changing time of at least \( \varepsilon(s, l_1, l_2) \). Finally, we return the smallest arrival time that was found in one of the recursive calls.

Issues and improvement of the recursive algorithm An issue with this naïve implementation is the running time, which might be exponential in \( |r| \) in the worst-case (if \( |l_i \cap l_{i+1}| > 1 \) for \( \Omega(|r|) \) many \( i \in \{1, \ldots, |r| - 1\} \)). However, remember our assumption that timetables respect the FIFO property: vehicles
serving the same line to not overtake each other. For a moment we also ignore
the fact that vehicles may leave out certain stops. Now, for every stop $s$ on a
line $l_i$, we can compute the earliest time $A_i(s)$ at which $s$ can be reached via the
route $\langle l_1, \ldots, l_i \rangle$ when $d$ is left at time $t_D$ or later. If $s$ cannot be reached via
$\langle l_1, \ldots, l_i \rangle$ (e.g., because $l_i$ visits $s$ before visiting any transfer stop from $l_{i-1}$),
we set $A_i(s) = \infty$. These times $A_i(s)$ can now be computed by considering the
stops on $l_i$ in increasing order (i.e., from the beginning to the end), and keeping
track of the earliest trip $\tau_i$ of $l_i$ that can somehow be reached via $\langle l_1, \ldots, l_i \rangle$.
For every stop $s$ that $l_i$ visits, we set $A_i(s) \leftarrow A(\tau_i, s)$. Moreover, every time
that we encounter a stop $s$ that also $l_{i-1}$ visits, we check whether an earlier
trip can be reached. This is the case if and only if there exists a trip $\tau'$ that
leaves $s$ at time $A_{i-1}(s) + \varepsilon(s, l_{i-1}, l_i)$ or later, and that leaves $s$ earlier than $\tau_i$
does. A somewhat similar idea was used in RAPTOR [24]; there, however, all
routes are taken into consideration while we are only interested in the arrival
time of one concrete route.

Observe that $A_i(s)$ need to be computed only for the stops $s \in l_i \cap l_{i+1}$. In
the following algorithmic formulation of the aforementioned method we define $A(\text{null}, s)$ to be $\infty$. To make the formulation more compact, we introduce two
virtual lines $l_0 = \langle d \rangle$ and $l_{k+1} = \langle t \rangle$ that only visit the departure and the target
stop, respectively.

**Algorithm 5: EarliestArrival($T, d, t, t_D, \langle l_1, \ldots, l_k \rangle$)**

1. $l_0 \leftarrow \langle d \rangle$; $l_{k+1} \leftarrow \langle t \rangle$; $A_0(d) \leftarrow t_D$; $\varepsilon(d, l_0, l_1) \leftarrow 0$
2. for $i \leftarrow 1, \ldots, k$ do
   3. \quad $\tau_i \leftarrow \text{null}$
   4. \quad for each stop $s$ on $l_i$ in increasing order do
      5. \quad \quad if $l_{i+1}$ visits $s$ then $A_i(s) \leftarrow A(\tau_i, s)$;
      6. \quad \quad if $l_{i-1}$ visits $s$ then
         7. \quad \quad \quad $T_i \leftarrow \{ \tau \in L^{-1}(l_i) \mid A_{i-1}(s) + \varepsilon(s, l_{i-1}, l_i) \leq D(\tau, s) < \infty \}$
         8. \quad \quad \quad $\tau' \leftarrow \arg \min_{\tau \in T_i} D(\tau, s)$
         9. \quad \quad \quad if $A(\tau', s) < A(\tau_i, s)$ then $\tau_i \leftarrow \tau'$;
3. return $A_k(t)$

**Computing appropriate transfer stops** Algorithm 5 just computes the ear-
liest possible arrival time, not a journey itself. The above algorithm can eas-
ily be extended to also compute the corresponding transfer stops: every time
that an earlier trip is found in step 9, we update $\tau_i$ and also store the stop $s_i = s$ by which $\tau_i$ can be reached first. Every time that some value $A_i(s)$
is computed, we also store the stop $\vartheta_j(s) = s_i$ by which the current trip was reached. Observe that if $\vartheta_i(s)$ exists, then it is contained in $l_{i-1} \cap l_i$. Hence, we set $s \leftarrow t$ and use $\theta_{k-1} := \vartheta_k(s)$ as the transfer from $l_{k-1}$ to $l_k$, set $s \leftarrow \vartheta_k(s)$ and use $\theta_{k-2} := \vartheta_{k-1}(s)$ as the transfer from $l_{k-2}$ to $l_{k-1}$, set $s \leftarrow \vartheta_{k-1}(s)$ and so on, until we computed the transfer $\theta_1$ from $l_1$ to $l_2$. Now, $(t_D, r, \theta_1, \ldots, \theta_{k-1})$ really is a journey: Since $A_0(d) = t_D$ and $\varepsilon(d, l_0, l_1) = 0$, the set $T_{l_i}$ contains only trips that leave $d$ at time $A_0(d) + \varepsilon(d, l_0, l_1) = t_D$ or later. For every $i \in \{1, \ldots, k\}$, let $\tau_i$ be the trip for which $s_i$ was set to $\theta_{i-1}$. Since $\theta_{i-1} = \vartheta_i(\theta_i)$ and the only stops for which $\vartheta_i(s)$ can be set to $s_i = \theta_{i-1}$ are those that $l_i$ visits after $s_i = \theta_{i-1}$, it follows that $l_i$ visits $\theta_{i-1}$ before $\theta_i$. Furthermore, an inductive argument over $i$ shows that $D(\tau_i, \theta_{i-1}) \geq A(\tau_{i-1}, \theta_{i-1}) + \varepsilon(\theta_{i-1}, l_{i-1}, l_i)$. For $i = 1$ we already showed that $D(\tau_1, d) \geq t_D$. Assume that $D(\tau_{i-1}, \theta_{i-2}) \geq A(\tau_{i-2}, \theta_{i-2}) + \varepsilon(\theta_{i-2}, l_{i-2}, l_{i-1})$, and assume that $A_{i-1}(\theta_{i-1}) = A(\tau_{i-1}, \theta_{i-1})$. Since $\tau_i \in T_{l_i}$ and $\tau_i$ was first reached at $\theta_{i-1}$, we have

$$D(\tau_i, \theta_{i-1}) \geq A_{i-1}(\theta_{i-1}) + \varepsilon(\theta_{i-1}, l_{i-1}, l_i)$$

(10)

$$= A(\tau_{i-1}, \theta_{i-1}) + \varepsilon(\theta_{i-1}, l_{i-1}, l_i),$$

(11)

hence a transfer from $\tau_{i-1}$ to $\tau_i$ at the stop $\theta_i$ is feasible. Moreover, since $l_i$ visits $\theta_i$ only after $\theta_{i-1}$, we obtain $A_i(\theta_i) = A(\tau_i, \theta_i)$.

**Correctness and running time** Let $t_{OPT}$ be the earliest arrival time among all journeys along $r = \langle l_1, \ldots, l_k \rangle$ that leave $d$ at time $t_D$ or later. We already argued in the previous paragraph that Algorithm 5 returns a number $A_k(t) \geq t_{OPT}$, but we did not prove yet that it always returns an optimal solution.

**Theorem 7.** Let $t_{OPT}$ be the earliest arrival time among all journeys along $r$ that leave $d$ at time $t_D$ or later. If Algorithm 5 returns a number $A_k(t) < \infty$, then $A_k(t) = t_{OPT}$. The overall running time is bounded by $O(kn \log |T_{max}|)$ where $n = \max_{i \in \{1, \ldots, k\}} |l_i|$ and $|T_{max}|$ is the maximum number of trips that a timetable $T_s$ contains, for any stop $s$ that some line $l_i \in r$ visits.

**Proof.** For $i \in \{1, \ldots, k\}$ and every stop $s$ that $l_i$ visits, let $A_i'(s)$ be the earliest time when $s$ can be reached via the route $\langle l_1, \ldots, l_i \rangle$. We argued earlier that $A_i(s) \geq A_i'(s)$. Moreover, we saw that the set $T_{l_i}$ contains only trips that leave $d$ at time $t_D$ or later. Since we chose the earliest one $\tau_1$ and since trips respect the FIFO property, we have $A_1(s) = A(\tau_1, s) = A_1'(s)$ for every stop $s$ that $l_1$ visits after $d$, and $A_1(s) = \infty = A_1'(s)$ for every other stop (including $s = d$ which is correct because we assumed that a line cannot be boarded and left at the same time). Now, assume that the algorithm computed some value $A_i(s) > A_i'(s)$, and let $i$ be the smallest value for which an $s \in l_i$ with $A_i(s)$
exists (i.e., \(A_j(s) = A'_j(s)\) for every \(j \in \{1, \ldots, i - 1\}\) and every \(s \in l_j\)). Let \(\tau_i\) be the trip by which \(s\) was reached in the algorithm, and let \(\tau'_i\) be the trip by which \(s\) was reached in an optimal journey along \(\langle l_1, \ldots, l_i\rangle\). Since trips do not overtake each other and \(A_i(s) > A'_i(s)\), it follows that \(\tau'_i\) is operated earlier than \(\tau_i\). There exists a transfer stop \(\theta'_i\) that \(l_i\) visits before \(s\), and by which \(\tau'_i\) was first reached (in an optimal journey). This implies that \(D(\tau'_i, \theta'_{i-1}) \geq A_{i-1}(\theta'_{i-1}) + \varepsilon(\theta'_{i-1}, l_{i-1}, l_i)\), and hence \(T_{l_i}\) would contain \(\tau'_i\). Now, if \(\tau_i\) was reached first at \(\theta'_{i-1}\) or some earlier stop, then \(T_{l_i}\) would also contain \(\tau_i\), and as soon as \(\tau'_i \in T_{l_i}\), the algorithm would use it as the currently earliest trip (and hence would set \(A_i(s) \leftarrow A(\tau'_i, s) = A'_i(s)\)). On the other hand, if \(\tau_i\) was reached only somewhere between \(\theta_{i-1}\) and \(s\), then the earliest trip was already set to \(\tau'_i\), and the algorithm would not substitute \(\tau_i\) for it. Hence it would also set \(A_i(s) \leftarrow A(\tau'_i, s) = A'_i(s)\), which contradicts our assumption that \(A_i(s) > A'_i(s)\).

The running time analysis is straightforward: The outer loop is repeated \(k\) times, the inner loop at most \(n\) times, and step 7 can be performed in time \(O(\log |T_{max}|)\). Since every other step can be performed in constant time, an overall running time of \(O(kn \log |T_{max}|)\) immediately follows.

**Handling trips that skip stops** A drawback of Algorithm 5 is that it may return \(\infty\) if \(T\) contains some trips that leave out certain stops. To see that, consider, for example, a route \(\langle l_1, l_2, l_3\rangle\) where \(l_1\) and \(l_2\) have only one common stop \(s\), and \(l_2\) visits all stops in \(l_2 \cap l_3\) after \(s\). Now, if the first trip of \(l_2\) (that leaves \(s\) only after it was reached by an appropriate trip of \(l_1\)) turns around in advance (before any stop in \(l_2 \cap l_3\) was reached), the algorithm returns \(\infty\). On the other hand, one may argue that in this special situation such an answer is justified: our interpretation of a journey (Chapter 2) assumed that a user takes “the next arriving vehicle [...] immediately”. We did not specify what happens if the next transfer stop is not reached by this vehicle. Hence, we slightly extend our definition: for a \(dt\)-route \(r = \langle l_1, \ldots, l_k\rangle\) and suitable transfer stops \(\theta_0 = d, \theta_1, \ldots, \theta_{k-1}, \theta_k = t\), at each transfer \(\theta_i\) we take the earliest departing trip which respects the transfer times and visits both \(\theta_i\) and \(\theta_{i+1}\), i.e. the earliest trip \(\tau_{i+1}\) where \(D(\tau_{i+1}, \theta_i) \geq A(\tau_i, \theta_i) + \varepsilon(\theta_i, l_i, l_{i+1})\) and \(A(\tau_{i+1}, \theta_{i+1}) < \infty\).

Now we modify Algorithm 5 as follows. Again, we consider the stops on every line \(l_i\) from the beginning to the end. Now, we keep track of a set \(S\) containing all transfers from \(l_{i-1}\) that we have seen so far. For every stop \(s\) that both \(l_i\) and \(l_{i+1}\) visit, we compute the earliest time \(A_i(s)\) at which \(s\) can be reached via \(\langle l_1, \ldots, l_i, l_{i+1}\rangle\) (or, as before, set \(A_i(s) = \infty\) can \(s\) not be reached using this partial route). This can be done by computing the set \(T_{l_i}^s\) of all trips of line \(l_i\) that visit both some stop \(s' \in S\) as well as \(s\), and that leave \(s'\) at time \(A_{i-1}(s') + \varepsilon(s', l_{i-1}, l_i)\) or later. Afterwards we set \(A_i(s)\) to the earliest time
at which some trip in $T_i^s$ reaches $s$. Moreover, if we find a stop that both $l_{i-1}$ and $l_i$ visit, we add $s$ to $S$. Hence we obtain the following algorithm.

**Algorithm 6: EarliestArrival**($T, d, t, t_D, \{l_1, \ldots, l_k\}$)

1. $l_0 \leftarrow \langle d \rangle$; $l_{k+1} \leftarrow \langle t \rangle$; $A_0(d) \leftarrow t_D$; $\varepsilon(d, l_0, l_1) \leftarrow 0$
2. for $i \leftarrow 1, \ldots, k$ do
   3.     $S \leftarrow \emptyset$
   4.     for each stop $s$ on $l_i$ in increasing order do
   5.         if $l_{i+1}$ visits $s$ then
   6.             $T_i^s \leftarrow \{ \tau \in L^{-1}(l_i) \mid (A(\tau, s) < \infty) \land (\exists s' \in S : A_{i-1}(s') + \varepsilon(s', l_{i-1}, l_i) \leq D(\tau, s') < \infty) \}$
   7.             if $T_i^s \neq \emptyset$ then $A_i(s) \leftarrow \min_{\tau \in T_i^s} A(\tau, s)$;
   8.         else $A_i(s) \leftarrow \infty$;
   9.     if $l_{i-1}$ visits $s$ then $S \leftarrow S \cup \{s\}$;
10. return $A_k(t)$

For the correctness of Algorithm 6, observe that at every iteration of the loop in step 4, $S$ contains exactly the stops in $l_{i-1} \cap l_i$ that $l_i$ visits before $s$. As before, the correctness is proven by induction over $i$. For $i = 1$ we have $S = \{d\}$ for every stop $s$ that $l_1$ visits after $d$, and $S = \emptyset$ otherwise. Hence, for every $s$ that $l_1$ visits after $d$, $T_1^s$ contains all trips that visit $d$ (because $D(\tau, s') < \infty$), that depart at $d$ not before time $t_D$ (because $D(\tau, s') \geq A_{i-1}(s) + \varepsilon(s', l_{i-1}, l_i) = t_D$), and that visit $s$ (because $A(\tau, s) < \infty$). For every other $s$ on $l_1$, $T_1^s$ is empty. Since step 7 selects a trip $\tau$ that reaches $s$ as early as possible, for every $s$ that $l_1$ visits $A_1(s)$ indeed denotes the earliest arrival time at $s$.

Now, assume that for every $s'$ that $l_{i-1}$ visits, $A_{i-1}(s')$ denotes the earliest arrival time at $s'$ via $\langle l_1, \ldots, l_{i-1} \rangle$. Let $s$ be the current stop of the loop in step 4, and let $\tau$ be the trip which reaches $s$ as early as possible (via $\langle l_1, \ldots, l_i \rangle$). Certainly, $A(\tau, s) < \infty$ (because $\tau$ visits $s$), and there is a stop $s'$ that $l_i$ visits before $s$ for which $D(\tau, s') < \infty$ (because $\tau$ visits $s'$) and $D(\tau, s') \geq A_{i-1}(s') + \varepsilon(s', l_{i-1}, l_i)$ (because $\tau$ respects the transfer times). Hence, $T_i^s$ contains $\tau$, and no other trip in $T_i^s$ reaches $s$ earlier than $\tau$ does (because otherwise $\tau$ would not reach $s$ as early as possible). Hence, step 7 computes $A_i(s)$ correctly.

To analyze the running time, we have to investigate how the steps 6 and 7 can be implemented. Observe that, since timetables respect the FIFO property, $T_i^s$ does not have to contain all possible trips, but for every $s' \in S$ only the one that visits $s'$ as early as possible (and also respects the transfer times at $s'$,
Measuring the Robustness of Routes

and also visits s). This set can be computed by iterating over the appropriate trips of line \( l_i \) in \( T_s \) that visit s and at least one stop \( s' \in S \). We also have to keep track of those stops from \( S \) for which we already found an appropriate trip. Since the minimum computation takes time \( \mathcal{O}(|S|) \subseteq \mathcal{O}(n) \), we obtain an overall bound of \( \mathcal{O}(kn(|T_{max}| + n)) \) where, as before, \( k \) is the length of the route, \( n = \max_{i \in \{1,...,n\}} |l_i| \) is the maximal length of a line and \( |T_{max}| \) denotes the maximum number of trips that a timetable \( T_s \) contains (for any stop s that some line \( l_i \) visits). This bound is not tight: in practice trips only rarely skip stops, and it happens hardly ever that many succeeding trips also skip these stops. Hence usually we don’t have to consider all trips in \( T_s \), but only very few of them. Now, if we also keep a copy of \( T_s \) that is ordered by the arrival times (in Chapter 2 we assumed \( T_s \) to be ordered by the departure times) and use binary search to disregard all trips that leave s before time \( \min_{s \in l_{i-1} \cap l_i} A_{i-1}(s) \), then the real running time should in many cases be bounded by \( \mathcal{O}(kn(\log |T_{max}| + n)) \). Indeed, our experiments in Chapter 5 will show that Algorithm 6 performs sufficiently well on real-world data.

4.3 Transfer Buffers

A naïve strategy to increase the reliability of a journey is to enforce an additional buffer time at each transfer or at the end of the trip. The Buffer-\( \xi \) approach uses the planned timetables \( T_{planned} \) to compute a journey that is planned to leave \( d \) as late as possible, arrives at \( t \) not later than at time \( t_A \), and that has an additional time of at least \( \xi \) at each transfer of the journey. This especially implies that if a line \( l_i \) is planned to arrive at a transfer stop \( v_i \) at time \( t_i \), then the next line \( l_{i+1} \) of the journey can only be taken at time \( t_i + \xi \) or later. Buffer-0 corresponds to an optimal journey in the planned timetable, so we refer to it as Opt-TT.

Given two stops \( d,t \in S \) and a departure time \( t_0 \in \mathbb{N} \), we can already compute the earliest arrival of a journey from \( d \) to \( t \) starting at time \( t_D \). From now on, we aim to compute the latest departure time at \( d \) when the latest arrival time \( t_A \) at \( t \) is given. Instead of modifying one of the algorithms from the previous section to operate in the reverse direction, we propose an algorithm that sweeps backwards in time and uses the previous algorithm Earliest-Arrival as a subroutine. This sweepline algorithm will later be extended to count routes (instead of computing a single one), and can therefore be used to apply the approach by Buhmann et al. [18] for finding robust routes.

The sweepline algorithm works as follows. We consider the trips departing at stop \( d \) before time \( t_A \), sorted in reverse chronological order. Every time we find a trip \( \tau \) of any line departing at some time \( t_0 \), we check whether there
exists a route \( r = \langle L(\tau), l_2, \ldots, l_k \rangle \in \mathcal{R} \) that starts with the line \( L(\tau) \). If yes, then we use one of the two algorithms proposed in the previous section to compute the earliest arrival time at \( t \) when we depart at \( d \) at time \( t_0 \) and follow the route \( r \). If the time computed is not later than \( t_A \), we found a possible solution. However, the algorithm cannot stop yet because there might be another trip leaving at the same time that can be extended to a journey that reaches \( t \) even earlier. Hence, we continue until a trip that departs earlier is found. If there are multiple journeys with the same travel time that all leave \( d \) at the same time, then all of them are returned.

### 4.4 A Similarity-Based Approach

**Idea** We will now describe how to compute robust journeys using ideas of Buhmann et al. [18]. The approach behind is discussed in more detail and investigated further in Chapter 6.

Let \( d, t \in S \) be the departure and the target stop of the journey, \( t_A \) be the latest arrival time at \( t \), and \( \mathcal{T} \) be a set of recorded timetables for comparable time periods (e.g., daily recordings for the past Mondays). For a timetable \( T \in \mathcal{T} \) and a value \( \gamma \), the approximation set \( A_\gamma(T) \) contains a route \( r \in \mathcal{R} \) if and only if there exists a journey along the route \( r \) that starts at \( d \) at time \( t_A - \gamma \) or later and arrives at \( t \) at time \( t_A \) or earlier (both times refer to the timetable \( T \)). The major advantage of this definition over classical approximation definitions (such as multiplicative approximation) is that we can consider multiple recorded timetables at the same time, and that the parameter \( \gamma \) still has a direct interpretation as the time that we depart before \( t_A \). Especially, if we consider approximation sets \( A_\gamma(T_1), \ldots, A_\gamma(T_k) \) for \( T_1, \ldots, T_k \in \mathcal{T} \), every set contains only routes that appear in the same time period and are therefore comparable among different approximation sets.

To identify robust routes when only two timetables \( T_1, T_2 \in \mathcal{T} \) are given, we consider \( A_\gamma(T_1) \cap A_\gamma(T_2) \): the only chance to find a route that is likely to be good in the future is a route that was good in the past for both recorded timetables. Remember that the parameter \( \gamma \) measures the time difference to the latest allowed arrival time \( t_A \). It determines the size of the intersection: if \( \gamma \) is too small (e.g., smaller than the minimum travel time of any \( dt \)-journey in \( T_1 \) or \( T_2 \)), the intersection will be empty. If \( \gamma \) is too large, the intersection contains many (and maybe all) \( dt \)-routes, and not all of them will be a good choice. Assuming that we knew the optimal parameter \( \gamma_{\text{OPT}} \), we could pick a route from \( A_{\gamma_{\text{OPT}}}(T_1) \cap A_{\gamma_{\text{OPT}}}(T_2) \) at random. Buhmann et al. [18] suggest to
Measuring the Robustness of Routes

Figure 17 An example with five lines \{1, \ldots, 5\} and two routes \(r_1 = \langle 1, 2, 3 \rangle\) (solid) and \(r_2 = \langle 4, 5 \rangle\) (dotted). The \(x\)-axis illustrates the stops \(\{d, s_1, s_2, s_3, t\}\), whereas the \(y\)-axis denotes the time. If a trip leaves a stop \(s_d\) at time \(t_d\) and arrives at a stop \(s_a\) at time \(t_a\), it is indicated by a line segment from \((s_d, t_d)\) to \((s_a, t_a)\). We have \(\mu^T_\gamma(r_1) = 3\) and \(\mu^T_\gamma(r_2) = 1\).

In combinatorial optimization, every solution is unique. Here, however, for a given \(t_A, \gamma,\) and a route \(r\), there might be multiple journeys along \(r\) that depart at time \(t_A - \gamma\) or later, and that arrive at time \(t_A\) or earlier. Our definition of an approximation should take this fact into account: a route \(r\) should be present more often if more realizing journeys in the corresponding time period \([t_A - \gamma, t_A]\) exist. Therefore we change the definition of \(A_\gamma(T)\) to a multiset of routes, and \(A_\gamma(T)\) contains a route \(r\) as often as it is realized by a journey starting at time \(t_A - \gamma\) or later, and arriving at time \(t_A\) or earlier. Figure 17 shows an example with five lines \(\{1, \ldots, 5\}\) and two routes \(r_1 = \langle 1, 2, 3 \rangle\) and \(r_2 = \langle 4, 5 \rangle\). We have \(\mu^T_\gamma(r_1) = 3\): taking the second 1 and the second 2 (from above) as well as taking the third 1 and the second 2 are counted as different journeys since the departure times at \(d\) differ. On the other hand, according to our definition of journey we always take the first arriving vehicle of any line, hence taking the first 1 and waiting for the second 2 is not counted.

Now the approximation set \(A_\gamma(T)\) can be represented by its multiplicity function \(\mu^T_\gamma : \mathcal{R} \rightarrow \mathbb{N}_0\), where for a route \(r \in \mathcal{R}\), \(\mu^T_\gamma(r)\) denotes the number of journeys starting at time \(t_A - \gamma\) or later, arriving at time \(t_A\) or earlier and following the route \(r\). Thus, we have \(|A_\gamma(T)| = \sum_{r \in \mathcal{R}} \mu^T_\gamma(r)\), and for two
4.4 A Similarity-Based Approach

recorded timetables $T_1, T_2$, we need to compute

$$
\gamma_{\text{OPT}} = \arg \max_{\gamma} \frac{\sum_{r \in \mathcal{R}} \min(\mu^{T_1}_\gamma(r), \mu^{T_2}_\gamma(r))}{\left(\sum_{r \in \mathcal{R}} \mu^{T_1}_\gamma(r)\right) \cdot \left(\sum_{r \in \mathcal{R}} \mu^{T_2}_\gamma(r)\right)}.
$$

(13)

Notice that the numerator simply denotes the cardinality of the intersection of the two approximation sets; the minimum operator has to be used because the approximation sets are multisets.

**Choosing from the intersection** After computing the value $\gamma_{\text{OPT}}$, we select a route $r$ from the intersection $A_{\gamma_{\text{OPT}}} (T_1) \cap A_{\gamma_{\text{OPT}}} (T_2)$ of the corresponding approximation sets. Buhmann et al. [18] suggest to simply pick a solution uniformly at random (ignoring the multiplicities of the routes). However, since we modified the definition of an approximation set to take multiplicities into account, it seems more appropriate to select a route $r$ according to the probability distribution defined by

$$
p_r := \frac{\min(\mu^{T_1}_{\gamma_{\text{OPT}}}(r), \mu^{T_2}_{\gamma_{\text{OPT}}}(r))}{\sum_{r \in \mathcal{R}} \min(\mu^{T_1}_{\gamma_{\text{OPT}}}(r), \mu^{T_2}_{\gamma_{\text{OPT}}}(r))}.
$$

(14)

From a practical perspective, routes with a larger number of realizations in the intersection seem to be more robust than those with few realizations, because these routes have “backup” trips if some transfer fails. Hence, we may also return the set of all routes $r$ that maximize $\min(\mu^{T_1}_{\gamma_{\text{OPT}}}(r), \mu^{T_2}_{\gamma_{\text{OPT}}}(r))$.

However, an experimental comparison between sampling uniformly at random and choosing the most realized route showed no significant difference in terms of arrival rate. We only compared these two “extreme” sampling strategies, because sampling routes according to (14) can be understood as a combination of these two methods. Hence, from now on we consider only the variant that chooses the most realized route, and we suggest to depart no later than at time $t_A - \gamma_{\text{OPT}}$. We refer to this method as **Similarity**.

**Computing the similarity** For $i \in \{1, 2\}$, we represent the function $\mu^{T_i}_\gamma$ by an $|\mathcal{R}|$-dimensional vector $\mu_i$ such that $\mu_i[r] = \mu^{T_i}_\gamma(r)$ for every $r \in \mathcal{R}$. We can compute the value $\gamma_{\text{OPT}}$ by a simple extension of the aforementioned sweepline algorithm. The modified algorithm again starts at time $t_A$, and considers all trips in $T_1$ and $T_2$ in reverse chronological order. The sweepline stops at every time when one or more trips in $T_1$ or in $T_2$ depart. Assume that the sweepline stops at time $t_A - \gamma$, and assume that it stopped at time $t_A - \gamma' > t_A - \gamma$ in the previous step. Of course, we have $\mu^{T_i}_\gamma(r) \geq \mu^{T_i}_{\gamma'}(r)$ for every $r \in \mathcal{R}$ and $i \in \{1, 2\}$. Let $\tau_1, \ldots, \tau_k$ be the trips that depart in $T_1$ or $T_2$ at time $t_A - \gamma$. The idea is to compute the values of $\mu_i$ (representing $\mu^{T_i}_\gamma$) from the values computed
in the previous step (representing $\mu_i^{T_j}$). This can be done as follows: for every trip $\tau_j$ occurring in $T_i$ and departing at time $t_A - \gamma$, we check whether there exists a route $r \in R$ starting with $L(\tau_j)$. If yes, we distinguish two cases:

1) If $\mu_i[r] = 0$, then $\mu_i^{T_j}(r) = 0$, thus $r \notin A_{\gamma'}(T_i)$. If there exists a journey from $d$ to $t$ along $r$ departing at time $t_A - \gamma$ or later, and arriving at time $t_A$ or earlier, then $A_{\gamma}(T_i)$ contains $r$ exactly once. Thus, if $\text{EARLIEST-ARRIVAL}(T_i, d, t, t_A - \gamma, r) \leq t_A$, we set $\mu_i[r] \leftarrow 1$.

2) If $\mu_i[r] > 0$, then $\mu_i^{T_j}(r) > 0$, thus $A_{\gamma'}(T_i)$ contains $r$ at least once. Thus, there exists a journey from $d$ to $t$ along $r$ departing at time $t_A - \gamma'$ or later, and arriving at time $t_A$ or earlier. Since $\tau_i$ is the only possibility to depart at $d$ between time $t_A - \gamma$ and $t_A - \gamma'$, $\tau_i$ is the first trip on a journey we never found before. Therefore it is sufficient to simply increase $\mu_i[r]$ by 1.

Up to now, we did not define when the algorithm terminates. In our implementation we decided to stop once $\gamma$ exceeds a certain value $\gamma_{\text{MAX}}$. In our experimental evaluation, we simply set $\gamma_{\text{MAX}}$ to be three hours before $t_A$, because there are no journeys with a larger travel time. Algorithm 7 shows the details.

Properties of the Approach Consider the situation where the best $dt$-journey $j$ in $T_1$ is identical to the best $dt$-journey in $T_2$. Assuming that $T_1$ and $T_2$ contain typical delays, common sense dictates to use the very same journey $j$ also in the future. This is exactly what this approach does as well. Observe that $S_{\gamma}(T_1, T_2) \leq |R|$. Let $r$ be the route along which $j$ travels. In our case, setting $\gamma$ so that $A_{\gamma}(T_1) = A_{\gamma}(T_2) = \{r\}$, we get that $S_{\gamma} = \frac{|R||A_{\gamma}(T_1)\cap A_{\gamma}(T_2)|}{|A_{\gamma}(T_1)||A_{\gamma}(T_2)|} = |R|$, and thus our approach computes the very same $\gamma$ and returns $r$. These considerations can be generalized to the cases where, e.g., $A_{\gamma}(T_1) = \{r\}, r \in A_{\gamma}(T_2)$, in which again $r$ will be computed.

If only a reliable journey is required, and the travel time is not an issue, then suggesting to depart few days before $t_A$ is certainly sufficient. However, our approach does not do this, but instead reasonably balances the two goals robustness and travel time. We consider the symmetric situation where both $|A_{\gamma}(T_1)|$ and $|A_{\gamma}(T_2)|$ grow with $\gamma$ in the same way, i.e., for every $\gamma$, $|A_{\gamma}(T_1)| = |A_{\gamma}(T_2)|$. Let us only consider discrete values of $\gamma$, and let $\gamma_1$ be the largest $\gamma$ for which $A_{\gamma_1}(T_1) \cap A_{\gamma_1}(T_2) = \emptyset$. Let $x = |A_{\gamma_1}(T_1)|$. Then, for every $\gamma > \gamma_1$, $S_{\gamma}(T_1, T_2) = |R| \cdot \frac{\Delta_{\gamma}}{(x + \Delta_{\gamma})^2}$ for some values of $\Delta_{\gamma}$. Simple calculation shows that $S_{\gamma}$ is maximized for $\Delta_{\gamma} = x$. We can interpret $x$ as the number of failed routes (that would otherwise make it if no delays appeared). Then, $S_{\gamma}(T_1, T_2)$ is maximized at the point that allows for another $\Delta_{\gamma} = x$ routes to join the
approximation sets $A_\gamma(T_i)$. Thus, the more disturbed the timetables are, the more “backward” in time we need to scan to obtain a robust route.

### 4.5 Function-Based Approaches

#### General Idea

Let $T_i \in \mathcal{T}$ be a recorded timetable, $r = \langle l_1, \ldots, l_\alpha \rangle \in \mathcal{R}$ be a route, $\tau_1, \ldots, \tau_k$ be the trips of line $l_1$ in $T_i$ and $D(\tau_j, d)$ be the departure time of the trip $\tau_j$ at $d$. We define

$$
\delta^r_i := \min_{j \in \{1, \ldots, k\}} \left\{ t_A - D(\tau_j, s) \mid \begin{array}{l}
\tau_j \text{ can be extended to a journey along } r \text{ that arrives in } T_i \text{ at stop } t \\
\text{at time } t_A \text{ or earlier}
\end{array} \right\}, \quad (15)
$$

which intuitively can be interpreted as follows: to arrive on time using route $r$ on the day at which $T_i$ is realized, one has to leave $s$ at least $\delta^r_i$ units of time before the latest allowed arrival time $t_A$. For a given penalty function $f : (\mathbb{R}^+)^{\left|\mathcal{T}\right|} \rightarrow \mathbb{R}$, we search for a route $r \in \mathcal{R}$ that minimizes $f(\delta^r_1, \ldots, \delta^r_{|T_i|})$.

---

**Algorithm 7: Compute-Maximum-Similarity($T_1, T_2, d, t, t_A$)**

1. $S^{\text{OPT}} \leftarrow 0$; $\mu_1 \leftarrow \mu_2 \leftarrow \mu_1^{\text{OPT}} \leftarrow \mu_2^{\text{OPT}} \leftarrow 0$; $\gamma^{\text{OPT}} \leftarrow 0$

2. for each stopping time of the sweepline do
   
3. \quad $\gamma \leftarrow$ Time difference of the sweepline to $t_A$

4. \quad $\{\tau_1, \ldots, \tau_n\} \leftarrow$ Trips departing at $d$ at time $t_A - \gamma$ in $T_1$ or $T_2$

5. \quad for $i \leftarrow 1, \ldots, n$ do

6. \quad \quad for each $r = \langle L(\tau_1), l_2, \ldots, l_k \rangle \in \mathcal{R}$ do

7. \quad \quad \quad if $\tau_i \text{ occurs in } T_1 \text{ then}$

8. \quad \quad \quad \quad if $\mu_1[r] > 0$ then $\mu_1[r] \leftarrow \mu_1[r] + 1$;

9. \quad \quad \quad \quad else if $\text{EarliestArrival}(T_1, d, t, t_A - \gamma, r) \leq t_A$ then

10. \quad \quad \quad \quad \quad $\mu_1[r] \leftarrow 1$

11. \quad \quad \quad else

12. \quad \quad \quad \quad if $\mu_2[r] > 0$ then $\mu_2[r] \leftarrow \mu_2[r] + 1$;

13. \quad \quad \quad \quad else if $\text{EarliestArrival}(T_2, d, t, t_A - \gamma, r) \leq t_A$ then

14. \quad \quad \quad \quad \quad $\mu_2[r] \leftarrow 1$

15. $S_\gamma \leftarrow (\sum_{r \in \mathcal{R}} \min(\mu_1[r], \mu_2[r])) / (\sum_{r \in \mathcal{R}} \mu_1[r] \cdot (\sum_{r \in \mathcal{R}} \mu_2[r]))$

16. if $S_\gamma > S^{\text{OPT}}$ then

17. \quad $\gamma^{\text{OPT}} \leftarrow \gamma$; $S^{\text{OPT}} \leftarrow S_\gamma$; $\mu_1^{\text{OPT}} \leftarrow \mu_1$; $\mu_2^{\text{OPT}} \leftarrow \mu_2$
In the following, we describe some possible choices for $f$, and for simplicity we abbreviate $f(\delta_1, \ldots, \delta_{|T|})$ by $f(r)$.

**Norm-Based Estimators** For a number $p \in [1, \infty]$, the Norm-$p$ estimator has the objective function

$$f^p_{\| \cdot \|}(r) = \left\| \delta_1, \ldots, \delta_{|T|} \right\|_p.$$  

It is easy to see that $f^1_{\| \cdot \|}$ selects all routes which in average (w.r.t. the recorded timetables in $T$) depart as late as possible. Moreover, $f^\infty_{\| \cdot \|}$ selects all routes minimizing the maximum time between the departure and the latest allowed arrival time $t_A$. Such routes can alternatively be seen as routes maximizing the earliest departure time necessary to arrive on time in all timetables in $T$. Thus, the Norm-$\infty$ estimator is related to the similarity-based approach from the previous paragraph in the following way. Let $\gamma_{FI} = \min \{ \gamma > 0 \mid \bigcap_{i=1}^{|T|} A_\gamma(T_i) \neq \emptyset \}$ be the smallest value for $\gamma$ such that the intersection of all $\gamma$-approximation sets is non-empty. One can observe that every route $r$ contained in $\bigcap_{i=1}^{|T|} A_{\gamma_{FI}}(T_i)$ minimizes $f^\infty_{\| \cdot \|}$ and vice versa. We note that these methods relate to strict robustness [34], but are based on a different solution concept, and learn from past observations given as daily recorded timetables (instead of specifying a set of possible delays).

Now, let $p \in [1, \infty]$ be arbitrary and let $r^p_j$ be a route minimizing $f^p_{\| \cdot \|}$. To determine how much in advance one has to depart when using $r^p_j$, we use our previous observations. For $p = 1$, it is reasonable to set $\gamma^p_j = f^1_{\| \cdot \|}(r^p_j)/|T|$ since $f^1_{\| \cdot \|}$ corresponds to averaging the departure times. For $p = \infty$, it is reasonable to set $\gamma^p_j = f^\infty_{\| \cdot \|}(r^p_j)$ since $f^\infty_{\| \cdot \|}$ considers the smallest time to leave in advance such that some route is contained in all approximation sets. For every other $p \in (1, \infty)$, we simply scale the time linearly with respect to $p = 1$ and $p = \infty$. More concretely, we set

$$\gamma^p_j = f^\infty_{\| \cdot \|}(r^p_j) - \left( f^p_{\| \cdot \|}(r^p_j) - f^\infty_{\| \cdot \|}(r^p_j) \right) \cdot \left( f^\infty_{\| \cdot \|}(r^p_j) - f^1_{\| \cdot \|}(r^p_j)/|T| \right).$$  

**A Mean-Risk Model** A different function-based estimator comes from the mean-risk model which was recently used for finding robust routes in private transportation [43]. Let $c \in \mathbb{R}^+_0$ be the risk-aversion coefficient, where $c = 0$ corresponds to the situation where the risk is being completely ignored. The objective function associated with the Mean-Risk-$c$ estimator is

$$f^p_{\text{MR}}(r) = \text{Mean}(\delta_1, \ldots, \delta_{|T|}) + c \cdot \sqrt{\text{Variance}(\delta_1, \ldots, \delta_{|T|})}.$$  


4.6 Conclusion

For a route $r_j$ minimizing $f_{\text{MR}}^c$, we simply set $\gamma_j = f_{\text{MR}}^c(r_j)$ as the time one has to depart in advance. Notice that Mean-Risk-0 is equivalent to Norm-1.

**Computational Aspects** The simplest way to find a route that minimizes (16) or (18) is to compute the values $\delta^r_i$ for every route $r \in \mathcal{R}$ and every recorded timetable $T_i \in \mathcal{T}$ using the sweepline algorithm described in Section 4.3 (where the input for this algorithm consists in $T_i$ instead of $T_{\text{planned}}$, and the additional buffer time is set to $\xi = 0$), and to find the route $r$ that minimizes $f(\delta^r_1, \ldots, \delta^r_{|T|})$ by iterating over all routes in $\mathcal{R}$.

An improved algorithm sweeps back in time over all recorded timetables $T_i$ simultaneously (as in Algorithm 7), and stops every time that a trip $\tau$ leaves $d$. Let $T_{\tau}$ be the timetable that contains $\tau$. We find all routes in $\mathcal{R}_i$ (initially, $\mathcal{R}_i = \mathcal{R}$ for every $i$) that start with $L(\tau)$ and that reach $t$ (in $T_{\tau}$) on time. For every such route $r$, we set $\delta^r_i \leftarrow t_A - D(\tau, d)$ and remove $r$ from $\mathcal{R}_i$. The algorithm stops if every $\mathcal{R}_i$ is empty, or if the sweepline is “sufficiently far” away from $t_A$. Depending on $f$, one may also stop earlier. For example, for the Norm-$\infty$ we can already stop if there is some route $r$ that was removed from every $\mathcal{R}_i$, and if the time difference between the sweepline and $t_A$ is larger than $f_{\|\cdot\|\|}(r)$. Analogously, for Norm-1 and Mean-Risk-$c$, we can maintain a lower bound on $f(\delta^r_1, \ldots, \delta^r_{|T|})$ for every route $r$, and stop the algorithm after the lower bound of every $r \in \bigcup_{i=1}^{|T|} \mathcal{R}_i$ exceeds the minimum objective value of a route $r \in \mathcal{R}\setminus\bigcup_{i=1}^{|T|} \mathcal{R}_i$.

4.6 Conclusion

The main contribution of this chapter are methods to find robust routes that use historic delays. The easiest methods ignore the historic data completely and enforce robustness only by adding some additional buffer time at each transfer or at the end of the journey. A more sophisticated strategy based on an idea by Buhmann et al. [18] first estimates how much in advance one has to depart, and then finds all routes for which each input instance contains at least one realization that left the departure stop after the estimated departure time, and that reaches the target stop on time. Among these, we select the set of the most realized ones. Since this method by design can accept only two input instances, we proposed norm-based generalizations, and also adapted the mean-risk model by Lim et al. [43] to public transportation.
Measuring the Robustness of Routes
Experimental Evaluation

In this chapter we experimentally investigate the model and the methods proposed in the previous chapters on real-world data of the urban public transportation network in Zürich. As a first step we investigate the given network itself, and we will show that it contains a considerable amount of delays. After that, we will investigate the quality of the methods proposed in the previous chapter with respect to, e.g., the arrival rates of the methods, the average departure and travel times and the influence of the number of required transfers. Moreover, we will see that the similarity-based approach helps to identify meaningful input instances, and we will use it to interpret the observed results.

5.1 Input Data

Provided databases  The real-world data that was used for the experimental evaluation was provided by Verkehrsbetriebe Zürich (VBZ), the transit company of Zürich. It contains the stops of the transportation network of the city of Zürich and the surrounding area, and historic delay data of the vehicles in the time period from April 1, 2013 until May 30, 2013. We were given five databases of which we used four. The first one stores the stops of the network. It contains 653 records (i.e., rows) where each record contains the ID of a stop (an integer, e.g. 736), its real name (a string, e.g., “Zürich, ETH/Universitätsspital”), its internal abbreviation (a string of length 4 or 6, e.g. ETHZ) and its corresponding coordinates (in Swiss coordinates, LV03). The remaining fields store internal information used by VBZ that can be ignored for our experiments.

The remaining four databases contain delay data of the city buses, agglomeration buses, trolley buses and trams. In reality, the public transportation network in Zürich also includes boats, funicular railways and suburban trains, but this data is not available to us. Hence we concentrated on the network of city buses, trolley buses and trams. We focused on the part of the network located in the city because there the number of alternative routes is sufficiently high. The agglomeration buses were ignored because they operate outside the city where the network usually is relatively thin there, and some buses operate
even at a low frequency. Therefore the number of possible alternative routes is smaller than in the city itself, and such a situation is not too interesting when one wants to evaluate the robustness of routes (we can only vary the departure time, not the route itself).

**Description of the delay data** The aforementioned four delay databases are structured as follows. Each record describes the stay of a vehicle at a stop. It contains the external line ID (an integer corresponding to the line number visible to a traveler, e.g., 10 to represent tram 10, 31 to represent bus 31, etc.), the direction of the line (1 or 2), the corresponding stop ID, the corresponding abbreviation of the stop, the actual arrival time at the stop, the planned arrival time at the stop, the actual departure time at the stop, the planned departure time at the stop, the next stop on the line (ID and abbreviation), the actual and real arrival and departure times at the succeeding stop, the operational day, the course number of the vehicle (an integer, e.g., 1 for the first bus of the corresponding line), and the trip ID (an integer). Times are represented by integers that measure the elapsed seconds since midnight. There are some additional fields such as a short explanation of this record (normal way, detour due to road work, etc.) which are not relevant for us. Notice that the delay databases contain multiple redundant fields: the stop abbreviations could also be extracted from the stop database using the corresponding ID, and each record also stores the arrival and departure times at the succeeding stop which are also stored in some other record.

**Cleaning and conversion of the delay data** As a first step we removed duplicate records, and we also removed all records in which the stop and the succeeding stop coincided. Notice that the delay databases just contain single observations. However, in Chapter 2 we defined a line as a sequence of stops, and we even explicitly distinguished the lines with the same stops that have opposite directions. Therefore we had to extract the lines and the trips on selected days manually from the delay databases.

For that we tracked every vehicle course (i.e., a walk in the transportation network) in the delay databases, and considered the sequence of the stops that it visits. From this sequence, the lines and trips were extracted. Identifying the end of a line (and the start of a new one) is easy, because the direction field changes. To identify the planned course of a vehicle, we considered the corresponding courses for every operational (week)day and selected the one that was realized most often. Since the planned arrival and departure times are stored for every stop on the course, we used them to extract the planned timetable (valid for weekdays).

Once the planned timetable and the lines were extracted, we also extracted the trips for the recorded timetables. Remember that we want to use these
5.1 Input Data

<table>
<thead>
<tr>
<th>Day</th>
<th>(T_{planned})</th>
<th>(T_1)</th>
<th>(T_2)</th>
<th>(T_3)</th>
<th>(T_4)</th>
<th>(T_5)</th>
<th>(T_6)</th>
<th>(T_7)</th>
<th>(T_8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall trips</td>
<td>9999 (!)</td>
<td>9912</td>
<td>9964</td>
<td>9967</td>
<td>9887</td>
<td>9931</td>
<td>9658</td>
<td>9931</td>
<td>8541</td>
</tr>
<tr>
<td>Regular trips</td>
<td>9439</td>
<td>9365</td>
<td>9406</td>
<td>9416</td>
<td>9336</td>
<td>9389</td>
<td>9109</td>
<td>9318</td>
<td>8005</td>
</tr>
<tr>
<td>Incomplete regular trips</td>
<td>–</td>
<td>235</td>
<td>140</td>
<td>249</td>
<td>111</td>
<td>183</td>
<td>194</td>
<td>174</td>
<td>280</td>
</tr>
</tbody>
</table>

Table 5.1 Overview of the number of trips in each instance, the number of regular trips (e.g., trips of regular lines), and the number of regular trips that left out at least one stop.

timetables for learning typical delays, and for that we always assumed that they are somewhat typical. Since the traffic on different days of the week is usually different, all recorded timetables should be realized on the same day of the week. We decided to compute the recorded timetables of eight consecutive Thursdays in the period from 4 April to 30 May 2013, ignoring 9 May (which was a public holiday and therefore had different traffic and a different planned timetable). Similar to the reconstruction of the planned timetable, for each of the aforementioned days we tracked the walk of each vehicle course in the transportation network. Sometimes it happened that a vehicle skipped certain stops or turned around in advance. Since each record does not only store the actual but also the planned arrival and departure times, we were able to identify the stops that were left out. For them, we set the arrival time to \(+\infty\) and the departure time to \(-\infty\). It even happened very rarely that complete courses were missing in the database. Problems with errors in real-life data have been reported before [1, 2, 26]. Table 5.1 shows details on how often these situations occurred in the recorded timetables.

Statistics After cleaning and processing the delay databases as described above we obtain a transportation network with 401 stops and 292 lines. Notice that the number of stops is smaller than before, because we ignored the agglomeration buses as well as the corresponding stops.

Issues in the data As already mentioned, few trips were missing or incomplete. Moreover, for one bus line we observed that in the evening it made a detour and visited a stop \(s_i\) twice, i.e. its stop sequence had the form \(\langle s_1, \ldots, s_i, s_{i+1}, s_i, s_{i+2}, s_{i+3}, \ldots \rangle\). This violates our simplifying assumption that no stop appears twice on a line, hence for incorporating such a situation one would have to modify the implementation. Since we observed such a situation only for one line and for this particular line even only rarely, we instead simply ignored the detour and considered \(\langle s_1, \ldots, s_i, s_{i+2}, s_{i+3}, \ldots \rangle\) instead.

Furthermore there was also some work on the tracks in April and May 2013, leading to detours of few tram lines. However, the majority of work was performed only in the evening. We, however, will mostly consider the situation
Figure 18  Distribution of the error coefficients grouped by planned travel time (a), and the average, minimum, maximum, 10th and 90th percentiles of the distribution of the error coefficients in each time slot (b). Times were measured in minutes. For the sake of clarity, (a) does not show journeys with an error coefficient greater than 8, because there are only few (< 0.1%). For the same reason journeys with more than one hour planned travel time are not shown in the figures.

during the day, especially during the rush hours. Moreover, the goal of this chapter is to evaluate the quality of the algorithms discussed in the previous chapters, and not so much to analyze the network itself. Therefore we think that the present data is sufficient.

5.2 Situation in Zürich

In the introduction we already mentioned the study of Firmani et al. [32] who observed that the timetable information and the movement of vehicles in the public transportation network of Rome are only mildly correlated. However, it is unclear whether their observations are also valid for Zürich. Hence, as a preliminary step, we shortly investigate the behavior of the public transportation network in Zürich. To make our results comparable to the results of Firmani et al., the methodology and notation of our preliminary study are similar to the methodology and notation in the original article [32].

We selected 5,000 departure and target stops $d,t$ uniformly at random, set the latest allowed arrival time $t_A$ to 8:00, and computed the $dt$-journeys $j$ that are optimal according to the planned timetable. For each journey $j$, we measured the planned travel times $t_p(j)$ as well as the actual travel times $t_a(j)$ (on 23 May 2013), and computed the error coefficient $t_a(j)/t_p(j)$. Figure 18(a) shows the distribution of the error coefficients grouped by the planned travel times $t_p(j)$. High error coefficients occur easily if the planned travel time is small and the vehicle of the planned journey leaves $d$ a bit too early so that
5.3 Experimental Setup

We observed that in reality many of the 292 lines have the same ID (such as, e.g., Tram 6, Bus 31, etc.). This is a consequence of our modeling: not only do we distinguish lines traveling in opposite directions, but there are also special lines coming from or going to the depot, lines whose corresponding vehicle is planned to turn around in advance, and lines that do not visit certain stops in the evening (see Section 2.6 for details). Since these special lines operate only on a low frequency and mostly only early in the morning or late in the evening, we ignored them and focused on the “standard” realizations. Hence we effectively used only 118 of the 292 lines. Although the network is rather small in comparison to the networks of other cities, it is well-suited for an experimental study on robustness for two reasons. First, the network is dense enough to provide many different routes between any two stops $d$ and $t$. Second, our study in the previous section showed that the network is affected by a considerable amount of delays, especially during the rush hours.

For each of the following experiments, we generated 1,000 (5,000 for the experiments on the number of transfers) departure/target pairs $(d, t) \in S^2$ with $d \neq t$ uniformly at random. For each such pair $(d, t)$, we computed the smallest $\beta \in N_0$ such that $R^\beta_{dt} \neq \emptyset$ and used this value for the maximum allowed number of transfers. If $\beta$ was below one or above three, we ignored the corresponding pair $(d, t)$. Although these decisions may sound arbitrary, there are reasons for them. The reason to forbid more transfers than necessary is motivated from a
practical point of view: in urban areas, travelers often are interested in routes with as few transfers as possible. Moreover, routes with more transfers than necessary are likely to be less robust due to the increased danger of missing a trip. The reason for ignoring pairs of stops between which a direct connection exists is that these pairs don’t give much insight into whether a route is robust or not due to the lack of alternative direct routes (in most cases). Pairs between which only routes with four or more transfers exist were ignored, because there are few of them (< 5%), and for these in reality a journey with fewer transfers exists, but this involves walking between nearby stops.

After computing $\beta$ and $R^\beta_{dt}$, we performed the corresponding experiment. Except otherwise stated the target arrival time $t_A$ was set to 18:00. We compared the optimal planned solution OPT-TT, the Buffer-$\delta$ methods with an additional transfer buffer of $\delta$ (see Section 4.3), Similarity in which the most frequent routes in the intersection were selected (see Section 4.4), the norm-based approaches Norm-1, Norm-2 and Norm-Inf (see Section 4.5) and the mean-risk model with risk aversity $c$ Mean-Risk-$c$ (see Section 4.5). Methods that enforce some additional buffer time at the end of a journey turned out to perform identical to the Buffer-$\delta$ methods which use an additional transfer buffer at each transfer, so the following figures don’t show them. Also, Norm-2 performed similarly to Norm-Inf, so the figures mostly omit also Norm-2. Notice that Norm-1 and Mean-Risk-0 are equivalent. Hence, the figures show only the results for the latter method. Unless otherwise stated, Buffer-$\delta$ used the planned timetable $T_{\text{planned}}$ as input, Similarity used $T_5$ and $T_6$ (recorded on 2 May and 16 May), and the function-based methods used $T_1, \ldots, T_6$ (recorded between 4 April and 16 May). Timetable $T_7$ (recorded on 23 May) was used to assess the quality of the proposed journeys. We ignored $T_8$ because the number of missing and incomplete trips is much higher than in $T_1, \ldots, T_7$.

Each of the aforementioned methods returns one or more routes and corresponding departure times, and for each of these suggestions we measured the corresponding arrival times in $T_7$ using Algorithm 6. If a method recommended more than one route, we computed the average arrival time over the recommendations. Notice that we directly evaluated the route $r$ and the departure time $t_D$ suggested by each method, and did not evaluate some journey obtained from $r$ and $t_D$. The reason is that we wanted to investigate the quality of the suggestions (i.e., the routes and departure times) itself. Taking transfers into account also shouldn’t change much: if two lines have only one common stop, then obviously nothing changes. If two lines had more than one transfer of which one would be clearly superior to the another (according to the planned timetable), then it is likely that in reality the very same stop would be the best transfer.
We observed that it rarely happened that Buffer-ξ, Similarity, Norm-Inf or Mean-Risk-1 propose solutions that arrive much too early or much too late in the test instance. In all of these cases this was caused either because of a highly non-typical situation in the input or the test instance (e.g., an accident), or because a line was chosen that was not realized regularly (e.g., less than once per hour). Hence we ignored all pairs \((d, t)\) for which at least one of these methods computed a solution that arrived more than one hour too early or more than one hour too late in the test instance.

The algorithms were implemented in Java 7, and the experiments were performed on one core of an Intel Core i5-3470 CPU clocked at 3.2 GHz with 4 GB of RAM running Debian Linux 7.8. For enumerating all \(dt\)-routes in \(R^\beta dt\), we used Algorithm 1 whose running time is negligible (less than 1ms). After computing \(R^\beta dt\), the buffer strategies have an average running time 3ms or less, the similarity-based approach 17ms, and the function-based approaches 54ms. The latter method was implemented using the simple strategy described in Section 4.5 where every \(\delta^r_i\) is explicitly computed for every route \(r\) and every recorded timetable \(T_i\).

5.4 Arrival Rate, Departure Time and Standard Deviation on the Arrival Time

Intuitively, an earlier departure time leads to a higher probability to arrive on time (i.e., a higher arrival rate). Figures 19 and 20 compare the proposed methods with respect to this aspect, and also relate the arrival rate to the standard deviation of the arrival time. They show that, independently of the considered method, there is a clear trade-off between the departure time and the arrival rate (a). For the trade-off between the arrival rate and the standard deviation (b), such a connection is less clearly visible, but one can observe that a higher arrival rate often leads to a higher standard deviation on the arrival time.

Observe that in Figure 19(a) and Figure 20(a) both parameter-based methods Buffer-ξ and Mean-Risk-c produce Pareto-optimal solutions (with respect to the departure time and the arrival rate). Clearly, Mean-Risk-c benefits from the additional information from the recorded input instances \(T_1, \ldots, T_6\), hence it is never dominated by Buffer-ξ in Figure 19 and Figure 20. Similarity does not require to choose some parameter (such as \(\delta\) for Buffer or \(c\) for Mean-Risk), and using only two recorded timetables at least for \(t_A = 8:00\) it still proposes solutions with a reasonable arrival rate that do not depart too early. Notice that Norm-Inf (the generalization of Similarity) also benefits from the knowledge of six past timetables, and without parameter adjustment it produces a solution
Figure 19 Comparison of various methods: arrival rate vs. average departure time (a), and arrival rate vs. standard deviation on the arrival time (b). The target arrival time was set of $t_A = 8:00$.

Figure 20 Comparison of various methods: arrival rate vs. average departure time (a), and arrival rate vs. standard deviation on the arrival time (b). The target arrival time was set of $t_A = 18:00$.

which gives a very reasonable trade-off between the departure time, the arrival rate and the standard deviation on the arrival time. Moreover, the solutions proposed by Norm-Inf performed quite well compared to all the competitors (which do require parameter adjustment).

We also investigated whether the arrival rates of different methods differ due to different departure times only, or whether the suggested route(s) also differ. In particular, for any two methods $M_1$ and $M_2$ that recommend a set of routes $R_1$ and $R_2$, respectively, we studied how often the suggested route(s) of $M_1$ and $M_2$ differ (i.e., how often $R_1 \neq R_2$). For some exemplary methods, Table 5.2 shows that this happens in roughly 14 to 45% of the cases. Notice that in roughly one third of the cases, the routes proposed by Similarity differ from the
5.5 Influence of the Similarity between Input and Test Instances

We just saw that recommendations computed by Similarity perform relatively poorly, with respect to both arrival rate as well as standard deviation on the arrival time. However, we have to take into account that these methods use only two recorded timetables as input: if both differ substantially from the test instance, then in general there is very little one can do. The generic approach of Buhmann et al. [18] requires both the input and the test instances to be typical, i.e., their mutual similarities have to be high. Thus we investigate the impact of high and low mutual similarities on the quality of the predictions.

First we note that the similarity $S_{OPT}^{\gamma}$ does not only depend on the two input instances but also on the departure stop $d$ and the target stop $t$, and on the target arrival time $t_A$. Thus, in the following experiments, we do not always use the same timetables $T_5$, $T_6$ as input and $T_7$ for testing, but select for every $(d, t)$ the timetables whose mutual similarities are as high or as low as possible. Let $\Upsilon$ be the set of all triples of recorded timetables $(T_i, T_j, T_k) \in T^3$ where $i, j, k$ are mutually different. For a given pair $(d, t)$ and two timetables $T_i, T_j \in T$, let $S_{ij}^{dt}$ be the similarity of $T_i$ and $T_j$ with respect to $d$ and $t$. We then select triples whose minimum (or maximum, respectively) pairwise

<table>
<thead>
<tr>
<th>Opt-TT</th>
<th>Buffer-3</th>
<th>Buffer-6</th>
<th>Buffer-9</th>
<th>Buffer-12</th>
<th>Norm-1</th>
<th>Norm-Inf</th>
<th>Similarity</th>
<th>Mean-Risk-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opt-TT</td>
<td>38.80%</td>
<td>30.77%</td>
<td>33.00%</td>
<td>36.76%</td>
<td>31.11%</td>
<td>42.53%</td>
<td>41.06%</td>
<td>35.63%</td>
</tr>
<tr>
<td>Buffer-3</td>
<td>38.80%</td>
<td>39.14%</td>
<td>33.82%</td>
<td>31.67%</td>
<td>33.94%</td>
<td>40.27%</td>
<td>41.52%</td>
<td>34.95%</td>
</tr>
<tr>
<td>Buffer-6</td>
<td>30.77%</td>
<td>39.14%</td>
<td>37.44%</td>
<td>33.82%</td>
<td>37.56%</td>
<td>43.67%</td>
<td>43.55%</td>
<td>39.03%</td>
</tr>
<tr>
<td>Buffer-9</td>
<td>33.00%</td>
<td>33.82%</td>
<td>37.44%</td>
<td>39.03%</td>
<td>34.73%</td>
<td>42.65%</td>
<td>41.40%</td>
<td>36.54%</td>
</tr>
<tr>
<td>Buffer-12</td>
<td>36.76%</td>
<td>31.67%</td>
<td>33.82%</td>
<td>39.03%</td>
<td>36.31%</td>
<td>44.80%</td>
<td>44.12%</td>
<td>38.57%</td>
</tr>
<tr>
<td>Norm-1</td>
<td>31.11%</td>
<td>33.94%</td>
<td>37.56%</td>
<td>34.73%</td>
<td>36.31%</td>
<td>28.73%</td>
<td>31.67%</td>
<td>14.03%</td>
</tr>
<tr>
<td>Norm-Inf</td>
<td>42.53%</td>
<td>40.27%</td>
<td>43.67%</td>
<td>42.65%</td>
<td>44.80%</td>
<td>28.73%</td>
<td>31.79%</td>
<td>21.61%</td>
</tr>
<tr>
<td>Similarity</td>
<td>41.06%</td>
<td>41.52%</td>
<td>43.55%</td>
<td>41.40%</td>
<td>44.12%</td>
<td>31.67%</td>
<td>31.79%</td>
<td>32.24%</td>
</tr>
<tr>
<td>Mean-Risk-1</td>
<td>35.63%</td>
<td>34.95%</td>
<td>39.03%</td>
<td>36.54%</td>
<td>38.57%</td>
<td>14.03%</td>
<td>21.61%</td>
<td>32.24%</td>
</tr>
</tbody>
</table>

Table 5.2 Overview of how often the route suggestions of two methods differ.
similarity is as high or as low as possible,

\[
(T_h^1, T_h^2, T_h^3) = \arg \max_{(T_i, T_j, T_k) \in \Upsilon} \min \{S_{ij}^{dt}, S_{ik}^{dt}, S_{jk}^{dt}\}
\]

(19)

\[
(T_l^1, T_l^2, T_l^3) = \arg \min_{(T_i, T_j, T_k) \in \Upsilon} \max \{S_{ij}^{dt}, S_{ik}^{dt}, S_{jk}^{dt}\}
\]

(20)

and use \(T_h^1\) and \(T_h^2\) as input and \(T_h^3\) for testing, and for comparison, use \(T_l^1\) and \(T_l^2\) as input and \(T_l^3\) for testing. Even though Mean-Risk-\(p\) and Norm-\(p\) could handle more instances, they were given just the two mentioned instances to investigate whether they also benefit from a high mutual similarity.

Figure 21(a) shows that all methods benefit when the similarity of the three instances is high. The arrival rates of both Norm-\(p\) and especially Similarity increase significantly. We observed that Similarity outperforms Norm-\(p\) when the similarity is low, which is reasonable: for a low similarity, the routes in the first intersection of the approximation sets as well as the route that maximizes the average departure time are too much influenced by the noise in the input instances. However, Similarity can still let the approximation sets grow beyond the first intersection so that more robust routes are identified (which Norm-\(p\) cannot). On the other hand, if the similarity is high, then there is so little noise in the data that \(S_\gamma\) is likely to be maximized already at the first \(\gamma\) for which the intersection is non-empty, thus Similarity and Norm-\(p\) perform nearly identical.

Of course these results cannot directly be used for designing an algorithm, since the test instance is unknown. Nevertheless we believe that the results
5.6 Influence of the Target Arrival Time

Figure 22 shows how the behavior of the methods, in terms of on-time arrival rate (a) and travel time (b), changes over the day. In particular, we can observe a clear influence of the morning and evening rush hours. Interestingly, the two rush hours affect the arrival rates of different methods differently. Specifically, the timetable-based method Buffer-ξ is greatly affected by both rush hours while the learning strategies are less affected by the evening rush hour.

To understand this behavior, consider Figure 21(b). The red curve shows how the value of the similarity of $T_{\text{planned}}$ and the test instance $T_7$ changes during the day. In particular, we see a significant drop of the similarity during rush hours. Notably, the two dips corresponding to morning and evening rush hour are of the same height. This suggests that on the day corresponding to $T_7$, during the morning rush hour, there was a similar amount of irregularities with respect to $T_{\text{planned}}$ as during the evening one. The blue curve in Figure 21(b) shows the changes during the day of the averaged value of similarity of $T_{\text{planned}}$ and each of the training instances $T_1 - T_6$. Also there the similarity drops during the rush hours, but we clearly see that the morning dip is significantly lower than the evening one. This suggests that in the recorded timetables $T_1 - T_6$ used for learning, the amount of irregularities (with respect to $T_{\text{planned}}$) was lower in the morning than in the evening. Thus, when comparing the two curves, we see a significant gap between them during the morning rush hour, but a relative match during the rest of the day. This suggests that the test instance...
$T_7$ contained during the day a similar amount of irregularities as expected on a typical day (represented by $T_1 - T_6$), with the only exception of the morning rush hour where it was less regular.

Let us now relate what we observed in Figures 21(b) and 22(a). Since \texttt{Buffer-}$\xi$ is based solely on $T_{\text{planned}}$, any irregularities with respect to $T_{\text{planned}}$ occurring in $T_7$ (captured by the red curve in Figure 21(b)) affect its arrival rate. This explains why the arrival rate of \texttt{Buffer-}$\xi$ drops both in the morning and evening rush hour and exhibits two dips of nearly the same height. On the other hand, the methods that use the information from the past observations (e.g., \texttt{Mean-Risk-c}) are trained to account for a certain amount of irregularities. Since the situation in $T_7$ in the evening is typical, the solutions proposed by these methods are prepared for it and their arrival rate is almost not affected by the evening rush hour. In contrast, morning rush hour cause the arrival rate to drop significantly and this maps to the discrepancy of the red and blue curve in Figure 21(b).

In Figure 22(b) we observe that during peak hours, the travel time increases. Interestingly, the required travel time does not seem to depend on the method nor on its arrival rate. Thus, to achieve higher arrival rate, one has to depart earlier (as seen in Figure 19(a) and Figure 20(a)), but does not need to increase the time spent traveling. We believe that this is the case because the network of Zürich is quite dense, hence there exist different alternative journeys with comparable travel times.

Goerik et al. [34] performed an experimental study on the german train network and observed that a higher buffer leads to a “considerable increase in travel time” and that “asking for 10 minutes of slack time for each transfer leads to an average increase in travel time of 65 minutes on the complete network”. One may wonder whether these statements contradict our observations above. The reason for such different results is that we use a completely different solution concept. Goerik et al. [34] use an event-activity network (a variation of a time-expanded graph), in which every subpath between a departure and the succeeding arrival vertex corresponds to a trip. We, however, consider sequences of lines whose corresponding trips are indistinguishable for travelers. Goerik et al. [34] assume that at every stop, one waits for the preselected trip, which is a totally reasonable assumption for train networks where one often may not use other than the preselected trains. In urban public transportation, however, lines have much higher frequencies and we simply board the next trip of the corresponding line that arrives. Therefore, we use the buffer only in the planning phase; in reality, if an earlier trip of the same line arrives, we can simply take it.
5.8 Choice of the Parameters for Buffer-\(\xi\) and Mean-Risk-\(c\)

Figure 23 Necessary parameter to achieve a specified arrival rate in \(T_7\) depending on the target arrival time.

5.7 Choice of the Parameters for Buffer-\(\xi\) and Mean-Risk-\(c\)

Figure 23(a) shows the minimum value of the parameter \(\xi\) of Buffer-\(\xi\) that is necessary to achieve arrival rates of 80% and 90% of the cases in \(T_7\), and how this value changes over the day. We see that, affected by the daily rush hours, this parameter varies significantly, suggesting that the Buffer-\(\xi\) strategies need a non-trivial amount of parameter adjustment. We observe that the dips corresponding to morning and evening rush hours are of the same height. Again, we can directly relate this behavior with the similarity of \(T_{planned}\) and \(T_7\) (captured by the red curve in Figure 21(b)).

Similarly, Figure 23(b) shows the value of the risk-coefficient \(c\) of Mean-Risk-\(c\) that is necessary to achieve arrival rates of 78.5%, 90%, and 92.5% in \(T_7\), and its development during the day. We observe that this value is greatly affected by the morning rush hour. On the other hand, the dip corresponding to the evening rush hour is visible, but not too significant. This again can be explained using Figure 21(b): Since the morning in \(T_7\) was not typical with respect to previous observations, the risk-coefficient \(c\) has to be quite large (see Figure 23(b)) to compensate for these unexpected irregularities. On the other hand, when the traffic behaves as expected (i.e., when the two curves in Figure 21(b) approximately match), the Mean-Risk-\(c\) method performs well and fine-tuning of the parameters is not crucial. For instance, a risk-coefficient \(c\) set to 1 already leads to reasonably robust solutions.
5.8 Influence of the Number of Transfers

Figure 24(a) shows that the arrival rate of Buffer-\(\xi\) (for \(\xi = 6\)) is quite sensitive to the number of transfers. This suggests that the number of transfers is another aspect (of possibly many aspects) which has to be taken into account when searching for the best parameter for Buffer-\(\xi\). In contrast, Figure 24 (b)–(d) shows that the influence of the number of transfers on the arrival rate of the learning-based methods Similarity, Norm-Inf and Mean-Risk-1 is clearly less visible. Especially for Mean-Risk-1, there is no need to fine-tune the risk-coefficient \(c\) to compensate for this aspect.

5.9 Conclusion

We observed a clear trade-off: to achieve a higher probability to arrive on time in a network with delays, one has to depart earlier and expect a higher standard
deviation on the arrival time. On the other hand, the average travel time itself does not change with robustness or the choice of a routing method.

Methods based solely on the planned timetable, where the robustness is achieved by adding buffer times, need a non-trivial parameter adjustment for which many aspects need to be considered (time of the day, number of transfers, etc.). Methods learning from the past benefit from the additional knowledge: if the test instance is typical with respect to the past observations, these strategies perform well, Mean-Risk-c does not need much fine-tuning, and Norm-Inf without parameter adjustment proposes a highly competitive solution with reasonable trade-offs. We saw that Similarity gives a good measure of the amount of irregularities in the network and can help to detect typical situations. Notably, it considers complex solutions (routes), and thus it has a potential to capture behavior that cannot be observed only locally. We believe that this measure is worth further exploring, and by considering various aspects (e.g., how different approaches would benefit if Similarity was used to preselect typical instances for training) it can bring us even closer to the goal of robust routing.

The existence of equally good alternative routes is one of the reasons why we believe that it was reasonable to choose the public transportation network of Zürich for our experiments, although the network is rather small in comparison to the public transportation networks of other cities. An interesting question is whether the algorithms are still sufficiently fast on larger networks. We believe that due to our solution concept (i.e., sequences of lines), the running time depends on the number of lines rather than the number of stops. In that respect, the network of Zürich is not exorbitantly small: For example, the public transportation network in Vienna has more than six times as many stops, but only less than two times as many lines. Hence, if the number of feasible $dt$-routes (with a bounded number of transfers) is not too large, the algorithms should still work fast. Otherwise one could try to generate meaningful alternative routes in advance. Investigating these aspects and also whether our qualitative results hold for other cities are clearly interesting questions that should be investigated further.
Approximation Set-Based Approaches: Models and Extensions

In this chapter we take a closer look at the approximation set-based methods that Buhmann et al. [17, 18] proposed, and that were also studied by R. Šramek in his Ph.D. thesis [62]. In the first section, we recall their ideas that are necessary to understand this chapter. After that, we compare their methods with the commonly-used joint minimizer (JM) and minimax-regret (MR). We will experimentally identify situations where these approximation-set based methods outperform the other two, and also give an example in which JM and MR are superior to the approximation-set based approaches. We will then give an analytic expression to estimate the similarity of two instances and experimentally evaluate the quality of this estimation. The original approach of Buhmann et al. is only designed to incorporate two input instances. Since in reality one often has more than two instances available, we will propose two possible generalizations of the approximation-set based approaches. We conclude the chapter with a summary of the results and some remarks about an application of the aforementioned methods to Linear Regression and Linear Programming.

6.1 Introduction

To make the thesis self-contained, this section briefly recalls the ideas necessary to understand the approximation set-based approaches by Buhmann et al. [17, 18]. It only describes the results known so far, and in particular it does not include new ideas by me. Large parts of this section are similar to Chapter 2 in the Ph.D. thesis of R. Šramek [62], where he explains his ideas in more detail, and to the original research article by Buhmann et al. [18] (which is a shortened version of Chapter 2 of the aforementioned Ph.D. thesis). This section especially adopts their terminology and notation.

Model of Uncertainty Classical optimization often neglects uncertainty and can be seen as follows. The input to a problem consists of an instance I from an infinite set of possible instances (a subset of a space of mathematical
structures) $\mathcal{I}$. The usual goal is to choose from a set of possible solutions $\mathcal{S}$ a solution $s^\perp \in \mathcal{S}$ that minimizes a given cost function $c : \mathcal{S} \times \mathcal{I} \to \mathbb{R} \cup \{\infty\}$. The structure of the solution space $\mathcal{S}$, of the instance $I$ and the instance space $\mathcal{I}$, and of the cost function $c$ depend on the problem to be solved, but they are assumed to be known (although at least $\mathcal{S}$ is usually represented only implicitly). When a solution $s \in \mathcal{S}$ is not feasible for an instance $I$ (i.e., not compatible with $I$), we simply set $c(s, I) = \infty$.

In the previous chapters we studied routing problems in public transportation networks. For example, if we want to travel from $d$ to $t$ and we want to start the journey at $d$ at time $t_D$, an instance $I$ is a (planned or recorded) timetable, the set of feasible solutions $\mathcal{S}_I$ is the set of all feasible $dt$-routes, and the cost of a solution (i.e., a $dt$-route) $c(s, I)$ is the time at which $t$ is reached if $d$ is left at time $t_D$ and the route $s$ is used (departure and arrival times are considered with respect to the timetable that $I$ represents). If the network didn’t contain any delays, then the real and the planned timetables would coincide, and the usual goal is to find a route by which $t$ is reached as early as possible. In the previous chapters we studied a slightly different setting in which we were given $d, t$ and a deadline $t_A$ by which $t$ has to be reached (instead of a departure time $t_D$ at $d$). Instances and the set of feasible solutions can be defined analogously, only does now $c(s, I)$ measure how much earlier than $t_A$ we would have to leave $d$ such that $t$ is still reached on time.

In many real-life problems, however, uncertainty is omnipresent due to congestions, imprecise measurements, etc. There are many ways to model optimization under uncertainty, and many robust optimization methods such as the earlier mentioned strict, light and recoverable robustness: these were not specifically designed for transportation problems but are general approaches to robust optimization (see [35] for an overview). In this chapter we take a closer look at the approximation-based methods that Buhmann et al. [17, 18] proposed, and that were also studied by R. Šramek in his Ph.D. thesis [62]. They model uncertainty by a problem generator $\mathfrak{PG}$ that generates a set $\mathcal{I}_{\mathfrak{PG}} \subset \mathcal{I}$ of related instances that differ slightly due to noise. For simplicity they assume that the instances in $\mathcal{I}_{\mathfrak{PG}}$ have the same set of feasible solutions $\mathcal{S}_{\mathfrak{PG}} \subset \mathcal{S}$, i.e., for every instance $I \in \mathcal{I}_{\mathfrak{PG}}$, we have $c(s, I) < \infty$ for every $s \in \mathcal{S}_{\mathfrak{PG}}$ and $c(s, I) = \infty$ for every $s \in \mathcal{S} \setminus \mathcal{S}_{\mathfrak{PG}}$. Hence in the following we simply write $\mathcal{S}$ instead of $\mathcal{S}_{\mathfrak{PG}}$.

The problem generator, and especially the type of the noise, are unknown. However, we know the set of feasible solutions $\mathcal{S}$, and we are given exactly two instances $I_1, I_2$ generated by $\mathfrak{PG}$ (Buhmann et al. studied the two-instance scenario because it is the smallest number of instances that in principle allows to distinguish the signal in the instance from the noise). Now, we want to
6.1 Introduction

identify solutions that are expected to perform “reasonably well” for future, yet unknown, instances by \( \mathcal{PG} \) (a more formal definition of “reasonably well” will be given soon). For combinatorial optimization problems, one often wants to identify solutions whose cost in future instances likely approximate the optimum cost. Another goal might be to extract underlying information about the instance generation process.

**Joint Minimizer (JM)** A straightforward attempt to solve such a noisy optimization problem is to compute the joint minimizer, i.e. the set of solutions \( S_{JM} \) that minimize the joint cost function 
\[
\begin{align*}
&\text{c}^{12}_{JM}(s, I_1, I_2) := (c(s, I_1) + c(s, I_2))/2.
&\end{align*}
\]
From the statistical point of view, the joint minimizer finds a solution based on the mean estimator of instance costs, see, e.g., [41].

**Approximation Sets** Buhmann et al. [18] introduced the following concept of an approximation set. For an instance \( I \) and a real number \( \rho \geq 1 \), the \( \rho \)-approximation set of \( I \) is defined as
\[
A_\rho(I) = \{ s \in S \mid c(s, I) \leq c(s^\perp, I) \cdot \rho \},
\]
i.e., \( A_\rho(I) \) contains all solutions that approximate the cost of a minimum solution \( s^\perp \) within a factor \( \rho \). Although multiplicative approximation is commonly used in the context of combinatorial optimization, some models and problems in this chapter allow solutions to also have negative cost, hence multiplicative approximation cannot be used. We therefore slightly modify the above definition: for a real number \( \gamma \geq 0 \) the \( \gamma \)-approximation set of \( I \) is the set
\[
A_\gamma^+(I) = \{ s \in S \mid c(s, I) \leq c(s^\perp, I) + \gamma \},
\]
i.e., we consider additive instead of multiplicative approximation. For the remaining part of this introductory section we keep the original definition (21) of Buhmann et al.; notice, however, that their ideas and concepts still work when additive approximation sets are used.

**Approximation Sets in the Two-Instance Scenario** Since we possess no knowledge about \( \mathcal{PG} \) except for the two given instances \( I_1 \) and \( I_2 \), we have to identify the solutions which performed well under both \( I_1 \) and \( I_2 \): only those are likely to be good for other instances. Hence, for a well-chosen parameter \( \rho^* \) we compute the set \( A_{\rho^*}(I_1) \cap A_{\rho^*}(I_2) \) and sample a solution from it uniformly at random. We now discuss how to choose an appropriate value \( \rho^* \) so that the solutions in the intersection of the corresponding approximation set are likely to be robust.

**First Intersection (FI) / Minimax Regret (MR)** An obvious choice for \( \rho^* \) is the smallest value \( \rho \) for which the intersection of the \( \rho \)-approximation sets
\( A_\rho(I_1) \cap A_\rho(I_2) \) is non-empty. In the context of additive approximation this method is known as *minimax-regret*. It was introduced by Savage [58].

**Maximizing the Similarity (SIM)** Choosing the smallest \( \rho^* \) for which the intersection of the corresponding \( \rho \)-approximation sets is non-empty might not be the best strategy: for a sufficiently large \( \rho^* \), the intersection of the corresponding approximation sets will always be non-empty, no matter whether the two instances are similar or not. Hence, Buhmann et al. [18] relate the number of solutions in the intersection to the expected number of solutions in the intersection. More concretely, they define \( F_k \) as the set of all feasible approximation sets of size \( k \), i.e. as the set of all subsets \( X \subseteq S \) of size \( k \) for which there exists a number \( \rho \geq 1 \) and an instance \( I \in \mathcal{I} \) such that \( X = A_\rho(I) \). They then propose to find the value \( \rho \) at which the so-called *(unexpected)* similarity of \( I_1 \) and \( I_2 \) at value \( \rho \),

\[
S_\rho(I_1,I_2) := \frac{|A_\rho(I_1) \cap A_\rho(I_2)|}{\mathbb{E}_{B \in \mathcal{F}_{|A_\rho(I_1)|}, C \in \mathcal{F}_{|A_\rho(I_2)|}}|B \cap C|
\]

is maximized, where \( B \) and \( C \) are chosen uniformly at random from \( \mathcal{F}_{|A_\rho(I_1)|} \) and \( \mathcal{F}_{|A_\rho(I_2)|} \), respectively. Let \( \rho^* = \arg \max_\rho S_\rho(S_1,S_2) \) be the value which maximizes (23). Now they define \( S(I_1,I_2) := S_{\rho^*}(I_1,I_2) \) as the *(unexpected)* similarity of \( I_1 \) and \( I_2 \). For simplicity, we will omit the term “unexpected” in the rest of this chapter. If there are multiple values \( \rho^* \) that maximize (23), then we (somewhat arbitrarily) define \( \rho^* \) as the smallest among these. Buhmann et al. [18] proved that

\[
S_\rho(I_1,I_2) \geq \frac{|A_\rho(I_1) \cap A_\rho(I_2)|}{|A_\rho(I_1)| \cdot |A_\rho(I_2)|},
\]

and they also showed that if the problem has the property that *every* subset of \( S \) is a feasible approximation set (which is the case for, e.g., the problem of finding a minimum in an array, and for clustering [17]), then the similarity at value \( \rho \) can be expressed as

\[
S_\rho(I_1,I_2) = \frac{|S||A_\rho(I_1) \cap A_\rho(I_2)|}{|A_\rho(I_1)| \cdot |A_\rho(I_2)|}.
\]

Notice, however, that many combinatorial optimization problems do not have the aforementioned property. We also note that the proofs of the above statements (24) and (25) do not exploit the fact that \( A_\rho(I) \) uses multiplicative approximation; hence, they still hold when additive approximation sets are used, i.e., when \( A_\rho(I_k) \) is replaced by \( A_\rho^+(I_k) \). However, as the following example shows, replacing multiplicative approximation by additive might have
a huge impact on the behavior of the approach. Imagine that some artificial problem had \( n \) solutions \( s_1, \ldots, s_n \), and we had two instances \( I_1 \) and \( I_2 \) where \( c(s, I) = i \) and \( c(s, I_2) = (2 - i/n)L \) for some large \( L > n^2 \). Obviously the solution that minimizes \( c(\cdot, I_1) \) is \( s_1 \) (with \( c(s_1, I_1) = 1 \)), and the solution that minimizes \( c(\cdot, I_2) \) is \( s_n \) (with \( c(s_n, I_2) = (2 - n/n)L = L \)). When additive approximation is used, then \( A^+_\gamma(I_1) \) changes at \( \gamma_0 = 0, \gamma_1 = n - 1 \), while \( A^+_\gamma(I_2) \) changes at \( \gamma_0 = 0, \gamma_1 = L/n, \gamma_2 = 2L/n, \ldots, \gamma_2n-2 = (n-1)L/n \). Since \( L > n^2 \) we have \( \gamma_i < \gamma_{i+1} \). Now, for every \( i \in \{0, \ldots, n-2\} \), we have \( A^+_{\gamma_i}(I_2) = \emptyset \), hence \( S_{\gamma_i}(I_1, I_2) = 0 \). For \( i = n-1 \) we have \( A^+_{\gamma_i}(I_1) = \{s_1, \ldots, s_n\} \) and \( A^+_{\gamma_i}(I_2) = \{s_n\} \), hence \( S_{\gamma_i}(I_1, I_2) = n/n = 1 \). For \( i \geq n \) we have \( A^+_{\gamma_i}(I_1) = \{s_1, \ldots, s_n\} \) and \( A^+_{\gamma_i}(I_2) = \{s_n, s_{n-1}, \ldots, s_{2n-i-1}\} \), hence \( |A^+_{\gamma_i}(I_1)| = n \) and \( |A^+_{\gamma_i}(I_2)| = |A^+_{\gamma_i}(I_1) \cap A^+_{\gamma_i}(I_2)| = i - n + 2 \), which implies \( S_{\gamma_i}(I_1, I_2) = (n(i - n + 2))/(n(i - n + 2)) = 1 \). Since we choose the smallest \( \gamma^* \) that maximizes \( S_{\gamma^*}(I_1, I_2) \), we obtain \( \gamma^* = \gamma_{n-1} = n-1 \), and the only solution in the intersection of \( A^+_{\gamma^*}(I_1) \) and \( A^+_{\gamma^*}(I_2) \) is \( s_n \). On the other hand, for multiplicative approximation, \( A_\rho(I_2) \) changes for \( \rho_0 = 1, \rho_1 = 1 + 1/n, \ldots, \rho_{n-1} = 1 + (n-1)/n \) while \( A_\rho(I_1) \) changes for \( \rho_0 = 1, \rho_n = 2, \ldots, \rho_{2n-2} = n \). A similar argument as before reveals that the smallest \( \rho^* \) maximizing \( S_{\rho^*}(I_1, I_2) \) is \( \rho^* = \rho_{n-1} = 1 + (n-1)/n \). Observe, however, that now \( A_{\rho^*}(I_1) = \{s_1\} \) and \( A_{\rho^*}(I_2) = \{s_1, \ldots, s_n\} \), hence their intersection contains exactly \( s_1 \) (and not \( s_n \) as before). Arguably, the above example is very artificial, and it was specifically designed to show that additive and multiplicative approximation might give completely different results. Observe that the instances in the example seem to be highly unrelated; for two typical instances in \( \mathcal{I}_{\Phi^\Phi} \) the costs of the solutions in the instances might be relatively close, hence the difference between additive and multiplicative approximation might be much smaller than the above example suggests.

**Similarity with Gibbs Weights (GSIM)**  Buhmann [17] originally introduced a different but related formulation of the aforementioned SIM approach which can be deduced as follows. For a number \( \rho \geq 1 \), an instance \( I \) and a solution \( s \) we define a 0-1-weight \( w_\rho(s, I) \) that is 1 if and only if \( c(s, I) \leq c(s^i, I) \cdot \rho \), and 0 otherwise. It is easy to see that

\[
|A_\rho(I_k)| = \sum_{s \in S} w_\rho(s, I_k) \text{ for } k = 1, 2, \text{ and } \quad (26)
\]

\[
|A_\rho(I_1) \cap A_\rho(I_2)| = \sum_{s \in S} w_\rho(s, I_1) w_\rho(s, I_2). \quad (27)
\]

Now, one can use an idea from statistical physics [45] and replace the 0-1-weights by *Gibbs* weights: for a real number \( \beta \geq 0 \), an instance \( I \) and a solu-
tion $s$, the Gibbs weight is defined as $w^G_\beta(s, I) := \exp(-\beta c(s, I))$. Analogously to (25), (26) and (27) we have to compute the largest value $\beta^*$ that maximizes the ratio

$$S_\beta(I_1, I_2) := \frac{\sum_{s \in S} w^G_\beta(s, I_1) w^G_\beta(s, I_2)}{(\sum_{s \in S} w^G_\beta(s, I_1)) \cdot (\sum_{s \in S} w^G_\beta(s, I_2))}.$$  

(28)

In SIM, after computing $\rho^*$ we return the set $\mathcal{S}_{SIM}$ of all solutions $s$ with $w^\square_\rho^*(s, I_1) = w^\square_\rho^*(s, I_2) = 1$, and solutions from this set were chosen uniformly at random. For solutions $s \in \mathcal{S} \setminus \mathcal{S}_{SIM}$, we have either $w^\square_\rho^*(s, I_1) = 0$ or $w^\square_\rho^*(s, I_2) = 0$, hence sampling a solution uniformly at random from $\mathcal{S}_{SIM}$ can be interpreted as sampling a solution from the whole solution space $\mathcal{S}$ with a probability proportional to $w^G_\beta^*(s, I_1) w^G_\beta^*(s, I_2)$. The same way of sampling can be used for GSIM: we return the whole solution space $\mathcal{S}$, and choose a solution $s$ with a probability proportional to $w^G_\beta^*(s, I_1) w^G_\beta^*(s, I_2)$. Buhmann originally applied a variation of this approach to clustering [17].

Observe that the parameter $\beta$ in (28) is somehow inverse to the parameter $\rho$ in (25): For $\beta = 0$, all solutions have the same weight 1 (corresponding to the case $\rho = \infty$ where the intersection contains every solution in $\mathcal{S}$), while for $\beta \to \infty$ the distribution concentrates on the solutions with the minimum joint cost. This also explains why $\beta^*$ is defined as the largest $\beta$ that maximizes (28) while $\rho^*$ was the smallest one maximizing (23). In statistical physics, $\beta$ is called the “inverse temperature”.

### 6.2 Experiments on the Two-Instance Scenario

**Motivation** One of the goals of this chapter is to identify conditions under which the approximation set-based methods (especially SIM and GSIM) outperform the joint minimizer (JM). Buhmann et al. [18] suggested a very simple “prototypic example” to compare SIM against JM and FI. They assumed that the solution set $\mathcal{S}$ consists of $g$ “good” solutions that always have cost $\hat{\rho} \in (1, 2)$, and $b$ “bad” solutions whose cost is either 1 with a certain probability, or 2 otherwise. The values $\hat{\rho}$ and $p$ are chosen such that the cost of a good solution is smaller than the expected cost of a bad solution, i.e. $\hat{\rho} < 2 - p$. The good solutions can be considered robust in the sense that their costs stay invariant. Although the cost of a bad solution might sometimes be smaller than $\hat{\rho}$, both the expected cost as well as the standard deviation are larger, hence bad solutions are not robust. For this specific model Buhmann et al. proved that SIM outperforms the other methods with high probability. In the following, we extend their model using more realistic distributions for good and bad solutions,
and experimentally evaluate for each of the methods proposed in Section 6.1 how often they choose a good solution. Notice that the original model uses multiplicative approximation while we will use additive approximation.

**Model** Like Buhmann et al. [18] we assume that the set of solutions \( S = \{1, \ldots, n\} \) is the disjoint union of a set \( G = \{1, \ldots, g\} \) of \( g \) good (i.e., robust) solutions and a set \( B = \{g + 1, \ldots, n\} \) of \( b \) bad (i.e., non-robust) solutions. An instance \( I \) is represented by a vector of length \( n = g + b \) where the \( i \)-th entry stores the cost of the solution \( i \) (i.e., the first \( g \) positions store the costs of the good solutions while the last \( b = n - g \) positions store the costs of the bad solutions). Since in this special model there is a one-to-one correspondence between an instance and its cost vector, for simplicity we will use the term “instance” instead of “cost vector of an instance”. For generating an instance \( I \), the generation process proposed in [18] is modified as follows:

1) the first \( g \) values of \( I \) are chosen at random from some (fixed, unknown to the algorithm) probability distribution \( F_G \);

2) the remaining \( b \) values of \( I \) are chosen at random from some (fixed, unknown to the algorithm) probability distribution \( F_B \).

Of course, \( F_G \) and \( F_B \) have to be chosen such that good solutions are superior to bad ones, e.g. because they have a smaller expected cost or a smaller variance (as for the aforementioned “prototypic example”). Now, similar to [18] our goal is the following: given \( S \) and two instances \( I_1, I_2 \) generated by \( \mathcal{P}\mathcal{G} \), identify at least one solution from \( G \).

Observe that, depending on \( F_G \) and \( F_B \), not every subset of \( S \) might be feasible. However, in the examples that are discussed in the following this problem occurs only in the case of the discrete noise model. Therefore we simply computed the value \( \gamma^* \) that maximizes the right hand side of (25) (where every occurrence of \( A_\rho(I_k) \) has to be replaced by \( A_+^\gamma(I_k) \) since we are using additive approximation), and return the set \( S_{\text{SIM}} = A_+^\gamma(I_1) \cap A_+^\gamma(I_2) \).

Each algorithm \( A \in \{\text{JM, MR, SIM}\} \) identifies a set \( S_A \subseteq S \) of candidates for solutions in \( G \), and since we assume that a solution from \( S_A \) is picked uniformly at random, we define the *success probability* of \( A \) with input \( I_1 \) and \( I_2 \) as

\[
P_A(I_1, I_2) = \frac{|S_A \cap G|}{|S_A|} \quad \text{for} \quad A \in \{\text{JM, MR, SIM}\}.
\]

Since GSIM chooses every solution \( s \in S \) with probability proportional to
where \( \beta^* \) is the largest value \( \beta \) that maximizes (28). Notice that the sums are computed over different sets of solutions in the numerator and denominator.

Notice that unlike in the experiments on the public transportation networks, here we do not test the success of each method on some test instance, but instead measure how often a good solution is selected. This simplification is reasonable because of our assumption that good solutions perform reasonably well for all instances.

We want to investigate how the success probabilities of the algorithms proposed in Section 6.1 evolve with increasing noise. We model this by defining a set of noise levels \( N \) (the concrete definition depends on the type of the noise). For a fixed noise level \( N \in N \), we randomly generate an instance as follows. Good solutions are drawn from a distribution with fixed mean \( \mu_G \) and fixed standard deviation \( \sigma_G \). Bad solutions are drawn from a distribution with mean \( \mu_B(N) \) and standard deviation \( \sigma_B(N) \). The distributions of the bad solutions are chosen in a way such that for every two noise levels \( N, N' \in N \) with \( N' > N \), we have \( \mu_B(N') < \mu_B(N) \) or \( \sigma_B(N') > \sigma_B(N) \) (i.e., a higher noise implies either a smaller expected cost, or a higher standard deviation, or both). The concrete choices of \( g, b \), the distributions and the set of noise levels, and the behavior that they exhibit are discussed in the next paragraphs.

Now, for each noise level \( N \in N \), we perform the following experiment: \( \Phi \) generates \( R = 1000 \) instance pairs \( (I_1^k, I_2^k)_{k \in \{1, \ldots, R\}} \) with noise level \( N \) (1000 repetitions turned out to be enough to exhibit the behaviors of the methods), and for each of these instance pairs we compute \( P_A(I_1^k, I_2^k) \) where \( A \in \{JM, MR, SIM, GSIM\} \). After that we set

\[
P_A(N) := \frac{1}{R} \sum_{k=1}^{R} P_A(I_1^k, I_2^k)
\]

(31)

to estimate the average success probability of the proposed methods in dependency of the noise level \( N \). Unless otherwise stated, \( S \) contains \( n = 1000 \) solutions.

**Discrete Noise Model** We start with the investigation of the discrete noise model proposed by Buhmann et al. [18]. In that context it is especially interesting to see how GSIM compares to the other methods. Good solutions always have cost \( \hat{\rho} = 1.25 \). Bad solutions either have cost 1, or cost 2. Our noise
Experiments on the Two-Instance Scenario

Figure 25  Experimental results for the discrete noise model. Notice that JM and MR are equivalent in this model; hence their success rates coincide.

levels should reflect the probability that a bad solution has cost 1. We define \( \mathcal{N} = \{0, 0.02, 0.04, \ldots, 0.74\} \), and the probability that a bad solution has cost 1 as \( N \). The upper bound of 0.74 seems arbitrary. However, as already proven in [18], the expected cost of a bad solution is \( \mu_B(N) = 1 \cdot N + 2 \cdot (1 - N) = 2 - N \). Since \( \mathcal{N} \) is discretized at steps of 0.02, 0.74 is the largest possible value for which the expected cost of a bad solution lies above the cost of a good one. Notice that for \( N = 0.74 \) the expected cost of a bad solution is only slightly higher than the cost of good solution, but in nearly 3/4 of the cases a bad solution has a smaller cost. Whether such solutions should be called “bad” is clearly debatable. The set \( \mathcal{N} \) contains such large noise levels only for completeness.

With the noise levels in \( \mathcal{N} \) we observe that 1) a higher noise \( N \) leads to a smaller expected cost, and 2) we have \( \mu_B(N) > \hat{\rho} \) for every noise level \( N \in \mathcal{N} \) (this explains why the largest noise level in \( \mathcal{N} \) is below \( 2 - \hat{\rho} \)). Since the variance of the cost of a bad solution is \( N - N^2 \), a higher noise also implies a higher standard deviation, at least when \( N \leq 0.5 \). We assume that 10% of all solutions are good. Figure 25 shows the experimental results. As proven in [18], SIM outperforms JM and MR in this discrete model. Notice that for this special model, JM and MR coincide [18]. Our experiments here indicate that GSIM outperforms JM and it even outperforms SIM when the noise is high enough. For smaller noise levels, SIM performs significantly better than any of the other approaches.

**Gaussian-Distributed Noise** Since the discrete noise model is rather artificial, we now study a Gaussian noise model. Good solutions are drawn from a Gaussian distribution with \( \mu_G = 1 \) and \( \sigma_G = 1 \). Now we would like the noise levels to reflect the standard deviation of a bad solution. Therefore we choose \( \mathcal{N} = \{1, 1.5, 2, 2.15, 2.3, 2.45, \ldots, 8.75\} \) (as our experiments will show, for small noise levels all methods perform well, hence a broader discretization
Figure 26  Experimental results for the Gaussian noise model where 5% (a), 10% (b), 20% (c) and 50% (d) of the solutions are good.

Figure 27  Experimental results for the Gaussian noise model with 10% good solutions over $n = 10000$ solutions (a), and over $n = 100000$ solutions (b). For (b), only $R = 100$ repetitions were performed.
is sufficient there). For each noise level \(N \in \mathcal{N}\), bad solutions are drawn from a Gaussian distribution with \(\mu_B(N) = 10\) and \(\sigma_B(N) = N\). As before, 10\% of all solutions are good. Figure 26 shows the experimental results. As in the case of the discrete noise model, we see that SIM outperforms both JM and MR. Also, observe that GSIM outperforms the latter two when the noise is high enough. Further experiments in which 20\%, 30\% and 50\% of the solutions are good indicate that GSIM works especially well when both the solution space as well as the fraction of good solutions over all solutions are large enough, and it even outperforms SIM. For optimization problems, however, we usually have a huge solution space but only relatively few good solutions. One may wonder whether considering \(n = 1000\) solutions is enough, because real optimization problems usually have much larger solution spaces. Figure 27 shows the results for \(n = 10000\) and \(n = 100000\), respectively. We see that the behavior of all methods is similar to the one shown in Figure 26, and we especially observe that SIM and GSIM are best-suited for such a situation while JM and MR perform substantially worse.

**Laplace-Distributed Noise** We now present some experimental results for Laplace-distributed noise. The distribution is somewhat similar to a Gaussian distribution but due to its explicit CDF it is potentially easier to analyze. Good solutions are drawn with mean \(\mu_G = 1\) and standard deviation \(\sigma_G = 1\), and we use the same set of noise levels \(\mathcal{N}\) as for the previous experiments. For each noise level \(N \in \mathcal{N}\), bad solutions are drawn with mean \(\mu_B(N) = 10\) and standard deviation \(\sigma_B(N) = N\). Again, 10\% of all solutions are good. Figure 28 shows that the experimental results for the Laplace noise model is comparable to the results that we obtained for Gaussian noise: SIM outperforms JM and MR, GSIM outperforms JM and MR when the noise is high enough, and it even outperforms SIM when both the solutions space and the fraction of good solutions are sufficiently large. For large solution spaces with relatively few good solutions, SIM performs best, although, again, GSIM is quite competitive.

**Exponential-Distributed Noise** To investigate whether an even simpler noise model would still work, we now replace the Laplace distributions by exponential distributions. Good solutions are chosen with parameter \(\lambda = 1\) (i.e., they have \(\mu_G = 1\) and \(\sigma_G = 1\)). The noise levels are defined as \(\mathcal{N} = \{0.1, 0.15, \ldots, 0.95\}\), and bad solutions are chosen with parameter \(\lambda(N) = N\) (i.e., they have \(\mu_B(N) = 1/N\) and \(\sigma_B(N) = 1/N^2\)). Hence a higher noise level implies a smaller mean, but also a smaller standard deviation. Figure 29 shows the experimental results when 10\% of all solutions are good: JM, SIM and MR perform nearly identical, and they outperform GSIM.

**Shortest Alternative Path Problem** In this paragraph we study an extremely simplified version of the shortest path problem, and show by an (arti-
Approximation Set-Based Approaches: Models and Extensions

Figure 28 Experimental results for the Laplace noise model where 5% (a), 10% (b), 20% (c) and 50% (d) of the solutions are good. They are comparable to the ones for the Gaussian noise model.

Figure 29 Experimental results for the exponential noise model. Observe that JM, MR and SIM perform nearly equally well.
6.2 Experiments on the Two-Instance Scenario

An example that SIM sometimes does not compute a robust solution. Let $G = (V, E, l)$ be a weighted directed graph consisting of exactly $n$ directed $dt$-paths $\pi_1, \ldots, \pi_n$ where $\pi_i \cap \pi_j = \{d, t\}$ for every $i, j \in \{1, \ldots, n\}, i \neq j$. Let $l(\pi_i)$ be the length of the path $\pi_i$. We consider the problem of finding a shortest $dt$-path among $\pi_1, \ldots, \pi_n$, hence the set of feasible solutions can be defined as $S = \{\pi_1, \ldots, \pi_n\}$. The obvious difference to the classical shortest path problem is that any two $dt$-paths in $G$ do not cross. Hence, unlike for the classical shortest path problem, every set $X \subseteq S$ is a feasible approximation set. Notice that in this paragraph multiplicative approximation is used.

Lemma 8. In the shortest alternative path problem as defined above, every subset $X \subseteq S$ is a feasible approximation set.

Proof. Let $\rho \geq 1$ be arbitrary. We define $l(\pi) := 1$ for every $\pi \in X$ and $l(\pi) := \rho + 1$ for every $\pi \in S \setminus X$. For $G = (V, E, l)$ we have $A_\rho(G) = X$.

We will now construct two graphs $G_n$ and $G'_n$ having a high similarity (eq. (25)) for which SIM does not compute a robust solution with high probability. Let $k \geq 36$ be even and such that $\sqrt{k} \in \mathbb{N}$. Let $n \in \mathbb{N}$ be larger than $k$.

The graphs $G_n$ and $G'_n$ consist of exactly $n$ $d$-paths:

- Robust paths $\pi_1^R, \ldots, \pi_{\sqrt{k}}^R$ have length 2 both in $G_n$ as well as in $G'_n$.
- $G_n$-optimal paths $\pi_1, \ldots, \pi_{(k-\sqrt{k})/2}$ have length 1 in $G_n$, and length $2^n$ in $G'_n$.
- $G'_n$-optimal paths $\pi'_1, \ldots, \pi'_{(k-\sqrt{k})/2}$ have length $2^n$ in $G_n$, and length 1 in $G'_n$.
- The remaining $n - k$ paths have length $2^{2n}$ in both graphs.

Obviously, when only $G_n$ and $G'_n$ are given, we want to select only robust paths, because they have the smallest mean and variance 0.

Lemma 9. Let $G_n$ and $G'_n$ be the graphs as defined above. Then, $S_{\rho^*}(G_n, G'_n) = n/k$, and $S_\rho(G_n, G'_n)$ is maximized for $\rho^* = 2^n$.

Proof. Since $G_n$ and $G'_n$ contain only the edge weights 1, 2, $2^n$ and $2^{2n}$, and since the optimum solution has cost 1 both in $G_n$ as well as in $G'_n$, the similarity $S_\rho(G_n, G'_n)$ might only change at $\rho_1, \ldots, \rho_4$ where

$$\rho_1 = 1, \rho_2 = 2, \rho_3 = 2^n, \rho_4 = 2^{2n}.$$ (32)

Since $A_{\rho_1}(G_n) \cap A_{\rho_1}(G'_n) = \emptyset$ we have $S_{\rho_1}(G_n, G'_n) = 0$. 
Figure 30  The graphs $G_n$ and $G'_n$. The numbers next to each path are the lengths of these paths in $G_n$ (first number) and $G'_n$ (second number). The blue area contains the robust paths.

For $\rho_2$, the intersection of the approximation sets contains exactly the robust paths $\pi_1^R, \ldots, \pi_{\sqrt{k}}^R$. $A_{\rho_2}(G_n)$ additionally contains the $G_n$-optimal paths $\pi_1, \ldots, \pi_{(k-\sqrt{k})/2}$ while $A_{\rho_2}(G'_n)$ additionally contains the $G'_n$-optimal paths $\pi'_1, \ldots, \pi'_{(k-\sqrt{k})/2}$. Thus, we have

$$|A_{\rho_2}(G_n)| = |A_{\rho_2}(G'_n)| = \sqrt{k} + (k - \sqrt{k})/2 = (k + \sqrt{k})/2,$$

and for the similarity at value $\rho_2$ we have

$$S_{\rho_2}(G_n, G'_n) = \frac{n\sqrt{k}}{((k + \sqrt{k})/2)^2} = \frac{4n\sqrt{k}}{(\sqrt{k}(\sqrt{k} + 1))^2} < \frac{4n}{k^{3/2}}. \quad (34)$$

For $\rho_3$, both $\rho_3$-approximation sets contain the robust paths as well as the $G_n$-optimal paths and the $G'_n$-optimal paths. Therefore we have

$$|A_{\rho_2}(G_n)| = |A_{\rho_2}(G'_n)| = k,$$

and the similarity at value $\rho_3$ is exactly $S_{\rho_3}(G_n, G'_n) = n/k$ which is larger than $4n/(k^{3/2})$ because we chose $k \geq 36$. Therefore we have $S_{\rho_3} > S_{\rho_2}$.

For $\rho_4$, every path is contained in both approximation sets, and hence we obtain $S_{\rho_4}(G_n, G'_n) = n^2/n^2 = 1$ which is smaller than $S_{\rho_3}(G_n, G'_n)$ since $n > k$. Therefore the similarity is maximized at $\rho^* = \rho_3 = 2^n$, and its value is exactly $n/k$.

Lemma 10. Let $G_n$ and $G'_n$ be the input for SIM applied to the shortest alternative path problem. The average cost of a solution computed by SIM is at least $(1 - (1/\sqrt{k})) \cdot 2^{n-1}$. The probability that SIM chooses a robust solution is $P_{SIM} = 1/\sqrt{k}$. 
6.3 Expected Similarity

Proof. By Lemma 9, the similarity $S_{\rho^*}(G_n, G'_n)$ is maximized at $\rho^* = 2^n$. The intersection of the $\rho^*$-approximation sets contains $k$ paths, but only $\sqrt{k}$ are robust. Hence we obtain $P_{SIM} = \sqrt{k}/k = 1/\sqrt{k}$. Furthermore we see that robust solutions have an average cost of 2 while the other solutions in the intersection have cost $(2^n + 1)/2$. Since we have $k - \sqrt{k}$ many of these other paths, the average cost of a solution is

$$\frac{1}{k} \left( \sqrt{k} \cdot 2 + (k - \sqrt{k}) \cdot (2^n + 1)/2 \right) \geq \frac{1}{k} \left( k - \sqrt{k} \right) \cdot (2^{n-1}) = \left( 1 - \frac{1}{\sqrt{k}} \right) \cdot 2^{n-1},$$

which finalizes the proof.

Now, if we choose $k = \sqrt{n}$ and apply the previous two lemmata, we obtain two instances $G_n$ and $G'_n$ which have $n^{1/4}$ common robust solutions, a similarity $S_{\rho^*}(G_n, G'_n) \to \infty$ for $n \to \infty$, and a probability to pick a robust solution $P_{SIM} \to 0$ for $n \to \infty$. Both JM as well as FI always choose a robust solution. This certainly is disappointing, especially because according to Buhmann et al. [18] a high similarity usually implies that the fraction of robust solutions in the intersection over all solutions in the intersection is large, which in this specific example is not the case.

On the other hand, we did not specify how the generator looks like. If, for example, $\mathcal{I}_{\mathcal{P}\mathcal{E}}$ contains only $G_n$ and $G'_n$, then more than half of the solutions in the intersection are a good approximation for any instance. Also, notice that the worst solutions with cost $2^{2n}$ are never contained in the intersection. Therefore, the example in this paragraph does not at all imply that SIM is useless. However, it should be considered as a warning example that in some cases FI or JM might give much better results than SIM, and that a high absolute value of $S_{\rho^*}(I_1, I_2)$ alone does not necessarily imply that most solutions in the intersection are robust in the common sense.

6.3 Expected Similarity

One of our main assumptions made in Section 6.1 was that the noise generating process of $\mathcal{P}\mathcal{E}$ is unknown to the predicting algorithms. As a first step towards a formal analysis of the model discussed in the previous section we investigate how the similarity (25) behaves in expectation (where the expectation is computed over all pairs of instances generated by $\mathcal{P}\mathcal{E}$), i.e. we analyze the function $S_{\gamma}^{\text{EXP}} = \mathbb{E}_{I_1, I_2 \sim \mathcal{I}_{\mathcal{P}\mathcal{E}}} S_{\gamma}(I_1, I_2)$. 

$$S_{\gamma}^{\text{EXP}} = \mathbb{E}_{I_1, I_2 \sim \mathcal{I}_{\mathcal{P}\mathcal{E}}} S_{\gamma}(I_1, I_2).$$
Approximation Set-Based Approaches: Models and Extensions

As in our previous experiments, we use additive approximation. For simplicity we introduce the calibrating assumption that the minimum solutions of both instances $I_1$ and $I_2$ have the same cost. Notice that this does not imply that the minimum solutions itself are the same. Now we obtain the following result.

**Theorem 11.** Let $\gamma > 0$, $V = |A^+_\gamma(I_1) \cap A^+_\gamma(I_2)|$, $W = |A^+_\gamma(I_1) \cdot |A^+_\gamma(I_2)|$, and $F_G(x)$ and $F_B(x)$ denote the cumulative density functions of the good and the bad solutions, respectively. Then, the expected similarity (38) can be approximated by the estimated similarity (ESIM)

\[
\mathbb{E}\left[\frac{V}{W}\right] \approx \frac{\mathbb{E}[V]}{\mathbb{E}[W]} - \frac{\text{Cov}(V,W)}{\mathbb{E}[W]^2} + \frac{\text{Var}[W] \cdot \mathbb{E}[V]}{\mathbb{E}[W]^3} =: S^E_\gamma,
\]

where

\begin{align}
\mathbb{E}[V] &= gF_G^2 + bF_B^2, \\
\mathbb{E}[W] &= (gF_G + bF_B)^2, \\
\text{Cov}(V,W) &= gF_G^2(1 - F_G^2) + 2g(g - 1)F_G^3(1 - F_G) \\
&\quad + 2gbF_G^2F_B(1 - F_G) + 2gbF_G^2F_B(1 - F_B) \\
&\quad + bF_B^2(1 - F_B^2) + 2b(b - 1)F_B^3(1 - F_B), \quad \text{and} \\
\text{Var}[W] &= g^2F_G^2(1 - F_G^2) + 2g^2(g - 1)F_G^3(1 - F_G) \\
&\quad + 2gb(b - 1)F_GF_B^2(1 - F_G) \\
&\quad + 2g(g - 1)bF_G^2F_B(1 - F_B) \\
&\quad + 2gbF_GF_B(1 - F_GF_B) \\
&\quad + b^2F_B^2(1 - F_B^2) + 2b^2(b - 1)F_B^3(1 - F_B) \\
&\quad + 4g^2bF_GF_B^2(1 - F_G) + 4gb^2F_GF_B^2(1 - F_B).
\end{align}

**Proof.** To make this rather technical proof more readable, we break it down into several steps.

1) **Preliminaries.** Let $m = \min_{s \in S} c(s, I_1) = \min_{s \in S} c(s, I_2)$. Let $s_1, \ldots, s_g$ denote the solutions in $G$ and $\bar{s}_1, \ldots, \bar{s}_b$ denote the solutions in $B$. We define

\[
X_{i,\gamma} = 1\{c(s_i, I_1) \leq m + \gamma\}, \quad 1 \leq i \leq g \\
X_{i,\gamma}'' = 1\{c(s_i, I_2) \leq m + \gamma\}, \quad 1 \leq i \leq g \\
Y_{j,\gamma}' = 1\{c(\bar{s}_j, I_1) \leq m + \gamma\}, \quad 1 \leq j \leq b \\
Y_{j,\gamma}'' = 1\{c(\bar{s}_j, I_2) \leq m + \gamma\}, \quad 1 \leq j \leq b.
\]
Now the components of the similarity (25) can be expressed as

\[|A_\gamma^+(I_1) \cap A_\gamma^+(J_2)| = \sum_{i=1}^g X'_{i,\gamma} X''_{i,\gamma} + \sum_{j=1}^b Y'_{j,\gamma} Y''_{j,\gamma},\]

\[|A_\gamma^+(I_1)| = \sum_{i=1}^g X'_{i,\gamma} + \sum_{j=1}^b Y'_{j,\gamma},\]

\[|A_\gamma^+(J_2)| = \sum_{i=1}^g X''_{i,\gamma} + \sum_{j=1}^b Y''_{j,\gamma}.
\]

For the rest of this proof we will simplify the notation as follows: 1) \(\gamma\) is omitted in the subscript because we can assume it to be the same throughout all considerations, 2) the limits in the sums are omitted; for good solutions we always sum up to \(g\) and for bad solutions to \(b\), and 3) by \(F_G\) and \(F_B\) we denote the cumulative density functions of good and bad distributions, respectively, evaluated at \(m + \gamma\):

\[F_G = F_G(m + \gamma), \quad F_B = F_B(m + \gamma).\] (44)

Observe that 1) \(E[X'_i] = E[X''_i] = F_G\) and \(E[Y'_j] = E[Y''_j] = F_B\), 2) the random variables in \(\{X'_i\}_i \cup \{X''_i\}_i \cup \{Y'_k\}_k \cup \{Y''_\ell\}_\ell\) are jointly independent, and 3) \((X'_i)^2 = X'_i, (X''_i)^2 = X''_i, (Y'_j)^2 = Y'_j\) and \((Y''_j)^2 = Y''_j\) because these are indicators. Also, remember that for jointly independent indicator random variables \(Z_1, Z_2, Z_3\) with \(E[Z_i] = z_i\) we have

\[\text{Cov}(Z_1, Z_2) = z_1z_2(1 - z_1z_2)\] (45)

\[\text{Cov}(Z_1 Z_2, Z_1 Z_3) = z_1 z_2 z_3 (1 - z_1)\] (46)

2) \textit{Taylor expansion of the expected similarity.} A second-order Taylor approximation of \(E[V/W]\) gives

\[
E \left[ \frac{V}{W} \right] \approx \frac{E[V]}{E[W]} - \frac{\text{Cov}(V, W)}{E[W]^2} + \frac{\text{Var}[W] \cdot E[V]}{E[W]^3}.
\] (39)

Remember that \(V\) denotes the size of the intersection while \(W\) is the product of the approximation set sizes. In the following, we will analyze each term of (39) separately.

3) \textit{Expected values of \(V\) and \(W\).}

\[E[V] = \sum_i E[X'_i] \cdot E[X''_i] + \sum_j E[Y'_j] \cdot E[Y''_j] = gF_G^2 + bF_B^2.\] (40)
Taking the independence of the random variables into account, for $\mathbb{E}[W]$ we obtain

$$\mathbb{E}[W] = \mathbb{E}\left[\sum_i X'_i + \sum_j Y'_j \right] \cdot \mathbb{E}\left[\sum_i X''_i + \sum_j Y''_j \right] = (gF_G + bF_B)^2.$$  \hspace{1cm} (41)

4) Analyzing the covariance of $V$ and $W$. Remember that

\begin{align*}
V &= \sum_i X'_i X''_i + \sum_i Y'_i Y''_i, \hspace{1cm} (47) \\
W &= \sum_{j,k} X'_j X''_k + \sum_{i,j} X'_i Y''_k + \sum_{j,k} Y'_j X''_k + \sum_{j,k} Y'_j Y''_k, \hspace{1cm} (48)
\end{align*}

hence

\begin{align*}
\text{Cov}(V, W) &= \sum_{i,j,k} \text{Cov}(X'_i X''_i, X'_j X''_k) + \sum_{i,j,k} \text{Cov}(X'_i X''_i, X'_j Y''_k) \\
&\quad + \sum_{i,j,k} \text{Cov}(X'_i X''_i, Y'_j X''_k) + \sum_{i,j,k} \text{Cov}(X'_i X''_i, Y'_j Y''_k) \\
&\quad + \sum_{i,j,k} \text{Cov}(Y'_i Y''_i, X'_j X''_k) + \sum_{i,j,k} \text{Cov}(Y'_i Y''_i, X'_j Y''_k) \\
&\quad + \sum_{i,j,k} \text{Cov}(Y'_i Y''_i, Y'_j X''_k) + \sum_{i,j,k} \text{Cov}(Y'_i Y''_i, Y'_j Y''_k)
\end{align*}

We will now analyze each of the single terms.

- In the first term $\sum_{i,j,k} \text{Cov}(X'_i X''_i, X'_j X''_k)$ only the summands with $j = i$ or $k = i$ are non-zero, hence we obtain

\begin{align*}
\sum_{i,j,k} \text{Cov}(X'_i X''_i, X'_j X''_k) &= \sum_i \text{Cov}(X'_i X''_i, X'_i X''_i) \\
&\quad + \sum_{i\neq j} \left[ \text{Cov}(X'_i X''_i, X'_i X''_j) + \text{Cov}(X'_i X''_i, X'_j X''_i) \right] \\
&= \sum_i \text{Cov}(X'_i X''_i, X'_i X''_i) + 2 \sum_{i\neq j} \text{Cov}(X'_i X''_i, X'_i X''_j),
\end{align*}

where the last equality holds due to (45) and (46).

- The next two terms $\sum_{i,j} \text{Cov}(X'_i X''_i, X'_j Y''_k)$ and $\sum_{i,j} \text{Cov}(X'_i X''_i, Y'_j X''_k)$
are equal to each other (due to the symmetry of $I_1$ and $I_2$), so their sum resolves to

$$2 \sum_{i,k} \text{Cov}(X'_i X''_i, X'_i Y''_k) \overset{(46)}{=} 2gbF_G^2F_B(1 - F_G). \quad (50)$$

- The next two terms $\sum \text{Cov}(X'_i X''_i, Y'_j Y''_k)$ and $\sum \text{Cov}(Y'_i Y''_i, X'_j X''_k)$ are both zero due to the independence of $X'_i X''_i$ and $Y'_j Y''_k$.

- The next two terms $\sum \text{Cov}(Y'_i Y''_i, X'_j X''_k)$ and $\sum \text{Cov}(Y'_i Y''_i, Y'_j X''_k)$ can be computed in exactly the same way as (50) where both $F_G$ and $F_B$ as well as $g$ and $b$ are interchanged. Hence, their sum equals

$$2 \sum_{i,k} \text{Cov}(Y'_i Y''_i, Y'_j X''_k) \overset{(46)}{=} 2gbF_G^2F_B(1 - F_B).$$

- The last term $\sum \text{Cov}(Y'_i Y''_i, Y'_j Y''_k)$ is computed similar as (49), performing the above-mentioned replacements, hence

$$\sum_{i,j,k} \text{Cov}(Y'_i Y''_i, Y'_j Y''_k) = bF_B^2(1 - F_B^2) + 2b(b - 1)F_B^2(1 - F_B).$$

5) Analyzing the variance of $W$. Finally we compute $\text{Var}[W] = \text{Cov}(W, W)$. When $W$ is expressed as (48), we obtain

$$\text{Cov}(W, W) = \sum_{i,j,k,\ell} \text{Cov}(X'_i X''_j, X''_k X''_{\ell}) + \sum_{i,j,\ell} \text{Cov}(X'_i Y''_j, X''_k Y''_{\ell})$$

$$+ \sum_{i,j,k,\ell} \text{Cov}(Y'_i X''_j, X''_k X''_{\ell}) + \sum_{i,j,k,\ell} \text{Cov}(Y'_i Y''_j, Y''_k Y''_{\ell})$$

$$+ 2 \sum_{i,j,k,\ell} \text{Cov}(X'_i X''_j, X''_k Y''_{\ell}) + 2 \sum_{i,j,k,\ell} \text{Cov}(X'_i Y''_j, Y''_k Y''_{\ell})$$

$$+ 2 \sum_{i,j,k,\ell} \text{Cov}(X'_i X''_j, Y''_k Y''_{\ell}) + 2 \sum_{i,j,k,\ell} \text{Cov}(X'_i Y''_j, Y''_k Y''_{\ell})$$

$$+ 2 \sum_{i,j,k,\ell} \text{Cov}(X'_i Y''_j, Y''_k Y''_{\ell}) + 2 \sum_{i,j,k,\ell} \text{Cov}(Y'_i X''_j, Y''_k Y''_{\ell}).$$

As before we analyze each of these terms separately.
• The first term \( \sum \text{Cov}(X'_i X''_j, X'_k X''_l) \) can be expressed as
\[
\sum_i \text{Cov}(X'_i X''_i, X'_i X''_i) + 4 \sum_{i \neq j} \text{Cov}(X'_i X''_j, X'_i X''_j) \\
+ 2 \sum_{i \neq j, i \neq k, j \neq k} \text{Cov}(X'_i X''_j, X'_i X''_k) + \sum_{i \neq j} \text{Cov}(X'_i X''_j, X'_i X''_j)
\]
where
\[
\sum_i \text{Cov}(X'_i X''_i, X'_i X''_i) = gF_G^2(1 - F_G^2),
4 \sum_{i \neq j} \text{Cov}(X'_i X''_j, X'_i X''_j) = 4g(g - 1)F_G^3(1 - F_G),
2 \sum_{i \neq j, i \neq k, j \neq k} \text{Cov}(X'_i X''_j, X'_i X''_k) = 2g(g - 1)(g - 2)F_G^3(1 - F_G),
\sum_{i \neq j} \text{Cov}(X'_i X''_j, X'_i X''_j) = g(g - 1)F_G^2(1 - F_G),
\]
and therefore
\[
\sum_{i, j, k, \ell} \text{Cov}(X'_i X''_j, X'_k X''_\ell) = gF_G^2(1 - F_G^2) + 2g^2(g - 1)F_G^3(1 - F_G). \quad (51)
\]

• The next two terms \( \sum \text{Cov}(X'_i Y''_j, X'_k Y''_\ell) \) and \( \sum \text{Cov}(Y'_i X''_j, Y'_k X''_\ell) \) are equal due to the symmetry in instances, hence their sum equals
\[
2 \sum_{i \neq k} \text{Cov}(X'_i Y''_j, X'_k Y''_\ell) + 2 \sum_{i \neq k} \text{Cov}(X'_j Y''_i, X'_k Y''_i) \\
+ 2 \sum_{i, j} \text{Cov}(X'_i Y''_j, X'_j Y''_i),
\]
where the the terms are computed as
\[
2 \sum_{i \neq k} \text{Cov}(X'_i Y''_j, X'_k Y''_\ell) \overset{(46)}{=} 2gb(b - 1)F_GF_B^2(1 - F_G),
2 \sum_{i \neq k} \text{Cov}(X'_i Y''_j, X'_k Y''_i) \overset{(46)}{=} 2g(g - 1)bF_G^2F_B(1 - F_B),
\]
\[2 \sum_{i,j} \text{Cov}(X'_iY''_j, X'_iY''_\ell) \overset{(45)}{=} 2gb\text{F}_G\text{F}_B(1 - \text{F}_G\text{F}_B).\]

- The next term \(\sum \text{Cov}(Y''_iY''_\ell, Y''_iY''_\ell)\) is computed analogically to (51) where good and bad solutions are interchanged, resulting in

\[
\sum_{i,j,k,\ell} \text{Cov}(Y''_iY''_\ell, Y''_kY''_\ell) = b^2F_B^2(1 - F_B^2) + 2b^2(b - 1)F_B^2(1 - F_B).
\]

- The next terms \(2 \sum \text{Cov}(X'_iX''_j, X'_iX''_\ell)\) and \(2 \sum \text{Cov}(X'_iX''_j, Y'_iX''_\ell)\) are equal due to the symmetry of the instances, hence their sum is

\[
4 \sum_{i,j,k,\ell} \text{Cov}(X'_iX''_j, X'_iY''_\ell) = 4 \sum_{i,j,k} \text{Cov}(X'_iX''_j, X'_iY''_\ell) \overset{(46)}{=} 4g^2bF_G^2\text{F}_B(1 - \text{F}_G).
\]

- The next terms \(2 \sum \text{Cov}(X'_iX''_j, Y'_iX''_\ell)\) and \(2 \sum \text{Cov}(Y'_iX''_j, X'_iX''_\ell)\) are both equal to zero due to the independence of \(X'_iX''_j\) and \(X'_iX''_\ell\), and of \(X'_iX''_j\) and \(X'_iX''_\ell\).

- The last terms \(2 \sum \text{Cov}(X'_iY''_j, Y'_iY''_\ell)\) and \(2 \sum \text{Cov}(Y'_iX''_j, Y'_iY''_\ell)\) are equal due to the symmetry the instances hence their sum can be computed analogically to (52) where good and bad solutions are interchanged. Hence, we obtain

\[
4 \sum_{i,j,k,\ell} \text{Cov}(Y'_iX''_j, Y'_iY''_\ell) = 4gb^2\text{F}_G\text{F}_B^2(1 - \text{F}_B).
\]

**Experimental Evaluation** To investigate how well the real and the estimated similarity coincide, we performed an experimental evaluation using Gaussian noise. The setting is similar to the one in Section 6.2 \((g = 100, b = 900,\mu_G = 1, \sigma_G = 1, \mu_B = 10, \sigma_B \in \{1, \ldots, 8.75\}\), only the instance generation was slightly changed for the experiments in this paragraph due to the calibrating assumption: since the minima of both instances have to be sufficiently close to each other, we disregarded each pair for which the minima \(v_1\) and \(v_2\) differed by more than \(\varepsilon = 10^{-4}\), and repeatedly generated a new pair until \(|v_1 - v_2| \leq \varepsilon\). For each such instance pair \((I_1^k, I_1^k)\), we computed \(\gamma^*\) using SIM and ESIM (the estimated similarity (39)), where the latter method was calibrated with \(m = (v_1 + v_2)/2\). As before we computed for both methods the intersection \(A_{\gamma^*}(I_1) \cap A_{\gamma^*}(I_2)\) and evaluated the resulting success probability using (29). Figure 31 (a) shows that for high noise, ESIM has a higher chance to pick a good solution than SIM and MR which is not surprising because it has knowledge.
Approximation Set-Based Approaches: Models and Extensions

Figure 31 Comparison of the success rates of JM and SIM with the one of the estimated similarity (a), and a comparison of the values $\gamma^*$ that SIM, ESIM and MR compute.

about the underlying process.

The weak behavior of ESIM for small noise seems to be more surprising. To understand why this happens, consider Figure 31 (b) which shows the average value of $\gamma^*$ that each method computes. Observe that for $\sigma_B < 2.5$, the average value of $\gamma^*$ that ESIM computes is below the one that MR computes, and since MR computes the smallest $\gamma$ for which the intersection of both $\gamma$-approximation sets is non-empty, ESIM nearly always underestimates $\gamma$. To understand why this happens, we investigate the situation for $\sigma_B = 1$ (low noise) and $\sigma_B = 5$ (moderate noise). For each of the $R = 1000$ experiments, we compared the values of $\gamma^*$ that SIM computes with the ones that ESIM computes. Figure 32 shows the distribution of the points $(\gamma_{SIM}^*, \gamma_{ESIM}^*)$ where a point is red if SIM outperformed ESIM, and green otherwise. We see that for low noise, ESIM nearly always underestimates $\gamma^*$. On the other hand, the choice of $\gamma^*$ for moderate noise is often better than the one by SIM. Hence, it seems that SIM is still too much influenced by the noise in the instances.

Nevertheless we did not yet give a convincing explanation of why for low noise ESIM underestimates $\gamma^*$ so much. Hence as before we generated $R = 1000$ instance pairs $(I_1^k, I_2^k)_{k \in \{1, \ldots, R\}}$, calculated the average similarity $S_{\gamma}^A = \frac{1}{R} \sum_{i=1}^{R} S_{\gamma}(I_1^i, I_2^i)$, and compared it to the estimated similarity (39). We note that we did not compute the average estimated similarity over all instance pairs, but instead calibrated (39) directly using the average minimum cost of the instance pairs, i.e., using $m = \frac{1}{R} \sum_{k=1}^{R} (v^k_1 + v^k_2) / 2$ where $v^k_j = \min_{s \in S} c(s, I^k_j)$. Figure 33 shows the results for $\sigma_B = 1$ and for $\sigma_B = 5$. We see that the estimated similarity matches the average similarity relatively well, especially for larger values of $\gamma$. However, it is important to understand the behavior for
6.3 Expected Similarity

Figure 32 Each point corresponds to the outcome of one experiment, where the x-coordinate denotes the value $\gamma^*$ computed by SIM and the y-coordinate denotes the value $\gamma^*$ computed by ESIM. A point is red if SIM outperformed ESIM, and green otherwise. The experiments were performed for $\mu_B = 1$ (a), and for $\mu_B = 5$ (b).

Figure 33 Average vs. estimated similarity for $\sigma_B = 1$ (a), and for $\sigma_B = 5$ (b).

small values of $\gamma$, especially for $\gamma_0 = 0$.

Let $I_1$ and $I_2$ be two arbitrary instances generated by the process described in section 6.2. If there exists a unique solution $s^\perp$ that simultaneously minimizes $c(s, I_1)$ and $c(s, I_2)$, then $S_0(I_1, I_2) = |S| = g + b$. Otherwise the intersection of the approximation sets is empty, thus $S_0(I_1, I_2) = 0$. For the Gaussian noise model with $g = 100$, $b = 900$, $\mu_G = 1$, $\sigma_G = 1$, $\mu_B = 10$ and $\sigma_B = 1$, if such a solution $s^\perp$ exists, then it is likely to be in $G$ (due to the small standard deviation of solutions in $B$). Therefore the chance that $s^\perp$ exists is roughly $1/g$, hence the estimated similarity at value $\gamma = 0$ is $S_0^E \approx (g + b)/g = 10$ (which Figure 33 (a) also shows). Since we choose the smallest $\gamma^*_E$ maximizing the (estimated) similarity and, as Figure 33 (a) shows, the estimated similar-
ity decreases monotonically for $\sigma_B = 1$, we choose $\gamma^*_E = 0$. But in roughly $1 - 1/g \approx 99\%$ of the cases, the intersection of the $\gamma^*_E$-approximation sets is empty!

For $\sigma_B = 5$, the situation is more difficult to analyze. However, Figure 32 (b) shows that the maximum of the estimated similarity is larger than the estimated similarity at $\gamma = 0$, and it also shows that the values $\gamma$ where the average and the estimated similarity, respectively, are maximized coincide well. Therefore the aforementioned problem of an empty intersection does not occur.

### 6.4 Incorporating Multiple Input Instances

So far we only considered the two-instance scenario. Although two is the smallest number of instances from which information can be extracted, in reality one often has many (noisy) input instances. Hence, the question how the approaches presented in Section 6.1 can be generalized to incorporate multiple input instances $I_1, \ldots, I_M$ generated by $\mathcal{G} \mathcal{G}$, naturally arises. Both JM and MR can be trivially extended to multiple instances.

**An Intersection-Based Similarity Generalization** For the similarity approach, a straightforward generalization is to compute the $\gamma^*_IG$ that maximizes

$$S^{IG}_{\gamma(I_1, \ldots, I_M)} = \frac{|A_{\gamma}(I_1) \cap \cdots \cap A_{\gamma}(I_M)|}{|A_{\gamma}(I_1)| \cdots |A_{\gamma}(I_M)|}. \quad (53)$$

Note that there is no theoretical justification for this generalization. The next generalization, pairwise similarity, allows for a much clearer interpretation.

**A Pairwise Similarity Generalization** Another possible generalization consists in computing the $\gamma^*_PG$ that maximizes an average similarity score, and then choosing a solution that appears in as many $\gamma^*_PG$-approximation sets as possible. More concretely, we divide the $M$ input instances into two (nearly) equally-sized sets $F_1 = \{I_1, \ldots, I_{[M/2]}\}$ and $F_2 = \{I_{[M/2]+1}, \ldots, I_M\}$, compute the similarity over all instance pairs in $F_1 \times F_2$, and take the geometric mean of these values. Hence we compute a value $\gamma^*_PG$ that maximizes

$$S^{PG}_{\gamma(I_1, \ldots, I_M)} = \left( \prod_{j=1}^{[M/2]} \prod_{k=[M/2]+1}^{M} S_{\gamma}(I_j, I_k) \right)^{1/[M/2][M/2]} \quad (54)$$

and compute the set $S_{PG}$ of the solutions appearing in as many $\gamma^*_PG$-approximation sets as possible (there is no guarantee that the intersection of all $\gamma^*_PG$-approximation sets is non-empty), and then sample a solution from $S_{PG}$ uni-
6.4 Incorporating Multiple Input Instances

Incorporating Multiple Input Instances

<table>
<thead>
<tr>
<th>Number of Instances</th>
<th>Success Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0.1</td>
</tr>
<tr>
<td>4</td>
<td>0.2</td>
</tr>
<tr>
<td>5</td>
<td>0.3</td>
</tr>
<tr>
<td>6</td>
<td>0.4</td>
</tr>
<tr>
<td>7</td>
<td>0.5</td>
</tr>
<tr>
<td>8</td>
<td>0.6</td>
</tr>
<tr>
<td>9</td>
<td>0.7</td>
</tr>
<tr>
<td>10</td>
<td>0.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of Instances</th>
<th>Value of $\gamma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>18</td>
</tr>
<tr>
<td>7</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>22</td>
</tr>
</tbody>
</table>

![Figure 34](image)

Figure 34 Success rates of the methods in dependency of the number of input instances (a), and the average value of $\gamma^*$ for the generalized similarity, the estimated similarity and minimax regret in dependency of the number of input instances (b).

Notice that this approach can be interpreted as computing the value $\gamma$ that maximizes the average logarithm of the similarities:

$$
\log S^G_{\gamma}(I_1, \ldots, I_M) = \log \left( \prod_{j=1}^{[M/2]} \prod_{k=[M/2]+1}^{M} S_\gamma(I_j, I_k) \right)^{1/[M/2][M/2]}
$$

$$
= \frac{1}{[M/2][M/2]} \sum_{1 \leq j \leq [M/2]} \log S_\gamma(I_j, I_k)
= \log S_\gamma(I_j, I_k).
$$

(55)

We also studied a generalization of the estimated similarity proposed in the previous section, although it cannot be used in practice (because the parameters of the generator are unknown). Remember that Theorem 11 used the calibrating assumption that both instances have the same minimum $m$. For each experiment in which the instances $I_1, \ldots, I_M$ were given, we computed as before the average minimum over all instances (i.e., $\frac{1}{k} \sum_{i=1}^{k} \min_{s \in S} c(s, I_M)$), and calibrated the ESIM using this value. ESIM then computes a value $\gamma^*_{ESIM}$, and we compute the set of solutions that appear in as many $\gamma^*_{ESIM}$-approximation sets as possible.

For our experiments, we generated $k \in \{2, 3, \ldots, 10\}$ instances using the Gaussian noise model from Section 6.2 (where $g = 100$, $b = 900$, $\mu_G = 1$, $\sigma_G = 1$, $\mu_B = 10$ and $\sigma_B = 8$). Figure 34 (a) shows the success probabilities of the methods in dependency of the number of input instances. As before, each
experiment was repeated $R = 1000$ times. The figure clearly shows that the approximation set-based methods outperform JM, and both the generalized as well as the estimated similarity outperform MR.

Figure 34 (b) shows the average approximation $\gamma^*$ computed by these approaches. Observe that the intersection-based similarity generalization tends to find a substantially broader approximation while even outperforming the pairwise-based generalization. Also, observe that the average $\gamma^*_{ESIM}$ that ESIM computes does not seem to be influenced by the number of instances. This, however, is not very surprising: ESIM is not made for incorporating multiple input instances. Instead, we somewhat reduced our $M$ instances to two by calibrating ESIM using the average minimum. What is a bit more surprising is the fact that ESIM still performs reasonably well for six and more input instances although the intersection of all $\gamma^*_{ESIM}$-approximation sets is likely to be empty. We believe that all these methods are worthwhile to be explored further to give a convincing explanation to these phenomena.

6.5 Conclusion

Achievements We extended the model by Buhmann et al. [18] that partitions the solution space into good and bad solutions to use more realistic probability distributions, and we experimentally showed that for Gaussian and Laplace noise the similarity-based approaches clearly outperform minimax-regret and the joint minimizer, especially for intermediate and high noise. We also saw at least for Gaussian noise that these results are even stronger visible when the number of solutions grows. On the other hand, for exponential noise we couldn’t observe any advantage of the similarity-based approaches. For a very artificial simplification of the shortest path problem and two very special instances of this problem, we even observed that the (0-1-weight-based) similarity approach usually gives a very bad solution (in terms of cost and standard deviation), although the similarity of the input instances is high. Although this was a very constructed example, one should keep in mind that in few cases the similarity-based approaches are clearly worse than the simple methods.

We also gave an analytic expression to estimate the expected similarity score. For that we used the calibrating assumption that the minimum costs of the two instances are equal. An experimental evaluation showed that average real similarity scores and the estimated similarities coincide well, however for finding the value $\gamma^*$ the estimated score worked well only for intermediate and high noise, not for small noise. The reason is that for a small noise and a small $\gamma$, the average similarity is around $1 + b/g$ (where $b$ and $g$ denote the number of good and bad solutions, respectively) while the most likely similarity
is 0 (because the intersection is empty).

Finally we discussed possible generalizations of the aforementioned approach to incorporate more than two input instances. The straightforward intersection-based generalization simply computes the $\gamma$ that maximizes the ratio of the intersection of all approximation sets and the product of the approximation set sizes. The other possibility is to divide the instances into two (nearly) equally-sized sets $F_1$ and $F_2$, and to compute the $\gamma$ that maximizes the geometric mean of the similarities between any two instances $(I_1, I_2) \in F_1 \times F_2$. For both approaches we saw that they outperform both the joint minimizer and minimax-regret, and it seems that the intersection-based generalization is even slightly better than the pairwise-based method.

**Application to Linear Regression** In his B.Sc. thesis [8] which I supervised, A. Binding studied how the approximation-set based methods can be applied to the well-known problem of linear regression. In the simplest (one-dimensional) version of that problem we are given a set of $n$ points (i.e., an instance) $I = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ where it is assumed that $y_i = \alpha_1^* x_i + \alpha_2^* + \varepsilon_i$ for some small random noise $\varepsilon_i$, and the goal is to estimate $\alpha^* = (\alpha_1^*, \alpha_2^*)$. One well-known solution to this problem consists in computing the ordinary least squares estimator $\hat{\alpha} = (\hat{\alpha}_1, \hat{\alpha}_2)$ which minimizes the function $c(\alpha, I) = \sum_{i=1}^{n} (y_i - \alpha_1 x_i - \alpha_2)^2$, and for which an explicit analytical expression is well-known to exist. Hence, we can regard linear regression as an optimization problem where the set of feasible solutions is the set of estimators $S = \mathbb{R}^2$, and for an instance $I$ to goal is to find an $\alpha$ that minimizes $c(\alpha, I)$.

In the two-instance scenario, we assume that the problem generator gives two instances $I_1$ and $I_2$ (as defined above), and for simplicity we assumed that both contain $n$ points. However, we allowed the $x$-coordinates of the points in $I_1$ to differ from those in $I_2$, only the underlying process $y_i = \alpha_1^* x_i + \alpha_2^* + \varepsilon_i$ is the same. The straightforward solution for estimating $\alpha^*$ is, of course, the joint minimizer which corresponds to the OLS estimator of the joint instance $I_1 \cup I_2$. For applying approximation set-based methods, Binding proved that the boundary of an approximation set $A_\rho(I) = \{\alpha \in S \mid c(\alpha, I) \leq c(\hat{\alpha}, I) \cdot \rho\}$ is an ellipse. However, when SIM is applied to linear regression as described above, we are faced with two problems: the solution space $S = \mathbb{R}^2$ is 1) continuous and 2) unbounded. While problem 1) can be easily solved by replacing the cardinalities in (23) by volumes, problem 2) is serious because the expected volume of the intersection of two finite-sized approximation sets (i.e., ellipses) in $\mathbb{R}^2$ is 0, hence (23) is always $\infty$. Therefore Binding concentrated on computing the first non-empty intersection which he showed to be easily computable by solving a convex optimization problem, and he experimentally compared FI to JM. His experiments indicate that for Gaussian noise JM outperforms FI (in
terms of variance), but when the noise \( \epsilon_i \) is drawn from an exponential distribution then FI outperforms JM. At first glance this result seems to contradict the experimental results in Section 6.2 where FI outperformed JM for Gaussian noise, but not for exponential noise. On the other hand, linear regression is a completely different problem, and we don’t have the clear separation of good and bad solutions. An interesting open question in this context is to give a convincing explanation of the observed behavior of linear regression. For more details and the necessary background, consider Binding’s thesis [8].

Shortly before the aforementioned thesis Conde [20] applied minimax regret to linear regression, i.e. he considered additive instead of multiplicative approximation. He assumed to have a (usually infinite) set of scenarios \( s \) in which \( x_1, \ldots, x_n \) are fixed and each \( y_i(s) \) is drawn from an interval \([y_i^-, y_i^+]\), hence \( I_{\Phi \Theta} = \{(x_1, y_1), \ldots, (x_n, y_n)\} \mid \forall i \in \{1, \ldots, n\} : y_i \in [y_i^-, y_i^+]\}. This modeling totally differs from our modeling: While Conde assumes the intervals \([y_i^-, y_i^+]\) and hence also \( I_{\Phi \Theta} \) to be known explicitly, we (and also Binding) assume that \( I_{\Phi \Theta} \) and even the type of the noise are unknown. Conde showed that in this special setting and for least squares regression, MR coincides with the OLS estimator of \( \bar{I} = \{(x_1, \bar{y}_1), \ldots, (x_n, \bar{y}_n)\} \) where \( \bar{y}_i = (y_i^- + y_i^+)/2 \) which here is equivalent to JM.

**Application to Linear Programming**

We also applied SIM to the well-known problem of linear programming which asks, given a matrix \( A \in \mathbb{R}^{(n \times d)} \), a vector \( b \in \mathbb{R}^n \) and a vector \( c \in \mathbb{R}^d \), to compute a solution \( x \in P := \{y \in \mathbb{R}^d \mid Ay \leq b\} \) that maximizes \( c^T x \). Hence, the set of feasible solutions \( S \) is the polyhedron \( P \), an instance is a vector \( I \in \mathbb{R}^d \), and the cost of a solution \( s \) in the instance \( I \) is simply \( c(s, I) = I^T s \). By this modeling, we also see that the set of feasible solutions is the same for every instance. As for the linear regression, we have a continuous solution space (hence we have to replace the cardinalities in (23) by volumes). Here we assume \( P \) to be a polytope such that (unlike for linear regression) the solution space is bounded.

Now, an approximation set \( A_\rho(I) \) is the set of all points \( p \in P \) which satisfy \( c(p, I) \leq \rho \cdot \max_{s \in P} c(s, I) \). Unfortunately, computing volumes of polytopes is known to be \#P-hard [30] (a randomized polynomial-time algorithm exists [44], but it does not seem to be usable for practical applications). Hence, we performed some preliminary experiments on round two-dimensional polytopes, but we were unable to define a noise model on the instances in which GSIM or SIM outperform JM (for GSIM we defined \( S \) to be the vertices of \( P \) to avoid computing volumes, allowing us to perform experiments also in higher dimensions). We don’t have a convincing explanation yet, it seems that only varying the cost function and not the polytope itself is not sufficient. On the other hand, we assumed that the set of feasible solutions remains invariant for
every instance, hence, for varying the polytope we don’t have the theory yet. Therefore we did not pursue this issue further.
7

Concluding Remarks

7.1 Robust Public Transportation

The thesis investigated the problem of finding robust journeys in urban public transportation networks. We developed the solution concept of a route, which simply is a (feasible) sequence of (bus or tram) lines, and proposed algorithms for listing all routes with a bounded number of transfers. We also proposed an algorithm for listing all paths in a transportation network which are induced by some route whose overall number of transfers does not exceed a given threshold. All these listing algorithms have a running time that is polynomial in the sum of the input size and the number of computed solutions (i.e., routes or paths). After listing all routes, we selected the most robust ones using various methods. Simple methods enforce robustness only by adding some additional buffer time at each transfer. More sophisticated learning-based methods use recorded timetables (i.e., historic vehicle schedules of one particular day in the past) and measure the performance of every route with respect to the given recorded timetables. For example, we studied how an approximation-set based approach, originally developed by Buhmann et al. [18] for combinatorial optimization, can be adapted to public transportation: a $\gamma$-approximation set of a given recorded timetable $T$ contains every route $r$ as often as some realization along $r$ exists that departs at most $\gamma$ minutes before the latest allowed arrival time and reaches the target on time. Now, assuming that exactly two recorded timetables $T_1$ and $T_2$ are given, one computes an appropriate value $\gamma^*$ that maximizes the fraction of robust routes in the intersection of the corresponding $\gamma^*$-approximation sets over all routes, and from this intersection chooses the most frequent routes. The method also computes a number that measures how similar the given timetables are. Clearly, the drawback of this approach is that it only considers exactly two recorded timetables where in reality often more data is available. A natural generalization to $k > 2$ recorded timetables consists in finding the smallest $\gamma^*$ such that the intersection of all $k$ approximation sets is non-empty, and of using the route in the intersection. Moreover, we adapted a mean-risk model: for every route, one computes for every recorded
timetable how much one would have to leave in advance to reach the target on time in that particular timetable, and then selects the routes that minimize the weighted sum of the mean and the standard deviation of these times. An experimental evaluation showed that the mean-risk model as well as using the route in the intersection of multiple approximation sets outperform the simple buffer-based strategies w.r.t. the departure time and the arrival rate. The approximation set-based method that uses only two recorded timetables performs relatively similar to the buffer-based methods, but our experiments showed that the similarity measure it computes is indeed meaningful and can help to distinguish typical recorded timetables from the non-typical ones. This might be of further interest, because for any method it does not seem to be useful to learn from non-typical historic data. We also saw that every method benefits from a high mutual similarity between the input and also the test instances. Our experiments showed that the actual time for traveling does not increase when robustness is taken into account. We believe that the reason lies in the existence of good alternative routes of which some are more robust than others, and learning-based methods are able to identify these. Moreover, we saw that buffer-based methods and also the mean-risk model need a nontrivial amount of parameter adjustment, while taking the route from the intersection of multiple approximation sets often leads to an on-time arrival with a reasonable trade-off between the departure time and the arrival rate, and without requiring the specification of any parameters. In the following, we outline some possible directions for future research.

**Walking between nearby stops** So far, we did not consider walking between nearby stops. In reality, however, allowing short walks can be useful to find reasonable journeys with a smaller number of transfers. Integrating walking might not seem to be hard from a conceptual point of view: for example, one could introduce special “walking” lines between nearby stops. However, with such a modification the number of feasible routes will be tremendous, leading to large query times. Hence it is an important open question how walking can be efficiently integrated into our model.

**Joining lines** In our model, every line is a unique sequence of stops. If some other line visits exactly the same stops but in a different order (e.g., in reverse order), we distinguish these two lines. In reality, however, there are many lines that differ only slightly, and that a user would therefore consider to be equivalent. It is definitely a challenging task to join these lines in a systematic way such that the concepts and algorithms presented in this thesis are still applicable.

**Incorporating real-time information** It was shown in [1] that the integration of real-time information improves both the predicted travel times as well as the
7.2 Optimization under Uncertainty

quality of the proposed journeys. Although we did not study explicitly how real-time information can be incorporated into the model, it seems that this might be easily achievable: for every delayed trip, one simply has to update the arrays of the departure times stored at each stop. Since delays in urban networks are relatively small (in comparison to those in train networks), only few entries have to be modified, and updating the timetable should not take too much time. This, however, is pure intuition and has to be investigated further.

Different networks Our experimental evaluation on real-world data from Zürich showed that our algorithms compute journeys within a few milliseconds (less than 55ms for computing the intersection of multiple approximation sets, and the same time in the case of the mean-risk model), even with a relatively simple implementation in JAVA. Although the urban public transportation network of Zürich seems to be typical for such networks of cities in Switzerland, it is relatively small, especially when comparing it to the London dataset that is commonly used for experimental evaluations (e.g., in [24, 26]). It would certainly be interesting to investigate how the model developed in this thesis performs on larger datasets. A bottleneck could be the explicit listing of all feasible routes, especially if the number of allowed transfers is larger than necessary. One possibly would have to use some heuristics to generate meaningful alternative routes. Another interesting problem is to investigate whether the concepts in this thesis can also be applied to other public transportation networks such as train networks. There, the difficulty comes from the fact that trains usually operate at much lower frequency, and that it often helps to compute a new journey once a planned transfer breaks. However, while the solution concept used in this thesis (i.e., routes) seems to work well for urban networks, it is not clear whether the same holds for other networks as well.

7.2 Optimization under Uncertainty

We also investigated the approximation set-based approach by Buhmann et al. [18]. We experimentally identified conditions where this method outperforms simple strategies, but currently no proofs exist. Proving superiority for some simple situations is clearly one of the most important next steps. As a first step towards this, we developed an analytic expression to estimate the average similarity score of two instances. Our experiments showed that the estimated and real scores coincide well. Since the method originally proposed by Buhmann et al. [18] considers only two input instances, we studied generalizations to incorporate multiple instances: one simply relates the size of the intersection of all approximation sets to the product of the approximation set
sizes, and the other partitions the set of the input instances into two (nearly) equally-sized sets $F_1$ and $F_2$, computes the $\gamma^*$ maximizing the geometric mean of the similarities of the instance pairs in $F_1 \times F_2$, and then chooses a solution appearing in as many $\gamma^*$-approximation sets as possible. Although the latter method is more theoretically sound, in our experiments the first method performed slightly better. The reason why this is the case and a theoretical derivation of the first method should be clearly investigated in future research.

### 7.3 Summary of Contributions and Publications

The ideas in this thesis were developed by me, but some ideas were developed jointly with colleagues. In the following I declare who contributed to the development of ideas (“joint work with”). Preliminary versions of parts of this thesis were published before, and these publications were written jointly with colleagues. I will therefore also declare where the corresponding text appeared before, and who contributed to the writeup (“written jointly with”).

**Chapter 1** Parts of Section 1.1 appeared in [11, 12, 13]. Section 1.2 appeared in [10] and was written jointly with Gustavo Sacomoto.

**Chapter 2** The chapter is joint work with Kateřina Böhmová and Matúš Mihalák. A preliminary shorter version appeared in [11, 12, 13]. Section 2.6 was written jointly with Kateřina Böhmová.

**Chapter 3** Section 3.3 is joint work with Kateřina Böhmová, and an early version was written jointly with her. She also created Figure 6. Section 3.6 is joint work with Gustavo Sacomoto and was written jointly with him. A preliminary shorter version of the Chapter (except for Section 3.3) appeared in [10]. A preliminary shorter version of Section 3.3 appeared in [12].

**Chapter 4** A preliminary version of this chapter appeared in [11, 12, 13]. Section 4.4 is joint work with Matúš Mihalák, and the last paragraph (“Properties of the Approach”) of this section was written jointly with him.

**Chapter 5** The chapter is joint work with Kateřina Böhmová and Matúš Mihalák. Kateřina Böhmová also helped with the implementation, but the strong majority of the experiments were implemented by me. A preliminary version of the chapter appeared in [11]. Sections 5.4, 5.6–5.7 and 5.9 were written jointly with Kateřina Böhmová.

**Chapter 6** The chapter is joint work with Joachim M. Buhmann, Alexey Gronskiy, Matúš Mihalák and Rastislav Šrámek. Alexey Gronskiy also helped with the implementation, but the vast majority of the experiments were implemented by me. Theorem 11 goes back solely to Alexey Gronskiy.
Feedback  Matúš Mihalák and Peter Widmayer gave feedback on all of the aforementioned publications [11, 10, 12, 13]. Additionally, Marie-France Sagot gave feedback on [10].


[18] Joachim M. Buhmann, Matúš Mihalák, Rastislav Srámeek, and Peter Widmayer. Robust optimization in the presence of uncertainty. In Innovations


[27] Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Delay-robust journeys in timetable networks with minimum expected arrival time. In 14th


Nomenclature

Mathematical Notation

\(\mathbb{N}\) Set of natural numbers \(1, 2, \ldots\)
\(\mathbb{N}_0\) \(\mathbb{N} \cup \{0\}\)
\(\mathbb{R}\) Set of real-valued numbers
\(\mathbb{R}^+\) Set of strictly positive real-valued numbers
\(\mathbb{R}_0^+\) Set of all non-negative real-valued numbers
\(\delta_{ij}\) 1 if \(i = j\), and 0 otherwise (Kronecker delta)

Public Transportation (Chapter 2–5)

\(\mathcal{S}\) Set of stops
\(\mathcal{L}\) Set of lines (i.e., sequences of stops)
\(\mathcal{R}_{dt}^\beta\) Set of \(dt\)-routes (i.e., a feasible sequence of at most \(\beta\) lines)
\(\mathcal{R}\) Abbreviation for \(\mathcal{R}_{dt}^\beta\) when \(d, t\) and \(\beta\) are clear from the context
\(\mathcal{N}_{\mathcal{S},\mathcal{L}}\) Transportation network induced by \(\mathcal{S}\) and \(\mathcal{L}\), i.e. a graph with vertex set \(\mathcal{S}\) and an arc \((s, s')\) for every stop \(s\) that is visited directly before \(s'\) by some line in \(\mathcal{L}\)
\(\mathcal{N}\) Abbreviation for \(\mathcal{N}_{\mathcal{S},\mathcal{L}}\) when \(\mathcal{S}\) and \(\mathcal{L}\) are clear from the context
\(|\mathcal{N}_{\mathcal{S},\mathcal{L}}|\) \(|\mathcal{S}| + \sum_{l \in \mathcal{L}} |l|\), i.e., the size of the input
\(\mathcal{L}_a\) lines that contain the arc \(a = (s, s')\), i.e., that visit \(s\) directly before \(s'\)
\(\mathcal{L}_s\) lines that visit the stop \(s\) and at least one other stop \(s'\) after \(s\)
\(l_1 \cap l_2\) Set of all common stops of \(l_1\) and \(l_2\)
\(s, s_i, s'\) Some stop
\(\theta, \theta_i\) Some transfer stop
\(d\) Departure stop
\(t\) Target stop
\(t_D\) Departure time
\(t_A\) Arrival time
\(l, l_i, l'\) Some line
Public Transportation (Chapter 2–5)

\[ r \]  Some route
\[ |r| \]  Length of route \( r \), i.e., the number of transfers in \( r \) plus one
\[ \beta \]  Maximal length a route, i.e., the maximum number of transfers plus one
\[ d_L(d,t) \]  Length of a shortest \( dt \)-route
\[ d_{\mathcal{N}}(s,t,l) \]  Length of a shortest \( st \)-route in \( \mathcal{N} \) that starts with \( l \)
\[ \tau \]  A trip
\[ L(\tau) \]  Line associated with the trip \( \tau \)
\[ L^{-1}(l) \]  Set of all trips \( \tau \) where \( L(\tau) = l \)
\[ A(\tau,s) \]  Arrival time of \( \tau \) at \( s \), can be \( +\infty \) if \( \tau \) does not visit \( s \)
\[ D(\tau,s) \]  Departure time of \( \tau \) at \( s \), can be \( -\infty \) if \( \tau \) does not visit \( s \)
\[ T_{\text{planned}} \]  Planned timetable
\[ T_i \]  \( i \)-th recorded timetable
\[ T \]  Some timetable, might be \( T_{\text{planned}} \) or \( T_i \)
\[ \varepsilon(s,l,l') \]  Minimum required transfer time to change at stop \( s \) from line \( l \) to line \( l' \)
\[ \pi_{st} \]  A path from a vertex \( s \) to a vertex \( t \)
\[ \mathcal{N}_G \]  Out-Neighborhood of \( v \), i.e. the set of all vertices \( w \) for which \( G \) contains an arc \((v,w)\)
\[ \Gamma[\mathcal{N}] \]  Auxiliary graph for computing \( d_{\mathcal{N}}(s,t,l) \)
\[ G_I \]  Line incidence graph, contains a vertex for every line and an undirected arc between two vertices \( l_i \) and \( l_j \) if the corresponding lines have at least one common transfer
\[ \nu(l,l',s) \]  Stop that \( l \) visits after \( s \), by which \( l' \) can be reached as early as possible (on \( l' \))
\[ \mathcal{P}_d^\beta \]  Set of all \( dt \)-paths \( \pi_{dt} \) in \( \mathcal{N} \) for which \( \mathcal{R}_d^\beta \) contains a route \( r = \langle l_1, \ldots, l_k \rangle \) such that \( \pi \) is the concatenation of suitable subpaths of \( l_1, \ldots, l_k \)
\[ S - \pi \]  \( S \) from which all stops in the path \( \pi \) have been removed
\[ l - \pi \]  Line \( l \) from which every vertex in \( \pi \) has been removed, union of (possibly degenerate) paths
\[ L - \pi \]  Set of all lines \( l - \pi \)
\[ \gamma \]  Time difference to \( t_A \)
\[ S_{\gamma}(T_1,T_2) \]  Similarity of the timetables \( T_1 \) and \( T_2 \) when \( d \) is left at time \( t_A - \gamma \) or later
\[ \gamma_{\text{OPT}} \]  Value of \( \gamma \) that maximizes \( S_{\gamma}(T_1,T_2) \)
Public Transportation (Chapter 2–5)

\( \delta_i^r \) Minimum required time difference to \( t_A \) such that leaving \( d \) at time \( t_A - \delta_i^r \) and following route \( r \) leads to an on-time arrival at \( t \)

Optimization under Uncertainty (Chapter 6)

\( S \) Set of feasible solutions
\( \mathcal{I} \) Set of possible instances
\( c(s, I) \) Cost of a solution \( s \in S \) in an instance \( I \in \mathcal{I} \)
\( s^\perp \) Solution with minimum cost
\( \mathcal{PG} \) Problem generator
\( \mathcal{I}_{\mathcal{PG}} \) Instances generated by \( \mathcal{PG} \)
\( A_\rho(I) \) (Multiplicative) approximation set containing all solutions whose cost in \( I \) approximates the minimum cost of a solution in \( I \) within a factor \( \rho \)
\( A_\gamma^+(I) \) (Additive) approximation set containing all solutions whose cost in \( I \) lies at most \( \gamma \) above the minimum cost of a solution in \( I \)
\( \mathcal{F}_k \) Set of all feasible approximation sets of size \( k \)
\( w_\rho^\square(s, I) \) 1 if \( s \in A_\rho(I) \), and 0 otherwise
\( w_\beta^G(s, I) \) \( \exp(-\beta c(s, I)) \) (Gibbs weight)
\( S_\rho(I_1, I_2) \) (Multiplicative) similarity of \( I_1 \) and \( I_2 \)
\( S_\gamma(I_1, I_2) \) (Additive) similarity of \( I_1 \) and \( I_2 \)
\( S_\beta(I_1, I_2) \) Similarity of \( I_1 \) and \( I_2 \) using Gibbs weights
\( g \) Number of good solutions
\( b \) Number of bad solutions
\( G \) Set of good solutions
\( B \) Set of bad solutions
\( \mathcal{S}_A \) Set of solutions computed by algorithm \( A \)
\( P_A(I_1, I_2) \) Success probability of the algorithm \( A \) with input \( I_1, I_2 \), i.e., the fraction of good solutions in \( \mathcal{S}_A \)
\( P_A(N) \) Average success probability of algorithm \( A \) at noise level \( N \), i.e., the average of \( P_A(I_1^k, I_2^k) \), where \( I_j^k \) are instances generated at noise level \( N \)
Curriculum Vitae

Tobias Pröger

Personal

Date of birth  September 5, 1987
Place of birth  Schwerte, Germany
Citizenship  German

Education

Diploma in Computer Science  (with honors)  October 2010
Technische Universität Dortmund  Dortmund, Germany
Abitur  June 2006
Ruhrtal-Gymnasium Schwerte  Schwerte, Germany

Publications


Robust Routing in Urban Public Transportation: Evaluating Strategies that Learn From the Past, ATMOS 2015, 68–81, 2015 (with K. Böhmová, M. Mihalák, P. Neubert and P. Widmayer)


On Efficient Implicit OBDD-Based Algorithms for Maximal Matchings, Information and Computation, 239, 29–43, 2014 (with B. Bollig)


An Efficient Implicit OBDD-Based Algorithm for Maximal Matchings, *LATA 2012*, LNCS 7183, 143–154, 2012 (with B. Bollig)