TECHNISCHE UNIVERSITÄT DARMSTADT

ULB

Efficient Algorithms for Intermodal Routing and Monitoring in Travel Information Systems

Gündling, Felix

(2020)

DOI (TUprints): https://doi.org/10.25534/tuprints-00014212

License:

CC-BY 4.0 International - Creative Commons, Attribution

Publication type: Ph.D. Thesis

Division: 20 Department of Computer Science

Original source: https://tuprints.ulb.tu-darmstadt.de/14212

Efficient Algorithms for Intermodal Routing and Monitoring in Travel Information Systems

Zur Erlangung des Grades eines Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigte Dissertation von Felix Johannes Gündling aus Lindenfels
Tag der Einreichung: 10.09.2020, Tag der Prüfung: 22.10.2020

1. Gutachten: Prof. Dr. rer. nat. Karsten Weihe
2. Gutachten: Prof. Dr. rer. nat. Ralf Borndörfer
D17 Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Computer Science
Department

Algorithmik

Efficient Algorithms for Intermodal Routing and Monitoring in Travel Information Systems

accepted doctoral thesis by Felix Johannes Gündling

1. Review: Prof. Dr. rer. nat. Karsten Weihe
2. Review: Prof. Dr. rer. nat. Ralf Borndörfer

Date of submission: 10.09.2020
Date of thesis defense: 22.10.2020

D17 Darmstadt

Bitte zitieren Sie dieses Dokument als:
URN: urn:nbn:de:tuda-tuprints-142127
URL: http://tuprints.ulb.tu-darmstadt.de/14212

Dieses Dokument wird bereitgestellt von tuprints,
E-Publishing-Service der TU Darmstadt
http://tuprints.ulb.tu-darmstadt.de
tuprints@ulb.tu-darmstadt.de

Die Veröffentlichung steht unter folgender Creative Commons Lizenz:
CC-BY 4.0 International – Creative Commons, Attribution
https://creativecommons.org/licenses/by/4.0/

Erklärungen laut Promotionsordnung

§8 Abs. 1 lit. c PromO

Ich versichere hiermit, dass die elektronische Version meiner Dissertation mit der schriftlichen Version übereinstimmt.

§8 Abs. 1 lit. d PromO

Ich versichere hiermit, dass zu einem vorherigen Zeitpunkt noch keine Promotion versucht wurde. In diesem Fall sind nähere Angaben über Zeitpunkt, Hochschule, Dissertationsthema und Ergebnis dieses Versuchs mitzuteilen.

§9 Abs. 1 PromO

Ich versichere hiermit, dass die vorliegende Dissertation selbstständig und nur unter Verwendung der angegebenen Quellen verfasst wurde.

§9 Abs. 2 PromO

Die Arbeit hat bisher noch nicht zu Prüfungszwecken gedient.

Darmstadt, 10.09.2020

F. Gündling

Acknowledgements

First of all, I want to thank my advisor Karsten Weihe for the opportunity to join the Algorithmics group, the support given throughout the years, and the positive environment. I am deeply grateful to Mathias Schnee who supported my research regarding routing algorithms since 2012. I thank Christoph Blendinger from Deutsche Bahn AG for supporting our work and for many fruitful discussions. I want to thank my co-authors Sebastian Fahnenschreiber, Pablo Hoch, Florian Hopp, Mohammad Keyhani, and Tim Witzel. I thank Tobias Raffel for his help with implementing the parser for the HAFAS Rohdaten timetable format in our intermodal routing system. I would like to thank Simon Gündling, Julian Harbarth, Pablo Hoch, Florian Hopp, and Tim Witzel their code implemented as student staff. I thank every student who implemented an experimental algorithm in a thesis I supervised: Sruthi Parvathy Subramanian, Mohan Kanth Dayanandan Jonas Schlitzer, Tim Witzel, Ann-Sophie Hahn, Markus Lehmann, Benedikt Naumann, Fabian Keßler, Pablo Hoch, Sebastian Fahnenschreiber, Marius Kaufmann. I also thank my co-workers Hans-Peter Zorn and Daniel Mäurer for interesting discussions and a productive environment.

Abstract

Millions of people use online journey planning systems. However, most of the currently available systems only support finding optimal journeys for one mode of transportation (i.e. only public transportation, driving by car, etc.). This makes it hard to plan intermodal journeys combining different modes of transportation to reach a destination. In this thesis we present a real-time multi-criteria intermodal travel information system supporting various transportation modes as well as different special use cases such as intermodal routing for people with disabilities and tourist tour planning.

Choosing the perfect parking to switch from private transportation (e.g. bicycle or car) to public transport (e.g. buses, trams, trains, etc.) is basically trial and error when using unimodal planning systems. This problem becomes even harder when the user wants to return to the starting point which is a common use case of intermodal travel. The optimal route for the outward trip may yield a suboptimal or even infeasible return trip and vice versa (due to the choice of the parking place). In this work, we present a novel and integrated multi-criteria approach to computing optimal journeys for both trips (outward and return trip) combined.

Previous routing approaches only provide very limited functionality for people with disabilities. Many elderly people as well as persons with heavy luggage or a baby buggy would like to avoid obstacles like stairs – however not at all costs (depending on the detour length). Our new multi-criteria approach computes all optimal trade-offs between the difficulty of the route and other optimization criteria (like travel time and the number of transfers). Additionally, we support restrictions that forbid the usage of certain obstacle types completely (like a person in a wheelchair cannot use stairs at all). The restrictions as well as the difficulty of each obstacle need to be adaptable to the profile of the person using the routing service. Our approach is customizable and computes optimal intermodal journeys in a fully integrated manner.

Another use case of intermodal mobility is the planning of a tourist trip in a foreign city. Here, several constraints such as opening hours of attractions need to be considered. If planning a tour for multiple days, we want to avoid redundancy. We present a novel combination of the Time Dependent Team Orienteering Problem with Time Windows (TDTOPTW) with the Orienteering Problem with Variable Profits (OPVP). Additionally, our modeling is the first to support several entries and exits per point of interest (PoI) which is relevant in practice because for large area sites like zoos or boardwalks the public transport stop at each entry/exit may be serviced by different lines.

In case of delays, cancellations, reroutings, or track changes, a journey may become infeasible. In this case, information is key to finding a solution to this problem. Informing travelers as soon as possible gives them the most options. We present an efficient approach to monitor millions of journeys in parallel. The selection of change notices to be communicated to a traveler may be flexibly adapted to the travelers individual needs. Additionally, the system is capable of providing intermodal real-time alternatives in case of a broken connection.

To make the functionality described above accessible to the end-user, we have built mobile (Android) as well as web-based user interfaces. We describe the distributed modular software architecture which can resemble micro-services as well as a monolithic setup enables us to provide the approaches in a scalable and efficient way.

Zusammenfassung

Millionen Menschen nutzen Online-Reiseplaner. Allerdings unterstützen die meisten aktuell verfügbaren Systeme nur die Routensuche für ein Verkehrsmittel (z.B. nur öffentliche Verkehrsmittel oder nur mit dem Auto fahren). Das erschwert die Suche nach intermodalen Reiseketten in denen verschiedene Verkehrsmittel kombiniert werden, um das Ziel zu erreichen. In dieser Arbeit stellen wir ein echtzeitfähiges, multikriterielles und intermodales Reiseinformationssystem vor, das verschiedenste Verkehrsmittel und Anwendungsfälle wie beispielsweise die intermodale Verbindungssuche für Menschen mit Mobilitätseinschränkung oder das Planen einer Touristen-Tour unterstützt.

Den perfekten Parkplatz zum Wechsel zwischen Individualverkehr (beispielsweise Fahrrad oder PkW) auf den öffentlichen Verkehr (Buasse, Straßenbahnen, Züge, usw.) zu finden, ist bei der Verwendung mehrerer unimodaler Reiseplaner nur durch Ausprobieren möglich. Das Problem wird weiter erschwert, wenn man den Rückweg in die Planung miteinbezieht. Dies ist ein weitverbreiteter Anwendungsfall intermodaler Mobilität. Die optimale Route für den Hinweg kann (durch die Wahl des Parkplatzes) einen suboptimalen oder sogar unfahrbaren Rückweg erzwingen. Selbiges gilt umgekehrt. In dieser Arbeit stellen wir einen neuartigen integrierten multikriteriellen Ansatz vor, mit dem (kombiniert) optimale Reiseketten für Hin- und Rückrichtung berechnet werden können.

Bisherige Routingansätze bieten nur sehr limitierte Funktionalität für Menschen mit Mobilitätseinschränkungen. Viele ältere Menschen sowie Menschen mit schwerem Gepäck oder einem Kinderwagen würden Hindernisse gerne vermeiden – aber nicht zu jedem Preis (abhängig von der Länge des Umwegs). Unser neuer multikriterieller Ansatz berechnet alle optimalen Kompromisslösungen zwischen der Beschwerlichkeit des Weges und den anderen Optimierungskriterien (wie zum Beispiel Reisezeit und die Anzahl der Umstiege). Zudem unterstützen wir Einschränkungen, die die Nutzung

von bestimmten Hindernistypen vollständig verhindert (wie zum Beispiel eine Person im Rollstuhl, die keine Treppen nutzen kann). Die Konfiguration dieser Ausschlüsse sowie der Beschwerlichkeit für jedes Wegstück müssen vollständig durch die Nutzer des Routenplaners anpassbar sein. Unser Ansatz ist frei konfigurierbar und berechnet optimale intermodale Reiseketten auf eine vollständig integrierte Art- und Weise.

Ein weiterer Anwendungsfall intermodaler Mobilität ist die Planung einer Touristen-Tour in einer fremden Stadt. Hierfür müssen einige Bedingungen, wie beispielsweise die Öffnungszeiten von Attraktionen, berücksichtigt werden. Bei der Planung von Touren für mehrere Tage möchte man Redundanz bei den Aktivitäten vermeiden. Wir präsentieren eine neuartige Kombination des *Time Dependent Team Orienteering Problem with Time Windows (TDTOPTW)* mit dem *Orienteering Problem with Variable Profits (OPVP)*. Zudem ist unsere Modellierung die erste, die Attraktionen mit mehreren Ein- und Ausgängen unterstützt. Diese Möglichkeit ist praktisch relevant, da insbesondere bei großflächigen Attraktionen wie beispielsweise Zoos oder Strandpromenaden die Haltestellen in der Nähe der Ein- und Ausgänge von verschiedenen Linien bedient werden.

Im Fall von Verspätungen, Ausfällen, Umleitungen, oder Gleiswechseln können Verbindungen brechen. In diesem Fall sind Informationen der Schlüssel um eine Lösung für das Problem zu finden. Die Reisenden möglichst frühzeitig über Probleme zu informieren vergrößert den Handlungsspielraum. Wir präsentieren einen effizienten Ansatz um Millionen von Verbindungen zeitgleich zu überwachen. Die Auswahl der Änderungsmeldungen, die den Reisenden übermittelt werden sollen, kann flexibel den Wünschen der Reisenden angepasst werden. Zudem ist das System in der Lage im Fall einer gebrochenen Reisekette Echtzeitalternativen zu berechnen.

Um die beschriebenen Funktionalitäten dem Endanwender zur Verfügung zu stellen, wurde eine mobile sowie eine web-basierte Nutzeroberflächen entwickelt. Die vorgestellte verteilte modulare Software-Architektur, die sowohl als Microservices als auch in einem monolithischen Setup verwendet werden kann, ermöglicht die skalierbare und effiziente Bereitstellung der Ansätze.

Contents

Introduction

On average, 31.2 million people traveled by public transportation in Germany in 2018 every day [VDV19]. However, none of these journeys started or ended at a bus stop or train station but rather at another address. This fact needs to be considered when designing journey planning algorithms. An intermodal travel information system computes optimal journeys from one address to another. These journeys may involve multiple modes of transportation like walking, ride sharing, bike sharing, taxi, flights, intercity buses, private car, as well as all sorts of public transportation (trams, buses, long and short distance trains).

Not all journeys go as planned: in Germany, more than six percent of all trains operated by the largest Germany railway company, Deutsche Bahn, had a delay of more than six minutes; only 75.9 percent of all long distance train arrivals were on time [DB19]. This does not take into account cancellation and rerouting of trains which can cause additional problems for customers. In those situations, information is key to finding a solution. Thus, an information system should not only compute optimal journeys for the scheduled timetable but also monitor booked journeys, inform the customer in case of a problem and provide real-time alternatives if necessary.

Most scientific work on the subject of finding shortest paths in transportation networks was focused on one predominant use case: computing a non-extensible set of Pareto optimal journeys regarding only travel time and number of transfers. In this thesis, we broaden this view by considering further very common use cases of intermodal routing such as the optimal integrated planning of two-way roundtrip journeys (including park and ride) as well as a personalized route planning algorithm for mobility impaired persons. Much effort was also put into a realistic data model that respects the specialities of public transport such as portion working, through trains, time zones and daylight saving time as well as fine grained transfer times. All this needs to be accomplished

while still maintaining acceptable computation times even when considering real time information such as delays, reroutings, cancellations, additional services and track changes. Thus, algorithms that require extensive preprocessing need to be ruled out.

This thesis proposes algorithms which solve practical problems regarding intermodal mobility. Classic Algorithmics does not necessarily lead to practical solutions: as we will also see in the course of this thesis, an improvement of asymptotic worst-case running time (often ignoring memory consumption) does not necessitate better running times on real hardware for realistic problem instances. One important aspect is that the classical von-Neumann machine model [Neu93] does not resemble the properties of modern hardware such as memory hierarchies or parallelism anymore. This has led to the development of a new paradigm: *Algorithm Engineering* [MS10; San09; SW11] which "is concerned with the design, theoretical and experimental analysis, engineering and tuning of algorithms" and "addresses issues of realistic algorithm performance by carefully combining traditional theoretical methods together with thorough experimental investigations" (posted in DMANET on May 17, 2001 by Giuseppe Italiano [MS10]). Based on Popper's scientific method [Pop35] which is driven by falsifiable hypotheses (e.g. about performance or quality metrics in our case) that are supported by experiments, all approaches in this thesis were reproducibly evaluated with practical implementations.

Main Contributions

In this thesis, we present a comprehensive realistic intermodal real-time journey planner. It supports a broad range of modes of transportation: services that are operated on a schedule (e.g. trains, flights, ferries, long distance busses, local busses, and trams) as well as individual transportation (e.g. walking, riding a bicycle, driving private car, taxi). Additionally, the system supports sharing mobility such as bike sharing and dynamic ride sharing. The approach to integrate dynamic ride sharing with public transport employs a preprocessing phase to efficiently compute ride sharing options at runtime [Fah+16]. [1]

Besides journey planning based on a schedule timetable, the system also supports

[1] Presented at the *European Transport Conference (ETC) 2015*.

updating the data model according to the real situation which allows to find intermodal journey alternatives in disturbed situations. This includes the processing of real-time messages such as delay updates, reroutings, additional services, cancellations, track changes, and free-text messages. Furthermore, we present a novel efficient approach to journey monitoring which can be used to inform the user about a personalizable set of changes (e.g. journey not feasible anymore, track changes, interchange alarm, later/earlier arrival/departure at the first/last stop, etc.) but also to monitor all journeys from a train operator's perspective in a decision support system. [2]

We present a novel personalized approach to intermodal journey planning that computes Pareto-optimal accessible journeys for people with disabilities in an integrated way. We introduce a new Pareto optimization criterion that reflects the individual difficulty (based on the user's profile) of an obstacle (like stairs). This enables us to compute all optimal trade-offs between difficulty and all other optimization criteria derived from the journey's properties. Furthermore, the approach also supports "hard" restrictions (e.g. the wheelchair profile excludes the use of stairs). [3]

This thesis considers aspects that are special to intermodal mobility: when planning unimodal journeys, two-way roundtrips (i.e. if the user would like to return to their starting point) can be split into an outward and a return trip where each trip can be optimized separately. This is not the case for intermodal journey planning: if the user uses a car or a bike for the first part of the outward trip, this introduces a dependency for the return trip. We present the first multi-criteria approach to optimize outward and return trip of an intermodal journey in an integrated way [GHW19].

Another special case of intermodal mobility is the planning of a tourist trip in a foreign city using public transport as well as walking to travel between points of interest. We propose several realistic extensions to the Time-Dependent Team Orienteering Problem with Time Windows (TDTOPTW) which are relevant in practice and present the first MILP representation of it. Furthermore, we propose a problem-specific preprocessing step which enables fast heuristic (iterated local search) and exact (mixed-integer linear programming) personalized trip-planning for tourists. [4]

[2] Presented at the RTDM (Oct 2018), Symposium on Rail Transport Demand Management.
To appear in the journal *Public Transport* published by *Springer*.

[3] To be presented at the *HEUREKA* 2020 conference in April 2021.
To appear in the database of the *Forschungsgesellschaft für Straßen- und Verkehrswesen (FGSV)*.

[4] To be presented at the 20th Workshop on Algorithmic Approaches for Transportation Modelling,

For all our contributions, the evaluation results show that the approaches are feasible in practice.

Outline

This thesis is structured as follows: In Chapter 1, we first discuss the current state-of-the-art in Section 1.1. After that, we introduce the public transport routing in Section 1.2 which forms the core of our routing engine. Here, we also introduce special service rules and describe a model extension to handle these cases and describe the available problem definitions which we extend in Section 1.4 by a problem definition that especially suits flexible users. Section 1.5 presents a generic approach to intermodal routing. Our approach to street routing is described in Section 1.6. The integration of dynamic ride sharing is discussed in Section 1.8. Extensions to our basic intermodal routing approach to provide a journey planning for people with disabilities is presented in Section 1.9. An integrated solution to compute optimal round trip journeys is described in Section 1.10. Optimal intermodal tourist tour planning is covered in Section 1.11.

In Chapter 2, we first describe the updating of the core routing graph according to real-time updates (delays, etc.) in Section 2.1. Thereafter, in Section 2.2, we introduce an efficient personalized approach to monitor millions of journeys in parallel.

Finally, we describe the software architecture in Chapter 3 (including a module distributed system described in Section 3.1 and an efficient (de-)serialization technique in Section 3.2), conclude, and list starting points for future work in Chapter 4.

Optimization, and Systems (ATMOS 2020).

1 Intermodal Route Planning

Intermodal route planning is the planning of an optimal route from one location to another using different means of transportation. By this definition, all commonly used public transport routing algorithms are already intermodal because they consider walking between stations and support several means of transportation such as buses, trams, trains, etc. However, in this thesis, we want to consider more advanced planning scenarios involving all available modes of transportation such as driving by car, riding a bicycle or using bike sharing, ride-sharing, etc. and define intermodal route planning accordingly. We aim to provide a broad selection of supported modes of transportation.

1.1 Related Work

There has been extensive research regarding unimodal routing as well as intermodal routing. Bast et. al. [Bas+16a] give a comprehensive overview of the current state-of-the-art.

In public transport routing it is common to compute a non-extensible Pareto set optimizing a set of selected criteria such as travel time and number of transfers. The time-expanded and the time-dependent graph model are the two basic variants to model a timetable with a graph [Pyr+08; Sch09] which allows to apply shortest path algorithms such as variations of Dijkstra's Algorithm [Dij+59] to find optimal connections. Recent advances in public transport routing were primarily the RAPTOR algorithm [DPW12], the Connection Scanning Algorithm (CSA) [Dib+13a], TripBased routing [Wit15], and Public Transit Labeling (PTL) [Del+15]. Previously, graph-based shortest path algorithms were predominant. There are two approaches to model a timetable with a graph: the time-expanded graph model (also called event-activity network) where nodes

represent events (e.g. departures and arrivals) and edges represent actions (driving, standing, changing vehicles, etc.) and the time-dependent graph model where each node basically represents a location (e.g. a station) and the edges are time-dependent; i.e. the edge weight is the sum of the waiting time and the driving time until the next arrival at the location represented by the head node of the edge. Note however, that all approaches (graph-based and table-based) allow footpaths to be used between stations. These footpaths are typically contained in the timetable dataset. Flights can be integrated into the public transport network graph, too [Del+09]. There has recently been extensive research to support computing optimal journeys without restricting walking to a fixed set of footpaths [WZ17; PV19; Bau+19].

There are several speedup techniques that require non-negligible preprocessing times preventing their use for real-time routing (i.e. routing according to the current situation including delays, reroutings, etc.). These include advanced versions of the CSA [SW14] (30 min preprocessing for Germany), RAPTOR [Del+17] (67:32 min preprocessing for the Netherlands), TripBased [Wit16] (231:16 h preprocessing for Germany). Also, Public Transit Labeling [Del+15] requires a significant amount of time for preprocessing (54min for the city of London). The same is true for the basic TripBased routing [Wit15] (39 min for Germany). Transfer Patterns [Bas+10] as well as a scalable version thereof [BHS16] is another speedup technique that takes too long to preprocess to be capable of real-time routing (16.5 h). It was shown that transfer patterns are robust against delays [BSS13] (e.g. of 50 000 queries, only 450 computed paths are not optimal). Note however, that this experiment was artificial and does not include reroutings, cancellations, additional services, or track changes. Berger et. al. developed the speedup technique SUBITO [BGM10] for which preprocessing is optional. Delling et. al. [DKP12a] propose a parallel algorithm for queries on a departure time interval.

There are several approaches for street routing. These allow to find shortest paths on continental sized road networks in the order of milliseconds or even microseconds [Bas+16a]. All approaches are graph-based. Speedup techniques (which mostly exploit hierarchy and/or goal direction) include Highway Hierarchies [SS12], Contraction Hierarchies [Gei+12], Hub Labels [Coh+03; Gav+04], Chase [Bau+10], ALT [GH05], Arc Flags [Hil+09], and SHARC [Del08]. PHAST [Del+13c] effectively implements a lookup table (shortest path tree).

Research regarding intermodal routing (also called multimodal routing) mostly reuses

concepts from street routing and public transit routing. Here, it is common to restrain the search to a given sequence of transportation modes. The corresponding problem is called *Label Constrained Shortest Path Problem* [MW95; BJM00]. A label is assigned to each edge. A sequence of edges from source to destination needs to fulfill certain constraints (regarding those labels) to form a valid path. [Bar+08] explores speedup techniques (A* [HNR68] and Bidirectional Search) for finding label constrained paths. Access Node Routing [DPW09] computes hierarchical journeys (e.g. driving, public transport, driving) and "skips" the road network at the start and the end of the journey. This is accomplished by precomputing so called *access nodes* (entry and exit nodes on the next hierarchy layer graph) for each node of the lower hierarchy layer. Core-based Access Node Routing [DPW09] combines Access Node Routing with Contraction Hierarchies. It introduces a core graph of the street network and precomputes access nodes only for this core graph. State Dependent uniALT (SDALT) [KLC12] accelerates the search for label constrained paths by computing state-dependent lower bounds.

However, these approaches optimize only one criterion. Delling et. al. [Del+13b] propose a multi-criteria approach to computing multimodal journeys by adapting the RAPTOR algorithm [DPW12]. To filter the result set to a reasonable size, they apply Fuzzy Logic [Zad88]. Bast et. al. [BBS13] present *Types aNd Thresholds* (TNT) which defines a set of reasonable journey patterns such as only car, public transport and walking without driving, or public transport with little to no walking but with car. In both cases [Del+13b; BBS13], the restrictions are used to reduce the search space and therefore improve runtime performance.

1.2 Public Transport Routing

The core of our intermodal routing system is the public transport routing. In this section, we present different existing approaches to public transport routing as well as new extensions to existing models.

1.2.1 Data Model

Computing shortest paths in public transport networks requires a formal model of the problem. Historically, shortest path problems were solved by using Dijkstra's Algorithm [Dij+59] (and extended versions thereof) on a digraph $G = (V, E)$ where V is the set of vertices and E is the set of edges. When modeling the search graph, there are two basic ways to deal with the fact that public transport vehicles operate on a schedule: the graph can either be *time-expanded* or *time-dependent* [Pyr+08; Sch09]. In the time-expanded graph model, every departure and arrival event is modeled as a node, whereas activities like driving or standing at a station are modeled by edges connecting the event nodes. This is also called an event activity network. In the basic time-dependent model, nodes correspond to public transport stops and the edges connection these stops are time-dependent – i.e. the edge weight is a function of time returning the sum of the waiting time and the travel time until the arrival at the next stop. Since the time-dependent graph model is more efficient regarding size as well as computation time [Sch09; Pyr+08], we focus on the time-dependent graph model as well as other table-based data models employed in the algorithms *Connection Scanning Algorithm (CSA)* [Dib+13b], *Round-bAsed Public Transit Optimized Router (RAPTOR)* [DPW12], and *TripBased Routing* [Wit15].

In the following, we describe how to integrate extensions such as merged services into the time-dependent model as well as how to derive the other data models (RAPTOR, CSA, and TripBased) from the time-dependent graph.

1.3 Changes to the Time Dependent Graph Model to Support Latest Departure Queries

This modeling change is also published in [GHW19].

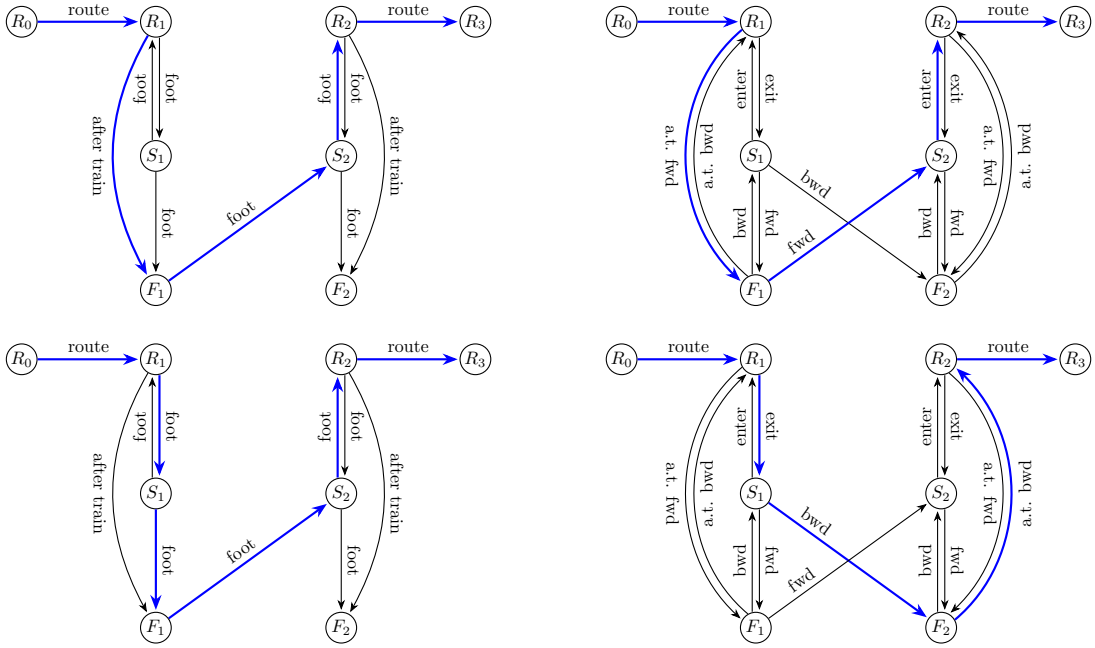


Figure 1.1: Changes to the Time Dependent Graph Model to Support Backward Search: The new model (right side) fixes the inconsistency (different costs for forward and backward search) of the old model (left side).

As we can see in Figure 1.1 (edge costs are listed in Table 1.1), the basic time dependent model presented in [DMS08a] is not consistent (i.e. equal graph costs for the same journey in forward and backward search) for routes containing walks between nearby stations: in the backward search the path includes the transfer costs of S_2 while in the forward search no transfer costs are included (which is the desired behavior). This problem arises because the after train edge is not symmetric for forward and backward search: a “before train” edge could be introduced but it would enforce expanded labels to use a route edge thereafter. A label with this restriction may not dominate other labels without this restriction. This would prevent domination in many cases and therefore

Table 1.1: Edge Type Costs for Forward and Backward Search in the Time Dependent Graph as (Travel Time, Transfer Count) tuples: costs marked with a star "*" are not feasible at edge expansion if the corresponding label did not use a route edge before. The symbol $\oslash$ indicates that the edge is not feasible in this search direction. ic_s is the transfer time for interchanges at station $s \in S$.

Edge Type	Forward Search	Backward Search
enter	$(0, 0)$	$(\text{ic}_s, 1)$*
exit	$(\text{ic}_s, 1)$*	$(0, 0)$
after train forward	$(0, 1)$*	$\oslash$
after train backward	$\oslash$	$(0, 1)$*
fwd	(x, y)	$\oslash$
bwd	$\oslash$	(x, y)

increase the algorithm runtime. Instead, we changed to the graph model: in the fixed model, a walk between two stations has the same costs in forward and backward search direction.

Rule Services

Besides normal services, there are special services in public transportation which operate coupled together on a part of their itinerary. These merge at one stop and split again at another stop. Another speciality of public transport services are extension services (also called through-service / through-train). Each section of a through train is serviced by the same physical vehicle. However, the itinerary is split into two or more separate services which are linked by a through-service rule. This is often the case for circular services where one separate service visits each station exactly once and each service is connected to the next with a through-service rule.

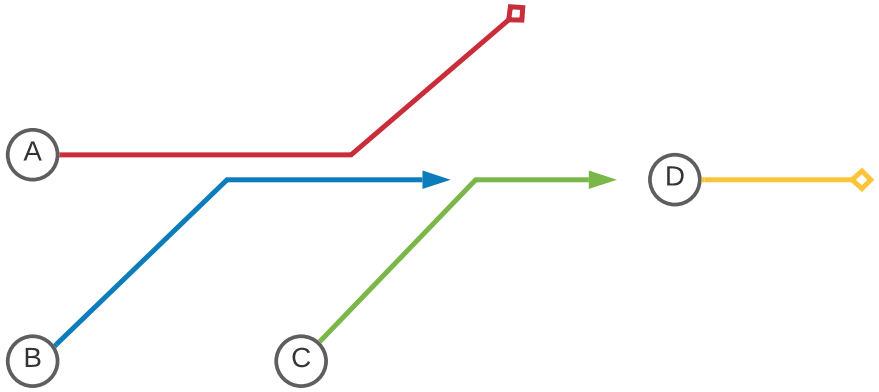


Figure 1.2: Rule Services Example: A and B have a joint section. B is connected with C by a through service rule. Again, C and D are also connected by a through service rule.

This has implications for the route planning: if we do not consider these special rules, we might count a transfer where there is no transfer because it is the same physical vehicle – just a different service. If the difference between the arrival time at the last stop of the first service and the first departure time of the next service is less than the transfer time at this stop, a connection is considered not feasible by the routing algorithm. The same is true for merge/split services: as depicted in Figure 1.2, services A and B have a section where trains are coupled together. Therefore, it is possible to travel from the first stop of service A to the last stop of service D without counting a transfer. It is also possible to travel from the start of service B to the end of service A without transfers.

Preprocessing The input in commonly used formats such as HAFAS Rohdaten[1] or GTFS[2] specifies a set of rules which are comprised of a pair of services as well as the rule type (through service or merge/split) and a bitfield of traffic days where a 1 indicates that the rule is active at this day. Thus, we have to consider three traffic day bitfields to consider: both, services as well as the rule itself each have traffic day bitfields. This is depicted in Figure 1.3. There, we see the input data on the lower two layers of the left side. To discover relations of more than two rules, we introduce *super rules* which connect rules that have common services.

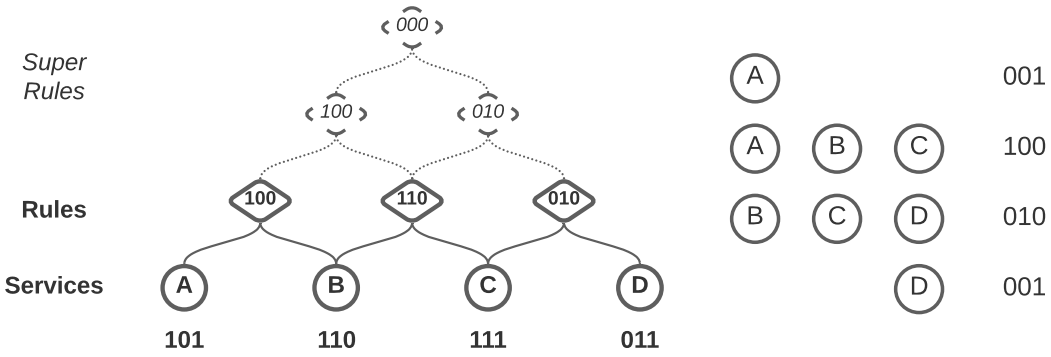


Figure 1.3: Rule Service Example with Traffic Day Bitfields: The left side shows the input in bold. Rules connect always two services. The *Super Rules* are computed by the preprocessing algorithm and connect rules that apply together. The right side shows the output of the preprocessing: each row contains a service combination that operates with the traffic day bitfields on the right.

.

A naive approach would be to create all super rules and iterate each layer beginning at the topmost node. In every iteration, one node (super node, rule node, or single service) is iterated for which every predecessor (nodes one layer above which are connected to this node) are already processed. The traffic day bits which are set in the current node are removed in all children that can be reached recursively and a rule service group with the traffic day bitfield of the initial node is pushed to the output. This way, we retrieve the output on the righthand side of Figure 1.3. Note that there might be

[1] A format used in the commercially available HAFAS system offered by Siemens AG.
[2] General Transit Feed Specification (GTFS): https://developers.google.com/transit/gtfs

days where a service that is otherwise involved in a rule, operates separately (like in Figure 1.3: services A and D at the first and last day). Since the number of nodes that need to be created this way, is quite large, we apply another approach which produces the same output: we apply a breadth-first search on each rule node and remember the intersection of every node visited. After the maximum relation has been discovered (i.e. the next intersection would yield a bitfield with only zeros), the intersection of the bitfields is removed from every node and the visited nodes together with the traffic day bitfield intersection are pushed to the output.

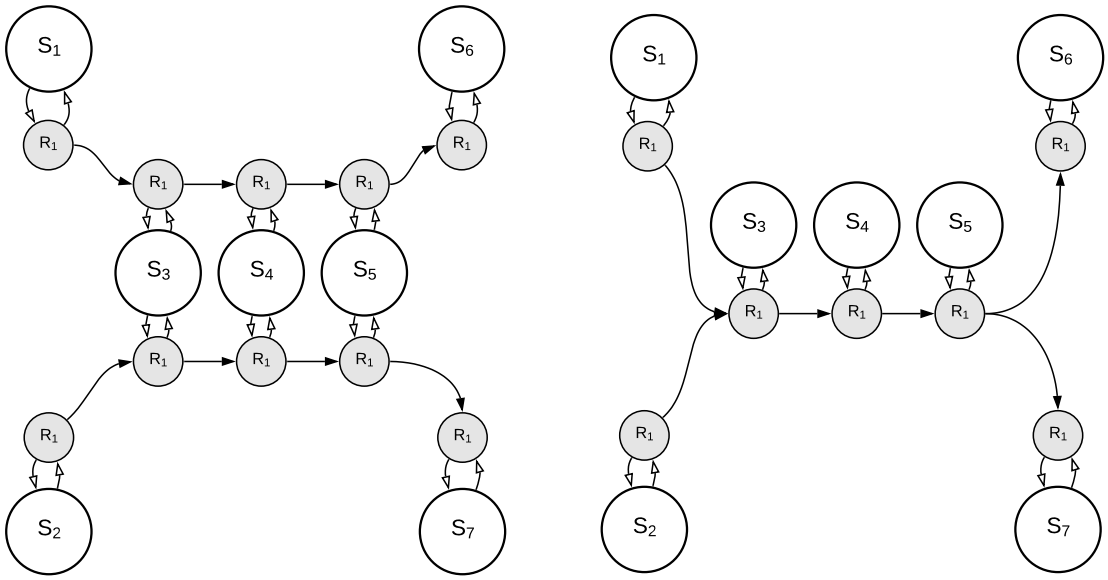


Figure 1.4: Comparison of Graph Models With and Without Portion Working: The left-hand side (where traveling from S_2 to S_6 or from S_1 to S_7 requires a transfer) shows the standard model without considering the portion working. The model on the righthand side enables the algorithm to find journeys from S_2 to S_6 or from S_1 to S_7 without transfers.

Graph Layout To enable finding the connections that are only feasible when considering the service rules described previously, the graph layout needs to be adjusted. Figure 1.4 shows how to adjust the graph in case of portion working (join / split) train. Here, we see that on the righthand side, it is possible to reach S_6 from S_2 without transfer whereas

on the lefthand side it requires one transfer. If the transfer time between route R_1 and R_2 on the lefthand side is shorter than the difference between departure and arrival the connection will not be feasible. Both services share one route on the righthand side.

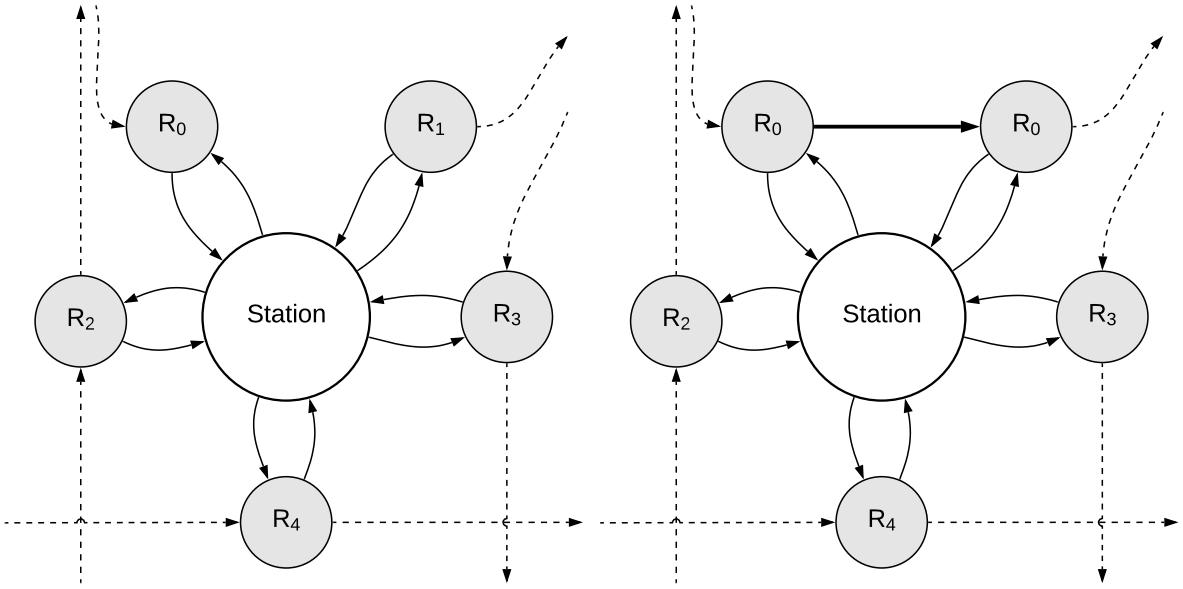


Figure 1.5: Comparison of Graph Models With and Without Through Train: routes R_0 and R_1 are serviced by the same physical vehicle. Thus, no transfer is necessary. The basic time-dependent graph model (left side) is extended by the bold through train edge on the right side to enable journeys without counting a transfer. R_0 and R_1 are merged into one route.

.

Figure 1.5 shows the previous time-dependent model on the lefthand side. Note that the vehicle that services route R_0 continuous on route R_1 and therefore there is no transfer required from R_0 to R_1. This is enabled by the additional bold edge on the righthand side. This bold edge does not count a transfer and has no travel time cost. Consequently, with the graph model on the righthand side the algorithm will be able to find through train connections without counting a transfer.

Rule Services for CSA, RAPTOR, and TripBased The solution proposed for the time-dependent graph cannot be adapted to newer table based routing approaches such as

CSA, RAPTOR, or TripBased routing. Our approach to enable the same journeys utilizing those algorithms is to generate all variations to travel, introduced by special service rules, as a new (artificial) trip. This can be accomplished by starting a depth-first-search at each first route node in the time-dependent graph model presented above. Each edge sequence leading to any last route node yields a new (artificial) expanded trip. Building these expanded trips and adding them to the set of trips for the CSA, RAPTOR, and TripBased routing algorithms enables us to find exactly the same journey sets as with the time-dependent multi-criteria Dijkstra.

1.3.1 More Precise Transfer Times

The current model only supports one minimum transfer time per station. However, the required transfer time between two tracks that are located at the same platform is much lower than the transfer time of tracks at different platforms. Therefore, we introduce two transfer times per station: one for tracks at the same platform and another for all other track pairs. This is also the granularity used for dispatching decision making at Deutsche Bahn AG. We introduce a new graph model which considers this fact in an efficient manner.

As we can see in Figure 1.6, each platform is modeled by a separate node which is connected to the station node with a transfer edge carrying the station transfer time. However, nodes which are connected to the same platform are reachable within the platform change time of the station.

Evaluation Our evaluation with 100,000 random 2 hour range queries on a timetable of Germany (covering two days 2019/10/01 - 2019/10/02) provided by Deutsche Bahn AG shows that the computation times an Intel Core i7 6850K (6x 3.6GHz) CPU and 64GB main memory slightly increase from 624ms to 651ms by approximately 4% which is acceptable. The average number of transfers stays nearly the same (4.2816 transfers vs. 4.2889 transfers). The same is true for the travel duration (419.5852 vs. 418.2726).

1.3.2 Problem Definitions

When routing in time-dependent networks, there are several problem definitions to consider:

- **Earliest Arrival**: Find the journey with the earliest arrival time given a earliest point in time for the departure. The waiting time until the departure time is counted towards the travel time.
- **Latest Departure**: Find the journey with the latest departure time given a latest point in time for the arrival. The duration between the last arrival and the provided search time is counted towards the travel time.
- **Profile Query and Range Query (Forward)**: Given a departure time interval, find all optimal journeys that depart within this departure time interval.
- **Profile Query and Range Query (Backward)**: Given an arrival time interval, find all optimal journeys that arrive within this arrival time interval.

Additionally, it can make sense to make use of *Meta Stations* (as introduced in [Sch09]) to allow routing from/to not just a single station but from/to a set of stations at the same time. Algorithmically, this can be solve by creating start labels at each source representative and accepting labels at each destination representative.

A routing query can from an address to another address, from an address to a station, from a station to an address, and from a station to another station.

Each problem definition described above can be applied with different sets of optimization criteria (travel duration, travel duration and number of transfers, etc.).

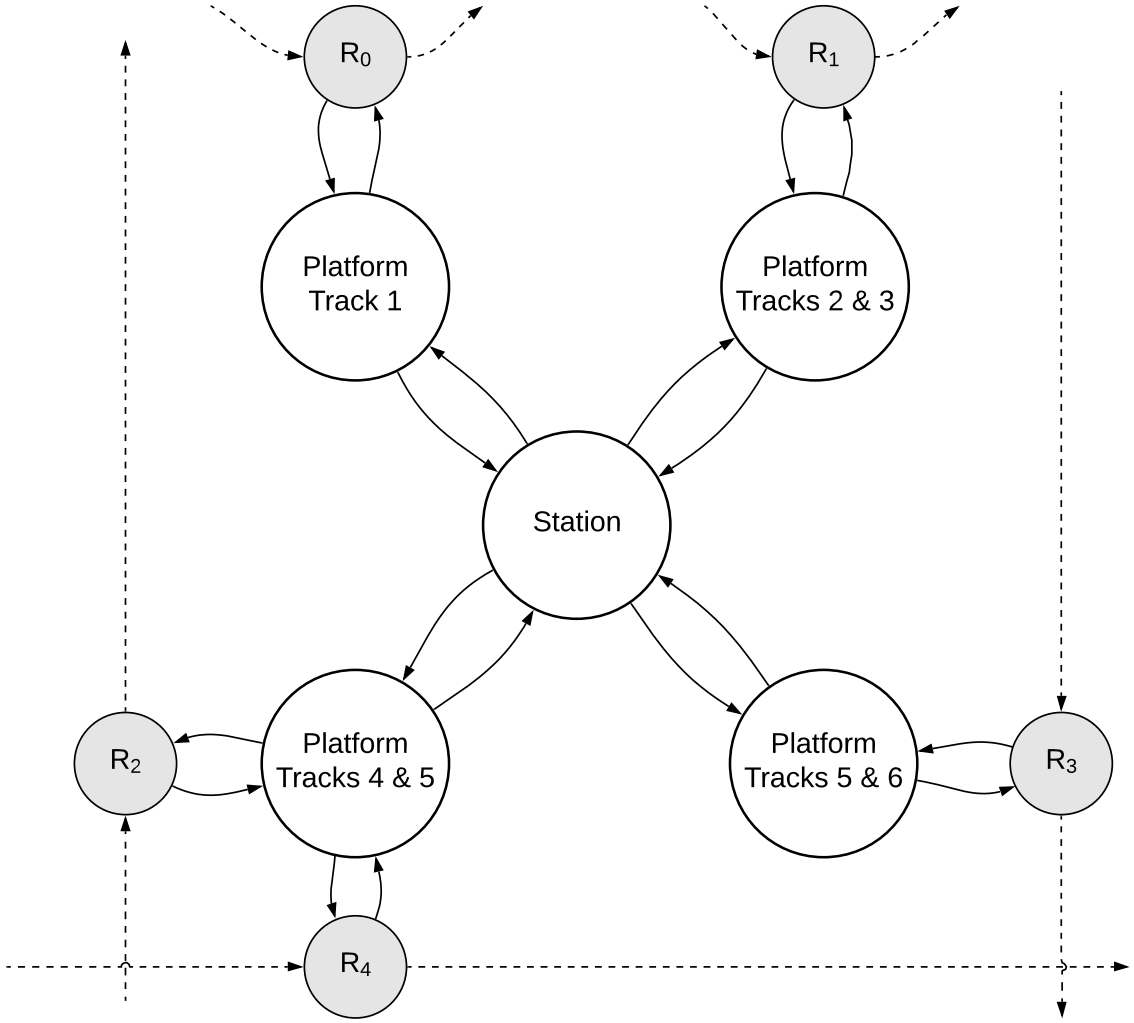


Figure 1.6: Platform Graph Model: route R_2 and R_4 share a platform. The edges from route node to platform node carry the lower platform transfer time whereas the nodes from platform node to station node carry the station change time minus the platform change time.

1.4 A Time-Dependent Routing Approach for Flexible Users

We consider a common use case in public transport routing where neither the earliest arrival nor the range problem yield satisfying results: the user has a rough idea at which time he would like to start but cannot specify a strict departure time interval. Contrary to standard approaches which apply a strict notion of dominance we also deliver “suboptimal” connections if they are sufficiently far away in time from all optimal ones. Our approach allows the user to scroll to earlier and later connections where the merged connection sets are still Pareto sets. If the merged connection set is not a Pareto set, it contains dominated journeys which may confuse the user.

1.4.1 Introduction

We study the problem of computing optimal connections in public transport. There are two common problem definitions: computing the earliest arrival for a given departure time or computing all optimal connections departing within a specified time interval. The former problem definition is suitable whenever the traveller is able to start the journey at the given time or later and wants to arrive as early as possible. Therefore, every minute until the first departure should be counted as travel time (even if the traveler departs later). The latter problem definition is suitable for users who are in a position to specify a strict time interval in which they definitely want to depart: not earlier but not later, either (even if there are connections departing later but arrive earlier than connections from the interval).

In this paper, we propose a definition targeting yet another, very common use case: the user knows roughly a point in time around which he would like to search for connections. Neither does he want to provide a fixed time range nor should the waiting time for the first departure be counted as travel time. It should be possible to search for earlier and/or later connections where the merged connection sets (including the connections from the initial plus those from the earlier/later search) should still be a Pareto set. Unlike most standard approaches which apply a strict notion of optimality, the user is also interested in “suboptimal” connections if they are sufficiently far away in time from

all optimal ones.

The example shown in Table 1.2 illustrates two problems of the standard interval approach: a search from Berlin to Frankfurt am Main [3] in a departure time interval from 11 pm to 1 am yields c_1, a very unattractive 10h 3min connection (with one transfer) where the user has to stay in Spandau for five hours (12:10 am - 5:10 am). This happens because the algorithm is forced to find connections departing in the specified time interval even if there is no sensible connection departing in this interval. Even if the user scrolls to later connections, moving the departure time interval to [5 am - 7 am], this does not yield the more attractive connection c_2, arriving with the same train but departing five hours later in Berlin. The reason for this is that it is dominated by c_3 which departs one hour later and takes 3min less time. All in all, with the standard range query problem definition, we do not find c_2, an attractive connection but rather display the unpromising alternative c_1. In this scenario it would make more sense to show c_2 and c_3 but not c_1. Even if c_3 would not exist and therefore c_2 would be displayed to the user, this could still be confusing because c_1 would still be displayed. To prevent c_3 from dominating c_2, we have to take into account the time difference between connections when applying dominance rules. In order to prevent c_1 from showing up in the result set, we need to consider connections departing outside the departure time interval (c_2 in this case). Note that these problems are not limited to overnight connections. It may be argued that two or more searches with smaller departure time intervals would show c_2 and c_3. However, smaller departure time intervals would constrain the algorithm to even less departures yielding more unsuitable connections like c_1. The standard range problem also yields highly "unstable" results: extending or moving the departure time interval by one minute may change the resulting connection set completely. The added departure may enable a connection which dominates every connection previously contained in the result set.

The challenge is to deliver those connections that meet the use case (but may be suboptimal regarding the strict definition of optimality) and to suppress suboptimal connections that do not. In this paper, we will argue that no approach known to us fulfills this challenge to a satisfactory level (Section 1.4.2). Subsequently, we present a

[3] Optimization criteria are travel time and the number of transfers. We assume all other relevant criteria to be equal; every aspect important for the user should be covered by a search criterion (see for example [GMS07] for a night train search optimizing continuous sleep time).

Table 1.2: Results from a traditional range query. Three connections: the first one has a transfer in Spandau, the second and the third connection have no transfers. The "Transp." column contains the unique train identifier, "Dep." shows the departure time and "Arr." shows the arrival time of the corresponding journey leg. The "Opt." column indicates whether the connection is optimal using a traditional range query.

	Search Interval	Transp.	From	Dep.	Arr.	To	Opt.
c_1	11 pm - 1 am	RB 1	Berlin	12:00 am	12:10 am	Spandau	✓
		IC 2	Spandau	5:10 am	10:03 am	Frankfurt	
c_2	5 am - 7 am	IC 2	Berlin	5:00 am	10:03 am	Frankfurt	✗
c_3	5 am - 7 am	IC 3	Berlin	6:00 am	11:00 am	Frankfurt	✓

new problem definition (Section 1.4.3), an algorithmic solution (Section 1.4.4), and demonstrate that it does fulfill the challenge. More specifically, in our approach, a connection a that is far away from a connection b in time only dominates b if a is much better than b. In particular, we define domination maximally generically, namely by an arbitrary Boolean function on two connections which has the time distance and the quality difference as inputs. Our approach works well with any reasonable domination function, that is, any domination function such that the required threshold quality difference of the two connections is monotonously increasing in the time distance.

Furthermore, our approach

- allows the user to specify a minimal number of connections to be delivered;
- goes well with multi-criteria optimization: in addition to travel time and number of transfers, other criteria such as price or crowdedness can still be optimized;
- can instantly be applied to arbitrary combinations of means of transportation including public transport, driving by car, cycling, etc.;
- also applies to the backward search case where the user selects an arrival time.

In Section 1.4.5 we present several speedup techniques; all of which preserve optimality. Our computational study (Section 1.4.6) covers all local (i.e. busses) and long-distance (trains) public transport in Germany (but excludes private transport). Finally, Section 1.4.7 summarizes and gives an outlook.

1.4.2 Related Work

In the field of public transport routing research, there are different problem definitions and approaches to compute optimal journeys: traditional algorithms are based on a graph (i.e. the time-expaneded [MS04; Pyr+08] and time-dependent [DMS08c; Pyr+08] graph model). Graph based speedup techniques include parallelization [DKP12b], frequency compression [BS14], transit node routing [AW12], and contraction [Gei10]. Some recent approaches like RAPTOR [DPW12], CSA [Dib+13a] and trip based routing [Wit15; Wit16] use other, more efficient data structures to represent the public transport schedule.

The most common problem definitions are the *earliest arrival problem* (e.g. [Dib+13a; AW12; Bas+10; Wit15; Wit16; DPW12; Gei10]) and the *range problem* (e.g. [Nac95; MS04; DMS08c; Gei10; DKP12b; DPW12; Dib+13a; BS14; Wit15; Wit16; BHS16]). Both approaches work well with one or more optimization criteria (i.e. single criterion, multi criteria, or weighted sum). The earliest arrival problem is to compute the journey with the earliest arrival at the destination from a given source departing not earlier than a given start time. The range problem asks for all optimal journeys departing in a specified departure time interval. Neither earliest arrival nor range query address our use case satisfactorily. In fact, earliest arrival is only useful when the user is tied to a specific point in time. On the other hand, the range problem demands the user to provide a fixed departure time interval and suppresses attractive connections based on the strict notion of optimality. By contrast, our approach applies a time difference dependent domination influence and considers optimality globally, not just related to a specific departure time interval.

Scientific publications differ slightly in the definition of the range problem: some compare connections with different departure times [BS14], others consider departure time as an additional optimization criterion allowing only connections departing later to dominate connections departing earlier [Wit16]. The rRAPTOR approach [DPW12] does implicitly consider the time difference when comparing connections: one connection may only dominate another connection if it departs later or at the same time and arrives earlier or at the same time. However, this can still yield inconsistent results in a sense that connections from the departure interval can be dominated by connections departing after the departure interval (see Section 1.4.3 for a precise definition of

consistency). Furthermore, this results in a large number of connections; many of them are not interesting for the user. For example a very long connection departing only a few minutes after an attractive connection with a short travel time is optimal and thus will be part of the result set. Comparing those approaches to the one presented in this paper regarding the problems described in the Section 1.4.1 it is obvious that all previous approaches share the property that they do not consider departures outside the departure time interval. This leads to the aforementioned inconsistencies.

Regarding the domination, we can distinguish two cases: approaches comparing connections departing at different points in time in the departure time interval will leave out interesting connections (i.e. those which take one minute longer but depart several ours earlier / later). Other approaches which do not compare connections with different departure times (or where only later connections may dominate) deliver a very large set of mostly uninteresting connections (i.e. a connection that takes ten times as long as another one departing one minute earlier).

1.4.3 Problem Definition

The input is the schedule on which the means of transportation operate and a query provided by the user. A query consists of the source s, the target t, a tentative point in time for the departure d, two boolean values e and ℓ indicating whether the user is interested in connections departing earlier than d (boolean parameter e), later than d (boolean parameter ℓ) or both, and an integer m stating a minimal connection count. So, a query is a six-tuple, $Q = (s, t, d, m, e, \ell)$.

The desired output needs to satisfy the following two requirements, *Consistency* and *Time Difference Dependent Domination Influence*. Note that these requirements are generic and allow for different implementations depending on the context: we do not require a specific set of criteria, nor do we restrict our approach to Pareto-dominance.

Consistency For an arbitrary, yet fixed partial order on all connections from s to t (such as Pareto dominance) with relation $\prec$ we define the output connection set C_{result} for Q:

1. Every connection in C_{result} needs to be minimal with respect to this partial order among all connections from s to t departing at any time.

2. Every connection in C_{result} has to depart at d or later if e is set to false.

3. Every connection in C_{result} has to depart at d or earlier if ℓ is set to false.

4. The result set has to be inclusion-maximal within the interval: $[min(d, \text{argmin}_{c \in C_{\text{result}}}(dep_c)), max(d, \text{argmax}_{c \in C_{\text{result}}}(dep_c))]$ where dep_c is the departure time of connection c.

5. $|C_{\text{result}}| \geq m$ if there are m or more connections fulfilling Items #1-4, otherwise C_{result} contains all connections fulfilling Items #1-4.

We use the terms optimality and dominance with regard to the partial order relation $\prec$ for the remainder of this paper: a dominates b means $a \prec b$; all minimal elements of the partial order are considered optimal.

Time Difference Dependent Domination Influence We need to adjust the notion of dominance so that the threshold quality difference of the two connections is monotonously increasing in the time distance (time difference between departures / arrivals). Basically, every monotonous dependency on the distance of two connections is suitable for our approach. Obviously, only monotonous dependencies are reasonable.

We define the distance between two connections a and b with their respective departure and arrival times dep_a, dep_b and arr_a, arr_b as $\Delta_{a,b} = min(|dep_a - dep_b|, |arr_a - arr_b|)$. If a overtakes b we set the distance $\Delta_{a,b}$ to 0. Additionally, we require that a connection with a longer travel time never dominates a connection with a shorter travel time. Since the relation describes only a partial order, this does not imply that a shorter connection always dominates a longer connection.

Running Example As an example we present an adjusted travel time criterion. This is based on the notion of *relaxed Pareto dominance* [MS04]. Consider connections a and b with their respective associated travel times t_a and t_b. Then journey a dominates journey b with respect to the travel time criterion only if

$$t_a + \alpha \cdot \frac{t_a}{t_b} \cdot \Delta_{a,b} < t_b \quad . \tag{1.1}$$

The parameter α enables us to scale the influence of journeys: large values make it harder to dominate (value on the left-hand side of the inequation increases) and therefore lead to a larger number of pairwise incomparable optimal journeys. Obviously, (1.1) is an antisymmetric, irreflexive relation; a proof that it is a transitive relation can be found in [Sch09]. Therefore, this relation is suitable as a Pareto criterion. This notion of dominance is independent of the given departure time interval. Note that comparing subpaths (partial connections) during the search (as most routing algorithms do) based on only this criterion is not feasible; specific preconditions for subpath domination in shortest path algorithms on time-dependent graphs are described in Section 1.5.2. As we can see, this example fulfills both requirements mentioned above:

- a connection with a longer travel time may never dominate a connection with a shorter travel time: as we add a non-negative value (here: $\alpha \cdot \frac{t_a}{t_b} \cdot \Delta_{a,b}$) to t_a on the left side of inequation (1.1), t_a may only dominate t_b if $t_a \leq t_b$.
- at an increasing distance $\Delta_{a,b}$ the influence connections have on each other diminishes: for a fixed α value and two connections a and b with travel times t_a and t_b it is obvious that if $t_a < t_b$ there is some point where (1.1) is true (e.g. if $dep_a = dep_b$) but when we increase the distance $\Delta_{a,b}$ (by moving the departure of either a or b), it will become false at a specific distance $\delta_{a,b} = \frac{1}{\alpha} \cdot (t_b - t_a) \cdot \frac{t_b}{t_a}$. Note that after this "threshold" it holds that neither a dominates b nor b dominates a; both are optimal and will appear in the result set.

Table 1.3 illustrates these properties and the role of the variable α. Connections c_1, c_2, and c_3 are already known from the initial example (Table 1.2). Since c_1 arrives at the same time as c_2, we know $arr_{c_1} - arr_{c_2} = 0$ which implies $\Delta_{c_1,c_2} = 0$. Consequently, c_2 will always dominate c_1, regardless of the chosen value for α. Every additional aspect possibly covered by additional search criteria (not applied in this small example) can introduce more incomparable optima, independent of the α value (e.g. if we would optimize for continuous sleep time as described in [GMS07]). Furthermore, we can see that at increasing α values, more and more connections are optimal despite their increased travel time (c_4 with $\alpha = 3$ and c_5 with $\alpha = 4$). Note that there cannot be a value for α where no connection is optimal. If a connection does not appear in the result set there needs to be at least one other connection that dominates it and therefore makes it obsolete.

Table 1.3: Concrete Number Example for the Running Example of Dominance, Inequation (1.1) with Different Values for α: The table on the left hand side contains departure times ("Dep."), arrival times ("Arr."), the duration and the number of transfers ("Tr."). The table on the right hand side lists whether a connection c_i is optimal for a specific value for α (✓means that it is optimal).

	Dep.	Arr.	Duration	Tr.
c_1	12:00 am	10:03 am	10h 3min	1
c_2	5:00 am	10:03 am	5h 3min	0
c_3	6:00 am	11:00 am	5h	0
c_4	5:30 am	10:50 am	5h 20min	0
c_5	6:05 am	11:20 am	5h 15min	0

	c_1	c_2	c_3	c_4	c_5
$\alpha = 0$			✓		
$\alpha = 1$		✓	✓		
$\alpha = 2$		✓	✓		
$\alpha = 3$		✓	✓	✓	
$\alpha = 4$		✓	✓	✓	✓

1.4.4 Approach

In this section, we present an algorithmic approach to solve the described problem. To do this, we first define and solve an auxiliary problem. With this algorithm as a subroutine, we continue to solve the main problem.

Auxiliary Problem

Definition Based on the problem definition in Section 1.4.3, we define a simplified version with a different type of query consisting of the source, the destination, and a time interval: $Q_{\text{aux}} = (s, t, I_{\text{input}})$. We replace the consistency definition from Section 1.4.3 with the following interval based definition. Every other aspect such as the requirement for time difference dependent domination influence and the generality (arbitrary partial order) still remains. C_{result} needs to satisfy the following requirements:

1. every connection departs in I_{input}
2. every connection is optimal among all connections inside and outside the time interval
3. the set is inclusion maximal subject to (1) and (2)

Since we also consider connections departing outside the time interval in this definition, the union of result sets from any departure time interval will still meet all three requirements.

Subroutine to Solve the Auxiliary Problem In this section, we will describe the algorithmic approach that yields consistent result sets for a given query $Q_{\text{aux}} = (s, t, I_{\text{input}})$ in Section 1.4.4. First of all, we define two kinds of optimality (based on the relation introduced in Section 1.4.3) regarding a fixed source-target combination:

- *Local optimality*: a connection is optimal regarding a specific departure interval. That is, considering all departures in this interval, there is no connection that dominates a locally optimal connection.
- *Global optimality*: a connection is optimal regarding the complete schedule. That is, considering all departures from the complete schedule, there is no connection that dominates a globally optimal connection.

Our approach is based on a traditional routing algorithm ϱ that calculates an inclusion-maximal set of locally optimal connections for a given input interval. Additionally, we compute a lower bound for the travel time $t_{\ell b}$ from s to t (i.e. obtained from a simplified time-independent graph). The algorithm consists of four steps:

1. Obtain a set C_{init} of locally optimal connections by executing ϱ on I_{input}.
2. Calculate a new search interval: we define $\delta_{a,b}$ as the maximal time difference $\Delta_{a,b}$ where one still may have $a \prec b$ (for the relation introduced in Section 1.4.3): $\delta_{a,b} = \max\{\Delta_{a,b} | a \prec b\}$ where t_b is fixed. So $\delta_{c,lb}$ is the maximal time difference $\Delta_{c,lb}$ at which a hypothetical connection with the travel time lower bound $t_{\ell b}$ as travel time could still dominate $c \in C_{init}$. We define

$$I_{filter} = \bigcup_{c \in C_{init}} [dep_c - \delta_{c,lb}, arr_c + \delta_{c,lb} - t_{\ell b}] \quad .$$

   For the running example from Section 1.4.3 we have $\delta_{c,lb} = \frac{1}{\alpha} \cdot (t_c - t_{\ell b}) \cdot \frac{t_c}{t_{\ell b}}$
3. Obtain a connection set C_{filter} by applying ϱ on C_{filter}.
4. Return result set $C_{\text{result}} = \{c \mid c \in C_{filter} \wedge dep_c \in I_{\text{input}}\}$

Observation 1 (Optimality)**.** *Every globally optimal connection is also locally optimal. A locally optimal connection is not necessarily globally optimal.*

Theorem 2 (Consistency). *The result connection set C_{result} is consistent as defined in Section 1.4.4*

Proof. To prove this, we show that all three requirements for consistency introduced in Section 1.4.4 hold:

Departure Within the Input Departure Time Interval The first requirement is obviously fulfilled because of Step (4) of the algorithm.

Global Optimallity We have to show that no connection from C_{result} can be dominated by any other connection in the schedule. We prove this by contradiction: a connection d dominating any connection $c \in C_{\text{result}}$ would have to depart outside I_{filter} because otherwise c would not be part of C_{result}. For any connection d with $dep_d \notin I_{filter}$ that dominates a connection $c \in C_{\text{result}}$ we would have $\Delta_{c,d} > \delta_{c,lb}$, which contradicts the way $\delta_{c,lb}$ was defined (since $t_d \geq t_{\ell b}$). Hence, there cannot be any such dominating connection d, and all connections in C_{result} are globally optimal. To show $\Delta_{c,d} > \delta_{c,lb}$, we distinguish two cases: 1. d departs later than c and 2. d departs earlier than c. The case that d and c have the same departure time cannot occur because $dep_d \in I_{\text{input}}$ follows from $dep_c \in I_{\text{input}}$. Thus, C_{result} cannot contain c if $dep_d = dep_c$ and $t_d < t_c$.

1. Later Departure Assume that d departs later than c. Connection d cannot depart later than c and at the same time arrive earlier than c because of the way we chose the filter interval: based on the fact that $dep_d > arr_c - t_{\ell b}$ (otherwise, $dep_d \in I_{filter}$ and d would dominate c in C_{result}), we can show that $arr_d > arr_c$. By adding arr_d on both sides, we get $arr_d - dep_d < arr_d + t_{\ell b} - arr_c$. Since we now have the travel time of d on the left-hand side, which is less or equal to the travel time lower bound $t_{\ell b}$, we get $arr_d > arr_c$. So we know that d departs later than c ($dep_d > dep_c$) and arrives later than c ($arr_d > arr_c$). Since d dominates c, we know that $t_d < t_c$. In case of a later departure, we have $dep_d > dep_c$ and $arr_d > arr_c$. Now, we can show that $|arr_d - arr_c| < |dep_d - dep_c|$: transposing $t_d < t_c$, where $t_d = arr_d - dep_d$ and $t_c = arr_c - dep_c$, we get $arr_d - arr_c < dep_d - dep_c$. Since we know that $dep_d > dep_c$ and $arr_d > arr_c$ it is obvious that $|arr_d - arr_c| < |dep_d - dep_c|$. Consequently, we have $\Delta_{c,d} = min(|dep_d - dep_c|, |arr_d - arr_c|) = |arr_d - arr_c|$.

We show $\Delta_{c,d} = |arr_d - arr_c| > \delta_{c,lb}$: If d departs later than c, it follows that $dep_d > arr_c + \delta_{c,lb} - t_{\ell b}$. Otherwise, dep_d would be in I_{filter} and d would have dominated c. Transposing yields $dep_d + t_{\ell b} - arr_c > \delta_{c,lb}$, where we can replace $dep_d + t_{\ell b}$ by arr_d (from $t_{\ell b} \leq arr_d - dep_d$ follows $arr_d \geq dep_d + t_{\ell b}$). Since we know that $arr_d > arr_c$ we get $|arr_d - arr_c| > \delta_{c,lb}$.

2. Earlier Departure This case is basically analogous to Case 1. In fact we have $arr_d < arr_c$ because if $dep_d < dep_c$ (earlier departure) and $arr_d > arr_c$ this would imply $t_d > t_c$, which contradicts our assumption that d dominates c. The last part is simpler because it does not involve the lower bound but dep_c itself.

Inclusion Maximality This follows directly from Observation 1: if we find every locally optimal connection, this implies that C_{result} also contains every globally optimal connection. Thus, it is inclusion-maximal.

□

Solving the Main Problem

Our goal is to compute connections that fulfill the requirements from Section 1.4.3. As input we have a query $Q = (s, t, d, m, e, \ell)$. We can use the algorithm described in Section 1.4.4 to accomplish this: we choose an arbitrarily sized initial search interval of x minutes (e.g. $x = 60$). We set $y = \frac{x}{2}$ if e and ℓ are both true, otherwise we set $y = x$. We set the initial interval bounds to $[a, b]$, where $a = d$ if $e = \text{false}$, otherwise $a = d - y$. The same applies to b: if $\ell = \text{false}$ we set $b = d$, otherwise $b = d + y$. Now, we can use $[a, b]$ as I_{input} in the query $Q_{\text{aux}} = (s, t, I_{\text{input}})$. Using Q_{aux} as input for the auxiliary algorithm described in Section 1.4.4 we get all globally optimal connections in the specified interval and add them to C_{result}. If $|C_{\text{result}}| \geq m$ is already fulfilled, we can return C_{result}. Otherwise, we extend C_{result} iteratively until it contains m (the minimal connection count) or more connections. We do this by shifting $[a, b]$ into the specified direction (earlier/later depending on the input parameters e and ℓ) and applying the subroutine on the shifted interval. If e and ℓ are both set to $true$, we can either alternatingly extend the departure time interval in one direction or we can extend it in both directions in each iteration. In our implementation we chose one hour extensions.

Independently of the selected value x, we obtain a set of connections C_{result} that satisfies all requirements defined in Section 1.4.3: based on the way we chose the departure interval, Items #2 and #3 are fulfilled. Global optimality (Item #1) and inclusion-maximality (Item #4) can be derived from the proven properties (Theorem 2) of the utilized subroutine from Section 1.4.4. The algorithm terminates when there are no more connections to compute (search interval reached schedule bounds) or if $|C_{\text{result}}| \geq m$. Therefore, also Item #5 is fulfilled.

1.4.5 Speed-Up Techniques

For the routing algorithm ϱ (introduced in Section 1.4.4) we assume any kind of label-based algorithm (i.e. any *Multicriteria Label-Setting* algorithm as described in [Bas+16a]) such that the runtime can be improved by reducing the number of labels. A label represents a path in the timetable. Thus, we will speak of labels departing at a certain point in time.

Initial Search

In the basic version of the algorithm (cf. Section 1.4.4), the first step is to apply the routing algorithm ϱ on I_{input}. Note that a larger I_{filter} interval has a negative impact on the search performance. Since the lower bound value $t_{\ell b}$ is fixed for a given source-target combination, $\delta_{c,lb}$ only depends on the travel time of the connections found in I_{input}. To prevent cases where a dominating connection departs just a few minutes after or before I_{input} we extend the search interval of the initial search. This enables us to dominate connections that would have unfavorable travel times (yielding high $\delta_{c,lb}$ values and thus a large I_{filter}) already in the first search step. Note that I_{input} stays the same, we just apply ϱ to an extended interval. Connections found in the initial search that do not depart in I_{input} will be filtered in Step 4 of the algorithm. In the computational study, we will evaluate interval extensions of 1, 2, 4, 8, and 12 hours.

Filter Search

The purpose of Step 3 of the auxiliary algorithm (cf. Section 1.4.4) is not to produce an inclusion-maximal set of optimal connections departing in I_{filter} but rather to eliminate

all locally optimal connections from C_{init} that are not globally optimal. Note that applying ϱ on I_{filter} cannot yield any new optimal results departing in I_{input} that were not discovered by applying ϱ on I_{input}. Based on this insight, we can significantly improve the performance of the second search step.

Early Termination Since the filter search should only prove that a connection found in I_{input} is either globally optimal or not, we can safely stop when there are no labels left at the destination that depart in I_{input}. In this case, the connection set C_{result} can only be empty because the filter search cannot yield any new connections with a departure time in I_{input}.

Discard Labels

Update Filter Search Interval When connections from the initial interval I_{input} get dominated, we can recalculate I_{filter} for the new reduced set of labels. This can only yield the same interval or a smaller interval. If I_{filter} is now smaller, there are chances that we now have labels in the search process that do not depart in I_{filter} anymore. Since those labels are no longer relevant we can safely discard them (i.e. no edge expansion).

Dominance We can safely discard a label from the filter search not only if it does not depart in the updated I_{filter} (anymore) but also when it is not able to dominate a label departing in the input search interval I_{input}. Thus, we only need to keep labels from the filter search that may still dominate at least one connection from C_{init} (which has not already been dominated by another label) with regard to the relation $\prec$ introduced in Section 1.4.3. So we discard it as soon as it is obvious that a label cannot dominate any connection from the initial search. To accelerate this process, it might be useful to calculate lower bounds (i.e. for each criterion in case of multi-criteria Pareto search). This enables us to discard unpromising labels early in the search process. Besides using the lower bound values, it is possible to utilize the *Time Difference Dependent Domination Influence* (cf. Section 1.4.3).

1.4.6 Experimental Results

In this section, we present a computational study evaluating the processing time and result quality of the approach. Since the problem description is generic, the routing steps #1 and #3 in the algorithm described in Section 1.4.4 can be performed with an arbitrary public transport routing algorithm ϱ (e.g. CSA [Dib+13a] or RAPTOR [DPW12]). Our experiments are based on a multi-criteria shortest path algorithm applied to a time-dependent graph model as described in [DMS08c]. This includes speed-up techniques to utilize lower bounds for each criterion to goal-direct the search and apply dominance by terminal labels (independent of those mentioned in Section 1.4.5). Since it does not require any preprocessing, this system can be applied to dynamic scenarios involving delays, additional trains, etc., but still yields reasonable query processing times: on average about 300ms per query as stated in Table 1.5 on the dataset described below. The system handles interchanges, time-zones, coupling and splitting of trains, walking between stations, etc., and can therefore compute realistic optimal connections. We optimize travel time and the number of transfers. To consider the time difference between two connections, we use the travel time criterion from the running example (cf. Section 1.4.3, "Time Difference Dependent Domination Influence") for all experiments. The evaluation was conducted on an Intel® Core™ i7-6850K CPU. The C++ software is compiled with GCC 5.4.0 (optimization flags enabled).

We use a timetable of Germany provided by Deutsche Bahn AG spanning two weeks. The queries were generated for two weekdays in the middle of those two weeks. Thus, the search intervals of the filter search can become sufficiently large in both directions. The schedule contains 256,495 stations (7,651 train stations, 248,241 local traffic stops), 106,599 footpaths (station pairs together with a time required to walk from one to another), and 137,972,035 elementary connections (one vehicle departing in one station and arriving in the next station) part of 7,194,529 trips. For most users it is only reasonable to search for connections within a very limited time range. So we chose to evaluate time intervals of one hour. Source, target, and the departure interval were generated uniformly at random. We ran all evaluations with 1,000 queries.

Table 1.4: Comparison of the number of locally optimal connections (traditional approach) and globally optimal connections (our approach) within the departure time interval (1 hour). Rows with $\pm xh$ show the number of locally optimal connections remaining in the original search interval when the search interval is extended by x hours in both directions.

	#Connections Traditional	#Connections Our Approach	Diff	Percent
±0h	2.0397	0.5785	1.4612	28.36%
±1h	0.9647	0.5785	0.3862	59.97%
±2h	0.8491	0.5785	0.2706	68.13%
±4h	0.6774	0.5785	0.0989	85.40%
±8h	0.5868	0.5785	0.0083	98.59%
±12h	0.5816	0.5785	0.0031	99.47%

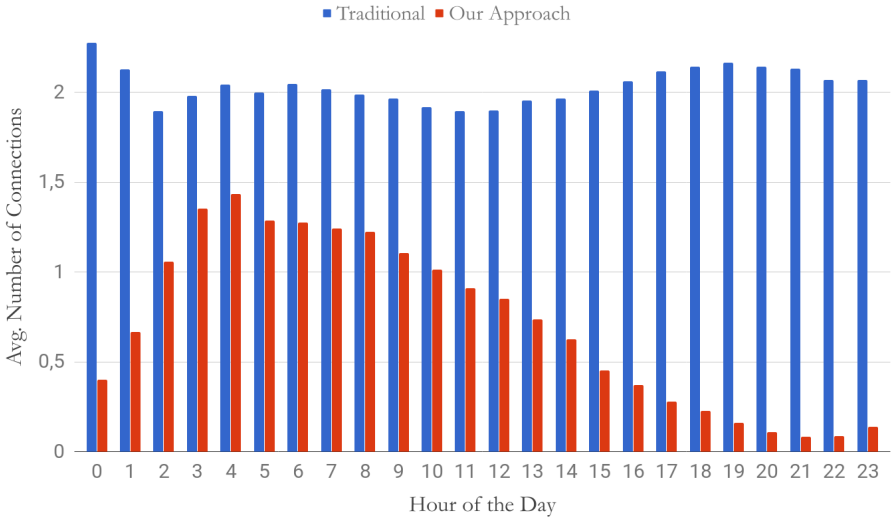
Table 1.5: Runtimes in Milliseconds (average, median, and 90% quantile) of the Initial Search for Different Departure Time Intervals. The base interval is one hour (i.e. $\pm 1h$ is a three hour search interval).

	Avg	Median	q90
±0h	294	178	729
±1h	568	370	1,283
±2h	849	565	1,899
±3h	1,385	898	3,184
±8h	2,505	1,557	5,898
±12h	3,663	2,250	8,931

Result Quality

As described in Section 1.4.1, our search addresses a common use case: the user cannot specify fixed departure time interval bounds. We compare the results from the traditional range query approach (all locally optimal connections from the time interval) and the approach presented in this paper (all globally optimal connections from the time interval). As shown in in Table 1.4, the traditional approach yields approximately two connections per query on average of which 0.58 are not only locally optimal but also globally optimal. This shows that a user has a great chance to pick a non-optimal connection if (s)he does not search for earlier and later connections. As we can see in Figure 1.7, the risk of getting suboptimal connections increases in the evening and night

Figure 1.7: Number of computed connections of the traditional approach (blue) and our approach (red) for each hour of the day. Departure interval size is one hour (i.e. from 12 am to 1 am).



hours.

A simple approach to overcome this problem could be to just extend the search interval by x hours in each direction (earlier and later) and only return connections from the original input time interval. As we can see in Table 1.4 and Table 1.5, this mitigates the problem with a trade-off between quality and response time (larger departure time intervals cause higher response times). Notably, extensions of 8 hours and more yield results that are close to the desired result. But as we will see in the next section, searching in the extended interval to eliminate suboptimal connections from the input interval is slower than the presented approach.

Response Times

Basically, the response time of a query comprises parts that are constant and others that depend on the query. The fixed part consists of calculating lower bounds for both criteria: 17ms for travel time lower bounds and 60ms for transfers lower bounds. So, in total, the response time is the actual computation time reported in the following plus 77ms.

The performance of the main algorithm introduced in Section 1.4.4 heavily depends on the performance of the auxiliary algorithm described in Section 1.4.4 and the query parameterization. As we can see in Figure 1.7, the number of iterations of the main algorithm (Section 1.4.4) depends on the time of day.

Since we have two distinct search steps in the algorithm described in Section 1.4.4, we report both times: the initial search time and the filter search time. The time required to respond to the query is the sum of these two times and the herein before mentioned fixed time of 77ms mentioned in the first paragraph.

Table 1.6: Runtimes in milliseconds (average, median, and 90% quantile) for the version without interval extension ($\pm 0h$, left table) and with an interval extension of 1 hour ($\pm 1h$, right table). The first row contains the runtime of the base version, the following rows contain the runtime of the algorithm with different speed-up techniques (ET = early termination, FIU = filter interval update, D = dominance with lower bounds, and $\hat{D}$ = dominance utilizing lower bounds and the *Time Difference Dependent Domination Influence* introduced in Section 1.4.3).

$\pm 0h$	Avg	q50	q90
base	37,062	3,047	73,233
ET	35,981	2,981	58,717
ET + FIU	35,257	2,762	50,498
ET + FIU + D	17,465	461	5,591
ET + FIU + $\hat{D}$	854	275	1,190

$\pm 1h$	Avg	q50	q90
base	4,104	1,940	7,239
ET	4,018	1,926	7,250
ET + FIU	4,078	1,915	7,207
ET + FIU + D	906	536	1,958
ET + FIU + $\hat{D}$	613	416	1,443

As shown in Table 1.5, the initial search takes 294ms on average (178ms median, 90% quantile is 729ms) for the input search interval of one hour. The computation time (of the initial search) increases to 568ms on average (370ms mean, 90% quantile is 1283ms) when we extend the search interval to three hours (1 hour initial search interval $\pm 1h$ in each direction).

Without Interval Extension The left hand side of Table 1.6 shows the runtimes for different variants of the algorithm without extension of the search time interval of the initial search: “base” is the basic version without any speed-up techniques. Here, we have very unattractive runtimes: ten percent of the queries take more than 73

seconds. With "ET" we refer to the speed-up technique described in Section 1.4.5, "Early Termination". Terminating early (when all connections from the initial search are dominated) yields slightly improved computation times (i.e. the 90% quantile can be improved by approximately 20%) that are still not viable for most practical purposes. For the label dominance based speedup technique (cf. Section 1.4.5, "Dominance") we evaluate two different versions: "D" denotes a simple version where we use the absolute lower bound values of the criteria; "$\hat{D}$" denotes a version that makes use of the time difference between two connections. As we can see, the application of these techniques yields significant speed-ups: even the simplified version (ET + FIU + D) is on average approximately twice as fast as the base version; the 90% quantile can be reduced to less than 10% of the base version. The more complex version $\hat{D}$ provides our fastest computation times (without interval extension) of 854ms on average. Half of the queries can even be answered in less than 275ms. Only 10% of the queries take longer than 1,190ms.

With Interval Extension As we can see in Table 1.5, the runtimes of the initial search reach the runtime of the sum of the initial search and the filter search already at an extension of two hours (849ms for the initial search with $\pm 2h$ time interval extension, 854ms for $\pm 0h$ in version ET + FIU + $\hat{D}$). Therefore, it is not reasonable to extend the interval by $\pm 2h$ or more. The runtimes for the $\pm 1h$ extension are shown on the right hand side of Table 1.6. Obviously, the filter search is much faster in the base version and most of the speed-up versions. The reason for this is that some unattractive connections are already discarded in the initial search and thus do not contribute to a large search interval for the filter search. The fastest version (ET + FIU + $\hat{D}$) requires 613ms on average. Interestingly, the median of this version exceeds the median of the fastest version without interval extension. This originates from the increased calculation time for the extended ($\pm$1h) initial search interval where the median is already 370ms (cf. Table 1.5) without the filter search.

1.4.7 Conclusion and Future Work

In this paper, we presented new problem definition and an algorithmic approach covering a common use case. The problem definition as well as the approach (including the

speed-up techniques) are generic and can therefore be adapted to a broad variety of scenarios. Our approach shows that, on average, less than one third of the results of the traditional range problem are relevant. Average response times of less than 700 milliseconds on a schedule containing all public transport within Germany (long distance as well as local traffic) were achieved. These are definitely times a user would accept.

In our experimental results we presented a version where the underlying algorithm was based on [DMS08c]. It would be interesting to analyze the response times if we base the approach on other routing algorithms such as RAPTOR [DPW12] or CSA [Dib+13a].

Start / Target	Station	Transport Mode	Duration	Costs
Start	Darmstadt Hbf.	Bike	15 min	0 Euro
Start	Darmstadt Hbf.	Car	5 min	2 Euro
Start	Darmstadt Ost	Car	10 min	4 Euro
Start	Darmstadt Nord	Car	8 min	3 Euro
Target	Berlin Hbf.	Taxi	11 min	14 Euro
Target	Berlin Spandau	Walk	15 min	0 Euro

Table 1.7: Example Input for the Public Transport Routing Algorithm: The previous individual transportation routing step has calculated the prices and durations shown in the table. For example, it is possible to reach "Darmstadt Hbf" by bike in 15 minutes (for free) or by car. The tradeoff is that the car is faster (five minutes) but not free (2 Euro).

1.5 Routing Algorithms

In the following, we describe multi-criteria routing algorithms that compute optimal solutions to the intermodal routing problem.

1.5.1 Multi Criteria Dijkstra

In the first phase, the system calculates every station that is a candidate for being the source or target station of an optimal connection. Since the user provided the coordinates and the means of transportation for start and destination, it is now possible to generate lists of stations that need to be considered for the respective means of transportation. A street routing algorithm is used for this purpose. This information is used as input for the second phase where we search for Pareto optimal connections from the start address to the destination address. A possible input for this phase is depicted in Table 1.7 (note: these lists can contain several hundred entries). As we can see, the input lists several possibilities to get from the source address to potential start stations and from potential target stations to the destination location. Each line contains the transport mode, the duration and the associated costs.

The time-dependent multi criteria shortest path algorithm is the essential core algorithm of our routing system. It takes the input format shown in Table 1.7. The algorithm

is a multi criteria version of Dijkstra's algorithm that finds Pareto optimal connections. The concept of Pareto optimality ensures that every user preference (weighting of the different criteria) is fulfilled. Thus, when comparing connections $A = (x_1, x_2, ..., x_n)$ and $B = (y_1, y_2, ..., y_n)$, A dominates ("is better than") B *iff* the following terms evaluate to true:

$$\forall i \in [1, n] : x_i \leq y_i \text{ and} \tag{1.2}$$

$$\exists j : x_j < y_j \tag{1.3}$$

The algorithm outputs a Pareto set of connections that cannot be dominated by any other connection (and are therefore Pareto optimal). In contrast to the regular Dijkstra algorithm, there can be many optimal solutions for one node. Consequently, we need to use labels (containing a vector of all applied criteria) instead of distance marks (containing only one criterion).
A generic version of the Pareto Dijkstra algorithm is shown in Algorithm 1: After the initialization where the start labels (generated from the input depicted in Table 1.7) have been inserted into the priority queue (which sorts in lexicographical ascending order of the chosen criteria) the algorithm iterates until the priority queue is empty. In every iteration the lexicographically smallest label is extracted. If this label has been dominated between its insertion into the queue and its extraction, we skip it. Otherwise, the algorithm iterates over all outgoing edges of the node the extracted label is located at and creates a new label at the target node of the edge ("$currentLabel.createNewLabel(edge)$"). This is done by adding the edge costs to the costs of the $currentLabel$, setting $newLabel.node$ to the head node of the edge and $newLabel.predecessor$ to the $currentLabel$ (this allows for a path reconstruction). Thereafter, all existing nodes of the head node of edge are iterated: if the new label is dominated by an existing label, the new label will be discarded. If the new label dominates an existing label, the existing label's dominated flag is set to $true$. This way, it will not be processed when extracted from the queue. When the algorithm finishes (because the priority queue is empty), it returns all labels that are located at the node representing the target station of the search. These labels represent the set of all Pareto optimal connections from the source node to the target

node.

ALGORITHM 1 : Pareto Dijkstra Algorithm

Data : V, E, startLabels, source, target
Result : Set of optimal terminal labels
$priorityQueue \leftarrow startLabels$;
for $v \in V$ **do**
    $labels(v) \leftarrow \begin{cases} startLabels, & \text{if } v = source \\ \emptyset, & \text{otherwise} \end{cases}$
end
while ***not*** $priorityQueue.empty()$ **do**
    $currentLabel \leftarrow priorityQueue.extractMin()$;
    if $currentLabel.dominated$ **then**
        continue;
    end
    for $edge \in \{(currentLabel.node, x) \in E \mid x \in V\}$ **do**
        $newLabel \leftarrow currentLabel.createNewLabel(edge)$;
        $dominated \leftarrow false$;
        for $existingLabel \in labels(edge.headNode)$ **do**
            if $dominates(existingLabel, newLabel)$ **then**
                $dominated \leftarrow true$;
                break;
            end
            if $dominates(newLabel, existingLabel)$ **then**
                $existingLabel.dominated \leftarrow true$;
            end
        end
        if ***not*** $dominated$ **then**
            $labels(edge.headNode) \leftarrow labels(edge.headNode) \bigcup \{newLabel\}$;
            $priorityQueue.insert(newLabel)$;
        end
    end
end

1.5.2 Preconditions for Subpath Dominance

Most multi-criteria range query routing algorithms eliminate unpromising subpaths during the search. It is important to consider important preconditions before discarding alternatives based on Pareto-dominance:

Let a and b be partial connections at the same node during the search. Regarding Pareto criteria a would dominate b. But we may only discard b if the following to conditions hold:

- a departs later than or at the same time as b: otherwise b may actually have a shorter travel time in case both connections arrive at the same time (which is still possible).
- a needs to arrive earlier or at the same time at the current node than b: otherwise, b may have the opportunity to take a faster train which a is not able to catch because a arrives later. This train may be the fastest possibility to reach the destination.

Thus, even if we apply a filter *after* the search, we need to use the criteria described above to dominate connections during the search.

1.6 Street Routing

Basically, the individual transportation mode routing is accomplished by the OSRM (Open Source Routing Machine) project [LV11]. In a previous step, the system has determined a set of possibly relevant stations (by using the Euclidean distance as an upper bound). Now, it is necessary to calculate exact distances and durations in order to initialize start labels and generate temporary virtual edges at the target and drop stations where the real duration exceeds the user defined maximum duration. To calculate the exact values, we can generate one query per station and means of individual transportation.

1.6.1 Many to One and One to Many Routing

As shown in Figure 1.8, we have two cases:

- **One to Many**: For the first part of the journey, we need to calculate exact distances and durations from the source location to every potential start station.
- **Many to One**: For the last part of the journey, we need to calculate distances and durations from every potential target station to the destination location.

Basically, individual transportation routing is currently carried out separately for each means of transportation (car, bike, etc.). Certain transport modes that are routed on the same graph (i.e. car, taxi, and getting a lift by somebody else) can be grouped. Thus, we can reduce the query count by reusing the results. But even when reusing the results, there are (depending on the actual user query) potentially hundreds of $1:1$ queries that need to be executed.

Therefore, we decide to develop a custom one-to-many and many-to-one contraction hierarchies [Gei+08] implementation applying the concepts from [Kno+07] that only calculates the durations and distances needed to proceed with the public transport routing. Thus, we need to search bidirectional. This part is similar to the $1:1$ search. From both directions the graph gets labeled by distance (in this case: duration) marks using Dijkstra's algorithm. Since OSRM provides us with a heap data structure, we call our queue "query heap". The forward direction can only use forward edges and the backward direction can only use backward edges (note: edges can be forward and backward edges at the same time here). Alternatingly, the forward search step and the backward search step gets executed (relaxing the edges of smallest entry of the query heap). If one direction hits a node that already got labeled by the other one, the sum of their corresponding entries represents a path. The search stops when the query heaps of both directions are empty. If the forward and backward heap did not meet at any node, there is no path between source and target. Otherwise, the result is the smallest sum of the values of forward and backward search as stated in [Gei+08]:

$$d(s,t) = min\{d(s,v) + d(v,t) : \textit{v is settled in both searches}\}$$

In order to implement a $1:N$ search, we now can just iterate the targets and employ $1:1$ searches. One important advantage is that we can reuse the forward query heap

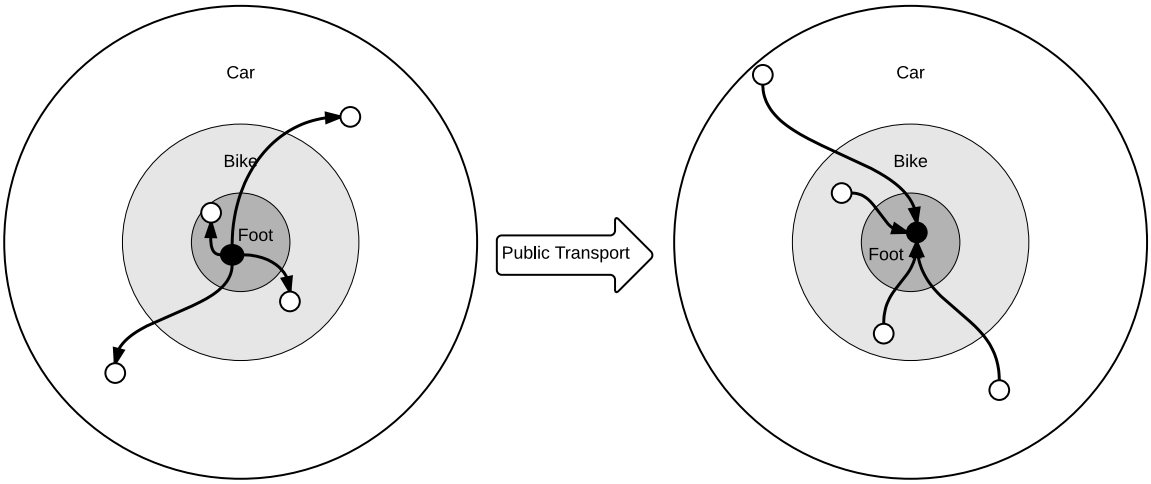


Figure 1.8: One to Many and Many to One Routing: At the source side the individual routing has one source location and many potential start stations: $1 : N$. For the last part of the journey we need the durations and distances between every potential target station and the destination location: $N : 1$ or "$1 : N$ backward".

in the $1 : N$ case and the backward query heap in the $N : 1$ case. This way, the "1" direction of both $1 : N$ or $N : 1$ does not have to do the same work over and over again (for each new target/source). When searching forward, the forward query heap can be reused. The backward heap gets cleared after each target and loaded with the initial entries of the next target. As described in the $1 : 1$ search case, the forward search is only allowed to use edges that are marked as forward edges and the backward search is only allowed to use edges that are marked as backward edges. In the $N : 1$ case there is only one backward search that reuses its query heap and the forward searches are iterated (without query heap reuse).

1.6.2 Evaluation

As depicted in Figure 1.9, the runtime of our $1 : N$ routing implementation (in red) scales linear with the number of sources / targets. The reason for this is that the loop over the targets ($1 : N$) / sources ($N : 1$) dominates the execution performance. In red,

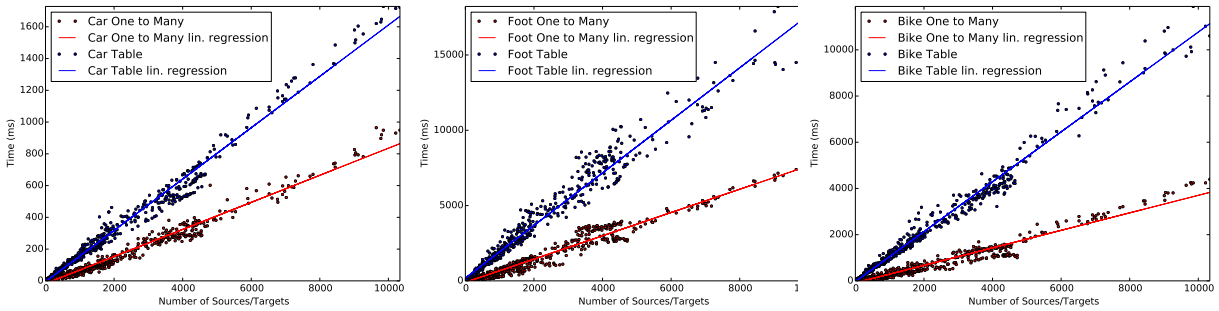


Figure 1.9: OSRM One-To-Many/Many-To-One Routing Performance: Comparison of Table-Routing and One-To-Many Routing regarding performance for a car graph, a foot graph and a bike graph.

we see the runtime of our implementation compared to the original table-routing plugin in OSRM.

1.7 Intermodal Routing

In this section, we describe the intermodal routing approach. The approach basically splits the journey into three sections: the middle section primarily consists of time-dependent modes of transportation (such as public transportation, ride sharing offers, etc.). Note however, that also non-time-dependent modes of transportation such as walking between stations are supported in this middle section. The first and the last section are the route from the start address/coordinate to the first station and from the last station to the destination address/coordinate. Here, primarily private transportation modes such as taxi, car, bicycle, walking, etc. are used. However, also time-dependent modes such as ride-sharing are supported for the first and the last section. The distinction of the three parts has mostly algorithmic reasons: the middle part is routed on a time-dependent routing model (e.g. time-dependent graph, RAPTOR [DPW12], CSA [Dib+13b], or TripBased Routing [Wit15]) whereas the first and the last part are routed separately. This enables us to use completely different (and incompatible) specialized data models and speedup techniques for the different segments of the journey. Street routing and public transport routing are inherently different [Bas09].

Therefore, splitting these parts to be able to apply approaches optimized for street routing (such as contraction hierarchies [Gei+08]) and public transport routing (such as RAPTOR [DPW12]) separately on their respective part of the journey, is reasonable. The routing results of the first and the last part are introduced as additional, query-specific edges into the time-dependent routing model. Note that, regardless of the separately routed first and last sections, we can guarantee a perfect result quality as long as the routing for the first and last part produces (additional) edges from the start address to *all* stations reachable within the time interval specified by the user and respectively (additional) edges to the destination address from *all* stations from where the destination is reachable within the time interval specified by the user to the destination address/location. Thus, if all potentially optimal subpaths of the first and last section are available as edges in the routing, the integrated routing is guaranteed to find all optimal journeys [BM09].

1.8 Dynamic Ride Sharing

This section is based on the paper *A Multi-Modal Routing Approach Combining Dynamic Ride-Sharing and Public Transport* by Sebastian Fahnenschreiber, Felix Gündling, Mohammad Keyhani, and Mathias Schnee (2016). It was accepted, presented, and published at the 43rd European Transport Conference (ETC), Frankfurt, Germany, September 28th–30th, 2015 (Transportation Research Procedia, 13, 176-183.).

Introduction On average 1.5 persons are utilizing one car [Fol+10]. Ride-sharing facilitates cheap and eco-friendly mobility and has the potential to improve the situation. Here, we concentrate on dynamic ride-sharing. In contrast to classical ride-sharing which basically works like a notice-board, dynamic ride-sharing allows a passenger to get a lift on a subsection of a driver's route and, if necessary, re-routes the driver. The new route proposed to the driver by the system may be totally different but not significantly longer than the route the driver originally planned to drive. On the one hand, this allows for more matches between driver and passenger(s). On the other hand, this makes it algorithmically challenging to integrate dynamic ride-sharing into the intermodal travel information system described in Section 1.7. In this section, we propose an efficient approach to compute optimal journey plans utilizing both, ride-sharing and timetable-based modes of transportation such as public transport. Furthermore, we extend this approach to allow for multi-passenger trips as well as real-time updates (i.e. creating, cancelling, and booking dynamic ride-sharing offers).

On a dynamic ride-sharing platform, drivers provide their routes and passengers specify queries consisting of departure and arrival location as well as a time for the journey. The platform computes suitable matches of driver routes and passenger queries, and proposes them to both parties.

State-of-the-Art platforms for public transport routing and dynamic ride-sharing provide unimodal connections but do not combine both transport modes. The challenge is that the driver routes are not static but could be changed significantly if the driver accepts the detour to pick up the passenger and drop them off at their destination. Thus, a driver's route may result in a high number of dynamic ride-sharing offers, namely all possible connections between pick up and drop off points with an acceptable detour for

the driver. In this section, we present a solution that integrates dynamic ride-sharing into the intermodal multi-criteria travel information system presented in Section 1.7. We solve two challenges: First, we allow dynamic ride-sharing between two train rides by connecting public transport stations by dynamic ride-sharing offers of drivers. For this, we integrate driver offers into our data model, which represents the public transport timetable. Second, we find suitable dynamic ride-sharing offers of drivers who can take the passenger from their start location to a public transport station or from a station to their destination location. In our computational study, we obtain a significant improvement of the results by combining public transport and dynamic ride-sharing compared to unimodal train connections.

The computational study was conducted with data provided by the startup *flinc GmbH* in 2015 which publicly provided a free dynamic ride sharing service until January, 2019. At the time of writing, flinc is part of the *Daimler AG* [AG18] and provides its service to offer ride sharing to employes.

This section is based on the peer-reviewed publication [Fah+16] which was presented by Felix Gündling on the 43rd European Transport Conference (in 2015), held in Frankfurt, Germany.

Motivation Current and future mobility requirements demand intelligent solutions. Ride-sharing is gaining in importance, and yields benefits compared to costly, eco-unfriendly individual car rides. Dynamic ride-sharing makes better use of existing resources by bringing travelers with matching routes together. Participating drivers and passengers can be matched on-demand or in advance. This is done by calculating the detour required for the driver to give the respective potential passenger a lift. If the result matches certain criteria, both the driver and the potential passenger receive an offer and decide whether to accept or decline.

Our intermodal multi-criteria routing system computes door-to-door connections by incorporating public and private transportation as described in Section 1.7. In this work, we present an approach to fulfill future individual traffic demands by proposing a journey information system, which adds dynamic ride-sharing to our routing system.

The motivation is that both these modes are complementary: On the one hand, public transport is quite sparse in some regions, but dynamic ride-sharing offers to reach areas with limited public transport connectivity. On the other hand, the number of dynamic

ride-sharing offers relevant for a query increases when allowing routes which bring the passenger to a station where they can continue their journey using public transport.

This way, dynamic ride-sharing as well as public transportation benefit from this combination: by offering dynamic ride-sharing to travelers using public transportation, the car utilization can be improved (reducing costs for the driver). Furthermore, the result quality of timetable information systems can be enhanced by additional optimal connections which use dynamic ride-sharing.

State of the Art There has been extensive research regarding ride-sharing. An overview is given in [Fur+13] which distinguishes different patterns based on the number of passengers (single/multi) and the potential matches. These range from *Identical Ridesharing* (named "perfect fit" in [DL13]) which requires start and destination to match exactly, over *Inclusive Ridesharing* and *Partial Ridesharing* which allow sharing a ride on a subsection of the drivers (fixed) route, to *Detour Ridesharing* (named "reasonable fit" in [DL13]) which allows pick-up and drop-off of the passenger to be at arbitrary locations. The definition of *Detour Ridesharing* is equivalent to what is commonly referred to as "dynamic ride-sharing".

Many publications consider the problem of optimal matchings between passenger and driver (e.g. [Sch+16; ÖW20; Pel+15]) based on their routes. In practice (e.g. at blablacar, the world's largest carpooling community [bla18]), users take many more aspects (e.g. the drivers/passengers rating on the platform, gender, etc.) into account when requesting a ride. An extension of matching one passenger to one driver is the multi-hop case [DL13] where a passenger uses multiple rides to reach the destination. There has been research that shows that meeting points are beneficial for more and better matchings [Sti+15].

The current state of the art regarding research in intermodal travel information systems and routing algorithms is presented in Chapter 1.1. To the best of our knowledge, non of the existing approaches support combining (dynamic) ride-sharing with other means of transportation such as public transport. This also applies to the available commercial systems.[4]

[4]Such as http://www.fromatob.de/ and https://www.qixxit.de/en/

Our contribution In this work, we address two problems to combine dynamic ride-sharing (aka *Detour Ridesharing*, the most complex case described in [Fur+13]) and public transport: connecting public transport stations by dynamic ride-sharing connections and connecting start and destination of a query to public transport stations by dynamic ride-sharing routes. To solve the first challenge we add station-to-station connections that are derived from dynamic ride-sharing offers to our graph representation in a preprocessing step. These connections can be updated when matches are agreed upon, offers are withdrawn, or new ones are available. The second challenge requires the selection of suitable ride-sharing offers that allow for a connection from the start location to a station or from a station to the destination location. Since the source and destination address are given in the individual queries and are not known beforehand, in contrast to the set of stations, this step needs to be carried out on-the-fly for each query. Our integration also enables "multi-hop" ride-sharing (a passenger utilizing multiple ride-sharing offers) described in [DL13].

Travel Information System We briefly recall the properties of the travel information system *TIS* this extension is based on. The following description is simplified but contains every information required to understand the ride-sharing integration. A detailed description can be found in Section 1.7.

The resulting journeys are Pareto-optimal regarding the optimization criteria of travel duration and number of interchanges. All computed journeys share the following structure: first, from the start location, private transportation is used to reach a station of the public transportation network. Second, with public transportation the passenger travels to a station near the destination. Last, from there the destination location is reached by private transportation again.

Primarily, TIS computes connections in the public transportation network including potential walks between stations which are close to each other. For the private transportation parts of the multi-modal connections, we use a third-party routing service, OSRM [LV11].

Here, we briefly explain the time-dependent graph model in TIS. This simplified description follows Disser et al. [DMS08b]. In the timetable, there is a set of transports $\mathcal{T}$ and a set of stations $\mathcal{S}$. Each transport $t \in \mathcal{T}$ consists of a set of elementary connections $\mathcal{E}(t)$. Hence, an elementary connection models a train run from a station to another

station without intermediate stops. The set of all elementary connections of all trains is denoted by $\mathcal{E}$. In a time-dependent graph $G = (V, E)$, for each station $s \in \mathcal{S}$, there is a *station node* $v_s \in V$ in the graph. There is an edge $e = (v_a, v_b)$ between two station nodes $v_a, v_b \in V$ if there is at least one elementary connection $c \in \mathcal{E}$ from station $a \in \mathcal{S}$ to station $b \in \mathcal{S}$. For a correct modeling of interchanges the graph contains additional node and edge types which are skipped here. A full description can be found in [DMS08b].

Each edge $e = (v_a, v_b) \in E$ has a duration and a cost. The duration is time-dependent and is determined during the search: If the edge is used at time τ, the duration of the edge equals the difference between the arrival time (at the head station of the edge) of the first connection departing after τ and time τ.

There are also *foot* edges in the graph to allow walkings from a station to other stations within walking distance. These edges are not time dependent but return a constant edge cost representing the time required to reach the other station by foot.

1.8.1 Dynamic Ride-Sharing

In this section, we will introduce our profit model for drivers, and describe the scenarios "connecting public transport stations" and "connecting start and destination of a query to public transport stations" by dynamic ride-sharing routes. In the first scenario, stations in our public transportation graph are connected to each other. In the second scenario, an address provided by the user is connected with a public transport station.

As shown in Figure 1.10, a driver provides a route from a *start* location to a *destination* location together with his departure time. Passengers are interested in a ride that does not necessarily start and end at the driver's start and destination. Therefore, a passenger has to be *picked up* and *dropped off* at his / her start and destination location. The driver may accept the detour in order to give the passenger a lift. She/He is rerouted to visit the pick up and drop off locations. The resulting times at these locations determine the availability of the dynamic ride-sharing connection and act as a schedule. To account for delays caused by heavy traffic, a safety margin is added to the duration of all ride-sharing connections, where the drop off location is not the final destination of the passenger.

The passenger bears part of the driver's costs. We assume a reasonable driver who maximizes the profit as defined in the following equation. By "dur_detour", we denote the difference between the duration of the original journey (from driver's start to driver's

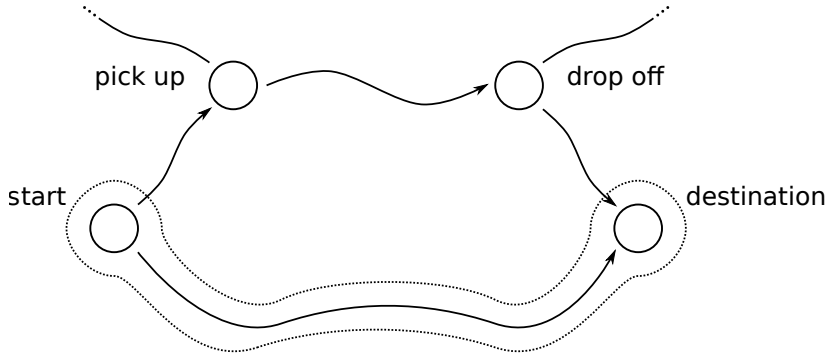


Figure 1.10: Rerouting a driver. The driver's journey is from *start* to *destination*. A potential dynamic ride-sharing match reroutes the driver via *pick up* and *drop off*. Note, that pick up and drop off do not need to be located close to the original route (dotted area) of the driver as long as the constraints regarding detour and profit hold.

destination) and the duration of the route via pick up and drop off. For the difference in distance we define "dist_detour" analogously. By M, we denote the cost per kilometer the driver has to pay for upkeep and gas. An hourly wage, denoted by W, compensates for the time required to take the detour.

$$\text{driver\_expenses} = \text{dist\_detour} \cdot M\,{}^{\text{ct}}\!/_{\text{km}} + \text{dur\_detour} \cdot W\,{}^{\text{ct}}\!/_{\text{min}}$$
$$\text{profit} = \text{passenger\_contribution} - \text{driver\_expenses}$$

We define an upper bound for the detour duration and distance and a lower bound for the driver's profit. Ride-sharing connections that do not satisfy these bounds are skipped since they are not attractive for drivers. In order to reduce the effort required to find reasonable reroutings, we conduct pre-computations using the straight line distance. Further calculations can be skipped if the results of those pre-computations already do not match the driver's criteria.

Supplement Scenario

Our system enables ride-sharing connections from one public transport station to another. After this ride, the passenger continues her/his journey with another ride-sharing

connection, public transportation or an individual transport to their destination location. Using an additional service "on-the-fly" during the search does not yield reasonable query times. For performance reasons the internal graph representation needs to be extended to contain all reasonable potential ride-sharing connections. Ride-sharing can therefore be seen as supplement to public transportation. In the following, this use case is referred to as *supplement* scenario. The driver has an own "schedule" and every potential pick up and drop off location combination can be seen as an elementary connection. Therefore, all reasonable routes from pick up to drop off stations can be added to the set of elementary connections. The time-dependent graph model as well as the RAPTOR and TripBased models require the connections to be grouped into routes. This also enables the algorithms to enforce minimum transfer times between routes. This property can also be used to enforce minimum transfer times between public transport and ride-sharing as well as between two consecutive ride-sharing offers in the multi-hop case.

One advantage of the supplement case is that all data is available statically beforehand. Therefore, we can preprocess all available driver's routes (start, destination, and time) in order to integrate all reasonable dynamic ride-sharing connections into the backbone graph. This saves valuable time when calculating optimal journeys for the user. The sheer amount of public transport stations leads to many potential pick up / drop off station pairs. Many smaller stations only lead to marginally different connections. Therefore, we decided to focus only on those stations that are served by trains (not only busses, trams, or subways).

Ensuring Subpath Optimality In general, all routing data models but the time-expanded graph and the Connection Scanning Algorithm require elementary connections to be grouped in routes. All other routing algorithms such as the (Pareto-)Dijkstra on a time-dependent graph, the RAPTOR Algorithm or TripBased Routing generally only consider the first departure from each route edge. Therefore, it needs to be the case that all further elementary connections on this edge cannot yield an optimal journey.

If we group the elementary connections derived from ride-sharing offers by their departure and arrival station, this property does not necessarily hold. There are two reasons for this: there can be elementary connections that are not optimal at all and there can be multiple optimal connections that need to be considered to find all Pareto-optimal

journeys.

When considering only the travel time criterion, there might be elementary connections that overtake each other. The reason for this is that the 20% reliability margin is applied to the whole route of the driver until drop-off of the passenger. Since not all drivers have the same initial starting location, this margin value can be different for each ride-sharing offer. Therefore, the travel duration from one station to another is not necessarily the same for each ride-sharing offer and overtaking is possible. This problem can be solved by removing all dominated elementary connections, i.e. those that depart earlier and arrive later than at least one other connection.

The second scenario, where more than one connection needs to be considered, can only happen in the multi-criteria case. The two most relevant criteria for ride-sharing are travel time and price. As noted before, the travel-time can be different for each elementary connection (based on the 20% reliability margin). This also applies to the price (which also depends on the detour length and duration). Thus, we can have several Pareto-optimal elementary connections (e.g. 2 hour / 2€ vs. 3 hours / 1€). Therefore, more than one route edge is required where each route edge only contains non-"conflicting" elementary connections. Ideally, the number of routes should be minimized to keep the graph as small as possible because the size of the graph impacts the routing performance. The perfect solution would be to compute all pairs of conflicting connections (i.e. mutually non-dominating connections where the one departing earlier is more expensive). Creating a conflict graph from those pairs (each pair can be interpreted as an edge), the problem to minimize the number of routes can be described as coloring problem with an unknown chromatic number (i.e. the number of routes). Since solving this problem is very computationally intensive, we decided to apply a fast greedy algorithm which iterates all elementary connections for one start/destination pair and moves conflicting connections to a new route.

Door Scenario

In case of connections from or to user provided locations (including direct connections from start to destination) we need to gather ride-sharing matches on-the-fly. Those calculations cannot be carried out beforehand and are at the same time time-critical because the user is waiting for the system to respond. Since this is a special case of

"*door* to *door*" routing (*door* to station or station to *door*) we refer to this case as the *door* scenario.

The basic approach is to gather all potentially useful ride-sharing offers that could take the passenger from their provided start location to a public transport station or from a public transport station to their provided destination location. All these connections are then temporarily integrated into the graph. The algorithm then finds connections that contain these rides if this yields an optimal connection.

It is beneficial for the system's performance to make use of data generated in the supplement preprocessing. We assume that the positions of all previously selected public transport stations are about evenly distributed. For our experiment with German data, this is the case. If this would not be the case, additional "virtual" stations would need to be generated. By reusing the routes computed in the preprocessing for the *supplement* case, we are able to select relevant ride-sharing offers very fast. Ride-sharing offers that could take the user from a public transport station to their destination location can easily be determined by gathering the rides with a drop off station in the area of the provided destination location. As shown in Figure 1.11 only the driving distances from the offer's start location via the user's location to the meeting point[5] need to be computed on-the-fly. Relevant ride-sharing connections from the user's start location to public transportation stations can be determined analogously.

The direct connection case can be seen as a special form of the *door* scenario. All ride-sharing offers that have pick up stations near the provided start location and drop off stations near the provided destination location are suitable to enable a direct ride-sharing connection if the departure time is contained in the user provided interval.

Additionally, temporal restrictions apply which can be utilized to further reduce the number of offers to consider as an additional search edge: only ride sharing offers that can be used at the start address after the earliest departure time provided by the user (potential passenger) and arrive at the destination before a maximum travel time (e.g. 24 hours would be a reasonable maximum travel time for journeys within Germany) are relevant.

[5] for example parking areas or easily accessible places

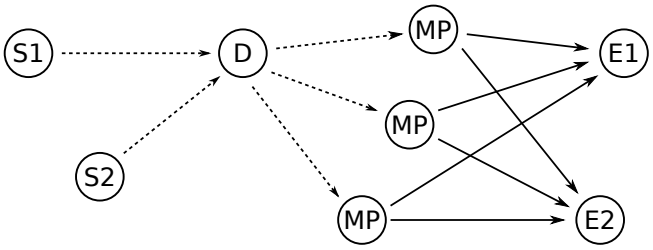


Figure 1.11: Rerouting the driver in the door scenario: Example situation for the passenger's departure **D**. Two offers (**S1** to **E1**, **S2** to **E2**) and three meeting points **MP** have been identified. Dotted distances need to be routed on the fly, solid ones are known from a preprocessing step.

1.8.2 Computational Study

Our prototype of the information system is implemented in C++ and evaluated on an Intel Xeon 3.4 GHz[6] quadcore CPU with 32 GB main memory. The public transport schedule were kindly provided by the German railway company, Deutsche Bahn, and contains all long-distance and regional trains in Germany as well as busses, trams, and underground trains of selected transportation authorities. The schedule contains 106,000 stations which results in 1.6 million nodes in the graph. Furthermore, the graph contains 6 million edges, where 1.4 million model train operations and the rest enables interchanges or walks between stations. The 6,455 long distance travel stations[7] in the schedule are selected as meeting points. The overall time window for the schedule was restricted to two weeks.

In our evaluation, we used real dynamic ride-sharing offers and real customer queries kindly provided by the dynamic ride-sharing service flinc. The data is anonymized to protect customer privacy: all coordinates are rounded to two decimal places. The dataset contains about 2,500 offers and more than 2,000 queries. The offers and queries are either one-off or defined by an interval and a weekly recurrence pattern. For our

[6] Intel(R) Xeon(R) CPU E3-1245 V2 @ 3.40GHz
[7] every station served by trains; not only busses, trams, and subways

evaluation, we mapped the queries and offers to the corresponding weekdays in our time period of two weeks. This results in 10,000 (temporally) unique offers and 7,000 queries. Street-routing is provided by OSRM [LV11], which operates on data from OpenStreetMap[8].

Preprocessing The approach for the supplement scenario relies on preprocessing, where street-routing has the largest performance impact. However, we profit from the fact, that the backbone of our system, the public transport stations, is static. Therefore, we choose to compute all possibly required distances[9] in one go and perform the actual processing of route approval according to the driver's profit model and edge generation afterwards.

The street-routing step is completed in just under two hours for the complete evaluation dataset, while adding another offer would take only ten seconds. The actual preprocessing runs in one minute and ten seconds (thereof 25 seconds for routes approval, 42 seconds for edge generation).

Evaluation Approach The computational study focuses on the extension of our existing information system with ride-sharing. As baseline, we configure our system with the public transport schedule and walkings up to 15 minutes from the start to the "first station" and from the "last station" to the destination. The evaluation is carried out in four different configurations: *no ride-sharing* at all; ride-sharing in the *supplement* scenario; ride-sharing in the *door* scenario; the *full system*, with both scenarios. Two questions are examined: How is the result quality affected in the various scenarios and what are the performance figures that directly impact the user experience?

In order to assess the result quality, the query set is partitioned: The first subset contains all queries for which the baseline delivers a connection. These are 81.6% of all queries, and they will be used to judge whether combining ride-sharing with public transport results in connections that are better compared to connections that only use public transport. In the rest of this section, we denote such better connections as *better alternatives*.

[8] Project OpenStreetMap, `https://www.openstreetmap.org`

[9] all offer starts to all meeting points, all meeting points to all offer ends, and the full meeting point distance matrix.

	criterion: duration		criterion: transfers	
	% better	avg. impr.	% better	avg. impr.
suppl.	6.0%	26.4 min	28.0%	1.18
door	21.3%	86.7 min	37.1%	1.27
full	22.3%	83.4 min	44.4%	1.33

Table 1.8: Improvements achieved by including dynamic ride-sharing: Rows show the improvements over the reference system without dynamic ride-sharing. Columns "% better" give the percentage of queries with better results. Columns "avg. impr." show the average improvement over these responses.

For other 14.7% of all queries, the baseline does not deliver a connection. For them, combining dynamic ride-sharing with public transport is a chance to *enable journeys* in the first place. However, for the remaining 3.7% of the queries no matching offers are available, which marks them as the unanswerable upfront. These queries are not considered in the rest of this study.

Better Alternatives In this evaluation, the information system processes the first subset of the query set, queries that can be answered by the baseline. Table 1.8 presents the improvements, when ride-sharing is included in the different configurations.

It has to be noticed, that the door scenario yields significantly more improvements than the supplement scenario. Yet, even for the supplement scenario, we could lower the minimum number of required interchanges for over a quarter of the queries by 1. In the door and full scenarios the fastest connection could be improved in over a fifth of the queries, on average by more than 80 minutes. For over a third of the queries, we were able to save one interchange on the connections with the least number of interchanges.

On the one hand, the high saving potential results from a number of moderate improvements. On the other hand, some results, where the information system barely found a connection without ride-sharing[10], benefited significantly from the new possibilities, and were turned into highly attractive travel opportunities. This shows that a system incorporating dynamic ride-sharing improves the overall quality of the connections offered to the users.

[10]Especially, when using door-scenario connections in the late hours of the day and to bridge areas with only sparse near distance transportation.

	new results	high quality
new in suppl.	2.2%	1.0%
new in door	94.1%	33.1%
new in full	94.3%	33.3%

Table 1.9: New results and their quality: Column “new results” shows the amount of newly enabled journeys. Column “high quality” gives the percentage of journeys with at most two times the car travel duration.

Enable Journeys In this evaluation, the information system uses ride-sharing to enable journeys, which are impossible with public transport alone (second subset of the query set). Table 1.9 confirms that this endeavor is successful in many cases. In the range-extending door-scenario 94% of these queries can now be answered. Unfortunately, the supplement scenario does not fare well at all.

To assure that the newly possible journeys are valuable to the passenger, their travel times were compared to twice the time required to travel from the start to the destination by car. For about a third of the queries the fastest connection does not exceed this bound and, thus, is considered high quality.

Information System Performance Table 1.10 shows the processing times in the different configurations. Either the supplement or the door scenario roughly double the computation time. The supplemental scenario is noticeably faster. The combination of both is slightly slower than three times the time required by the baseline version.

All in all, the performance impact of the various scenarios is definitely noticeable by the user. However, the average processing time of under seven seconds for the full configuration stays within reasonable bounds.

	none	suppl.	door	full
average [s]	1.8	3.5	4.1	6.2
90% quantile [s]	2.5	7.2	6.1	11.4

Table 1.10: Computation times of the graph search in the various scenarios.

1.8.3 Conclusion

We presented and evaluated the integration of dynamic ride-sharing into our multi-modal travel information system.

The extended system is capable to find optimal connections combining public transport and dynamic ride-sharing. These connections use dynamic ride-sharing either between public transportation (supplement scenario) or from the start or to the destination address (door scenario). The computed connections are Pareto-optimal in terms of travel duration and number of interchanges, while the price is also taken into account. Within reasonable computation time, we find additional attractive connections. Either alternatives to connections consisting only of public transportation or new connections for queries that could not be answered successfully without dynamic ride-sharing. Many of these connections are able to compete with individual modes of transportation.

While we have shown that ride-sharing is a valuable addition, its success depends on the available offers. The following aspects warrant further investigation: First, including offers with via locations and allowing multiple passengers per car, which introduces new constraints on the driver's schedule and driver-passenger matching. Second, implementing an online mode, which re-routes the driver during an ongoing journey. This could increase coverage and even might allow to reroute passengers on their public transport journey in case of connection failures.

1.9 Personalized Routing for People With Disabilities

This section is based on a translation of the German paper *Eine personalisierte multikriterielle multimodale Verbindungsauskunft für Menschen mit Mobilitätseinschränkung* by Sebastian Fahnenschreiber, Felix Gündling, Pablo Hoch und Karsten Weihe (2020). It is accepted at the HEUREKA 2020 which is postponed to April 2021.

According to the World Health Organization (WHO), approximately 15 percent live with a disability [Org11]. This often limits the options for social participation. One important aspect of social participation is mobility: even when it is theoretically possible to travel from one location to another, it might be hard to find this journey plan in practice. We present a personalized approach for computing intermodal connections that tackles this problem. At first, we describe a personalized pedestrian routing for people with disabilities which is then integrated into the intermodal travel information system introduced in Section 1.7.

First of all, we distinguish between "hard" limitations (e.g. wheelchairs are not well suited for stairs) and "soft" limitations: someone with a knee problem would probably like to avoid stairs if possible. However, probably not at any price: depending on the length of the detour (without stairs), it might be possible that using the stairs could be the better option.

Different preferences resulted in the broad usage of multi-criteria optimization algorithms to compute optimal journeys. This is especially true for public transport where both, travel time as well as the number of transfers are relevant for most passengers. This is true also for intermodal travel information systems where one might also want to minimize other criteria like the walking distance. Computing the whole Pareto set is the default in this area.

The approach presented here extends the multi-criteria approach by a new optimization criterion which quantifies the difficulty (of journey segments) depending on the personal profile of the traveler. Thus, we assign a difficulty value to every segment of the routing network based on the user's profile. This includes stairs, crossing of streets, inclines, and more potentially problematic sections. The difficulty value is freely configurable based on the properties of the obstacle. For instance, the difficulty value for stairs can depend on the number of steps, the direction (up or down), and whether

there are handrails available or not. For compatibility reasons, low values represent a low difficulty (and vice versa). This enables us to minimize all objective values (travel time, number of transfers, and difficulty) at the same time.

The pedestrian routing also supports street crossings. This requires a data model which distinguishes two street sides. For every unmarked street crossing, the shortest detour via a marked crossing (e.g. a pedestrian crossing or pedestrian lights) is computed in a preprocessing step. This value is then stored for use at query time. Unmarked street crossing edges in the routing graph are only considered feasible if the detour length exceeds the maximum detour length for unmarked crossings configured in the personal user profile. This enforces the usage of a nearby marked crossing if the detour is acceptable. The maximum detour length can be set separately for each street type (e.g. a living street can be more easily crossed than a country road). Alternatively, the usage of unmarked crossings can also be disabled completely for each street type. With this approach, we aim to be able to resemble the behavior of most pedestrians.

To enable the computation of fully accessible journey plans for people with disabilities depending on their profile, we integrate this pedestrian routing into the intermodal travel information system introduced in Section 1.7. The pedestrian routing will be utilized for every footpath contained in the journey. This includes footpaths between tracks for interchanges between different vehicles of the public transport as well as the first and the last section of the journey (from the start location to the first public transport station and from the last public transport station to the destination location). Routes that contain non-accessible vehicles (based on the personal user profile) are excluded from the search.

The employed data model (time dependent graph) can be updated with real-time updates like delay updates, cancellations, additional services, track changes or reroutings. This enables us to compute real-time alternatives based on the current situation. We extend this approach to also support elevator failures as well as changes of the accessibility of the vehicles provided by the transportation company. This way, the system is capable of providing real-time alternatives for affected people with disabilities in those cases.

The computational study based on the real German public transport timetable (including trains, buses, trams, subway and metro trains) and OpenStreetMap data shows that this approach is feasible even for complex nationwide transportation networks.

1.9.1 Related Work

Many approaches do not respect the disabilities of the user. The overview of the State-of-the-Art research regarding route planning (on roads and public transport timetables) by Bast el al. [Bas+16b] does not mention this topic. A first multi-criteria approach is described in [VW08]. However, it does not compute the Pareto-set but combines all criteria in a weighted sum. Thus, the user cannot chose from the whole set of trade-offs. The approach presented in [SM06] is capable of handling certain "hard" restrictions. To the best of our knowledge, there is no multi-criteria approach (in the sense of computing a non-extensible Pareto set) that considers "soft" restrictions (avoidance of obstacles as a trade-off - as mentioned above) modeling it as an additional Pareto-criterion. Weyrer et al. [WHP14] present a intermodal pedestrian routing for people with disabilities based on the OpenTripPlanner software platform. Their evaluation based on the data of the city of Villach (Austria) was carried out using two routing queries. Unlike our approach, they do not consider personalized transfer times from track to track. Müller-Hannemann and Schnee introduce a real-time timetable information system in [MS09] which incorporates real-time updates such as delay data, train cancellations, additional trains.

1.9.2 Contribution

This section presents a novel approach to accessible intermodal door-to-door routing. In particular, the integrated optimization using the additional difficulty Pareto criterion which includes the ways used for transfers has not been part of any publication yet. To the best of our knowledge, there is also no real-time routing available (neither commercial nor publicized research) which is capable of handling elevator failures, track changes or changes of the accessibility of public transport vehicles employed by the transportation company. None of the existing approaches to accessible routing have been evaluated on a dataset of the size of the German transportation network.

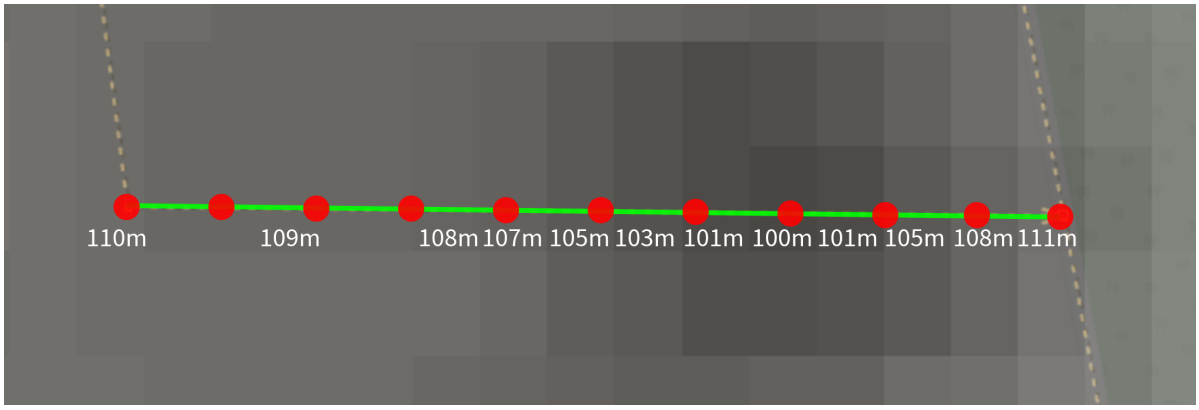


Figure 1.12: Example for Height Sampling of an Edge: the height is depicted in grey scale (background). The edge (green) with grid points (red) has a height difference from start to destination of 1m whereas the total height difference is -10m and +11m.

1.9.3 Pedestrian Routing

Preprocessing

The pedestrian routing for people with disabilities is based on OpenStreetMap (OSM) data. OpenStreetMap data consists of nodes, ways, and relations. The first step is to extract all ways and areas (e.g. public squares) that are open for pedestrians and transform them into a directed graph.

Simplification of the Routing Graph and Generation of Sidewalks To reduce the number of nodes and edges, we remove all nodes with a degree of two in each direction. Both edges are replaced by one. For display purposes, we keep the coordinates of all removed nodes on the replacement edges. Depending on the attributes of the way from OpenStreetMap, one or two sidewalk paths are created. OpenStreetMap ways can carry the information that the left, right, or both sidewalks are missing.

Height-Profile The elevation profile of a footpath is important especially for people with disabilities. Since OpenStreetMap data does not contain exhaustive terrain elevation data, we need to look for other data sources. Elevation profiles are often stored as

rasterized data. Our system supports reading data from the publicly available *Shuttle Radar Topography Mission* (SRTM) dataset provided by the NASA[11] (National Aeronautics and Space Administration) as well as other input data which uses the same format. An intuitive approach would be to look up only the height of the routing graph nodes and to use the height difference between two nodes as the edge attribute However, as depicted in Figure 1.12, this can be misleading: an edge might contain several slopes. Thus, the exact height profile is being determined by sampling along the course of the edge. This way, the precise geometric topology of the edge can be reproduced. We compute and store two values per edge: the ascend and the decline.

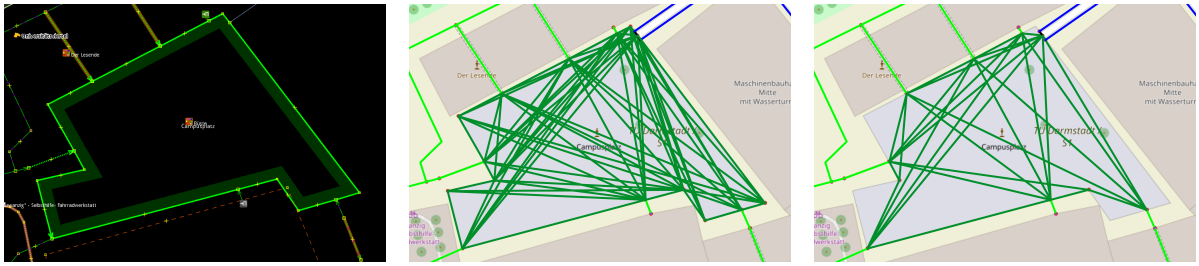
Combining Multilane Streets Different lanes of the same street or highway are often represented as separate ways in OpenStreetMap. This is mostly the case for highways but can also occur for larger city streets. Each OpenStreetMap way representing a lane is marked as one-way street. In this case, our preprocessing approach would create sidewalks for each lane which is undesirable. To prevent this, we introduce a check to efficiently find (more or less) parallel one-way streets (on the same map layer - i.e. not one underground and one overground or vice versa) that are within close proximity of each other.

Street Crossings Due to the conditional sidewalk generation and the distinction of the sidewalks on both sides of the street, it is now possible to implement street crossings correctly for the routing. Basically, we distinguish between marked and unmarked street crossings. Depending on the user profile, unmarked street crossings can either be used if the length of the detour via the marked crossings exceeds the maximum detour length (configured in the profile) or can be marked completely infeasible if the computed routes should not contain unmarked crossings at all. Every setting (maximum detour length and whether unmarked crossings are feasible at all) is separately configurable for each street type. In the preprocessing, we determine for each unmarked crossing the shortest path (using Dijkstra's algorithm [Dij+59]) to the destination of the unmarked crossing (on the other side of the street) using only marked crossings. This distance is then stored as the detour length attribute for the unmarked crossing. The actual routing

[11] Source: https://www2.jpl.nasa.gov/srtm/

algorithm then decides on whether this crossing can be used by comparing the stored detour length with the maximum detour length for the respective street type stored in the user profile.

Places To be able to compute optimal paths through areas efficiently at query time, we precompute shortest paths over all places of the dataset. Here, we consider every border node which is connected with streets, footpaths, as well as further places. To compute paths over non-convex places and places which contain inner polygons (e.g. fountains) we employ the concept of the Visibility Graph [LW79]. All nodes which do not have an obstacle for the direct sight connection are connected with each other by an edge. An example is depicted in Figure 1.13c; the initial input is shown in Figure 1.13a. In the next preprocessing step, we use the Floyd–Warshall algorithm [Flo62] to compute shortest paths between all pairs of border nodes on the previously built graph of sight connections. Based on Bellman's optimality principle all sub-paths of Pareto-optimal paths are also Pareto-optimal paths [BM09], we can eliminate every edge of the visibility graph which is not part of any shortest path found by the all-pairs shortest path algorithm. This results in a reduced graph as shown in Figure 1.13c. Reducing the graph improves the memory footprint of the routing at query time and improves the runtime of the final routing algorithm. The edges of this reduced visibility graph are then integrated into the final routing graph.



(a) Input Polygon from OpenStreetMap

(b) Fully Connected Visibility Graph

(c) Reduced Graph for Routing.

Figure 1.13: Processing steps for non-convex places.

Routing

In this section, we discuss the routing algorithm which takes the routing graph generated in the preprocessing (discussed in Section 1.9.3), a user profile, as well as start and destination coordinates. The algorithm can route either from one location to many other locations using a single-source shortest path (SSSP) algorithm or from many source locations to one destination location using a SSSP algorithm (one to many with reversed edges). The algorithm terminates as soon as all target nodes have been extracted from the queue.

Deriving Graph Nodes from Start and Destination Coordinates First of all, the coordinates provided by the user (e.g. by clicking on a map or entering a street address) need to be mapped to a routing graph node to initialize the shortest path (SSSP / SDSP) algorithm. Choosing the routing graph node that has the smallest Euclidean distance to the given coordinate may result in absurd looking routes where the first and/or last segment is a (short) detour in the wrong direction. To prevent this, the source and destination location (which can be seen as a "virtual node" as these locations are not contained in the routing graph) are connected with the closest edge by dropping a perpendicular from the coordinate provided by the user to the vector induced by the two endpoints of the edge. This distance has to be added to the shortest path. If a provided coordinate is located inside a place area, the virtual node representing this coordinate will be connected with all "visible" [LW79] corners of the polygon. If the coordinate is outside but nearby a place, it will be projected to the closest point on the outline of the polygon. If start and destination are located in the same or neighboring areas, this is handled separately by computing the beeline distance. There is not always a path from all start nodes to all destination nodes. This is the case if the graph has "islands" that are not connected with the rest of the graph. To counter this problem, we add more nodes around the start and destination location to the queue and repeat the SSSP/SDSP algorithm.

User Profile The user profile supports exclusions as well as arbitrary obstacle types that are contained in OpenStreetMap. Furthermore, it is possible to assign a difficulty value to each path element extracted from OpenStreetMap. This difficulty value can be derived

from arbitrary attributes (e.g. the number of stairs) of the objects in OpenStreetMap. Here, we distinguish between constant, linear, and quadratic relations between the attributes of obstacles and the difficulty that should be used in for the route computation. For example, this allows us to set a quadratic cost function for stairs and apply a constant penalty term if there is no handrail available. These values can be set separately for each direction (e.g. ramp/stairs up and down). Cost factors for street crossings can be set separately for unmarked crossings, crosswalks, as well as traffic lights. Additionally, the user profile can define maximum walking durations as well as maximum detours for the usage of unmarked street crossings (cf. preprocessing).

In practice, we can provide preconfigured profiles for common disabilities. Additionally, would be possible for experts like nurses or physicians (possibly after a specialized training) to configure individual profiles for their patients. For our evaluation, we used predefined profiles since this should be sufficient to determine algorithm runtimes. In general, it is possible to select an individual profile for each query.

Multi Criteria Shortest Paths The user profile defines an edge weight vector depending on the edge attributes and the settings of the user profile for each edge of the graph. These can now be used to determine the full Pareto-set using a multi-criteria shortest path algorithm (as found in [Sch09]). Therefore, we get every optimal trade-off between walking duration and difficulty.

Intermodal Door-to-Door Routing

The previously introduced personalized pedestrian routing can now be used to compute optimal first and last journey legs (the way from the start address to the first public transport stop and from the last public transport stop to the destination address) as well as footpaths between stop positions of vehicles involved in a transfer. A transfer can include a footpath between nearby stations.

Continuous Accessible Intermodal Routing The routing core is described in Section 1.7 and in [Gün+14]. This includes support for restrictions for entering/exiting vehicles at specific stops, through services, merge/split services, timezones, and many more. As discussed in Section 1.7, the first step is to compute all optimal routes from

the start address to all stations that are reachable within a given time limit using the specified modes of transportation for the first journey leg and from all stations from where it is possible to reach the destination address within the given time limit for the chosen modes of transportation at the destination. The start location and destination location can be seen as virtual nodes that are considered only for this particular query. The options to get from the start location to the first public transport station can be interpreted as virtual edges. The same is true for the travel options for the last journey leg. Using the shortest path Pareto set from the pedestrian routing as edges enables us include the difficulty (and duration) of the first and last segment in the Pareto-optimization of the complete journey. The edge weight vector of these additional edges is comprised of the walking duration and difficulty value. Again, Bellman's optimality principle tells us that all sub-paths of Pareto-optimal paths are also Pareto-optimal paths [BM09]. Therefore, it is sufficient to only include Pareto optimal paths for the first and the last segment which is exactly what the pedestrian routing algorithm computes.
The OpenStreetMap format is basically suitable to model the structure of buildings. Therefore, it can be used for indoor routing, outdoor routing, as well as paths that contain both, indoor as well as outdoor sections. For transfers, all shortest paths between all stop positions (which can be at any level inside or outside a building) of a station are precomputed using the pedestrian routing described previously. This yields all optimal trade-offs between transfer duration and difficulty.
Overall, this approach enables us to optimize the whole journey from start address to destination address.

Route Definition The route definition needs to be adapted to support transfer times that are precise to the level of stop positions. Previously, a separate route needed to be created for all trips traversing the same sequence of stations with the same enter/exit restrictions. To be able to consider the exact stop position, a route is now defined by the sequence of stop positions (instead of stations). Thus, two trains serving the same stations sequence may end up in two different routes if the visited stop positions within those stations do no match. This is required because shortest path algorithms (both, multi-criteria as well as single criterion) on a time-dependent graph [DMS08b; Pyr+08] as well as the RAPTOR algorithm [DPW12] and the TripBased routing algorithm [Wit15] only consider the first departure of a route in their respective "expansion" step. The

reason for this is that a later departure can only yield a longer journey with equal or more transfers and therefore cannot be Pareto-optimal considering travel time and the number of transfers as optimization criteria. Now, however, taking a later departure on the same route may result in a different transfer time at one of the following stations. This can lead to optimal journeys that cannot be found using the algorithms mentioned above if we do not change the route definition to include the stop position sequence. With the adjusted route definition, we can still guarantee the correctness of the routing results.

Routing Graph The basic time dependent graph model for modeling timetables has a central station node which is connected to all route nodes [DMS08b; Pyr+08]. The transfer edges between the station node and the route nodes carry the transfer costs (i.e. one transfer and the minimal station transfer time). To support stop position to stop position transfer times, we need to change the graph model: each stop position is modeled by a node which is connected to every other stop position node. There needs to be one edge for each Pareto-optimal footpath between two stop positions. This way, a station is now modeled by a fully connected subgraph of all stop positions. Each stop position node is again connected to every route node of routes that visit this stop position. An example is depicted in Figure 1.14: here, we see four fully-connected stop positions. Route R_2 and R_4 both service the same stop position.

Modeling transfer times at the stop position granularity is not only possible with the time-dependent graph but also with other multi-criteria routing algorithms such as RAPTOR [DPW12] or CSA [Dib+13b]: here, we could model each stop position as a separate station. Each transfer edge can be represented as footpath between those stations.

Real-Time For our experiments, we apply a multi criteria version of Dijkstra's algorithm on a time-dependent graph [DMS08a]. This has the benefit that it does not require any preprocessing which enables us to update the routing graph in order to represent the current situation rather than the schedule timetable. More details regarding the real-time update are given in Chapter 2. The approach for the time-expanded graph is described in [FMS08]. The previous approach supports delays, additional services, cancellations and reroutings. However, transfer times between stop positions can now

be affected by either track changes, disabled infrastructure (e.g. broken elevator or escalator), or a combination of both. To incorporate these changes, first the foot routing graph is updated to reflect the real situation. In the next step, we can compute updated shortest paths for the affected footpaths. In order to avoid recomputing all footpaths of a station, we store which infrastructure element is contained in which stop position relation. Thus, we can limit the recalculation to pairs of stop positions that are affected by a real-time update. Note that this is only relevant for infrastructure changes (e.g. a broken elevator); a track change can be seen as a rerouting. Thus, the affected trip needs to be removed from its original route and then, a new route that contains only the affected "rerouted" trip with the changed stop position is added.

In summary, we can now inform users that are on an affected journey (i.e. one that contains broken infrastructure elements, train delays, etc.) by monitoring journeys as described in Chapter 2. With the approach introduced here, we can compute real-time alternatives in case the planned journey is not feasible anymore (or has become unattractive due to delays). The real-time alternatives are fully accessible and are guaranteed to be Pareto-optimal regarding the selected optimization criteria travel time, number of transfers, and difficulty.

1.9.4 Evaluation

In this section, we present an computational study based on a prototypical implementation. The software is implemented in C++ and has been compiled using the Clang compiler (Version 8.0.0 with `-O3` and `-march=native` parameters for optimization) and executed on a system running Ubuntu 18.10. The system was equipped with an Intel Core i7 8700K (6x 3.70Ghz) CPU and 64GB of main memory (RAM).

The dataset was provided by Deutsche Bahn AG and covers Germany including all public transport services (bus, tram, subway, metro, regional, long distance, and high-speed trains). The timetable covers the dates 2019-07-02 and 2019-07-03. For the pedestrian routing, OpenStreetMap data covering the same region (Germany) was used.

The timetable contains 271,329 stations, 1,509,787 trips on 184,244 unique routes. In total, there are 53,792,814 arrival and departure events. The resulting routing graph has 4,063,195 nodes and 11,544,006 edges. The routing graph for the pedestrian routing is comprised of 28,829,081 nodes and 56,842,104 edges.

Since the building plans contained in OpenStreetMap data are often not precise enough (or suited for routing at all) and do not contain a link to the tracks in the timetable provided by Deutsche Bahn AG, it is not easily possible to make use of this data for routing. In order to still be able to evaluate the runtime characteristics of the routing algorithm, we randomly generate the transfer graph for every station contained in the timetable. Each stop position has attributes for whether it is ground-level or accessible by ramp (i.e. here: accessible for wheelchairs) or not and whether it is reachable via stairs and/or elevator. We assume that all tracks are numbered in ascending order and that neighboring track numbers are neighboring tracks. The transfer time and difficulty edge weights between two stop positions depends on the difference of the track numbers as well as the generated attributes. Only Pareto-optimal edges are generated.

Pedestrian Routing Preprocessing

Table 1.11 shows the preprocessing duration and the graph size for datasets of different sizes (Frankfurt, Hesse, and Germany). A significant increase in computation time is obvious. However, for most use cases (including our particular use case) the preprocessing time of approximately 13 minutes for a country like Germany is sufficient.

	City of Frankfurt	Hesse	Germany
Preprocessing	0:02 min (2008ms)	0:39 min	7:05 min
Serialization	0:00 min (528ms)	0:15 min	4:10 min
Generation of R-Trees	0:00 min (240ms)	0:05 min	1:07 min
Number of Nodes	130,063	2,086,705	28,829,081
Number of Edges	252,587	4,244,350	56,842,104

Table 1.11: Preprocessing Times and Graph Size of the Generating Pedestrian Routing Graph

Intermodal Routing

Table 1.12 shows the average runtime for different routing profiles. Note that the system can provide commonly used profiles but also allows for setting customized settings. Here, we evaluated three common profiles that should be of general use. However, we

do not expect more specialized profiles to have significantly different runtimes. The "Standard" profile does not optimize difficulty at all and allows all edges to be used. For the "Optimize Difficulty" profile, the difficulty value of each edge is taken into account and optimized as a Pareto criterion. The "Wheelchair" profile can only use edges that are accessible with a wheelchair (i.e. no stairs in our scenario). The computation time for the "Optimize Difficulty" is nearly ten times the standard computation time. This conforms to the number of labels which increased approximately in the same order of magnitude.

	Runtime	#Labels	#Optimal Paths
Standard	1,243ms	1,819,566	2.9723
Optimize Difficulty	9,129ms	8,793,709	33.9685
Only Wheelchair	3,119ms	3,949,925	15.1000

Table 1.12: Average Computation Time, Number of Created Labels, and Number of Optimal Paths for Different Search Profiles

Figure 1.15 shows the routing times of these profiles dependent on the distance between source and destination (left) and the maximum allowed footpath duration (right). In both cases a clear correlation between the chosen parameters is obvious. Thus, it is possible to estimate the computation duration based on the query parameters. The reason for the increased computation time for the larger footpath radius is that it increases the number of options to start the journey. Assessing all those options is computationally expensive. The same is true for larger distances between start and destination: on average, the number of options to be assessed (i.e. compared against each other) by the algorithm increases.

A more detailed view is provided by the box-whisker-diagram in Figure 1.16: here it is apparent that not only the average runtime increases with the distance between start and destination (beeline) but the higher quantiles increase, too. Starting with a 200km distance, approximately every forth query takes longer than 10 seconds.

1.9.5 Conclusion and Future Work

In this section, we introduced a personalized accessible intermodal real-time door-to-door routing and have shown that this approach is feasible in practice even on large-scale datasets like the German public transportation network. The approach is characterized by the ability to handle "soft" restrictions as a Pareto-criterion in a multi-criteria optimization algorithm. Furthermore, it supports "hard" exclusions of selected obstacle types (such as stairs for persons in a wheelchair). We take real-time updates such as infrastructure changes (e.g. elevator failures), track changes, as well as accessibility changes of the operating vehicles, delays, cancellations, reroutings, and additional services into account. This enables the user to find real-time alternatives in a disturbed situation when traveling with public transportation.

Although the computation times are feasible for practical use, it would be interesting to further improve the runtime performance of the algorithm. Other approaches like the RAPTOR algorithm [DPW12] could be suitable as a starting-point. The evaluation has shown that the number of results is quite high. To improve the user experience, it would be interesting to introduce an intelligent filtering mechanism that reduces the number of results shown to the user while at the same time retaining result diversity and quality.

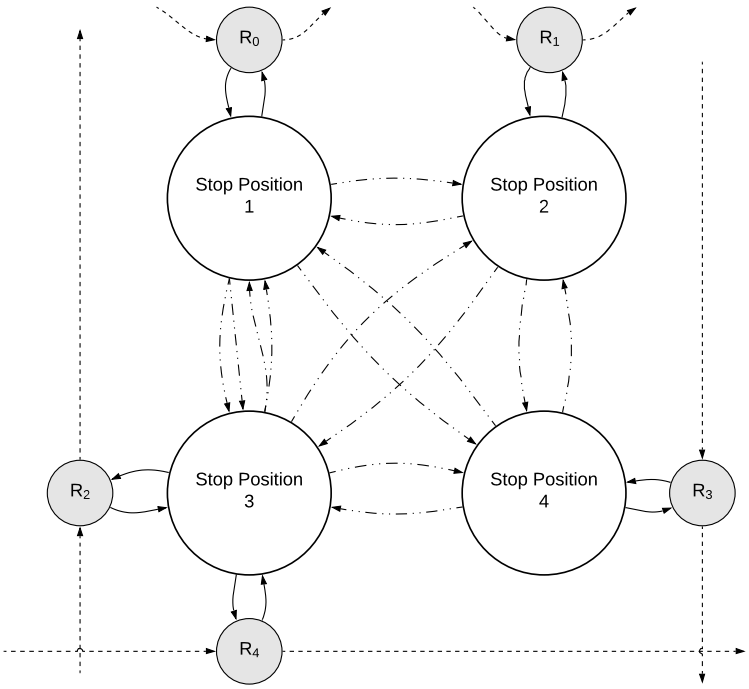
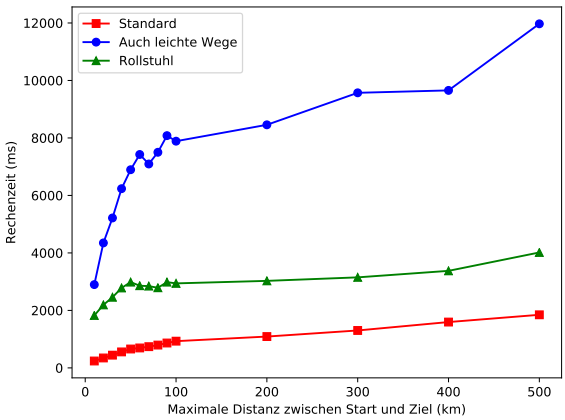
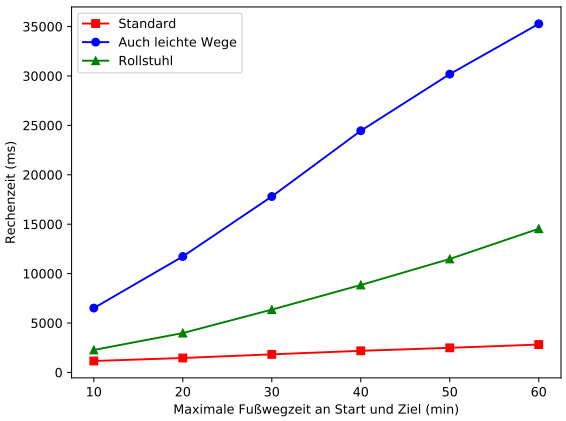


Figure 1.14: Station Model with Stop Positions: fully connected stop position graph with four stop positions. Stop Position 3 is serviced by routes R_2 and R_4. Stop Position 1 and Stop Position 3 are connected by multiple edges in each direction; each represents an Pareto optimum (i.e. one that has a shorter duration and more difficulty and the other one vice versa) computed by the pedestrian routing between these stop positions. Filled circles represent route nodes which are connected by route edges (dashed) to the route node at the next station.



(a) Maximum Walking Duration at Origin/Destination vs. Runtime in milliseconds



(b) Distance Between Origin and Destination vs. Runtime in milliseconds

Figure 1.15: Algorithm Runtimes.

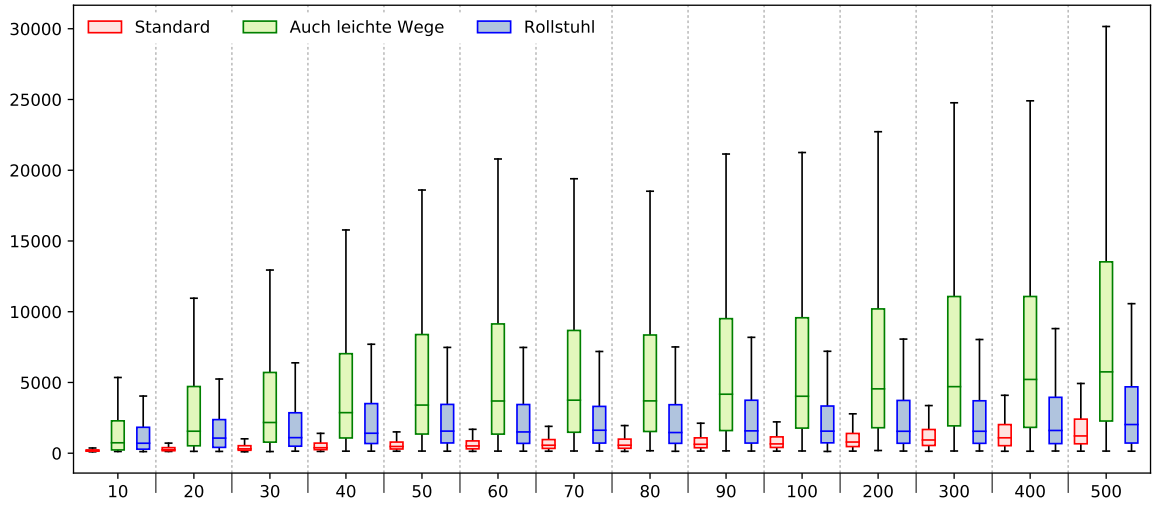


Figure 1.16: Total Computation Time in Milliseconds Depending on the Distance Between Start and Destination

1.10 Planning Optimal Two-Way Round-Trips with Park and Ride

This section is based on the paper *Multi Objective Optimization of Multimodal Two-Way Roundtrip Journeys* by Felix Gündling, Pablo Hoch, and Karsten Weihe (September, 2019). It was accepted, presented, and published at the 8th International Conference on Railway Operations Modelling and Analysis (ICROMA), Norrköping, Sweden, June 17th–20th, 2019 (No. 069, pp. 350-360).

Many journeys do not consist of one-way trips. On the contrary, in many cases travelers return to the starting point (home for private, office for business trips) [BTD04]. We consider a very common practical use case of multi modal routing: optimizing outward and return trip of a journey involving both private (e.g. a private car or bike) and public transportation (e.g. busses, trains, etc.). Planning a journey with commonly available online systems, that calculate optimal one-way trips, becomes quite cumbersome: finding the optimal P&R parking place (or an optimal place to park the bike) is not trivial. A parking place that was optimal for the outward trip might yield a suboptimal or infeasible return trip and vice versa. Considering multiple optimization criteria such as the number of transfers and travel time (accumulated for both trips), there might even be multiple optimal solutions. The reason for this is the time-dependent and directed nature of public transportation: an optimal route on the return trip does not necessarily include the parking place used in the outward trip. Combining independently optimized journeys may thus yield suboptimal or infeasible journeys. Consequently, optimizing both trips in a combined manner is required to compute optimal journeys for this use case. The fastest journey may not always be the most attractive one for everyone: in addition to a short travel time, some users prefer a cheap and/or convenient travel that minimizes the number of transfers. Since priorities of those optimization targets differ from traveler to traveler, our approaches compute a complete Pareto set considering convenience (number of transfers) and travel time as criteria. One variant additionally considers the price of the journey as optimization criterion: the total price is the sum of the costs of private transportation including time dependent costs for parking as well as the public transport ticket price. For the public transport ticket price, our price model assigns a

separate milage price per means of transportation (high speed trains are more expensive than local public transport). Parking prices are based on the parking duration. Our evaluation is based on real data: the public transport timetable is provided by Deutsche Bahn and covers Germany including trains (long distance as well as local), metro, busses and streetcar services. Street routing is based on Project OSRM [LV11] using an OpenStreetMap dataset covering the same geographic area as the public transport timetable. To the best of our knowledge there are neither scientific publications nor commercial systems offering this functionality. The presented algorithms are suitable for use in online routers and mobile routing applications.

1.10.1 Related Work

To the best of our knowledge, there are no publications that solve the described problem in a Pareto-optimal way optimizing multiple criteria. The first solution solving the problem [BCE09] considers a single optimization criterion: travel time. An improved bi-directional shortest path algorithm to solve the problem is described in [Hug+13]. Another approach based on access node routing [DPW09] is presented in [Spi15]. All three publications optimize travel time as single criterion and apply their algorithm to datasets covering a single city: Paris including its suburbs [Hug+13], the rural area around Lyon [BCE09], and Milano [Spi15]. Recent advances in public transport routing and multi modal routing such as RAPTOR/MCR[12] [DPW12; Del+13a], CSA [Dib+13b], TripBased [Wit15] were not extended to compute Pareto-optimal journeys for the multi modal park and ride two-way roundtrip problem.

1.10.2 Contribution

In this paper, we present various algorithms to solve the two-way park and ride roundtrip problem optimizing multiple criteria in a Pareto-optimal way. We compare different solutions based on a time-dependent graph model [DMS08a] with an algorithm based on connection scanning [Dib+13c] and another algorithm which is based on TripBased routing [Wit15]. All approaches optimize travel time as well as the number of transfers. Furthermore, we propose a variant that additionally optimizes prices.

[12]Round bAsed Public Transit Optimized Router, Multimodal Multi Criteria RAPTOR

We evaluate all algorithms on a realistic nationwide network: a complete public transport schedule for all of Germany including all modes of public transportation (e.g. busses, street cars, all kinds of trains) kindly provided by Deutsche Bahn. Our computational study shows that our algorithms are suitable to be deployed in online or mobile multimodal routing systems.

1.10.3 Preliminaries

This section describes the problem definition, static and dynamic/user inputs, and how they are preprocessed to be used as input for our core routing algorithms.

Problem Definition

We consider computing Pareto optimal solutions to the problem $\alpha \xrightarrow{t_{\text{out}}} \omega@[t_1, t_2] \xrightarrow{t_{\text{ret}}} \alpha$ where we call the outward trip t_{out} and the return trip t_{ret}. α and ω are locations (addresses / geographic coordinates). α might be the user's home address and ω the office address. The time interval $[t_1, t_2]$ is the minimal time range to stay at ω (e.g. office hours). Thus, our journeys have one of the following two structures:

$$\alpha \xrightarrow{\text{car}_1} p \xrightarrow{\text{walk}_1} s_w \xrightarrow{\text{pt}_1} s_x \xrightarrow{\text{walk}_2} \omega@t_1 \quad \dots \quad \omega@t_2 \xrightarrow{\text{walk}_3} s_y \xrightarrow{\text{pt}_2} s_z \xrightarrow{\text{walk}_4} p \xrightarrow{\text{car}_2} \alpha \tag{1.4}$$

$$\alpha \xrightarrow{\text{walk}_1} s_w \xrightarrow{\text{pt}_1} s_x \xrightarrow{\text{walk}_2} \omega@t_1 \quad \dots \quad \omega@t_2 \xrightarrow{\text{walk}_3} s_y \xrightarrow{\text{pt}_2} s_z \xrightarrow{\text{walk}_4} \alpha \tag{1.5}$$

The first one is most interesting to us. However, enabling the approach to find journeys with the second structure is necessary to avoid presenting unreasonable journeys to the user: it is not reasonable to use the car[13] if the trip between α and s_w over p ($\alpha \longleftrightarrow p \longleftrightarrow s_w$) takes longer than walking directly between α and s_w ($\alpha \longleftrightarrow s_w$). By allowing both structures, the journey involving the unnecessary car leg (Structure 1.4) will be superseded by the walking journey (Structure 1.5).

[13] Bad weather or mobility impairments could be reasons to use the car regardless of longer travel time. However, weather dependent routing and routing for handicapped persons is not addressed in this paper.

We minimize the combined travel time sum of t_{out} and t_{ret} as one Pareto criterion and the combined number of transfers of t_{out} and t_{ret} as another. The travel time includes the time from the start with the car at α until t_1 for the outward trip and the time from t_2 until α is reached again for the return trip. This includes waiting times at ω.

Note that the stations s_y and s_z as well as the stations s_w and s_w do not need to match but the parking place p is required to be the same for outward trip t_{out} and return trip t_{ret}. The user specifies α, ω, t_1, t_2, maximum driving distance d_{max} and maximum walking distance w_{max}. This naturally limits the number of parking places (candidates for p) and stations (for Journey Structure 1.5) reachable from α (car_1 and car_2 / walk_1 and walk_4), the number of candidate stations for s_w and s_z reachable from a parking place (walk_1 and walk_4), and the number of candidate stations for s_x and s_y reachable from ω (walk_2 and walk_3).

Inputs

Basically, an algorithm to solve the problem described above requires information about the road network, the locations of suitable parking places P, and the public transport timetable. The road network as well as the locations of parking places are extracted from OpenStreetMap. The public transport timetable consists of a set of stations S where each is associated with a geographic coordinate and a transfer time, *trips* (a vehicle visiting a stop sequence with associated departure and arrival times) and a set of *footpaths* that connect stations which are in close proximity so that walking between them is feasible. Furthermore, the timetable data contains information such as track names, service head signs, train category and service attributes like wireless internet availability or bicycle carriage. All presented algorithms require the trips to be grouped into *routes*: all trips in a route share the same sequence of stations. Additionally, trips in a route are not allowed to overtake each other. Otherwise, the route needs to be split into two separate routes. Grouping into routes is done as a preparation step.

Preprocessing

Since driving and walking is only available for the first and the last leg of both trips t_{out} and t_{ret}, we do not need to integrate both networks (timetable and road network).

This allows us to use specialized models and algorithms for each network: contraction hierarchies for the road network and Time Dependent/CSA/TripBased routing for public transport (cf. Section 1.10.4). Consequently, we can split the procedure to compute optimal roundtrip journeys into two parts without losing optimality: the preprocessing step computes all possibilities for the first and last leg of t_{out} and t_{ret}. This is the input for the actual core routing algorithm described in Section 1.10.4.

Procedure `preprocess_roundtrip()` shown in Listing 1.1 computes three sets W, C, and D: these enumerate all possibilities to reach a public transport station from α (sets W and C) and ω (set D) respecting the journey structure and user supplied driving and walking limits d_{max} and w_{max}. The preprocessing makes use of the following data structures and procedures:

- The procedures `car_route` and `foot_route` compute shortest paths (optimizing travel time) on the car/foot street network. They return the required time. Our `car_route` routine makes use of [LV11]. The `foot_route` routine is a specialized implementation based on OpenStreetMap data. Routes by foot are computed by a specialized algorithm that considers stairs, crossing roads, elevators, and many more elements.

- The table `dist` contains precomputed foot path durations between parking places and nearby stations. `dist[`p`][`s`]` is the time it takes to walk from parking place p to station s (and vice versa).

- `get_stations` and `get_parkings` are geographic lookup functions taking a coordinate and a radius. They return all stations/parkings where the distance to the given coordinate is less than the provided radius. The functions can be efficiently implemented using a spacial data structure such as an R-tree or a quadtree.

C contains all possibilities to get to a public transport station from α and vice versa (required to find journeys with Structure 1.4). Note that the entries store also the parking location. This is important because the core routing algorithm needs to match parking locations from t_{out} and t_{ret}. W contains all possibilities to walk between α and nearby public transport stations within w_{max} distance. W is required to find journeys

with Structure 1.5. D contains all possibilities to walk between ω and nearby public transport stations within $w_{\max}$ distance.

The code in Listing 1.1 can be improved by computing the routes to all targets in one step. Since Dijkstra-like algorithms (like contraction hierarchies employed in `car_routing`) are inherently "multi target"-algorithms, we calculate the walking/driving times to all candidates in one single step instead of running one one-to-one query for each target in a loop. Thus, we change the interface of `route_car` and `route_foot` to take a single location and a set of targets as input and return the travel time to each target as result.

1.10.4 Approaches

This section describes the different approaches for the core routing procedure. Each algorithm takes the same input computed in the preprocessing phase (in addition to the public transport timetable): the sets W and C which connect α with the public transport network through walking/driving, and D which connects ω with the public transport network through walking.

Time Dependent Graph

One established way to compute multi criteria shortest paths on public transport timetable networks are label correcting algorithms on graph data structures representing the timetable (i.e. time expanded and time dependent graphs). The time dependent graph is more compact (as compared to the time expanded graph) and therefore better suited to cope with large timetables containing not only trains but also streetcars and busses. So our first approach is based on the time dependent graph as described by [DMS08a]. This algorithm uses goal direction and domination by early results to speed up the search from one source to one target node.

Baseline

In this section, we will describe an algorithm that is purely based on an unchanged base algorithm: the time dependent earliest arrival problem. We extend the graph model so that it fits the problem. Basically, the sets W, C, and D can be seen as edges which

extend the time dependent graph. Consequently, we need to add nodes to represent α and ω. In the following, α and ω refer to those additional nodes if they are used in the graph context. Routing t_{out} and t_{ret} independently with all additional edges at once could yield suboptimal or unfeasible journeys due to non-matching parking places.

Since the edges from the set D (connecting ω with public transport stations and vice versa) do not introduce any dependencies, they are added for every search. It is sufficient to add those that match the search direction ($\omega \rightarrow s \in S$ for t_{ret} and $s \in S \rightarrow \omega$ for t_{out}). However, to prevent the interference between parking places, we conduct one multi-criteria search for each parking place separately for t_{out} and t_{ret}: the graph gets extended by all edges (one for each station that is reachable from p) that lead over the selected parking. These are earliest arrival problems $\omega @ t_2 \rightarrow p_i$ in case of t_{ret} and latest departure problems $p_i \leftarrow \omega @ t_1$ for t_{out} (for every parking i). This generates all optimal trips $T_{\text{out}}^{p_i}$ for t_{out} and $T_{\text{ret}}^{p_i}$ for t_{ret} for every potential parking place p_i. Waiting time (arriving earlier than t_1 or departing later than t_2 at ω) is considered travel time and is therefore minimized as described in Section 1.10.3. Note that every overall optimal roundtrip needs to be a combination of optimal trips t_{out} and t_{ret} for one of those potential parking places. Otherwise (if an optimal roundtrip would not be a combination of optimal individual trips), it could obviously be improved by an optimal one. Consequently, the combination of all computed trips $\bigcup_{p \in P} T_{\text{out}}^{p} \times T_{\text{ret}}^{p}$ contains all optimal roundtrips. Removing all roundtrips that are superseded by others (including duplicate ones) yields the final set of optimal roundtrips.

Edges from the set W representing all options to walk from α to a public transport station (for t_{out}) and vice versa (for t_{ret}) are added in a separate search (to enable the system to find journeys of Structure 1.5). Since there are no constraints to use the same parking place in t_{out} and t_{ret}, they can all be used in one search.

This approach requires $2(|\Pi| + 1)$ invocations of the basic time dependent routing routine (earliest arrival / latest departure) where Π is the set of all considered parking places: for each direction one invocation for every parking place candidate and one with all edges from W. This is certainly not optimal regarding computational effort (compared to the approaches presented later on). However, this approach is still useful for the practical verification of other approaches.

Parallelization

Since all searches are independent, they can be trivially parallelized. In theory, if the number of parallel processors is equals to two times the number of parkings, this can reduce the overall calculation time to the time it takes to respond to one routing query. However, since most systems (besides super compute clusters) cannot provide this level of parallelism, this is not a feasible approach, either.

Combined Search for t_{out} and t_{ret}

The basic Dijkstra algorithm computes shortest paths not only to the target node but to all nodes in the graph. Since the basic algorithm presented by [DMS08a] makes use of goal direction and domination by terminal labels[14], this property does not apply anymore: when the algorithm terminates, only the labels for one destination node will be correct (in the sense that they necessarily represent the non-extensible Pareto set).

The first step of the combined search approach is to compute the shortest paths from/to every single parking place like in the baseline approach described in Section 1.10.4 for one direction t_{out} or t_{ret}. For the opposite direction, we now can make one combined search: instead of adding just the edges for only one parking, we add all parking edges but combine the edge cost with the criteria computed for the opposite direction in the first step. Since the first routing can yield more than one optimal trip for one parking, we have one additional edge for each optimal trip. Assuming we chose t_{out} in the first step, we add one edge from $s \in S \rightarrow \alpha$ for each optimal t_{out} journey using parking $p_i \in P$ for each station reachable from p_i. The edges carry the following costs: $(\text{dist}[p][s] + \text{car\_route}(p, \alpha) + \text{tt}_i, \text{ic}_i)$ where ic_i is the number of transfers and tt_i is the travel time for journey i in t_{out}. If t_{ret} was chosen for the first step, the approach works analogously.

This approach allows us to reduce the complexity of the baseline approach from $2(|\Pi| + 1)$ invocations of the time dependent routing routine to $|\Pi| + 1$ invocations: in one direction (outward or return) we need to route to/from every parking. In the return direction, only one query is required. The invocation with all edges from W in the first step stays the same. The search in the opposite direction is conducted with all edges in

[14] labels are partial journeys that are used in the routing algorithm

W.

As with the baseline approach, this approach can also be parallelized. However, this approach has one constraint on the ordering: the invocation for the opposite direction requires all results from the first step.

No Terminal Domination and Worst Bounds

Applying domination by terminal labels (in combination with lower bounds and goal direction) in the time dependent graph routing is a very effective speedup technique for queries to a single target. However, in this setting (preventing interference of labels that use different parking places), domination by terminal labels as implemented in our basic time dependent routing algorithm demands a high number of invocations as we have seen in the previous three sections. Now, we want to further reduce the number of invocations by computing all optimal journeys over all parkings in one run of the algorithm (as opposed to $|\Pi|$ invocations in the first step of Section 1.10.4). Simply disabling the domination by terminal labels and routing with all additional edges (W, C, and D) would be one option.

However, domination by terminal labels can be replaced by a different technique that still allows us to discard labels early in the search process: those that are worse or equal to the combined worst (i.e. numerically greatest assuming optimization criteria are minimized) value of each optimization criterion over all parking places (“worst bound”) cannot contribute a new optimum. Consequently, this requires at least one terminal label for each parking place. Until this precondition is met, we cannot discard any label.

To implement this, we have a list of parkings that were not yet reached. This list is initialized with all parkings (identified by a unique index) reachable from α. Every time a label reaches the target node, the used parking is removed from this list if it is the first to use this parking. If the list is empty (i.e. every parking was reached), this means that domination by worst bounds can be applied. To track the worst bounds, one variable per optimization criterion is introduced and updated every time a label is created on the target node. If every parking was reached, every newly (through edge extension) created label is compared to the stored combined worst bounds and discarded if its criteria values are equal or greater. The same check will be applied upon queue extraction since the worst bounds can change between queue insertion and extraction. Labels that

were created through expansion of edges carrying different parking indices are deemed incomparable to prevent domination of options that may be part of an optimal round trip but are not optimal for this search direction. Walking options from W are always added. They can be implemented as "virtual" parking directly at α. Thus, no driving is required.

The routing for the opposite direction can be implemented as described in Section 1.10.4 and therefore benefit from unconditional dominance by terminal labels. This approach cannot be parallelized. Altogether this approach further reduces the number of invocations to two, albeit more complex calls: one for each direction t_{out} and t_{ret}.

Concurrent

In this section, we present an algorithm that handles the search in both directions (for t_{out} and t_{ret}) in an interleaved manner. Basically, we still have two multi criteria Dijkstra algorithms with the addition that they exchange information at runtime. Thus, every data structure (such as the priority queue, lower bounds, etc.) is redundant: one for each search direction.

Instead of the standard domination by terminal labels, we maintain a list of complete roundtrips: every time a new label has reached the target node in one direction, it is combined with each terminal label of the opposite direction that has a matching parking place. The resulting valid round trips are then added to the list of complete round trips if they are Pareto optimal. Previously added roundtrips that are worse than the newly added roundtrip are removed. These complete round trips can then be used to dominate labels in both search directions at the creation of new labels and after queue extraction: if a partial roundtrip is already worse (in the Pareto sense) than a complete roundtrip, it can be discarded. Instead of comparing the label values (here: travel time and the number of transfers) directly with those of the terminal label, we can employ lower bounds to discard suboptimal labels as early as possible: a label with travel time t in the t_{out} routing takes at least $t + lb_{t_{\text{out}}}[n] + lb_{t_{\text{ret}}}[\omega]$ minutes for the complete roundtrip where $lb_{t_{\text{out}}}$ and $lb_{t_{\text{ret}}}$ are precomputed lower bounds for every node in both search directions. This is analogous for the t_{ret} search: $t + lb_{t_{\text{ret}}}[n] + lb_{t_{\text{out}}}[\omega]$.

All in all, we reduced the number of invocations from $2(|\Pi| + 1)$ for the baseline approach to just one. This comes with an increased complexity of the queries. However,

the combination of information (instead of separate invocations of the basic time dependent routing procedure) as described here reduces the total number of steps required to compute all optimal round trips.

Connection Scan

In this section, we present an algorithm that is based on the Connection Scanning Algorithm (CSA) by [Dib+13c]. As opposed to the time dependent routing algorithm, it does not require a graph to represent the timetable, neither does it depend on a priority queue. The timetable model is a simple array of all elemental connections (departure and arrival of a trip with no intermediate stops in between) of the timetable sorted by departure time. The algorithm iterates through the array and updates earliest arrival times at the stations visited by the iterated connections accordingly. The algorithm also handles footpaths between stations and transfer times between transport services.

As the basic variant of CSA just iterates "through time" (sorted connections) it is not directed towards a specific target station. Therefore, it is well suited to be adapted as a multi target algorithm without a performance penalty. This can be utilized: in the first step, we ignore the actual driving and walking times from D and W that connect α with the public transport timetable. Instead, we search from all stations in D to all stations in W and C.

The original publication does not describe a multi-criteria version of the earliest arrival problem or journey reconstruction for this type of search nor does it describe the latest departure problem or multi source and multi destination routing. Consequently, we need a specialized version of the CSA algorithm for our use case:

- *Multiple Start Stations*: In the basic version, only one station is initialized with the desired start time. In our use case, every station in D is initialized with the walking time (between ω and $s \in S$) as offset that is added to t_2 (for t_{ret}) and subtracted from t_1 (for t_{out}).

- *Multiple Destination Stations*: Basically, this is what the algorithm does anyway if we omit the early termination mechanism which stops when the departure time of the currently iterated connection exceeds the earliest arrival at the destination.

- *Latest Departure Problem*: For t_{out}, we need to solve the problem $(s \in W \cup C) \leftarrow \omega @ t_1$. This can be done analogously to the forward search. For example, the connection array is sorted by descending arrival time and footpath walk times are subtracted instead of added.

- *Multi Criteria*: To support the optimization of the number of transfers as additional Pareto criterion, we do not only store a single earliest arrival time for each station but instead one for each number of transfers. The same applies to the array T which indicates whether a trip can be reached or not: instead of single reachable bit, one bit per number of transfers is stored. The n^{th} bit indicates whether the trip can be reached with n transfers.

- *Reconstruction*: Since additional journey pointers (which would need to be maintained for every number of transfers) as described in [Dib+13c] slow down the search (scan running time), we chose to adapt the version that works without them. As our implementation of the algorithm supports the optimization of the number of transfers as Pareto criterion, we need to reconstruct one journey for each optimal number of transfers. The recursive call with n transfers at the next interchange stop continues with $n - 1$ transfers. Similarly, the trip reachable array needs to be looked up at the bit referring to the current number of transfers. Not knowing where the journey may have started imposes additional complexity: we need to iterate every possible station and check whether the travel time matches the walk ($\omega \leftrightarrow s \in D$) for this station.

Now that we have a variant that handles multiple departure stations, multiple destinations and multiple criteria in both search directions (earliest arrival / latest departure), we can utilize it to find optimal round trips: For each direction t_{out} and t_{ret}, we execute one search. Both searches are independent and can therefore be executed in parallel. We execute one latest departure query (starting at $t_1 @ \omega$) for t_{out} and one earliest arrival query (starting at $t_2 @ \omega$) for t_{ret}. The results of those queries are then merged to complete roundtrips by iterating every parking place and combining all journeys from t_{out} and t_{ret}. Since not every roundtrip is necessarily optimal, we remove all that are not Pareto optimal. This yields the full set of optimal roundtrip journeys.

TripBased

As with the Connection Scanning Algorithm, TripBased routing as presented by [Wit15] is inherently a multi target routing algorithm: it can be seen as a breadth first search on a graph-like data structure consisting of trip sections and transfers between those trip sections. Similarly to the RAPTOR algorithm [DPW12], it operates in iterations/“rounds” where the n^{th} iteration computes all optimal connections with n transfers. Each round updates the trip sections that are reachable through one additional transfer from previously reachable trip sections.

We adapt the algorithm to be able to compute optimal journeys to multiple targets. Therefore, we need to keep one result set J for each target station. Additionally, the earliest arrival time τ_{min} needs to be kept separately for each target station to check whether a trip reaching the target is optimal. A new trip segment needs to be added to the queue only if its arrival time does not exceed the maximum earliest arrival time τ_{min} over all target stations. Otherwise, it can be discarded because it cannot be optimal for any target station anymore: every slower connection with less transfers was already discovered in a previous iteration.

The additional footpaths between ω and nearby public transport stations can be handled analogously to those already contained in the basic static timetable.

In addition to the changes required to compute optimal journeys to multiple targets, the basic TripBased algorithm needs to be adjusted to compute connections for the latest departure problem, not just the earliest arrival problem. Since the preprocessed transfers (transfer reduction step) differ for the forward (earliest arrival) and reverse (latest departure) direction, we need to have one transfer set T for each search direction. This doubles the preprocessing workload. Otherwise, the latest departure computation is analogous to the earliest arrival computation described in [Wit15].

As we now have an algorithm with properties similar to the adapted CSA algorithm (multi criteria, multi source, multi target, earliest departure, earliest arrival), we can use it to compute optimal roundtrip journeys as described in Section 1.10.4.

1.10.5 Price as an Additional Optimization Criterion

In this section, we present a version that optimizes not just travel time and the number of transfers but also the price. The price of the complete roundtrip is comprised of the costs for parking at p (depending on the parking duration), the driving costs (of car_1 and car_2), the public transport ticket price (of pt_1 and pt_2) and an hourly wage to eliminate cheap but exceedingly long journeys that are unattractive from a practical perspective.

Since public transport pricing models are very complex, constantly changing and different for every area, we decided to use two artificial pricing models. Both are milage and vehicle class based: a high speed train (such as a German ICE or French TGV) costs \$0.22 per kilometer, a local train costs \$0.18 per kilometer and short distance transports such as busses and trams cost \$0.15 per kilometer. Additionally, we introduce an hourly wage of \$4.80 (converted to the atomic timetable time unit, minutes). The first model computes just the sum of those costs. The second, more advanced model, introduces a special ticket that allows the passenger to use arbitrary local transports (local trains, busses, trams) for a flat price (\$42.00 here). All mentioned values are freely configurable.

All algorithms need an updated route definition which takes the vehicle class into account because otherwise later departures (which will not be considered by the algorithms) may yield a cheaper connection. In the following, we describe the extensions to the approaches presented in Section 1.10.4 that enable price optimization for the two price models described above.

Graph Based

Extending the graph based approaches (Baseline, Parallelized Baseline, Worst Bound and Concurrent) to support price as additional Pareto criterion is mostly straightforward: the edge weight vector as well as the individual labels carry the price as additional entry. However, we need to also adjust the label comparison. Before, a label a dominated label b if and only if every criterion value of a was less than or equal to the corresponding criterion value in b. The criteria were $(\text{arr}_i, -\text{dep}_i, \text{transfers}_i)$ where arr_i and dep_i are the arrival and the departure time of label i.

Instead of just adding the price to this comparison, the hourly wage requires special treatment to retain correctness of the search. Assume we compare two labels a and b

where a has a higher ticket price than b but a lower total price because it arrived earlier and did accumulate less costs due to the hourly wage. Consequently, from a Pareto perspective a dominates b (lower price, earlier arrival with same departure time). Since b arrived later, it now has to wait less for the next departure. Due to the hourly wage, the edge costs less for b than it does for a. So after edge expansion, a does not dominate b anymore which implies that we lost an optimal connection. To prevent this, we need to add the hourly wage price of the travel time difference to the price of a when comparing a with b. This way, the waiting time disparity is compensated.

The additional edges derived from the set C (connecting α with public transport stations through a parking) now carry the according kilometer based price for the car route. Additionally, the parking itself can be modeled as a time dependent edge: coming back later to the parking increases the costs by $2.00 per hour (staircase function).

Connection Scanning

Data Structures Extending the Connection Scanning algorithm to support price as an additional optimization criterion in the Pareto sense requires more effort than for the baseline approach because the data structures were designed with only travel time and number of transfers in mind. Before, the data structures holding the earliest arrival time for each station (S - note that we use the nomenclature of the CSA publication in this section) and the trip reachable bits for each trip T were both two-dimensional arrays with one entry for each number of transfers. This was sufficient for two criteria (number of transfers and travel time) because for each number of transfers only the fastest journey was relevant. Now, when additionally optimizing prices, there can be an arbitrary number of optimal journeys for each number of transfers (all optimal trade-offs between travel time and price). Thus, each entry of $S[\text{station}][\text{transfers}]$ now maintains an array with all Pareto optimal travel time / price tuples for this station instead of just the minimal travel time for this number of transfers.

The array T holds a bit (for each number of transfers n) that indicates whether a trip is reachable with n transfers. However, this is not sufficient because it is not known at which cost the trip can be reached. Note that the price to reach the trip is not the same for each section of the trip. Consequently, we need to maintain the cheapest price for each trip section for each trip for each number of transfers. This is necessary to compute

the correct journey price when iterating the connections array in the main loop of the Connection Scanning Algorithm.

Algorithm When initializing S with the offsets from D (foot routes between ω and nearby public transport stations) the price (incurred by the hourly wage) needs to be initialized, too. Furthermore, the main loop of the algorithm needs to be adjusted: if a connection is reachable through the station and the trip reachable flag is set (i.e. it has a price entry for the corresponding trip section), the cheaper solution is selected. If entering the trip at this station is the cheaper solution, the price of the following trip segments in the T array needs to be updated with the cheaper price including the hourly wage. Only Pareto optimal entries (travel time / price tuples) are added to $S[\text{station}][\text{transfers}]$ removing superseded ones. Footpaths also incur costs due to the hourly wage.

Reconstruction Naturally, the journey reconstruction step also needs to be adapted to the new data structures: when looking up S and T entries, not only the travel time and the number of transfers but also the price of the entry needs to match.

Trip-Based

Preprocessing The preprocessing to eliminate unnecessary transfers was aimed at transfers and travel time as optimization criteria. Thus, transfers that lead to cheap connections may be discarded. To prevent this, the transfer reduction step is omitted. Even transfers to later trip sections of the same trip (in case the trip visits a station two or more times) and other trips of the same line can save money. U-turn transfers are still being removed.

Data Structures To track the price of each journey, queue entries now also carry the current journey price (in addition to the trip segment and the number of transfers). Similarly to the CSA extension, the cheapest price to reach each trip segment is maintained: the data structure $R(t)$ which previously maintained for each trip the first reachable stop now holds the cheapest price to reach each stop of the trip with the corresponding trip.

Algorithm The algorithm now tracks the price of each queue entry. Updating trip t entails maintaining the cheapest prices in $R(t)$ as well as the cheapest prices of all trips of the same route with later departure times.

Pruning We extend the implementation to track the latest arrival time and most expensive price over each target station. Journeys exceeding these limits are discarded and therefore not added to the queue to be processed in the next iteration.

1.10.6 Computational Study

Our C++ implementation (compiler: LLVM/Clang 6 with "-O2" optimizations) of the presented algorithms was evaluated on a computer with an Intel Core i7 6850K (6x 3.6GHz) CPU and 64GB main memory. The public transport timetable was provided by Deutsche Bahn and covers all services (busses, trams, trains, etc.) operated in Germany. For foot and car routing the complete OpenStreetMap dataset of Germany was loaded.

The timetable spans the 27th and 28th of November 2018. It contains approximately 30M departure and arrival events (60M events total) that take place in 1.7M trips on 224,832 routes.

Queries are generated by choosing a random t_1 and a random t_2 30min to 4h after t_1. To generate coordinates that yield a high chance of non-empty result sets, we randomly select a public transport station that has at least one arrival event in the time interval $[t_1 - 60\text{min}, t_1]$ and at least one departure event in the time interval $[t_2, t_2 + 60\text{min}]$. Then, a random coordinate in a radius of w_{max} around this station is selected as ω. α is a random coordinate located in a 200km radius around destination. Both coordinates need to be within Germany which is checked with the help of a polygon that resembles Germanys borders.

All algorithms compute set of connections as described in Section 1.10.3. For each response, it was evaluated that every algorithm produces exactly the same result.

Preprocessing

Extracting all parking places and calculating optimal foot paths between public transport stations and nearby parking places takes 41 minutes and 44 seconds. However, this

needs to be done only once for every dataset. At runtime, a fast lookup table with the precomputed foot path times is used. Our OpenStreetMap dataset contains 319,361 parking places. On average, 5.18 stations are reachable from a parking place (median 4, 99% quantile 23).

The execution of the preprocessing step described in Section 1.10.3 takes place at query runtime. Street routing between α and all parking places in the selected radius takes 383ms. The lookup times for stations/parkings in a specified radius around a coordinate are negligible (below 1ms). Lookup of precomputed foot routes between parking places and nearby public transport stations takes 3.68ms. Since ω is a user input, foot paths between ω and nearby public transport stations (set D) cannot be precomputed. Computing W takes 27.4ms at runtime. The sets W, D and C can be computed in parallel. So in total, 387ms of the runtime are due to preprocessing. The next sections report runtimes including preprocessing times. Therefore, to obtain the total core routing runtime, approximately 0.4s need to be subtracted from the runtimes reported below.

Baseline Algorithms

Table 1.13: Runtimes of Baseline Algorithms without Price Optimization in Milliseconds

	avg	Q(99)	Q(90)	Q(80)	Q(50)
Baseline	659 700	2 732 816	1 676 236	924 110	398 985
Combined	399 353	1 455 144	868 975	630 930	258 519
Parallel	276 666	761 288	561 843	444 319	215 487
Comb. Par.	193 793	624 920	402 261	301 287	159 545

The baseline algorithms were executed with 100 randomly generated queries. As depicted in Table 1.13, parallel execution of the baseline approach yields a reasonable 2.4x speedup on average. The “trick” of an integrated optimization for one of the two directions (including parallel execution for the non-integrated search direction) yields another 1.4x speedup on average. Nonetheless, the baseline approach and its variations described in Section 1.10.4 (parallel implementation) and Section 1.10.4 (combined

search) are not really of any practical use because they require many invocations of the time dependent routing routine. Users of online services are not eager to wait more than three minutes for their routing result. However, due to their simplicity those approaches are useful for validation of the other implementations.

Advanced Algorithms

Table 1.14: Runtimes of Advanced Algorithms in Milliseconds

	avg	Q(99)	Q(90)	Q(80)	Q(50)
No Terminal Dominance	4800	11 430	7969	6615	4403
Worst Bound	4762	11 268	8042	6564	4363
Concurrent	3573	10 305	6439	5000	3156
CSA	1697	3384	2322	1999	1577
CSA SIMD	908	2453	1353	1113	806
TripBased	816	2644	1302	975	689

All advanced algorithms were executed with 1,000 randomly generated queries. In this section, we present the results of the advanced algorithms: different Dijkstra-based algorithms (with worst bounds, without terminal dominance and the interleaved / concurrent approach) on the time dependent graph model (introduced in Section 1.10.4), Connection Scanning (Section 1.10.4), and TripBased routing (Section 1.10.4). Additionally, we have implemented a CSA version that makes use of SIMD instructions.

As we can see in Table 1.14, the concurrent (interleaved) search in both directions (outward and return trip) brings the runtimes on the time dependent graph down from more than three minutes to 3.5 seconds. However, one percent of the queries take more than 10 seconds to answer. All non-graph-based approaches (CSA and TripBased) yield better runtime performance: the CSA SIMD variant as well as the trip based routing have average runtimes under one second. The data-parallel SIMD implementation of the CSA algorithm yields nearly a 2x speedup compared to the basic CSA version. Note that the CSA SIMD version is even faster than the TripBased approach when it comes to the 99% quantile (2.45 seconds vs 2.64 seconds) indicating that it has a more predictable performance profile.

Parking Radius In this section, we analyze the relation of the runtime of the approaches presented in this paper with the $d_{\max}$ parameter which essentially determines the number of parkings to consider. We analyze this relation for two and three optimization criteria.

As we can see in Figure 1.17, the runtime scales mostly linearly with the parking radius for all approaches regardless of the optimization criteria. However, the increase in runtime is different for the presented approaches without price optimization: while the Connection Scan SIMD and TripBased runtimes rise minimally with an increased parking radius and stay below one second, Concurrent and Worst Bound runtimes have a steep increase with a growing parking radius.

Note the different ordinate scale of the right graph of Figure 1.17: price optimization imposes a heavy toll on query runtime. When optimizing prices, the runtimes of the graph based approaches (Worst Bound and Concurrent) for short distances are better than those of CSA and TripBased. However, this changes for $d_{\max}$ values greater than 23km where CSA delivers the fastest (almost constant) runtimes. A mixed approach could pick a graph based algorithm for smaller radii and switch to CSA for larger radii. As the TripBased algorithm is tailored to two optimization criteria (travel time and number of transfers), the runtimes with three optimization criteria lack behind the other approaches.

Table 1.15: Runtimes for Different α/ω Distances in Milliseconds

	50km	200km	900km
No Terminal Dominance	2236	4800	5308
Worst Bound	2234	4762	5263
Concurrent	1544	3573	4876
CSA	1778	1697	1656
CSA SIMD	952	908	853
TripBased	878	816	730

Distance Analysis In Table 1.15 we see that the runtimes of the CSA and TripBased approaches are insensitive to changing distances between α and ω. All runtimes of graph based approaches grow with larger distances. Note that for short distances, the

average runtimes of the Concurrent approach are lower than those of the basic CSA approach. This is not the case anymore for higher distances.

Price Optimization

Table 1.16: Base Scenario, No Price vs. Simple Price vs. Regional Price

	no price	simple price	regional price
No Terminal Dominance	4800	19 505	17 492
Worst Bound	4762	19 249	17 316
Concurrent	3573	17 718	15 141
CSA	1697	18 314	23 245
TripBased	816	40 670	39 542

In this section, we analyze the impact price optimization has on the runtimes of the different approaches. We evaluated both public transport price models introduced in Section 1.10.5: one is based only on distance and vehicle class (called “simple” in this section), the other model adds a special regional ticket with a flat price (called “regional price”). The changed route definition (described in Section 1.10.5) leads to 0.61% more routes. As we can see in Table 1.16, this additional search criterion increases the runtimes of all algorithms between 4x and 50x compared to the two criteria implementations (regardless of the concrete pricing model).

Since the price optimizing implementation of TripBased disabled most of the speedup it gained through the preprocessing (transfers reduction), it switched from being the fastest implementation to being the slowest implementation. Note that while Worst Bound and Concurrent could gain more than 10% speedup through the regional price model, CSA was slowed down by it (by more than 25%).

1.10.7 Conclusion

We presented several novel approaches to compute Pareto optimal solutions to the 2-way park and ride roundtrip problem. In addition to two criteria optimization (travel time and number of transfers), we introduce variants of the approaches which additionally

optimize the journey price. Since many journeys follow this pattern (e.g. for commuters), the developed algorithms are useful in practice. The approaches are based on state-of-the art algorithms for public transport routing such as TD [DMS08a], CSA [Dib+13c] and TripBased routing [Wit15]. Our evaluation on a dataset covering all of Germany shows that the approaches offer query runtimes below 2 seconds which makes them suitable for use in online or mobile app information systems.

Listing 1.1: Preprocessing Procedure: computes edge sets W, C, and D to connect α and ω with the public transport network.

```
dist[p ∈ P][s ∈ S]

fn car_route(from, to) do ... return driving_time done
fn foot_route(from, to) do ... return walking_time done
fn get_parkings(coordinate, radius) do ... return parking_set done
fn get_stations(coordinate, radius) do ... return station_set done

fn preprocess_roundtrip(α, ω, d_max, w_max) do
  W := ∅ // possibilities for α ⟷ s ∈ S via foot
  walking_candidates := get_stations(α, w_max)
  foreach s ∈ walking_candidates do
    walking_time = foot_route(α, s)
    W := W ⋃ {(α → s, walking_time), (s → α, walking_time)}
  done

  C := ∅ // possibilities for α → p ∈ P → s ∈ S and s ∈ S → p ∈ P → α
  Π := get_parkings(α, d_max) // parking candidates
  foreach p ∈ Π do
    c_out := car_route(α, p) // driving time outward
    c_ret := car_route(p, α) // driving time back
    station_candidates := get_stations(p, w_max)
    foreach s ∈ station_candidates do
      w := dist[p][s] // walking time between parking and station
      C := C ⋃ {(α → p → s, c_out + w), (s → p → α, c_ret + w}
    done
  done

  D := ∅ // set of possibilities ω ⟷ s ∈ S via foot
  destination_station_candidates = get_stations(ω, w_max)
  foreach s in destination_station_candidates do
    walking_time := foot_route(s, ω)
    D := D ⋃ {(s → ω, walking_time), (ω → s, walking_time)}
  done

  return (W, C, D)
done
```

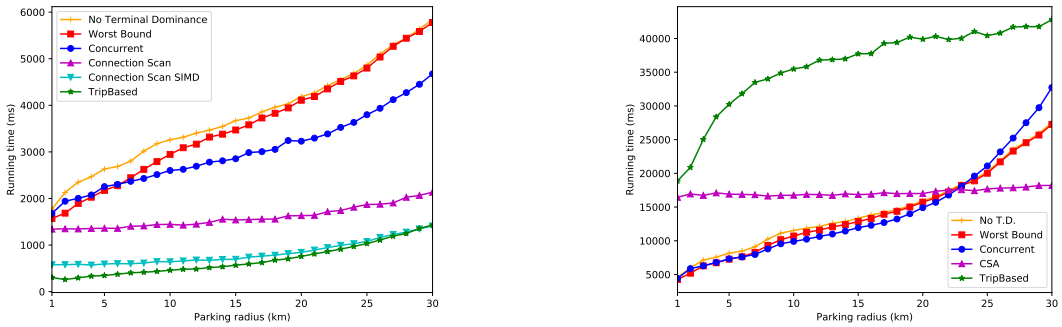


Figure 1.17: Runtime Subject to Parking Radius Distance d_{max}: the left figure shows the basic optimization with travel time and number of transfers. The right figure shows the runtime for optimization with all three criteria: travel time, number of transfers and price.

1.11 Time-Dependent Tourist Tour Planning with Adjustable Profits

This section is based on the paper *Time-Dependent Tourist Tour Planning with Adjustable Profits* by Felix Gündling and Tim Witzel (2020). It is accepted at the ATMOS 2020 conference.

Planning a tourist trip in a foreign city can be a complex undertaking: when selecting the attractions and choosing visit order and visit durations, opening hours as well as the public transit timetable need to be considered. Additionally, when planning trips for multiple days, it is desirable to avoid redundancy. Since the attractiveness of activities such as shopping or sightseeing depends on personal preferences, there is no one-size-fits-all solution to this problem. We propose several realistic extension to the Time-Dependent Team Orienteering Problem with Time Windows (TDTOPTW) which are relevant in practice and present the first MILP representation of it. Furthermore, we propose a problem-specific preprocessing step which enables fast heuristic (iterated local search) and exact (mixed-integer linear programming) personalized trip-planning for tourists. Experimental results for the city of Berlin show that the approach is feasible in practice.

1.11.1 Introduction

When planning a tourist trip to a foreign city, there are often many activities to choose from. Selecting a subset of these, while keeping in mind their opening hours as well as the alternatives to get from one point of interest (PoI) to the next, can be a daunting and time-consuming task. Planning activities for multiple days (each day within a fixed time horizon) is even more challenging because one probably wants to avoid redundancy.

Opening hours may have potentially zero (closed) to multiple different time windows each day. While public statues and monuments can be visited anytime of the day, a special place to enjoy the sunset should be visited when the sun goes down. Public transport (containing regular as well as irregular services) is a popular option to move between PoIs. Thus, the problem definition has to be time-dependent. In addition to

time-dependent means of transportation (public transit), many attractions are reachable by non-time-dependent means of transportation such as walking.

While some PoIs, like a statue, can be experienced within minutes, others (like a zoo or museum) can be entertaining for hours. Events like a theater or opera have a fixed start and end time. Modeling these properties requires a duration dependent profit function for each PoI. This profit function needs to be capable of enforcing a minimum required visit time and be able to model a "saturation effect". It should not only take into account the type of PoI but also the personal preferences of the tourist: a family with children probably will not want to spent the same amount of time at an art museum as an elderly person.

To realistically model PoIs, it is important to consider multiple locations for entries and exits. The problem definition has to respect the time required to get from one entry/exit to another. For example, a large zoo, park, or shopping street can have various entries where each one can be reached with different public transport lines. Additionally, such areal PoIs may contain further PoIs (like statues or famous shops, bars, cafes).

In this section, we propose a mathematical formulation of the aforementioned problem in the form of a mixed integer linear program (MILP). Furthermore, we present an iterated local search (ILS) approach to solve the problem fast enough for practical planning purposes (i.e. in a web-based or mobile planning service for tourists).

The remainder of this section is organized as follows: Section 1.11.2 gives an overview over related work. Section 1.11.3 outlines our contribution to the topic of realistic tourist trip planning. In Section 1.11.4 we describe how we model the Time-Dependent Team Orienteering Problem with Time Windows (TDTOPTW) with our problem specific extensions as a Mixed Integer Linear Program (MILP). Section 1.11.5 contains a description of our approach to solve the problem. In Section 1.11.6, we present the results of our experimental study with data from the city of Berlin. Finally, Section 1.11.7 contains a conclusion and outlines ideas for future work.

1.11.2 Related Work

The (informal) problem description from Section 1.11.1 is close to the functionalities of the Next Generation Mobile Tourist Guide (MTG) envisioned in [VV07], and can be formally defined as a variation of the Orienteering Problem (OP) (also known as the

selective traveling salesman problem [LM90]) which is proven to be NP-hard [GLV87]. There has been extensive research regarding the OP and extensions thereof. In this section, we will discuss the general algorithmic research regarding the OP as well as the literature that specifically deals with tourist trip planning.

For a much more detailed overview of the state of the art, we refer to the mentioned survey papers [Gav+14a; GLV16; VSV11] as well as the recent textbook [VG19]. The first computationally feasible mathematical formalization of the sport of orienteering [CGW96a] is given in [Tsi84]: participants have limited time to visit predefined checkpoints starting and finishing at a specific control point. Each checkpoint is associated with a score. The goal is to maximize the total score of all visited checkpoints. From this basic problem definition, several variations evolved. In the following, we will discuss those variants that are relevant for the problem introduced in Section 1.11.1.

Optimizing multiple tours (each limited in time) with the requirement that every checkpoint should still be visited only once is called the Team Orienteering Problem (TOP) which was introduced in [CGW96b]. The restriction that checkpoints may only be visited within specified time windows was first introduced in [BFG07]. The multi-period OP with multiple (arbitrary) time windows is presented in [Tri+10]; [Sou+13] shows an extension with extra knapsack constraints. The combined problem is named (Team) Orienteering Problem with Time Windows (T)OPTW which is closely related to the Selective Vehicle Routing Problem with Time Windows (SVRPTW) [Gue99]. The SVRPTW limits the vehicle capacity as well as the maximum distance traveled. The Time-Dependent Orienteering Problem (TDOP) is presented in [FL02]. The combination of the aforementioned problems is the Time Dependent Team Orienteering Problem with Time Windows (TDTOPTW) which was first presented in [Gar+13]. As some PoIs require a specific continuous amount of time spent for the visit, this induces the OP with Variable Profits (OPVP) which is studied in [EL13] and applied in [Yu+14] for the city of Istanbul with 20 PoIs to maximize time at PoIs and minimize time spent to travel between PoIs. However, the other extensions (time dependency, “team” version, time windows) are missing here.

One of the practical applications of the OP besides vehicle routing is the Tourist Trip Design Problem (TTDP). The basic OP can be regarded as the most simplistic TTDP [VSV11]. However, to model realistic tours, the variations described before are useful: the team version to compute multiple tours with non-overlapping sets of activities, time-

dependency to support using public transport between points of interest, as well as time windows to consider opening times of attractions. An overview of the latest research regarding the TTDP can be found in [Gav+14b; GLV16]. Most approaches used to solve realistic instances of the TTDP employ heuristic algorithms such as evolutionary genetic algorithms [AS11; AKE17] ([AS11] was evaluated with data of the city of Tehran; [AKE17] was evaluated on 15 major cities in Iran – both employ a shortest path routing routine as subroutine of the tour optimization), iterated local search (ILS) [Aya+17; Gar+10; Gav+15b; Van+09], or simulated annealing [LV12]. There are formulations in the form of a Mixed Integer Linear Program (MILP) of some variations of the OP (e.g. the TDOP in [GLV16] and the OPVP [Yu+14]). The system proposed in [Sub+18] takes real-time information such as traffic and queue length at the attractions (manually provided by administrators of the system) into account.

1.11.3 Contribution

In this section, we propose several realistic extensions to state-of-the-art variations of the orienteering problem. These extensions are specifically relevant to compute practical solutions when optimizing tourist trips. To the best of our knowledge, we present the first combination of the TDTOPTW and the OPVP with arbitrary time windows. The profit functions are personalized depending on the properties of each PoI as well as the preferences of the respective tourist. Additionally, our formulation of the problem supports multiple entries and exits for PoIs covering a widespread area. This is relevant in practice because especially for large PoIs like a zoo or a park, each entry/exit may be served by different public transport lines. Solutions computed by our approach respect the time required to walk from the entry to the exit of the PoI. Existing models associate each PoI with exactly one geographic coordinate which can lead to suboptimal routes in such cases.

We present the first Mixed Integer Linear Program (MILP) representation of the TDTOPTW with the aforementioned extensions. Furthermore, we present a iterated local search (ILS) algorithm that solves the problem fast enough for practical purposes (e.g. as a backend for a web-based or mobile app service for tourists). Approaches based on ILS have been proven to be well suited to efficiently compute feasible solutions for the TDTOPTW and to produce high quality results [Aya+17; Gav+15a; Van+09]. In

our evaluation on data for the city of Berlin with 41 diverse PoIs we compare the results of the ILS-based approach with the results of a MILP solver.

Visiting a park is worthwhile on its own. Therefore parks can be a PoI. However, parks may as well contain more PoIs (e.g statues). Thus, our model also supports PoIs in parks or other areal PoIs.

1.11.4 Modeling the Problem

Profit Function for Points of Interest

In this section, we define a generalized profit function which takes the visit time as input and returns the profit gained when visiting the PoI for this amount of time. As mentioned in Section 1.11.1, there are many different profit functions. Most PoIs require a certain amount of time to achieve any profit. Then, the accumulation of profit will flatten out and finally staying longer at a PoI will not yield any further profit. To model this behavior, we introduce a piecewise linear function

$$p(t) = \begin{cases} 0 & t < t_{\text{minvisit}} \\ p_{\text{min}} + (t - t_{\text{minvisit}}) \cdot \text{ppt} & t_{\text{minvisit}} \leq t \leq t_{\text{maxvisit}} \\ p_{\text{max}} & t_{\text{maxvisit}} < t \end{cases}$$

where:

- t_{minvisit} is the minimum time a tourist needs to visit a PoI before profit can be gained
- t_{maxvisit} is the maximum amount of time. Staying longer should not accumulate any further profit.
- p_{min} is the minimum profit a tourist gains when staying at least t_{minvisit} at the PoI
- p_{max} is the maximum profit a tourist can gain by visiting the PoI
- ppt is the profit per time unit gained at the PoI after the minimum visit time is exceeded. It can be calculated with the two points $(t_{\text{minvisit}}, p_{\text{min}})$ and $(t_{\text{maxvisit}}, p_{\text{max}})$.

A movie theater would have p_{min} set equal to p_{max} to ensure the visit does not last shorter but at the same time also not longer as the movie duration (plus some time buffer). As the attractiveness of a PoI depends on the preferences of the user, we multiply the profit values p_{min}, p_{max} with the preference value of the user for this PoI. For each category (e.g. “shopping”, “museum” or “public monument”), the user rates their interest on a scale from 0 (not interested) to 10 (highly interested). Each PoI is tagged by at least one category. The preference value of the user for each PoI is then calculated by dividing the sum of the preference values given by the user for each category of the PoI by the total number of categories of the PoI (as a normalization to not give weight to PoIs with multiple categories).

Mixed Integer Linear Program

In our definition of the MILP, we make use of the following notation. Inputs are noted as capital letters, output variables are lower case. A “location” is used synonymously to entry/exit of a PoI and is therefore associated with exactly one PoI. ℓ is used as an arbitrary large number. As input variables we use:

P	set of all PoIs
M	set of all tours
N_m	set of all locations for tour m
Z_{ip}	1 if location i belongs to PoI p, otherwise 0
T_m	set of all discrete timeslots for tour m
T_{ij}^{walk}	walking duration from location i to j
T_{ijt}^{travel}	travel time from location i to j departing in timeslot t
T_t^{slot}	starting time of timeslot t
T_m^{start}	starting time for tour m
T_m^{max}	time limit for tour m
$F_i(x)$	profit function for location i
P_i^{max}	maximum profit at location i
$P_i^{\text{min}_t}$	minimum visit time at location i
W_{im}	set of time windows for location i and tour m
O/C_{iwm}	opening/closing time for location i, tour m, time window w

As output variables we use:

p_{im}	profit accumulated at location i in tour m
y_{ijmt}	location i is left to location j in timeslot t for tour m
x_{pm}	number of visits to PoI p in tour m
t_{im}^{poi}	time spent visiting location i in tour m
s_{im}	arrival time at location i in tour m
j_{im}	helper variable for our piecewise linear profit function
g_{iwm}	helper variable for modeling multiple time windows

We define the objective function as: $\max \sum_{m \in M} \sum_{i \in N_m} p_{im}$ which sums up the profit gained over all PoIs in all tours. The profit is 0 if the PoI is not visited on the tour.

Otherwise, the gained profit is the value of the profit function defined in Section 1.11.4. Locations 1 and N are the starting and ending point provided by the user. These may differ for each tour. All other locations are fixed. Constraint 1.6 ensures that the first location is left exactly once and the last location is reached exactly once:

$$\sum_{j \in N_m} \sum_{t \in T_m} y_{1jmt} = \sum_{i \in N_m} \sum_{t \in T_m} y_{iNmt} = 1 : \forall m \tag{1.6}$$

To ensure that every PoI is entered at most once and left the same number of times (i.e. once or not at all), we introduce the following constraints ($y_{ijmt} \cdot Z_{jp}$ connects locations used in y with their PoIs in p):

$$\sum_{i \in N_m} \sum_{j \in N_m} \sum_{t \in T_m} y_{ijmt} \cdot Z_{jp} = \sum_{i \in N_m} \sum_{j \in N_m} \sum_{t \in T_m} y_{ijmt} \cdot Z_{ip} = x_{pm} : \forall m, \forall p \tag{1.7}$$

$$\sum_{m=1}^{M} x_{pm} \leq 1 : \forall p, \tag{1.8}$$

As we allow for entering and leaving PoIs at different locations, we introduce the following constraint to ensure that the minimum walking time between the two locations of the PoI is taken into account. The product of $Z_{kp} \cdot Z_{ip} \cdot T_{ik}^{\text{walk}}$ yields 1 only for locations of the same PoI.

$$(1 - \sum_{l \in N_m} \sum_{t \in T_m} y_{klmt} - \sum_{x \in N_m} \sum_{t \in T_m} y_{ximt}) \cdot Z_{kp} \cdot Z_{ip} \cdot T_{ik}^{walk} \leq t_{i,m}^{poi} \forall i, k, p, m \tag{1.9}$$

The following constraint ensures that each tour duration is limited to T_m^{max} given by the user. The duration of each tour is the sum of the time spent at PoIs and the time required to travel between PoIs. The time at the last location of the tour (e.g. the hotel) should be smaller than the sum of the start time and the maximum travel time.

$$s_{Nm} \leq T_m^{\text{start}} + T_m^{\text{max}} \tag{1.10}$$

The following constraints are needed to ensure that the correct departure time is used at each PoI - i.e. the PoI is left after the visit is finished (and not before). Both constraints are automatically fulfilled by the right hand side (given any big number M),

if the corresponding y variable is 0 or in case of Constraint 1.11 if locations k and i do not belong to the same PoI.

$$s_{im} + t_{im}^{\text{poi}} \leq T_t^{\text{slot}} + \ell(1 - y_{kjmt} \cdot Z_{ip} \cdot Z_{kp}) : \forall i, j, k, p, m, t \tag{1.11}$$
$$T_t^{slot} + T_{ijt}^{\text{travel}} \leq s_{jm} + \ell(1 - y_{ijmt}) : \forall i, j, m, t \tag{1.12}$$

To ensure that the time-dependent travel times between PoIs are respected, the following constraint is required. This constraint is automatically fulfilled by the right hand side (given any big number ℓ), if either k is never left towards j or if k and i are not locations of the same PoI.

$$s_{im} + t_{im}^{\text{poi}} + T_{kjt}^{\text{travel}} - s_{jm} \leq \ell(1 - y_{kjmt} \cdot Z_{ip} \cdot Z_{kp}) : \forall i, j, k, p, m, t \tag{1.13}$$

To model the profit function for each location, we use the following constraints. We introduce j as helper variable which (due to Constraint 1.16) is 1 *iff* the visiting time is less than the minimum visit duration. Therefore, Constraint 1.18 forces the gained profit to be 0 if this is the case.

$$\begin{aligned}
p_{im} &\leq F_i(t_{im}^{\text{poi}}) + \ell j_{im} && \forall i, m && (1.14)\\
p_{im} &\leq P_i^{\max} x_{im} && \forall i, m && (1.15)\\
t_{im}^{poi} + \ell j_{im} &\geq P_i^{\min_t} && \forall i, m && (1.16)\\
p_{im} &\geq 0 && \forall i, m && (1.17)\\
p_{im} &\leq \ell - \ell j_{im} && \forall i, m && (1.18)
\end{aligned}$$

To model multiple time windows per PoI/location and ensure that every visit of a PoI takes place within a time window of this respective PoI, we introduce the following constraints. Constraint 1.19 ensures, that a visit s_{im} at location i in tour m starts after an opening time O_{iwm} (w is the index of the specific time window) whereas Constraint 1.20 does this analogously for closing times. Constraint 1.21 ensures only opening and closing times of the same time window are matched by introducing variable g_{iwm}. Thus,

either index i in O_{iwm} and C_{iwm} matches or Constraints 1.19 and 1.20 are always true

$$O_{iwm} \cdot g_{iwm} \leq s_{im} \qquad \forall i, w, m \tag{1.19}$$

$$s_{im} \leq C_{iwm} + \ell(1 - g_{iwm}) \qquad \forall i, w, m \tag{1.20}$$

$$\sum_{w \in W_{im}} g_{iwm} \leq 1 \qquad \forall i, m \tag{1.21}$$

Finally, we set the starting location and ensure positive visit times through the following constraints:

$$s_{1m} = T_m^{start} \tag{1.22}$$

$$t_{im}^{poi} \geq 0 \quad \forall i, j, m, t \tag{1.23}$$

This form is suitable for MILP solvers and was programmed in Gurobi [Gur20] for the experimental study of this section presented in Section 1.11.6.

1.11.5 Approach

In this section, we will describe the preprocessing required to deliver real-time response times for user queries as well as different heuristic approaches to solve the problem defined mathematically in Section 1.11.4.

Preprocessing

Like in [Gar+13], we use an offline preprocessing step which does not have real-time requirements. We precompute intermodal time-dependent shortest paths for public transport and walking connections from every location to every other location for every timeslot. Our routing is based on [Gün+14][15] and respects realistic transfer times, allows for walking between stations, and does not assume periodicity of the timetable. We use a realistic pedestrian routing based on OpenStreetMap data for the path between the PoI location and the next public transport stop. Therefore, we precompute shortest

[15]The latest version is available as Open Source Software at https://motis-project.de/

paths from every PoI location to every public transport stop and vice versa.

The fact that most shortest path algorithms for public transit routing (like RAPTOR [DPW12], Connection Scanning [Dib+13b], and all graph-based Dijkstra variations like [DMS08b]) are inherently computing shortest paths to all targets at the same time, can be exploited here. Here, each shortest path problem is independent from every other shortest path problem. Thus, this task is perfectly suited to be carried out in parallel.

The result is a time-expanded directed acyclic event-activity graph where every node is either an arrival or a departure at a PoI location (in a discrete timeslot). Arrivals and departures at the same location are connected by *visit* edges. To model entering and leaving a PoI at different entries/exits, arrivals and departures at different locations of the same PoI are connected by *intra* edges. *Intra* edges and *inter* edges need to conform to the computed walking durations and travel times respectively. *Visit* edges and *intra* edges are only created for visit times greater or equal the minimum visit time specified for the respective PoI profit function. Finally, *inter* edges connect departure and arrival nodes of different PoIs.

Start and end location are both provided by the user. Thus, these can either be limited to a set of known hotels and public transport stations where tourists typically start their trip, which allows us to include them in the preprocessing. If this is not an option, four additional queries need to be carried out online at query time: from the start location to every location, using the start time as earliest departure time (forward search one to all) and *to* the last location from every location, taking the start time plus the maximum trip time as latest arrival time (backward search all to one). These two queries need to be done for the pedestrian routing (to all locations and public transport stops) as well as the public transport routing (initiating the labels with the results from the pedestrian routing). Both, forward and backward direction can be computed in parallel. These nodes and edges are added to the event-activity graph.

Heuristic Algorithm

To supplement computing solutions to problem using a MILP solver (which is time-consuming as discussed in the Section 1.11.6), we develop heuristic algorithms: as a baseline, we present a Basic Greedy Algorithm (BGA) and an Advanced Greedy Algorithm (AGA). As outlined in Section 1.11.2, Iterated Local Search (ILS) based approaches were

able to compute high-quality results in real-time. Therefore, we decided to implement an ILS approach to solve the problem at hand on the graph described in Section 1.11.5. We present our Basic Iterated Local Search (BILS) as well as our Specialized Iterated Local Search (SILS).

Basic Greedy Algorithm

This algorithm solves the problem sequentially, building a tour from start to end by adding one greedily chosen activity at a time. For each expansion, the BGA has to choose the next PoI, a visit time, and a location to exit the PoI at. This can be done efficiently on the event-activity graph described in Section 1.11.5. To compute valid tours where no PoI is visited twice, the algorithm has to keep a set of already visited PoIs and prevent expansions which would result in revisiting PoIs which were already visited. This set is kept for all tours that will be planned. Additionally, the algorithm requires a "dead-end protection": there are PoIs from where it is not possible to reach the end location within the maximum tour duration. To prevent this, the graph can be pruned by removing nodes that have no transitive path to the end location. A backward BFS starting from the end location nodes can mark all feasible nodes. Nodes not marked within this run are omitted in the expansion step of the BGA. Another solution would have been to introduce a backtracking step if the algorithm visited a dead-end. The BGA chooses the next PoI based on a weight function $p/(w \cdot T_{\text{travel}} + T_{\text{visit}})$ where w controls the influence of the travel time. Note that this algorithm greedily selects only steps that look locally promising. However, a globally optimal solution may contain steps which will not be chosen with any w value.

Advanced Greedy Algorithm

To improve upon the BGA, the AGA also makes (locally) suboptimal steps and keeps a list of multiple active solutions. The basic properties of the BGA (duplicate PoI prevention and dead-end protection) stay the same. However, it makes multiple expansions in each step - each one with a different value for w. After each complete step, it cuts off all solutions with a lower profit per time duration than the best solution times the cutoff threshold. A high cutoff threshold implies many cut off paths and therefore a better computing time but also a decreased chance to find better solutions (i.e. paths from

the start location to the end location in the event-activity DAG). A cutoff threshold of zero combined with a unlimited list of active solutions would result in listing all feasible solutions. This would yield the optimal solution but is not feasible in practice for realistic problem instances.

Basic Iterated Local Search

A ILS basically uses a Local Search to find a local optimum. After that, the local optimum is perturbed sufficiently enough to be able to escape the previous local optimum and find a local optimum. The algorithm terminates if the Local Search cannot find a new local optimum after a certain number of perturbations.

In our case, the search can be either seeded with an empty route (respectively multiple empty routes if we are planning more than one tour) from start to finish (without visiting PoIs) or the result of one of the previously described greedy algorithms. We define our neighborhood for the local search step as all solutions which can be produced by integrating a visit to a new PoI while still keeping the solution feasible. All existing visits keep their arrival and departure time. We decide to insert always the (locally) best PoI visit (i.e. the PoI which has the best profit per time including travel time) using the maximum visit time. This will be done until it is not possible to add yet another PoI. The perturb step removes a varying number of PoI visits from the current solution. The remaining PoIs are then shifted forward in time (i.e. towards the start of the route) as much as possible. The number of removed PoI visits is incremented (to improve the chance to find a new local optimum) if the new solution is equal or worse (regarding the profit value) than the previous solution.

Advanced Iterated Local Search

Since the previously presented heuristic algorithms always select the maximum visit time (by locally optimizing the profit value), it would be interesting to introduce options to lengthen (in the local search) or shorten (in the perturb step) a visit at a PoI. We add options to extend the visit of a PoI to the Local Search neighborhood. This is done by moving the arrival time to an earlier point in time or by moving the departure time to a later point in time. The visit time is extended by 5 minutes in each step. Since

the extension step is called repeatedly, the algorithm should eventually be able to find new optima. The perturbation step is now capable of shortening all PoIs from the front or back. As previously noted, only feasible solutions are allowed as Local Search and perturbation step result. Still, the best neighbor is chosen and the Local Search step continues until the neighborhood does not contain any improvement.

1.11.6 Experimental Results

In this section, we will present the results of our experiments. As MILP solver, Gurobi [Gur20] was used and executed on a computer with an Intel® Xeon® CPU E3-1245 V2 processor (3.4GHz) and 32 GB of RAM. Everything else was run on a computer with an Intel® Core® M i3-5005U processor and 8 GB of RAM. The greedy and ILS algorithms are implemented in C++.

The test instance are 41 hand-picked PoIs in Berlin from various categories with manually researched opening and closing times [16]. The main categories were defined as "Museum", "Monument", "Panorama", and "Experience". More details can be derived from the theme category "Art", "Nature", "History", "Famous", and "Shopping". Each PoI can have multiple categories. It is also possible for a tourist to set a high preference value for only a single category - e.g. if they are interested in a tour of famous landmarks of the city of Berlin (e.g. the Brandenburger Tor, pieces of the Berlin Wall, etc.). This could also be used to generate interesting ideas for theme tours for so called "Hop-On Hop-Off" buses (albeit with a street routing algorithm to generate the event activity DAG).

We manually picked 25 different queries covering a diverse set of combinations of maximum duration (between 2-10 hours), number of tours (one or two) with four possible start and end locations. We chose to evaluate the algorithms with a balanced profile as a single high preference value for one category eliminates all but a few PoIs which produces unvaried tours and makes the problem much easier to solve. This would not make for a good benchmark.

The timetable for the city of Berlin was kindly provided by Deutsche Bahn for research purposes.

[16]The data is freely available at https://github.com/motis-project/berlin-pois

Preprocessing Step	Computing Time
Data Initialization	46 ms
Calculating Walk Times between PoIs	11.5 min
Calculating Travel Times between PoIs	2.5 min
Building Event-Activity DAG	6,6 s
Precomputing Paths for Query Positions	3.5 min
Integrate Query Data	12 ms
Total	15 min 23 s

Table 1.17: Preprocessing Computation Times

	Granularity 1 min	Granularity 5 min	Granularity 10 min
Arrival Nodes	81,452	67,963	58,815
Departure Nodes	88,470	17,694	8847
Inter Edges	9,382,766	1,877,691	939,597
Visit Edges	5,130,816	929,892	438,198
Intra Edges	4,044,902	727,959	329,015

Table 1.18: Graph Information for Different Granularity Settings

Preprocessing In Table 1.17, we report the time it takes to finish the preprocessing step described in Section 1.11.5. The total duration (approximately 15min) is in a range where the preprocessing can even be repeated with minimal effort when the timetable or pedestrian routes change.

Table 1.18 shows the size of the event-activity DAG for different granularities of the timeslots.

MILP Solver We ran every query with the MILP solver for 10 hours each. The solver was seeded with the best greedy solution. For the simplest query (a two hour tour), the solver did find an optimal solution. For all other queries, the solver provided the best solution known so far as well as an upper bound for the profit of the best solution possible. The difference between the upper bound and the currently known best solution ranges between 6% and 20%.

Greedy Algorithm The BGA from Section 1.11.5 was evaluated using 13 different travel time weights (0-10, 15, and 20). In general, extreme travel time weights such as 0, 15, and 20 performed badly as it is not reasonable to chose only very close PoIs or only high profit PoIs (ignoring travel time completely). For long tours, lower travel time values seem to outperform higher values whereas for short tours, the opposite is the case. This makes sense because for short tours, long travel times leave not much time for the actual visits. Therefore, it could be useful to select the travel time weight depending on the tour length. The BGA from Section 1.11.5 takes about 2-4ms to complete. Comparing the result with those from the MILP solver, we see that the MILP solver consistently outperforms the BGA by 5-10 pp. For one query, the gap is even 16 pp. The gap is especially high for queries with long maximum travel times or even multiple tours because the solution space increases drastically.

The AGA from Section 1.11.5 was tested with different numbers of active solutions (100, 1.000, 10.000), different cutoff thresholds (0.25 and 0.5) as well as different numbers of chosen candidates (1, 3, 5). This yields 18 variations which we supplemented with one further combination: 100.000 active solutions, cut-off threshold 0.5 and 3 chosen candidates. The best solutions were found with the latter parameterization (15 times), closely followed by $10.000/0.25/5$ (active solutions / threshold / chosen candidates). The best configuration with 1.000 active solutions was $1.000/0.25/5$ which produced the best known solution in 12 cases. The main driving factor for the processing time is the number of active solutions: the AGA takes around 1 second for 100 solutions, 10 seconds for 1.000 solutions, 50 seconds for 10.000 solutions, and 500 seconds for 100.000 solutions. The other parameters do not influence the processing time significantly. Comparing the AGA with the MILP solver, the solver still outperforms the result quality of the greedy algorithm by a huge margin of up to 15 pp. Interestingly, for five queries, the AGA was able to compute slightly better solutions (2-3 pp) than the MILP solver (which was halted after 10 hours).

Iterated Local Search The BILS presented in Section 1.11.5 was not able to improve upon the seeds from the best greedy algorithm except in one case, where the profit was marginally improved (from 1231 to 1235 profit). As the best greedy algorithms are also very slow, we differentiate more between the different greedy algorithm parameterizations and observe that the BILS is quite capable of improving upon bad seeds from

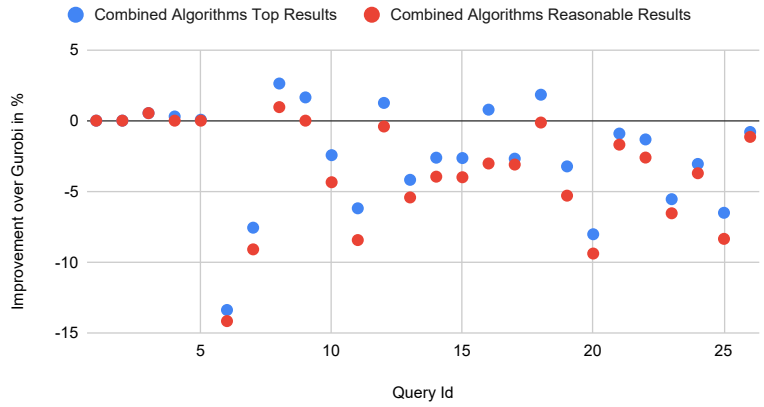


Figure 1.18: Improvement over Gurobi Results for the Combined Algorithms: the highest profit solution in blue and the highest solution which can be computed in real-time in red.

extreme travel time weights (0, 15, and 20) for the BGA. Although the improvement upon the highest quality seeds is marginal, the BILS is nonetheless interesting due to its fast computation times averaging around 1.5 seconds.

The AILS described in Section 1.11.5 was not able to improve upon the previously known best solutions of our heuristic algorithms except for a slight improvement for one query (1402 to 1403). Compared with the basic ILS, the query runtime of 5 seconds on average does not yield a worthwhile benefit.

Overall Comparison Figure 1.18 shows an overall comparison of the heuristic algorithms with the best solution found by the MILP solver after 10 hours. For simple queries (1-5), the solutions do not differ much. However, there are queries where the best solution found by a heuristic algorithm is not even close (more than 10 pp difference) to the solution found by the solver. Interestingly, in some cases, the heuristic algorithms were able to find slightly better solutions (2-3 pp) in some cases.

Granularity Analysis Previous instances were reported with a 5 minute granularity for timeslots. Now, we also vary the granularity and test the values 1 minute, 5 minutes,

and 10 minutes. The results show a strong correlation between query runtime and granularity: computing results with a one minute granularity takes about 5 times as long as for the 5 minute granularity while at the same time, the 10 minute granularity made the processing about twice as fast as the 5 minute granularity. The profits for the 1 minute granularity only improve between 0.5 pp to 1 pp (depending on the query) compared to the 5 minute granularity. However, the increase of profit value from the 10 minute granularity compared to the 5 minute granularity ranges from 0.5 pp to 5 pp. All in all, the 5 minute granularity seems to be a good trade-off between result quality and processing time.

1.11.7 Conclusion and Future Work

In this section, we presented several realistic extensions to the previously known definition of the TDTOPTW (a variation of the TTDP) to make tourist trip planning more feasible in practice and combine the TDTOPTW with the OPVP to account for variable personalized PoI profit functions. For instance, the problem definition presented in this section supports multiple entries and exits for each PoI. We presented the first MILP modeling of the TDTOPTW including the described realistic extensions. The approach is split into two phases: the preprocessing phase has no real-time requirements and computes a time-expanded event-activity DAG by routing optimal public transport and walking connections from every PoI entry/exit to every other PoI entry/exit at every time with different granularity (here, we used 1, 5, and 10 minutes). This allows for efficient trip planning at query time and eliminates the need for repair steps as required by most previous approaches.

As the MILP solver takes quite long with the current definition, an interesting research direction would be to search for ways to improve the representation in order to solve the problem online in real-time.

In the future, the system could be extended to support adaptions of the profit functions of PoIs depending on the weather forecast (i.e. prefer indoor activities for rainy days). Additionally, the tour can be split further into smaller parts to allow for lunch and/or dinner. Note that both of these extensions neither require any adjustment of the MILP nor any changes to the ILS algorithm but can be encoded into the input. Furthermore, the algorithm described and implemented in this section could be used as a backend

service for interfaces presented in [BMV14]. Combining our approach with [Mor+13] to guess preferences based on previous ratings and activities can make for an even more satisfying tourist experience.

2 Real Time Support

Many transportation systems suffer from delays and other service disruptions such as cancellations, track changes, or reroutings. To counter these problems, service operators may want to provide additional services. All these changes are not considered in a schedule journey planner. In this chapter, we discuss how to update the data model to be able to provide real-time alternatives in case of a broken connection. Furthermore, we present an approach to efficiently monitor millions of journeys in parallel.

2.1 Real-Time Update

The time-dependent graph model does not require any preprocessing to be able to find shortest paths on it. Therefore, it is well suited to be adapted to reflect real-time updates.

Updating the Time-Dependent Graph The real-time update of the time-dependent graph works analogously to the update of the time-expanded graph presented in [FMS08; Sch09]. When updating the time-dependent graph, we need to additionally keep the following properties:

1. For trip i on a route with a sequence of departures and arrivals $\{d_1^i, a_2^i, d_2^i, ..., d_{N-1}^i, a_N^i\}$, the times $t(d_x^i), t(a_x^i)$ need to be ascending: $t(d_1^i) \leq t(a_2^i) \leq t(d_2^i) \leq ... \leq t(d_{N-1}^i) \leq t(a_N^i)$.
2. Routes are not allowed to contain overtaking trips. Thus, if for two trips i and j it is true that $t(d_1^i) \leq t(d_1^j)$ this needs to imply that $t(a_j^i) \leq t(a_j^j)$ for every following stop $j > i$. If overtaking on a route takes place after a real-time update, one trip needs to be moved to a separate route.

If the first property is not satisfied, this is a data error in the real-time updates. But since this is not uncommon, we need to handle these data errors and ensure that the routing graph stays valid. Since overtaking can take place in reality, this is not necessarily a data error. The overtaking property can easily be checked (and potentially fixed) after all real-time updates have been applied.

Delay Propagation and Graph Repair The system receives real-time updates for events (arrivals, departures) that just took place as well as (potentially inconsistent or incomplete) delay forecasts. Thus, we propagate delays along the trips as well as along waiting relationships if it is known beforehand that one trip will wait for another train for a certain amount of time. Implicitly, we use the concept of the dependency graph from [FMS08]. However, we do not explicitly create node and edge objects for this task in memory because this would be another time-expanded graph which would be very inefficient regarding memory usage. As described in [FMS08], the maximum value from the schedule time, the propagated time, and the received delay forecast is taken if there is no update that the event took place at a specific time. If there is such an update message available, this time overrides the maximum.

In addition to the timestamp types used in [FMS08], we introduce a special *repair* timestamp type that overrides real-time updates in case different real-time updates contradict each other regarding the first property mentioned above. To repair a trip, each event's definite timestamp is interpreted as a minimum for all the following events in the trip and as a maximum for every preceding event in this trip. If we do this for every event of the trip, we can set the repair timestamp t_{repair} accordingly (i.e. $\min \leq t_{\text{repair}} \leq \max$).

Service Changes For every trip, we introduce an attribute which indicates whether the trip section is valid. This needs to be respected when routing (i.e. in the time-dependent edge weight function). Now, we can simply disable every trip that needs to be update (i.e. additional events, cancelled events, or a complete cancellation) and build a new route with the new stop/event order.

Evaluation In our evaluation on real-time updates provided by Deutsche Bahn from September 2019, we count approximately 3.5M real-time updates per day. Updating a

whole day (i.e. fast-forward from the schedule timetable to the end of the day real-time timetable) takes 15 seconds. This means, updating the timetable model takes is 0,01% days per day which is negligible for most use cases. The routing times do not differ from those of the schedule timetable.

2.2 Scalable Monitoring of Journeys

This section is based on the article *Efficient Monitoring of Public Transport Journeys* accepted for publication by the journal *Public Transport*.

Many things can go wrong on a journey. From minor disturbances like a track change to major problems like train cancellations, everything can happen. The broad availability of smartphones enables us to keep the traveler up-to-date with information relevant for her journey. This way, the traveler can react to changes as early as possible and make well-informed decisions. Naive approaches are too inefficient to monitor a large number of journeys in real-time. This paper presents an efficient way to monitor millions of journeys in parallel. In our approach, the selection of change notices to be communicated to a traveler may be flexibly adapted to the travelers individual needs.

Every day, millions of travelers use public transportation to get to their destinations. Not all of those journeys run smoothly. Problems may be caused by delays, reroutings, cancellations, track changes, etc. If they occur, information is key to finding a solution. The earlier a problem is communicated by the transportation provider, the more options are available to the customer to react. With the advent of smartphones, it is now possible to inform the user as soon as new information about the situation becomes apparent. This imposes some constraints on the data processing: once a real-time update (e.g. a delay) is available, the system needs to determine the affected journeys in a timely manner. Due to the large number of travelers, the real-time monitoring of all current and future (i.e. booked) journeys is a challenging task for the transportation provider.

On the one hand, the customer always wants to be up-to-date regarding the status of her journey. On the other hand, no one wants to annoy the customer with journey updates she is not interested in: while some customers might be interested in a notice about an upcoming transfer, others only want to be bothered with essential information

regarding the feasibility of the journey. Here, the remaining time until the corresponding event (e.g. the transfer in question) is important: a traveler is probably not interested in a predicted change in arrival time of two minutes when there are still two hours left until the interchange.

This would probably trigger many unnecessary alerts since forecasts so far in the future are inherently inaccurate. Not only currently active journeys need to be monitored. Changes may also concern all future journeys already booked by customers. For example, a schedule change due to planned construction work may change the arrival time at the destination, which should be communicated to all affected customers who already booked their journey.

In this paper, we address both of the above mentioned challenges: our system is capable of monitoring millions of customer journeys in real-time on commodity hardware; every customer can set an individual profile for each journey. This profile specifies precisely about which changes she wants to get informed. Separate profiles for each journey, not one per customer, enables the customer to specify a different "alert level" for different journeys. For example, for her way to work she is only interested in cancellations and delays above 15 minutes but when traveling to an important appointment, she might choose a more verbose setting.

2.3 Comparison to Related Work

Previous work [FMS08; MS09] focuses on incorporating real-time information into a graph model which represents the public transport schedule. This way, it is possible to provide feasible real-time alternatives to customers. However, [FMS08; MS09] do not address any topics related to personalized connection monitoring. In contrast, our approach builds on top of an up-to-date real-time graph and provides a scalable but personalized connection monitoring.

As shown in recent studies conducted in Chicago [TT12] and New York [BMW15], even information in the form of a real-time arrivals boards may increase ridership and customer satisfaction. A study from Stockholm [Cat+11] suggests that real-time information may change the paths chosen by passengers.

The system presented in this paper takes real-time information one step further:

instead of providing information statically through displays at stations or on customer request in an smartphone application, the system monitors all customer journeys. Based on a personalized monitoring profile, it informs the customer (e.g. through smart phone push notification, SMS, or email) about the events she is interested in.

Regarding disruption management, [Jes+09; Tör07] discuss how advanced mathematical models which are already in use in the airline industry can be applied to disrupted railway situations. The approach to passenger information discussed here can be integrated into disruption management systems for all transport modes to improve the communication with the passenger. This will result in improved outcomes for both, the service provider as well as the passenger.

2.4 Monitoring Profile

All interchanges as well as the first departure and the last arrival of a connection are particularly important for the user. Besides the changes concerning the time of an event which is part of an interchange, a user may be concerned about the time between the arrival and departure of an interchange. More specifically, she might not want to be informed about changes where departure and arrival of an interchange are both delayed by N minutes: she has still the same time buffer for the interchange. However, she could be interested in a delayed arrival when the connecting train departs on schedule because this would make the interchange inconvenient or even infeasible. Note, that both aspects can be configured separately. Psychologically, it might feel saver to get a message when any of those aspects change - which is supported the system described here.

The system stores the attribute value (e.g. the timestamp of an interchange event, the interchange time buffer, the overall feasibility of the journey, etc.) most recently communicated to the user for each attribute for each aspect of the stored connection. The initial values are taken from the journey stored by the user. Usually, these are the scheduled values. However, in case of a real-time alternative for a missed connection, the initial values may differ from the scheduled values. If the latest value announced to the user differs from the actual real-time value more than a specified threshold, the user gets informed about this change. After an update has been sent to the user, the updated

value will be stored by the system. The system allows one to monitor the following attribute value changes (relative to the value most recently announced to the user) separately:

1. For the first departure: later departure, earlier departure
2. For the last arrival: later arrival, earlier arrival
3. For the events of an interchange: earlier arrival, later arrival, earlier departure, later departure, and the time difference between departure and arrival
4. For an interchange: a notice about the oncoming interchange a fixed time before the arrival
5. Messages to customers using a specific vehicle. These messages address specific stops of the stop sequence of a vehicle.

The monitoring profile is comprised of one two-dimensional look-up table for each aspect described above.

Another important factor for the decision of whether to inform the user about a change or not is the remaining time until the event will take place. The closer to the event in question, the more urgent it is to inform the user and the more precise predictions can be. For example, an anticipated delay increase of two minutes might be relevant 30 minutes before the actual event. However, predicted delays in a complex public transport network (e.g. the German railway network) cannot be very precise on an extended time horizon of several hours and are thus subject to fluctuations. We introduce the following rule set as means to avoid informing the user about changes that are not (yet) relevant and subject to change: the system requires a two dimensional look-up table where for each remaining time until the actual event (row) and change in minutes (column), it is specified whether the user should get informed about that change. Note that this is not necessarily the way, an end-user would need to specify this: a mobile app might come completely pre-configured or condense these detailed settings into a few profiles to suite different user groups.

Changes in different directions (increase of delay / decrease of delay) might be of different importance to the user. Consequently, the system incorporates different

matrices for each value (listed above) for each direction. For example, a change of five minutes delay increase might have a larger impact for the arrival event of an interchange than for the departure event.

2.5 Basic Terminology

We introduce the notion of departure events e_{dep} and arrival events e_{arr}. Important events are events where the traveler either enters or exits a vehicle. Monitoring these events is sufficient to monitor all relevant aspects (e.g. feasibility) of a journey. We define an interchange as a pair of an arrival and a departure event $(e_{\text{dep}}, e_{\text{arr}})$ where both events have to take place either at the same station or at two stations in convenient walking distance. As our system allows walks between nearby stations to occur in connections, the station(s) of these events do not need to match.

For each station, the schedule provides a minimal transfer time $\text{tt}(s)$ required to get from one vehicle to another. Depending on the input data, this function can be fine-grained and adjustable for elderly or handicapped people. However, for our evaluation we did not have access to data at this level of detail. Furthermore, the schedule contains stations between which it is possible to walk: this is defined as a tuple (s_1, s_2, t) where s_1 and s_2 are two stations and t is the corresponding time to walk from s_1 to s_2.

Thus, an interchange $(e_{\text{dep}}, e_{\text{arr}})$ between two vehicles at the same station s is considered valid if $t(e_{\text{dep}}) - t(e_{\text{arr}}) \geq \text{tt}(s)$. For foot walks, the estimated walking time between the two stations may not be undercut.

2.6 Real-Time Data

2.6.1 Identifier

Real-time data (for example provided by the operator - in our case Deutsche Bahn) contains data that references *stations*, identified by their unique station id. Later on, we will make use of the following entities:

- *vehicles*, identified by the train number[1], first stop id, and first stop departure time

- *events*, identified by the vehicle (see above), the station the event takes place, its schedule time and the event type (either arrival or departure)

This data will be used as key in our data structures (e.g. a hash map) to uniquely identify the referenced entity.

2.6.2 Message Types

In this section, we will outline the structure of the real-time data processed by the system:

- A delay message contains either anticipated or real (measured) delays of public transport vehicles. It refers to the vehicle in question as well as a list of events with their corresponding delay (which can be zero or even negative if the event took place earlier than planned).

- A rerouting message refers to a specific vehicle and contain a list of added stops with their corresponding schedule time and a list of removed stops.

- A cancellation message is analogous to rerouting messages except that there are no added stops.

- A track change message refers to a specific event and contains the new track, the vehicle arrives / departs at.

- A free text message contains a text written by the operator and targets a specific section (identified by the first and the last stop) of a vehicle. It can be used to inform the travelers about important issues (e.g. guidance in critical situations).

[1]The train number by itself is only unique for one traffic day.

2.6.3 Delay Propagation

The information basis for the connection monitoring is the schedule timetable on the one hand and a continuous stream of real-time messages on the other hand. The timetable is represented as a time-dependent routing graph (as introduced in [DMS08a]). This graph is updated according to the received real-time data to represent the current and predicted real-time situation. The basic update approach is described in [MS09]. This approach propagates delays along the vehicle path, respects waiting time rules among vehicles and fixes resulting data inconsistencies (e.g. delay update messages that permute the event order of a vehicle). Note that the application of waiting time rules enables the propagation of delays from one vehicle to another. This way, delays can spread not only along the vehicle path but throughout the entire network.

One objective of the system is to warn users about anticipated problems, not those that are already obvious from the real (measured) delays from the past. The real-time data stream does not contain the propagated delays. Thus, a monitoring approach based on the real-time data itself will not work. Therefore, the system uses both, delays as well as propagated delays for the connection monitoring.

2.7 Connection Monitoring

In this section, we will discuss the two approaches to monitor a set of stored connections. Section 2.8 evaluates and compares the performance of both approaches.

2.7.1 Periodic Approach

One method to monitor a set of stored connections (regarding the values listed in Section 2.4) is to check each connection separately in a fixed period of time (e.g. every minute). The first step is to check for each connection whether it is affected by any received real-time change. The second step is to check if this real-time change needs to be communicated to the user based on her individual monitoring profile (introduced in Section 2.4). Later on, we will refer to this approach as the *periodic* approach. But since most stored connections will not have relevant changes that need to be communicated to the user, this may result in a waste of runtime.

2.7.2 Event-Based Approach

The basic idea of our event-based approach is to determine the set of connections that need to be checked based on the events that changed during the real-time update of the routing graph. Assuming that not every real-time update affects every event contained in the stored connections, this is much less computational effort compared to the periodic approach. This assumption is reasonable considering that most stored (i.e. booked) connections will not take place right now or in the next few hours. Even if the periodic approach would only check the connections of the current, previous, and next day (to detect problems in overnight connections), the system would need to check many connections that are not actually affected by the real-time update.

Handling Cancellations, Reroutings, Delays, and Track Changes

To efficiently determine the set of connections affected by incoming real-time updates, we utilize data structures which allow for a fast lookup. Our goal is to create a function δ_{ev} that takes a set of updated events (either updated directly or through propagation) R as input and returns the subset of these connections that is affected by at least one of the changes. For every changed event e, we aim to implement a lookup of affected stored connections $E[e]$ where E is an efficient $O(1)$ lookup from an event to a set of connections (i.e. a hash map data structure). Using the information that uniquely identifies an event as described in Section 2.6.1 as key, we can build a data structure E mapping an event to the set of connections that contain this event. Regarding the algorithmic complexity, a hash map is a sound choice for this task. With the help of this lookup table E, we can implement $\delta_{\text{ev}}(R) = \bigcup_{e \in R} E[e]$.

To monitor the properties described in Section 2.4 (besides the free text messages – the handling of which will be described in Section 2.7.2), it is sufficient to check the interchanges of a journey as well as its first and last stop. Instead of monitoring every event contained in every stored connection, we can focus on $2N_i + 2$ events per journey where N_i is the number of interchanges $(e_{\text{dep}}, e_{\text{arr}})$ of journey i. This accounts for all interchanges as well as the first departure and last arrival.

Since a single connection i is associated with several keys ($2N_i+2$ events) which wastes memory, we create an indirection: the map just stores a unique integer that references the

associated connection. To lookup the details of a connection, we introduce an additional map data structure mapping from the unique connection integer to the details (stop sequence, used vehicles, walk times, etc.) describing the associated connection. When adding a new connection to the set of monitored connections, this connection needs to be indexed: every important event (first departure, last arrival, and all interchanges) gets extracted and added to the mapping.

Handling Free Text Messages

For the special case of free text messages, the approach described in Section 2.7.2 is not suitable because free text messages can target arbitrary events in the journey, not just interchanges, the first departure, and last arrival. Fortunately, instead of indexing all events of the journey, indexing the vehicles used in a journey is sufficient. Analogously to E which maps events to connections, hash map V maps vehicles onto a set of connections containing the respective vehicle. Thus, we can define a lookup function for vehicles $\delta_{\mathrm{v}}(T) = \bigcup_{t \in T} V[vehicle(t)]$ where T is a set of free text messages and $vehicle(t)$ extracts the vehicle in question from a free text message. Uniting the results of both functions, δ_{ev} and δ_{v} yields the set of affected connections.

Putting the Pieces Together

After the affected connections have been determined (through δ_{ev} and δ_{v}), the next step is to check for each connection whether the change exceeds the configured threshold introduced in Section 2.4. Thus, the system checks every aspect mentioned in the monitoring profile (described in Section 2.4). Basically, the system needs to store the value known to the user. For example, as described in Section 2.4, for each interchange, the system stores the arrival time, the departure time and the interchange buffer. For each stop, it stores, the free text messages that were communicated to the user. This way, it is possible to check whether the user needs to be informed: if a connection is affected by a real-time change (which is determined using δ_{ev} and δ_{v}), each stored value is compared to the current value. Since the time until the event will take place is also known, both row (change in minutes) and column (remaining time until the actual event) of the change two dimensional look-up table (introduced in Section 2.4) are

available. Consequently, the system can compare the current change in each entity with the threshold value from the two dimensional look-up table.

Deferred Checking

If the difference between the value (e.g. interchange time buffer for a specific interchange of the connection) last known to the user and the current value determined by the system (based on the real-time schedule and delay propagation) exceeds the threshold from the two dimensional look-up table, the system informs the user instantly. Otherwise, the system needs to check the connection again later: as time goes by, the time until the monitored event decreases. Therefore, other (smaller) threshold values become relevant in the two dimensional look-up table. By iterating the rows (time until event) in the column (computed change) from the current row in decreasing order, we can determine the first row where the user would need to get informed if the computed change stays the same. Therefore, the system may setup a timer to check the connection again at this time. This can be seen as an event-driven simulation. Since these are potentially many timers, the system collects all theses events in a queue which is sorted by the timer expiry (earliest check first). Since every user has his own personal preferences on time thresholds, each event may occur multiple times (one for each stored connection). This way, only one timer for the first element in the queue is required. When the timer expires, the system iterates all entries from the queue until the first entry where the expiry value is not exceeded anymore. When checking those iterated entries it may be the case that the user already was informed because of another change that occurred between the insertion into the vector and the iteration. It may also be the case that the user needs to be informed. In both cases, the entry is removed from the vector. Lastly, the value may have changed between the insertion into the vector and the iteration so that the user does not need to get informed now. In the last case, the entry will be reinserted into the queue with the new timer expiry value. Comparing the event-based approach to the periodic approach, the event-based approach reduces the number of connections to look at by checking on demand, not periodically.

2.8 Evaluation

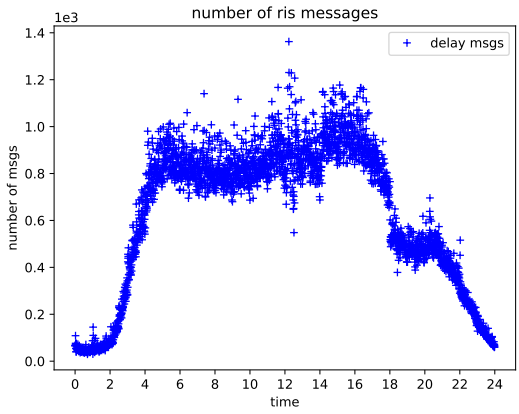
In this section, we present an experimental study of the concepts presented in Section 2.7. We evaluate the event-based version and the periodic version. Additionally, we measure the runtimes of parallel versions of both implementations. All versions are implemented in C++ and run on a machine with an Intel® Core™ i7-6850K CPU and 64GB of RAM.

2.8.1 Schedule Timetable and Real-Time Data

Both, the schedule timetable as well as the real-time data are provided by Deutsche Bahn. For our evaluation, we use the full public transport schedule timetable of Germany. This includes buses, streetcars, subway, suburban trains, regional as well as long distance trains. We analyze a timespan of one week. In this timespan, $250,000$ stations are served with an average of 24M events per day. The time-dependent timetable graph has 4.9M nodes and 15.1M edges. A cumulative real-time update is sent every 30 seconds by Deutsche Bahn. Note that our system (especially the event-based approach) is also capable of handling updates at arbitrary times or update periods. Every real-time update package sent by Deutsche Bahn contains an average of approximately 666 real-time updates (1.9M updates per day). Those updates cover all trains (including suburban railway) operated by Deutsche Bahn as well as the real-time data of certain local public transportation authorities (e.g. for Berlin or the Rhein/Ruhr area). The system propagates these primary delays through the timetable network which results in another 6713 forecasted delays per update that need to be processed by the connection monitoring system.

Figure 2.1 shows the delay distribution over a regular Tuesday (number of messages against time of day). Note that a delay message is not only sent for delayed arrival and departure events but also for events that take place as scheduled. Thus, there is at least one delay update for each event of vehicles operated by Deutsche Bahn plus several delay forecast updates that need to be processed by the system. Each dot represents the messages from a 30 second time interval. As is clearly visible, the delays are not uniformly distributed but raw delays received from Deutsche Bahn (left plot) dip to below 100 update messages (per 30 seconds) at night times while staying above 900 update messages every 30 seconds for several hours in the daytime. The reason for this

(a) delay messages received



(b) other messages received

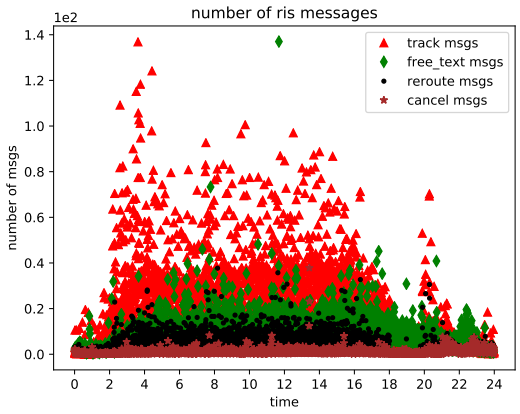


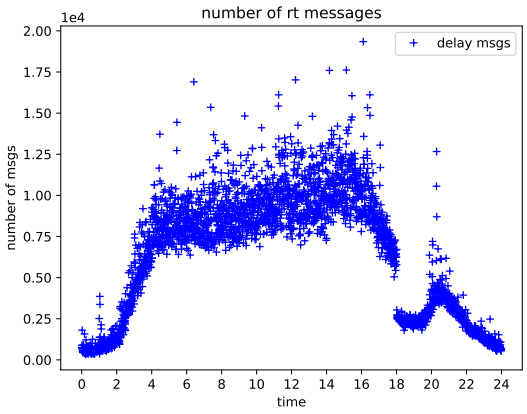
Figure 2.1: Distribution of real-time (delays, cancellations including reroutings, track changes and free text updates) messages over the day: one dot represents all messages that occurred in a 30 seconds time period. (a) shows delay messages received from Deutsche Bahn. (b) all other message types without the delay messages

is that Deutsche Bahn operates more trains at daytime than at night time. Peaks at 1200 messages to process in one turn are possible and should not lead to any disruptions. Message spikes are caused by systems that are not under our control. Thus, the reasons for these deviations are not transparent to us. The plot in the middle shows the number of total delays including raw delays as well as propagated delays. Here, we can see that propagation adds roughly one order of magnitude to the message count: raw messages (left plot) peak around 1200 messages every 30 seconds against $12,500$ to $17,500$ when propagating delays. The number of other messages (track change messages, free text messages, and reroute messages) is comparatively small. The pattern of decreased message volume at night times is visible here, too.

2.8.2 Performance Comparison

In this section, we will analyze both approaches presented in Section 2.7 (periodic and event-based) regarding the number of required check operations as well as runtime performance. For this, we generated one million connections by using a multi-criteria

(a) messages propagated



(b) messages without delays propagated

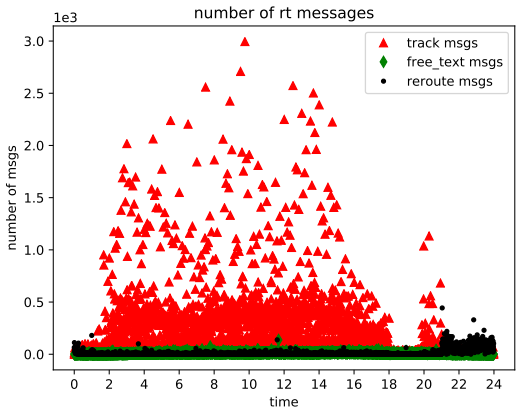


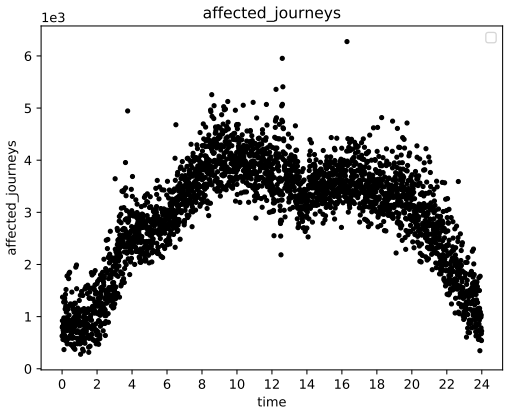
Figure 2.2: Distribution of real-time (delays, cancellations including reroutings, track changes and free text updates) messages over the day: one dot represents all messages that occurred in a 30 seconds time period. (a) shows all messages including those propagated by our system. (b) shows all messages without the delay messages including those propagated by our system.

shortest path routing algorithm. Source, destination, and departure time interval are chosen randomly with a uniform distribution over all stations in the schedule and the complete schedule period of one week. These connections were stored and indexed by both approaches: for the periodic approach no further processing is required whereas for the event-based approach all important events need to be indexed. Event indexing took 19 minutes for $1\mathrm{M}$ journeys. However, note that this needs to be done only once and usually happens gradually when users book their journeys. Indexing a single journey takes $1.15ms$ on average.

Using the periodic approach, all one million connections need to be checked in every iteration. Figure 2.3 shows the number of connections affected by real-time messages (either directly received from Deutsche Bahn or generated by delay propagation). Note that if a connection is affected by a delay, this does not necessarily mean that the user needs to get informed. As we can see, from the one million stored connections, only approximately 10,000 connections need to be checked more closely.

Figure 2.4 shows the runtime of the event-based approach over the day. The increased

(a) affected journeys one day



(b) affected journeys one week

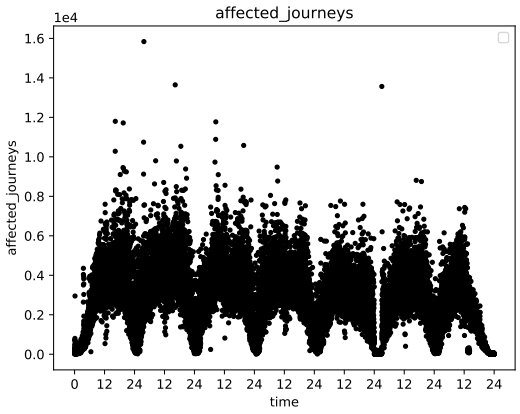


Figure 2.3: The number of connections affected by real-time changes over time for the event-based approach.

message volume at daytime compared to nighttime results in increased runtimes at daytime. The runtime of the periodic approach is roughly constant at close to 10 seconds per check run. The efficient lookup data structures used in the event-based approach lead to a runtime reduction of approximately two orders of magnitude at daytimes (0.1 seconds against nearly 10 seconds).

Timer Queue Performance

As described in Section 2.7, the system keeps a list of connections that need to be checked again at some point in the future. Since the timetable data provided by Deutsche Bahn has a granularity of one minute, the system only works at discrete points in time. Figure 2.5 shows (a) the number of checked journeys and (b) the runtime at each minute of the day for two consecutive days. As there are less trains operated at night times, the pattern of reduced work load at night times is similar to those visible in Figure 2.4.

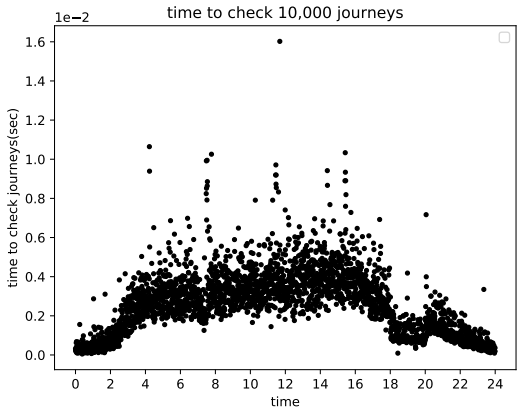


Figure 2.4: Runtimes of the event-based connection checking approach in 30 second batches (as received by Deutsche Bahn).

2.8.3 Improvements

The performance of both approaches can be further improved through better resource usage of the underlying hardware. An analysis of better memory usage (RAM in addition to disk) and CPU usage (multi-core instead of single-core) follows.

On-Disk against In-Memory Basically, all connections need to be stored persistently so they can be loaded after system restarts. This on-disk database can be used for the connection monitoring, too. Faster access times can speedup the checks. The difference is shown in Figure 2.6. The evaluation was conducted with different numbers of connections (100, 1k, 10k, 100k, 1M). Obviously, both approaches perform better with in-memory storage. While the periodic approach is less influenced by the storage medium, the event-based approach is very sensible in this regard: for 10,000 connections, the performance does not show a big difference. However, for 100,000 connections and 1,000,000 connections the runtimes leap. This difference in behavior may be explained by the access patterns of both approaches: the periodic approach on the one hand iterates all connections sequentially. On the other hand, the event-based approach does not access connections in a particular order. Thus, the event-based approach benefits more from the fast random-access read performance of RAM storage. Finally, we can

(a) queue checked journeys (b) queue time

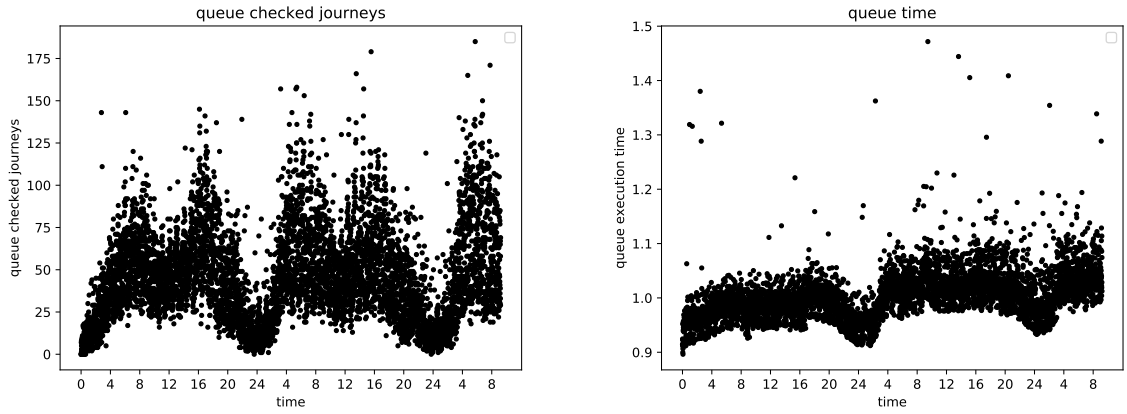


Figure 2.5: Timer Queue for 1 Million stored Connections over two days. (a) shows the queue size, (b) shows the journeys, which are checked again and (c) shows the time the queue needed for execution.

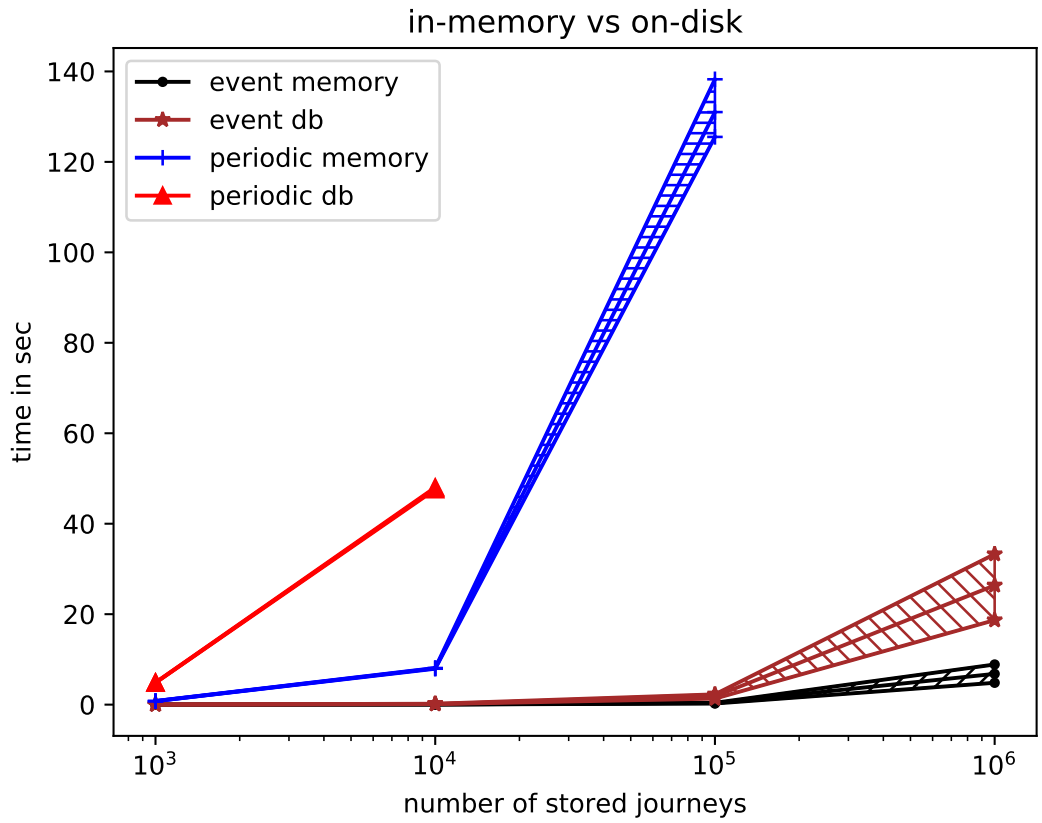
see that in practice, the periodic approach scales poorly and is not able to keep up with the real-time message input (one update every 30 seconds vs. more than two minutes for each check) for 100,000 registered connections. The event-based approach scales easily up to 1M connections.

Parallelization Modern server CPUs have many cores. Utilizing all cores is essential to make use of the compute resources the hardware provides. Both approaches presented in this paper are parallelizable in a straightforward manner: the checks for each real-time update (going through the map lookup data structures / iterating all connections) do not depend on each other. Therefore, several stages of the connection monitoring can be processed in parallel. Figure 2.7 shows the results of an evaluation conducted on a 6-core CPU: overall the average time required to process one real-time update was reduced more than 25%. Since the persistent storage medium does not benefit from a multi-core CPU, all I/O bound processes (time to write updated journeys as well as the time to write latest value known to the user) do barely benefit from the parallelization. However, all computational steps (identifying and updating affected journeys as well as updating the latest value known to the user) can be performed much faster.

2.9 Conclusion and Outlook

We presented two different approaches to monitoring booked connections of public transport travelers: the naive approach checks every stored connection periodically whereas the event-based approach reduces the number of connections that need to be checked by several magnitudes (e.g. from 1M to below 10.000). The system provides an elaborate rule system to configure which information should be pushed (e.g. via an mobile application, SMS, or e-mail) to the user (on a per-connection basis) to avoid annoying the user with updates she is not interested in. Our evaluation, which is based on real data provided by Deutsche Bahn (schedule timetable as well as real-time updates), shows that the event-based system is capable of monitoring millions of connections on a default desktop workstation. Different improvements (parallel real-time update processing as well as in-memory storage) further enhances the runtime performance of the system. In our future work, we aim to incorporate real-time updates of other means of transportation (e.g. flight data or traffic flow updates) into our system.

Figure 2.6: Analysis of how the storage medium (on-disk against in-memory) influences the runtime performance (25% quantile, median, and 75% quantile) of both approaches for different counts of stored connections.



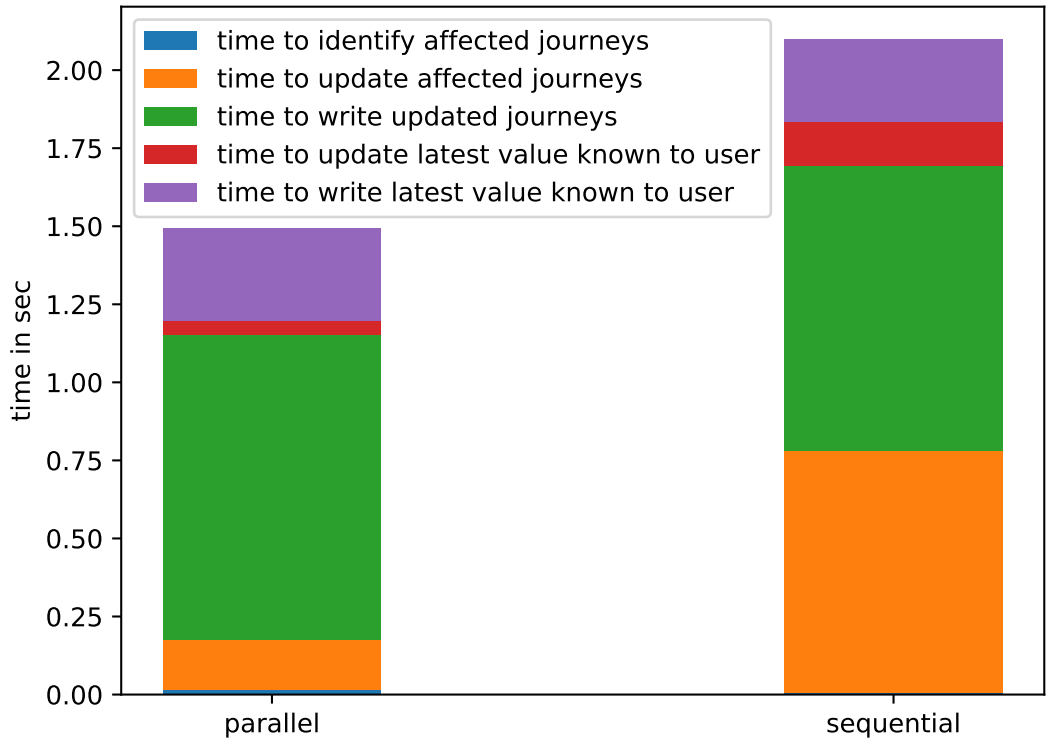


Figure 2.7: Different Parts and Their Time Requirements

3 Software Design and Architecture

In this section, we present a server software architecture which is suitable for an intermodal travel information system with the functionality presented in the previous chapters. The routing approach (presented in Chapter 1) which computes optimal sections for different modes of transportation separately allows us to take an modularized approach to software development. Additionally, we gain that different functionalities can be carried out on different servers enabling us to scale the system horizontally (to many servers) if needed.

3.1 Module System

A module is an isolated code unit that provides a certain functionality via a defined input/output interface. Each active module registers its available operations in a central registry. An active module can be instantiated locally or remotely. The system takes a list of modules to instantiate locally as well as a list of remote instances. These remote instances will be queried at startup on which operations they provide. The architecture is depicted in Figure 3.1. Local operations are dispatched to the local thread pool whereas remote operations are dispatched via the network to the respective remote server.

3.2 Efficient and Convenient Data Serialization

To enable efficient loading of data, we introduce a new approach to data serialization.[1] Common high-performance serialization approaches (such as FlatBuffers[2] or Cap'n

[1] Source code available at `https://github.com/felixguendling/cista`
[2] Google's Flatbuffers library `https://google.github.io/flatbuffers`

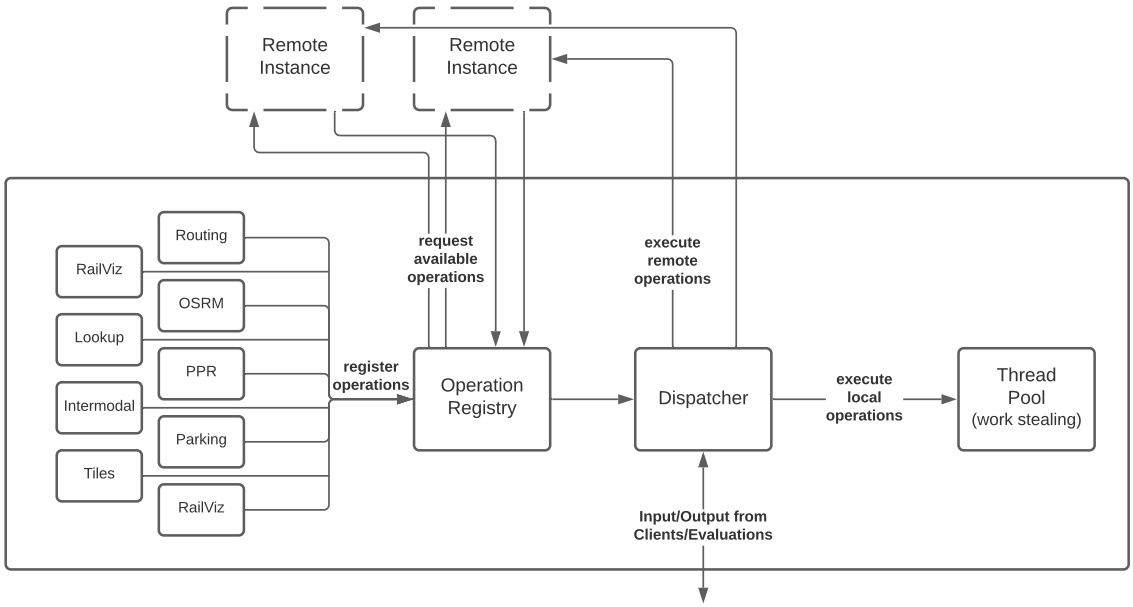


Figure 3.1: Multi-Server Modular Architecture with Multi-Threading and: Each instance has a separate registry which modules register their functionality in. Registry contents can be queried by instances. Each instance can either process requests locally or remotely.

Proto[3]) rely on an external data structure definition file to generate code for different programming languages. In contrast, our approach works by defining C++ data structures. We "abuse" structured bindings[4] as described in [Fah16] to iterate all members of a struct recursively and write its memory representation to disk. We introduce a special *offset pointer* class which is used as a drop-in replacement for raw memory pointers. An *offset pointer* stores the address it points to relative to its own address in memory (named "this" in C++). This makes the serialized memory buffer position independent because an offset pointer within a serialized buffer pointing to an address within the serialized buffer keeps the same value regardless of the position the memory buffer is located at in the main memory. This enables us to start using a buffer of serialized data structures immediately without further deserialization. When using a memory mapped file, huge data files from disk can be used without any further ado.

Comparison with Current State-of-the-Art Libraries To compare our serialization approach to other approaches, we perform a benchmark on a random graph data structure with 2000 nodes and a probability of 90% that each two nodes are connected by an edge. The results are shown in Table 3.1: our approach (Cista) outperforms all other approaches regarding all metrics but deserialize where Cap'n Proto is faster. However, if we look at the combination of deserialize and traversal, Cap'n Proto is 2-3x slower than Cista. Using raw pointers instead of the previously described *offset pointers* saves one addition operation `this + offset` to compute the actual address at pointer access and is therefore faster for traversal (112ms for raw pointers vs 132ms for *offset pointers*).

3.3 User Interaction

To make the intermodal travel information system developed in this thesis available to end-users, we developed user interfaces: one web-based and an application for the Android platform. The web interface is depicted in Figure 3.2. It provides a map overview of the current status of the network as well as input and output for journey

[3] `https://capnproto.org/capnp-tool.html`
[4] `https://en.cppreference.com/w/cpp/language/structured_binding`

Library	Serialize	Deserialize	Fast Deserialize	Traverse	Deserialize & Traverse	Size
Cap'n Proto	105 ms	0.002 ms	0.0 ms	356 ms	353 ms	50.5M
cereal	239 ms	197.000 ms	-	125 ms	322 ms	37.8M
Cista `offset`	72 ms	0.053 ms	0.0 ms	132 ms	132 ms	25.3M
Cista `raw`	3555 ms	68.900 ms	21.5 ms	112 ms	133 ms	176.4M
Flatbuffers	2349 ms	15.400 ms	0.0 ms	136 ms	133 ms	378.0M

Table 3.1: Serialization, Deserialization, and Traversal Performance Statistics: Our approach (Cista) provides high performance for all tasks while maintaining the smallest memory-footprint.

planning. The Android application is depicted in Figure 3.3. Both are available at `https://motis-project.de`.

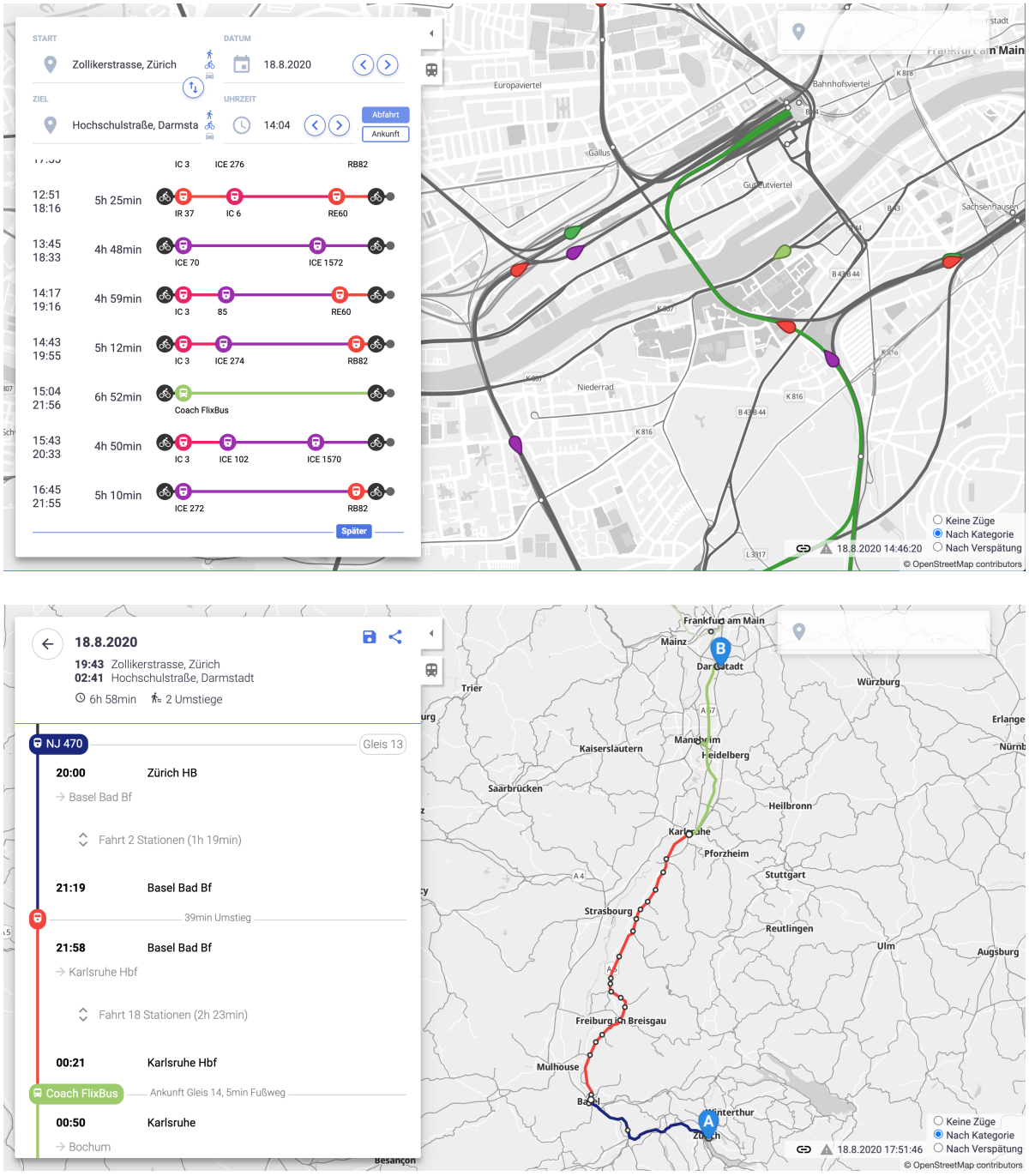


Figure 3.2: Screenshot of the MOTIS Web Interface

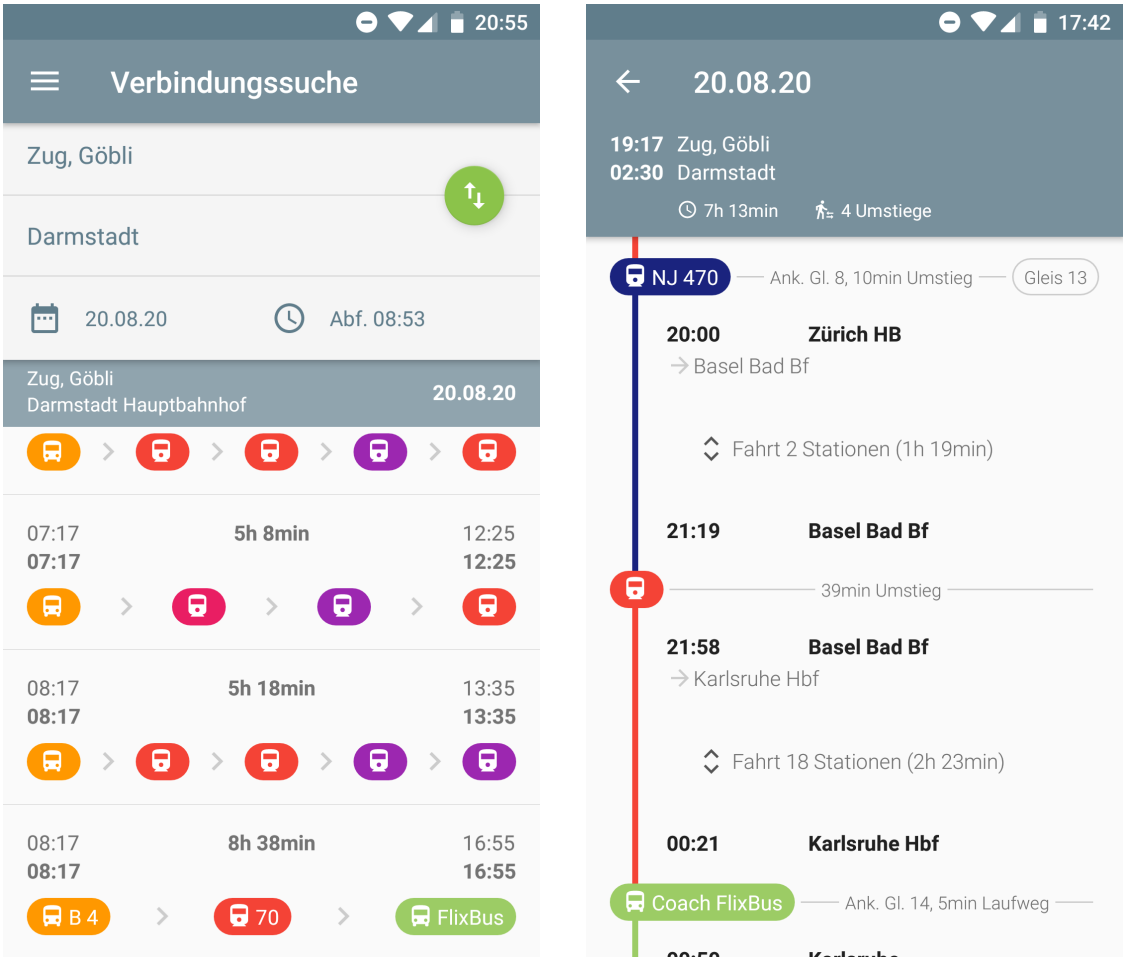


Figure 3.3: Screenshot of the MOTIS App Interface

4 Conclusion and Outlook

In this thesis, we presented a fully realistic multi-criteria intermodal routing system. Specifically, we support all aspects such as portion workings, through trains, realistic transfer times (precise to the stop position), and do not make any simplifying assumptions such as timetable periodicity. All approaches were implemented and our computational study has shown that they are feasible in practice.

We introduced several novel algorithms to solve common use cases of intermodal travel.

Dynamic Ride Sharing Unlike normal ride sharing, dynamic ride sharing offers much more flexibility to reroute the driver if the detour is acceptable. This results in a large number of options which made it algorithmically challenging to integrate into our intermodal travel information system. We introduced a preprocessing step for the "station to station" case (supplement for public transport) which employs fast filter techniques to eliminate uninteresting options (i.e. where the cost-benefit ratio for the driver is unreasonable) efficiently. We reused the preprocessed ride sharing matching options to quickly retrieve interesting offers for the "address to station" / "station to address" / "address to address" scenario at runtime.

Personalized Accessible Routing for People with Disabilities We developed a fully-integrated accessible intermodal routing. Here, we leveraged a multi-criteria approach to not only support "hard" restrictions (such as stairs for a person in a wheelchair) but also optimize "soft" restrictions where a user would like to avoid stairs but not at all costs. We consider the detour to avoid a "soft" obstacle as a trade-off. The algorithm computes all Pareto-optimal trade-offs between travel time, number of transfers, and path difficulty (sum of the difficulty of all obstacles on the path). We introduced a new

graph model to allow for personalized transfer edge weights between arbitrary stop positions of a station. Transfer paths as well as the path from the start address to the first station and the path from the last station to the destination are routed with our novel personalized multi-criteria pedestrian routing. Therefore, the resulting journeys are optimized regarding all optimization criteria (including difficulty) in a fully integrated way.

Multi-Criteria Two-way Roundtrip Park and Ride Optimization We considered a very common use case of intermodal travel where the user wants to return to the starting point and uses a bike or car for the first part of the outward trip and last part of the return trip. A conventional routing algorithm is not capable of optimizing outward and return trip in an integrated way. We presented the first integrated multi-criteria optimization approach to this problem and compared approaches with specialized versions of the CSA algorithm [Dib+13b], the TripBased routing algorithm [Wit15] as well as the multi-criteria Dijkstra on the time-dependent graph [DMS08a]. All approaches delivered exactly the same results but had different performance characteristics. We discovered that the TripBased routing algorithm performed best with two criteria (travel time and number of transfers) but with three criteria (additional price criterion), the Pareto-Dijkstra implementation outperformed the TripBased routing algorithm.

Realistic Tourist Trip Planning Another common use case of intermodal mobility not covered by conventional journey planning approaches is the planning of a tourist trip in a foreign city. We presented a novel, more realistic modeling which employs a piecewise linear function to model "saturation", i.e. the fact that the accumulation of profit flattens out and finally staying longer at a point of interest (PoI) should not yield any further profit. The presented approach is a combination of the Time Dependent Team Orienteering Problem with Time Windows (TDTOPTW) with the Orienteering Problem with Variable Profits (OPVP). Additionally, our modeling supports several entries and exits per PoI which is relevant in practice because for PoI like zoos or boardwalks the public transport stop at each entry/exit may be serviced by different lines. We introduced different greedy algorithms, two versions of an iterated local search (ILS) as well as a mixed integer linear program (MILP) representation which we implemented in Gurobi, a commercial solver. Our computational study for tour planning in the city of Berlin has

shown that the ILS approach is feasible (regarding result quality as well as computation times) for a practical online/mobile application.

Real-Time Support and Efficient Journey Monitoring The system was developed with real-time support in mind. Its model can be updated faster than real-time. This enables us to find real-time alternatives when connection is not feasible anymore due to delays, track-changes, reroutings, or cancellations. Additionally, we developed an approach to monitor millions of journeys efficiently in parallel to be able to inform the traveler about all kinds of changes as soon as possible. In our approach, the selection of change notices to be communicated to a traveler may be flexibly adapted to the travelers individual needs.

4.1 Future Work

The following topics could be interesting further research directions.

Extended Time Periods The journey planning and real time information system presented in this thesis focused on a short time horizon. Evaluations were limited to a few days. Loading larger time periods is possible, albeit not memory efficient. It would be interesting to develop speedup techniques to efficiently compute optimal intermodal journeys on a timetable covering at least a full year. Here, integrating traffic day bitfields (as used in Section 1.3) in an efficient algorithm such as RAPTOR [DPW12] or TripBased routing [Wit15] would be promising. Note that an intermodal information system that covers a longer period like one year does not need to be capable to incorporate real-time updates. There are usually no real-time updates for trips that are more than a few days in the future. Thus, the queries can either be directed to the real-time system (if they are in the near future) or to the long-term system (otherwise).

Real-Time Street Traffic Data and Time-Dependent Street Routing The modular structure of the system proposed in this thesis allows for easy replacement of the current (time-independent) street routing by a real-time time-dependent street routing. There are many existing approaches for time-dependent street routing [Bas+15]. This

can improve the driving time predictions and optimize for shorter driving durations for the usage of private car as well as for taxi usage.

Optimization with Realistic Prices The system is currently only capable to optimize artificial distance based prices. In reality, there are many quite complex pricing systems with a broad range of different rules. Integrating these rules and all kinds of discounts would be algorithmically challenging as well as commercially relevant. A mathematical model to describe pricing systems as a tariff graph is described in [Bor+18]. It would be interesting to integrate this approach in an intermodal travel information system like the one we developed here.

Use-Cases for Operators The intermodal travel information system we presented here is primarily focused on the end user. However, parts of it can be useful for operators for planning as well as for real-time dispatching purposes. For example, computing optimal roundtrip journeys with park and ride [GHW19] for the whole population of a specific area (with assumed activities) with different locations of park and ride parkings would enable to optimize the placement of parking places. Analogously, simulating routing requests of an assumed population of people with disabilities with their mobility demand enables to optimize the placement of elevators in stations to maximize the effect (i.e. reduction in travel time and number of transfers for the assumed population). Monitoring all journeys (e.g. derived from the tickets sold) is a relevant task in the context of dispatching decision making [Rüc+17].

List of Figures

List of Tables

Bibliography

[AG18] Daimler AG. *Daimler übernimmt Mitfahr-Pionier flinc*. https://media.daimler.com/marsMediaSite/de/instance/ko/Daimler-uebernimmt-Mitfahr-Pionier-flinc.xhtml?oid=29639908. [Online; accessed 22-July-2020]. 2018
Cited on page 46.

[AKE17] Ali Azizi, Farid Karimipour, and Ali Esmaeily. "Time-dependent, activity-based itinerary personal tour planning in multimodal transportation networks". In: *Annals of GIS* 23.1 (2017), pp. 27–39
Cited on page 102.

[AS11] Rahim A Abbaspour and Farhad Samadzadegan. "Time-dependent personal tour planning and scheduling in metropolises". In: *Expert Systems with Applications* 38.10 (2011), pp. 12439–12452
Cited on page 102.

[AW12] Leonid Antsfeld and Toby Walsh. "Finding multi-criteria optimal paths in multi-modal public transportation networks using the transit algorithm". In: *Intelligent Transport Systems World Congress*. 2012, p. 25
Cited on page 21.

[Aya+17] Victor Anthony Arrascue Ayala et al. "A delay-robust touristic plan recommendation using real-world public transportation information". In: *CEUR Workshop Proceedings*. Vol. 1906. 2017
Cited on page 102.

[Bar+08] Chris Barrett et al. “Engineering label-constrained shortest-path algorithms”. In: *International conference on algorithmic applications in management*. Springer. 2008, pp. 27–37
Cited on page 7.

[Bas+10] Hannah Bast et al. “Fast routing in very large public transportation networks using transfer patterns”. In: *European Symposium on Algorithms*. Springer. 2010, pp. 290–301
Cited on page 6, 21.

[Bas+15] Hannah Bast et al. “Route Planning in Transportation Networks”. In: *CoRR* abs/1504.05140 (2015). URL: http://arxiv.org/abs/1504.05140
Cited on page 149.

[Bas+16a] Hannah Bast et al. “Route planning in transportation networks”. In: *Algorithm Engineering*. Springer, 2016, pp. 19–80
Cited on page 5, 6, 29.

[Bas+16b] Hannah Bast et al. “Route planning in transportation networks”. In: *Algorithm engineering*. Springer, 2016, pp. 19–80
Cited on page 61.

[Bas09] Hannah Bast. “Car or public transport—two worlds”. In: *Efficient Algorithms*. Springer, 2009, pp. 355–367
Cited on page 43.

[Bau+10] Reinhard Bauer et al. “Combining hierarchical and goal-directed speed-up techniques for dijkstra’s algorithm”. In: *Journal of Experimental Algorithmics (JEA)* 15 (2010), pp. 2–1
Cited on page 6.

[Bau+19] Moritz Baum et al. “UnLimited TRAnsfers for Multi-Modal Route Planning: An Efficient Solution”. In: *27th Annual European Symposium on Algorithms (ESA 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2019

Cited on page 6.

[BBS13] Hannah Bast, Mirko Brodesser, and Sabine Storandt. “Result diversity for multi-modal route planning”. In: *13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2013
Cited on page 7.

[BCE09] Aurelie Bousquet, Sophie Constans, and Nour-Eddin El Faouzi. “On the adaptation of a label-setting shortest path algorithm for one-way and two-way routing in multimodal urban transport networks”. In: *International Network Optimization Conference*. 2009
Cited on page 76.

[BFG07] Sylvain Boussier, Dominique Feillet, and Michel Gendreau. “An exact algorithm for team orienteering problems”. In: *4or* 5.3 (2007), pp. 211–230
Cited on page 101.

[BGM10] Annabell Berger, Martin Grimmer, and Matthias Müller-Hannemann. “Fully dynamic speed-up techniques for multi-criteria shortest path searches in time-dependent networks”. In: *International Symposium on Experimental Algorithms*. Springer. 2010, pp. 35–46
Cited on page 6.

[BHS16] Hannah Bast, Matthias Hertel, and Sabine Storandt. “Scalable transfer patterns”. In: *2016 Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM. 2016, pp. 15–29
Cited on page 6, 21.

[BJM00] Chris Barrett, Riko Jacob, and Madhav Marathe. “Formal-language-constrained path problems”. In: *SIAM Journal on Computing* 30.3 (2000), pp. 809–837
Cited on page 7.

[bla18] blablacar. *BlaBlaCar unveils new search engine, logo and visual identity*. `https://blog.blablacar.com/newsroom/news-list/new-search-logo-visual-identity`. [Online; accessed 22-July-2020]. 2018
Cited on page 47.

[BM09] Annabell Berger and Matthias Müller-Hannemann. "Subpath-Optimality of Multi-Criteria Shortest Paths in Time-and Event-Dependent Networks". In: (2009)
Cited on page 44, 64, 67.

[BMV14] Joan Borràs, Antonio Moreno, and Aida Valls. "Intelligent tourism recommender systems: A survey". In: *Expert Systems with Applications* 41.16 (2014), pp. 7370–7389
Cited on page 117.

[BMW15] Candace Brakewood, Gregory S Macfarlane, and Kari Watkins. "The impact of real-time information on bus ridership in New York City". In: *Transportation Research Part C: Emerging Technologies* 53 (2015), pp. 59–75
Cited on page 122.

[Bor+18] Ralf Borndörfer et al. *Ein mathematisches Modell zur Beschreibung von Preissystemen im öV*. deu. Tech. rep. 18-47. Takustr. 7, 14195 Berlin: ZIB, 2018
Cited on page 150.

[BS14] Hannah Bast and Sabine Storandt. "Frequency-based search for public transit". In: *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM. 2014, pp. 13–22
Cited on page 21.

[BSS13] Hannah Bast, Jonas Sternisko, and Sabine Storandt. "Delay-Robustness of Transfer Patterns in Public Transportation Route Planning". In: *13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*. 2013, pp. 42–54

Cited on page 6.

[BTD04] Daniel Baumann, Alexandre Torday, and A-G Dumont. “The importance of computing intermodal roundtrips in multimodal guidance systems”. In: *Proceedings of the 4th STRC Swiss Transport Research Conference*. LAVOC-CONF-2008-029. 2004
Cited on page 75.

[Cat+11] Oded Cats et al. “Effect of real-time transit information on dynamic passenger path choice”. In: *Transportation Research Record* 2217 (2011), pp. 46–54
Cited on page 122.

[CGW96a] I-Ming Chao, Bruce L Golden, and Edward A Wasil. “A fast and effective heuristic for the orienteering problem”. In: *European journal of operational research* 88.3 (1996), pp. 475–489
Cited on page 101.

[CGW96b] I-Ming Chao, Bruce L Golden, and Edward A Wasil. “The team orienteering problem”. In: *European journal of operational research* 88.3 (1996), pp. 464–474
Cited on page 101.

[Coh+03] Edith Cohen et al. “Reachability and distance queries via 2-hop labels”. In: *SIAM Journal on Computing* 32.5 (2003), pp. 1338–1355
Cited on page 6.

[DB19] Deutsche Bahn AG. *Deutsche Bahn - Daten & Fakten 2018*. 2019
Cited on page 1.

[Del+09] Daniel Delling et al. “Efficient route planning in flight networks”. In: *9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS’09)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2009
Cited on page 6.

[Del+13a] Daniel Delling et al. “Computing Multimodal Journeys in Practice”. In: *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*. Vol. 7933. Lecture Notes in Computer Science. Springer, 2013, pp. 260–271. DOI: 10.1007/978-3-642-38527-8_24
Cited on page 76.

[Del+13b] Daniel Delling et al. “Computing multimodal journeys in practice”. In: *International Symposium on Experimental Algorithms*. Springer. 2013, pp. 260–271
Cited on page 7.

[Del+13c] Daniel Delling et al. “PHAST: Hardware-accelerated shortest path trees”. In: *Journal of Parallel and Distributed Computing* 73.7 (2013), pp. 940–952
Cited on page 6.

[Del+15] Daniel Delling et al. “Public transit labeling”. In: *International Symposium on Experimental Algorithms*. Springer. 2015, pp. 273–285
Cited on page 5, 6.

[Del+17] Daniel Delling et al. “Faster transit routing by hyper partitioning”. In: *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2017
Cited on page 6.

[Del08] Daniel Delling. “Time-dependent SHARC-routing”. In: *European symposium on algorithms*. Springer. 2008, pp. 332–343
Cited on page 6.

[Dib+13a] Julian Dibbelt et al. “Intriguingly simple and fast transit routing”. In: *International Symposium on Experimental Algorithms*. Springer. 2013, pp. 43–54
Cited on page 5, 21, 31, 36.

[Dib+13b] Julian Dibbelt et al. “Intriguingly simple and fast transit routing”. In: *International Symposium on Experimental Algorithms*. Springer. 2013, pp. 43–54
Cited on page 8, 43, 68, 76, 109, 148.

[Dib+13c] Julian Dibbelt et al. “Intriguingly simple and fast transit routing”. In: *International Symposium on Experimental Algorithms*. Springer. 2013, pp. 43–54
Cited on page 76, 85, 86, 96.

[Dij+59] Edsger W Dijkstra et al. “A note on two problems in connexion with graphs”. In: *Numerische mathematik* 1.1 (1959), pp. 269–271
Cited on page 5, 8, 63.

[DKP12a] Daniel Delling, Bastian Katz, and Thomas Pajor. “Parallel computation of best connections in public transportation networks”. In: *Journal of Experimental Algorithmics (JEA)* 17 (2012), pp. 4–1
Cited on page 6.

[DKP12b] Daniel Delling, Bastian Katz, and Thomas Pajor. “Parallel computation of best connections in public transportation networks”. In: *Journal of Experimental Algorithmics (JEA)* 17 (2012), pp. 4–4
Cited on page 21.

[DL13] Florian Drews and Dennis Luxen. “Multi-hop ride sharing”. In: *Sixth annual symposium on combinatorial search*. 2013
Cited on page 47, 48.

[DMS08a] Yann Disser, Matthias Müller–Hannemann, and Mathias Schnee. “Multi-criteria shortest paths in time-dependent train networks”. In: *International Workshop on Experimental and Efficient Algorithms*. Springer. 2008, pp. 347–361
Cited on page 9, 68, 76, 80, 82, 96, 127, 148.

[DMS08b] Yann Disser, Matthias Müller–Hannemann, and Mathias Schnee. “Multi-criteria shortest paths in time-dependent train networks”. In: *International Workshop on Experimental and Efficient Algorithms*. Springer. 2008, pp. 347–361
Cited on page 48, 49, 67, 68, 109.

[DMS08c] Yann Disser, Matthias Müller-Hannemann, and Mathias Schnee. “Multi-criteria shortest paths in time-dependent train networks”. In: *Proceedings of the 7th international conference on Experimental algorithms*. Springer-Verlag. 2008, pp. 347–361
Cited on page 21, 31, 36.

[DPW09] Daniel Delling, Thomas Pajor, and Dorothea Wagner. “Accelerating multi-modal route planning by access-nodes”. In: *European Symposium on Algorithms*. Springer. 2009, pp. 587–598
Cited on page 7, 76.

[DPW12] Daniel Delling, Thomas Pajor, and Renato F Werneck. “Round-Based Public Transit Routing”. In: *2012 Proceedings of the Fourteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM. 2012, pp. 130–140
Cited on page 5, 7, 8, 21, 31, 36, 43, 44, 67, 68, 72, 76, 87, 109, 149.

[EL13] Güneş Erdoğan and Gilbert Laporte. “The orienteering problem with variable profits”. In: *Networks* 61.2 (2013), pp. 104–116
Cited on page 101.

[Fah+16] Sebastian Fahnenschreiber et al. “A multi-modal routing approach combining dynamic ride-sharing and public transport”. In: *Transportation Research Procedia* 13 (2016), pp. 176–183
Cited on page 2, 46.

[Fah16] Björn Fahller. *Serializing structs with C++17 structured bindings*. 2016. URL: `https://playfulprogramming.blogspot.com/2016/12/serializing-structs-with-c17-structured.html`

Cited on page 143.

[FL02] Fedor V Fomin and Andrzej Lingas. “Approximation algorithms for time-dependent orienteering”. In: *Information Processing Letters* 83.2 (2002), pp. 57–62
Cited on page 101.

[Flo62] Robert W Floyd. “Algorithm 97: shortest path”. In: *Communications of the ACM* 5.6 (1962), p. 345
Cited on page 64.

[FMS08] Lennart Frede, Matthias Müller-Hannemann, and Mathias Schnee. “Efficient on-trip timetable information in the presence of delays”. In: *8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'08)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2008
Cited on page 68, 119, 120, 122.

[Fol+10] Robert Follmer et al. *Mobilität in Deutschland 2008 - Ergebnisbericht*. Tech. rep. `http://www.mobilitaet-in-deutschland.de/pdf/MiD2008_Abschlussbericht_I.pdf`, FE-Nr. 70.801/2006. infas Institut für angewandte Sozialwissenschaft GmbH and Deutsches Zentrum für Luft- und Raumfahrt e.V. Institut für Verkehrsforschung, 2010
Cited on page 45.

[Fur+13] Masabumi Furuhata et al. “Ridesharing: The state-of-the-art and future directions”. In: *Transportation Research Part B: Methodological* 57 (2013), pp. 28–46
Cited on page 47, 48.

[Gar+10] Ander Garcia et al. “Personalized tourist route generation”. In: *International Conference on Web Engineering*. Springer. 2010, pp. 486–497
Cited on page 102.

[Gar+13] Ander Garcia et al. “Integrating public transportation in personalised electronic tourist guides”. In: *Computers & Operations Research* 40.3 (2013), pp. 758–774
Cited on page 101, 108.

[Gav+04] Cyril Gavoille et al. “Distance labeling in graphs”. In: *Journal of Algorithms* 53.1 (2004), pp. 85–112
Cited on page 6.

[Gav+14a] Damianos Gavalas et al. “A survey on algorithmic approaches for solving tourist trip design problems”. In: *Journal of Heuristics* 20.3 (2014), pp. 291–328
Cited on page 101.

[Gav+14b] Damianos Gavalas et al. “Mobile recommender systems in tourism”. In: *Journal of network and computer applications* 39 (2014), pp. 319–333
Cited on page 102.

[Gav+15a] Damianos Gavalas et al. “Heuristics for the time dependent team orienteering problem: Application to tourist route planning”. In: *Computers & Operations Research* 62 (2015), pp. 36–50
Cited on page 102.

[Gav+15b] Damianos Gavalas et al. “The eCOMPASS multimodal tourist tour planner”. In: *Expert systems with Applications* 42.21 (2015), pp. 7303–7316
Cited on page 102.

[Gei+08] Robert Geisberger et al. “Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks”. In: *Proceedings of the 7th International Conference on Experimental Algorithms*. WEA’08. Provincetown, MA, USA: Springer-Verlag, 2008, pp. 319–333. ISBN: 3-540-68548-0, 978-3-540-68548-7. URL: http://dl.acm.org/citation.cfm?id=1788888.1788912
Cited on page 41, 44.

[Gei+12] Robert Geisberger et al. “Exact routing in large road networks using contraction hierarchies”. In: *Transportation Science* 46.3 (2012), pp. 388–404
Cited on page 6.

[Gei10] Robert Geisberger. “Contraction of Timetable Networks with Realistic Transfers.” In: *SEA*. Springer. 2010, pp. 71–82
Cited on page 21.

[GH05] Andrew V Goldberg and Chris Harrelson. “Computing the shortest path: A search meets graph theory.” In: *SODA*. Vol. 5. 2005, pp. 156–165
Cited on page 6.

[GHW19] Felix Gündling, Pablo Hoch, and Karsten Weihe. “Multi Objective Optimization of Multimodal Two-Way Roundtrip Journeys”. In: *RailNorrköping 2019. 8th International Conference on Railway Operations Modelling and Analysis (ICROMA), Norrköping, Sweden, June 17th–20th, 2019*. 069. Linköping University Electronic Press. 2019, pp. 350–360
Cited on page 3, 9, 150.

[GLV16] Aldy Gunawan, Hoong Chuin Lau, and Pieter Vansteenwegen. “Orienteering problem: A survey of recent variants, solution approaches and applications”. In: *European Journal of Operational Research* 255.2 (2016), pp. 315–332
Cited on page 101, 102.

[GLV87] Bruce L Golden, Larry Levy, and Rakesh Vohra. “The orienteering problem”. In: *Naval Research Logistics (NRL)* 34.3 (1987), pp. 307–318
Cited on page 101.

[GMS07] Thorsten Gunkel, Matthias Müller-Hannemann, and Mathias Schnee. “Improved Search for Night Train Connections”. In: *PROCEEDINGS OF THE 7TH WORKSHOP ON ALGORITHMIC APPROACHES FOR TRANSPORTATION MODELING, OPTIMIZATION, AND SYSTEMS (ATMOS 2007*. Citeseer. 2007
Cited on page 19, 24.

[Gue99] Cyrille Gueguen. “Méthodes de résolution exacte pour les problèmes de tournées de véhicules”. PhD thesis. Châtenay-Malabry, Ecole centrale de Paris, 1999
Cited on page 101.

[Gün+14] Felix Gündling et al. “Fully Realistic Multi-Criteria Multi-Modal Routing”. In: *tuprints - TU Darmstadt publication service* (2014). URL: `http://tuprints.ulb.tu-darmstadt.de/4298/`
Cited on page 66, 108.

[Gur20] LLC Gurobi Optimization. *Gurobi Optimizer Reference Manual*. 2020. URL: `http://www.gurobi.com`
Cited on page 108, 112.

[Hil+09] Moritz Hilger et al. “Fast point-to-point shortest path computations with arc-flags”. In: *The Shortest Path Problem: Ninth DIMACS Implementation Challenge* 74 (2009), pp. 41–72
Cited on page 6.

[HNR68] Peter E Hart, Nils J Nilsson, and Bertram Raphael. “A formal basis for the heuristic determination of minimum cost paths”. In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107
Cited on page 7.

[Hug+13] Marie-José Huguet et al. “Efficient algorithms for the 2-way multi modal shortest path problem”. In: *Electronic Notes in Discrete Mathematics* 41 (2013), pp. 431–437
Cited on page 76.

[Jes+09] Julie Jespersen-Groth et al. “Disruption management in passenger railway transportation”. In: *Robust and online large-scale optimization*. Springer, 2009, pp. 399–421
Cited on page 123.

[KLC12] Dominik Kirchler, Leo Liberti, and Roberto Wolfler Calvo. “A label correcting algorithm for the shortest path problem on a multi-modal route network”. In: *International Symposium on Experimental Algorithms*. Springer. 2012, pp. 236–247
Cited on page 7.

[Kno+07] Sebastian Knopp et al. “Computing Many-to-Many Shortest Paths Using Highway Hierarchies”. In: *Algorithm Engineering and Experimentation*. 2007
Cited on page 41.

[LM90] Gilbert Laporte and Silvano Martello. “The selective travelling salesman problem”. In: *Discrete applied mathematics* 26.2-3 (1990), pp. 193–207
Cited on page 101.

[LV11] Dennis Luxen and Christian Vetter. “Real-time routing with OpenStreetMap data”. In: *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. GIS ’11. Chicago, Illinois: ACM, 2011, pp. 513–516. ISBN: 978-1-4503-1031-4. DOI: 10.1145/2093973.2094062. URL: http://doi.acm.org/10.1145/2093973.2094062
Cited on page 40, 48, 55, 76, 79.

[LV12] Shih-Wei Lin and F Yu Vincent. “A simulated annealing heuristic for the team orienteering problem with time windows”. In: *European Journal of Operational Research* 217.1 (2012), pp. 94–107
Cited on page 102.

[LW79] Tomás Lozano-Pérez and Michael A Wesley. “An algorithm for planning collision-free paths among polyhedral obstacles”. In: *Communications of the ACM* 22.10 (1979), pp. 560–570
Cited on page 64, 65.

[Mor+13] Antonio Moreno et al. “Sigtur/e-destination: ontology-based personalized recommendation of tourism and leisure activities”. In: *Engineering applications of artificial intelligence* 26.1 (2013), pp. 633–651
Cited on page 117.

[MS04] Matthias Müller-Hannemann and Mathias Schnee. “Finding all attractive train connections by multi-criteria Pareto search”. In: *Proceedings of the 4th international Dagstuhl, ATMOS conference on Algorithmic approaches for transportation modeling, optimization, and systems*. Springer-Verlag. 2004, pp. 246–263
Cited on page 21, 23.

[MS09] Matthias Müller-Hannemann and Mathias Schnee. “Efficient timetable information in the presence of delays”. In: *Robust and Online Large-Scale Optimization*. Springer, 2009, pp. 249–272
Cited on page 61, 122, 127.

[MS10] Matthias Müller-Hannemann and Stefan Schirra. *Algorithm Engineering*. Springer, 2010
Cited on page 2.

[MW95] Alberto O Mendelzon and Peter T Wood. “Finding regular simple paths in graph databases”. In: *SIAM Journal on Computing* 24.6 (1995), pp. 1235–1258
Cited on page 7.

[Nac95] Karl Nachtigall. “Time depending shortest-path problems with applications to railway networks”. In: *European Journal of Operational Research* 83.1 (1995), pp. 154–166
Cited on page 21.

[Neu93] John Von Neumann. “First Draft of a Report on the EDVAC”. In: *IEEE Annals of the History of Computing* 15.4 (1993), pp. 27–75
Cited on page 2.

[Org11] World Health Organization. *World report on disability*. 2011
Cited on page 59.

[ÖW20] Erhun Özkan and Amy R Ward. “Dynamic matching for real-time ride sharing”. In: *Stochastic Systems* 10.1 (2020), pp. 29–70
Cited on page 47.

[Pel+15] Dominik Pelzer et al. “A partition-based match making algorithm for dynamic ridesharing”. In: *IEEE Transactions on Intelligent Transportation Systems* 16.5 (2015), pp. 2587–2598
Cited on page 47.

[Pop35] Sir Karl Popper. *Logik der Forschung - Zur Erkenntnistheorie der modernen Naturwissenschaft*. Springer, 1935
Cited on page 2.

[PV19] Duc-Minh Phan and Laurent Viennot. “Fast Public Transit Routing with Unrestricted Walking through Hub Labeling”. In: *International Symposium on Experimental Algorithms*. Springer. 2019, pp. 237–247
Cited on page 6.

[Pyr+08] Evangelia Pyrga et al. “Efficient models for timetable information in public transportation systems”. In: *Journal of Experimental Algorithmics (JEA)* 12 (2008), pp. 1–39
Cited on page 5, 8, 21, 67, 68.

[Rüc+17] Ralf Rückert et al. “PANDA: a software tool for improved train dispatching with focus on passenger flows”. In: *Public Transport* 9.1-2 (2017), pp. 307–324
Cited on page 150.

[San09] Peter Sanders. “Algorithm engineering–an attempt at a definition”. In: *Efficient algorithms*. Springer, 2009, pp. 321–340
Cited on page 2.

[Sch+16] Maximilian Schreieck et al. “A Matching Algorithm for Dynamic Ridesharing”. In: *Transportation Research Procedia* 100.19 (2016), pp. 272–285
Cited on page 47.

[Sch09] Mathias Schnee. “Fully realistic multi-criteria timetable information systems”. PhD thesis. Technische Universität, 2009
Cited on page 5, 8, 16, 24, 66, 119.

[SM06] Adam D Sobek and Harvey J Miller. “U-Access: a web-based system for routing pedestrians of differing abilities”. In: *Journal of geographical systems* 8.3 (2006), pp. 269–287
Cited on page 61.

[Sou+13] Wouter Souffriau et al. “The multiconstraint team orienteering problem with multiple time windows”. In: *Transportation Science* 47.1 (2013), pp. 53–63
Cited on page 101.

[Spi15] Sergio Spinatelli. “Minimal Effective Time Two-Way Park and Ride Problem”. MA thesis. 2015
Cited on page 76.

[SS12] Peter Sanders and Dominik Schultes. “Engineering highway hierarchies”. In: *Journal of Experimental Algorithmics (JEA)* 17 (2012), pp. 1–1
Cited on page 6.

[Sti+15] Mitja Stiglic et al. “The benefits of meeting points in ride-sharing systems”. In: *Transportation Research Part B: Methodological* 82 (2015), pp. 36–53
Cited on page 47.

[Sub+18] V Subramaniyaswamy et al. “Dynamo: Dynamic Multimodal Route and Travel Recommendation System”. In: *2018 International Conference on Recent Trends in Advance Computing (ICRTAC)*. IEEE. 2018, pp. 47–52
Cited on page 102.

[SW11] Peter Sanders and Dorothea Wagner. “Algorithm engineering”. In: *it-Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik* 53.6 (2011), pp. 263–265
Cited on page 2.

[SW14] Ben Strasser and Dorothea Wagner. “Connection scan accelerated”. In: *2014 Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM. 2014, pp. 125–137
Cited on page 6.

[Tör07] Johanna Törnquist. “Railway traffic disturbance management—An experimental analysis of disturbance complexity, management objectives and limitations in planning horizon”. In: *Transportation Research Part A: Policy and Practice* 41.3 (2007), pp. 249–266
Cited on page 123.

[Tri+10] Fabien Tricoire et al. “Heuristics for the multi-period orienteering problem with multiple time windows”. In: *Computers & Operations Research* 37.2 (2010), pp. 351–367
Cited on page 101.

[Tsi84] Theodore Tsiligirides. “Heuristic methods applied to orienteering”. In: *Journal of the Operational Research Society* 35.9 (1984), pp. 797–809
Cited on page 101.

[TT12] Lei Tang and Piyushimita Vonu Thakuriah. “Ridership effects of real-time bus information system: A case study in the City of Chicago”. In: *Transportation Research Part C: Emerging Technologies* 22 (2012), pp. 146–161
Cited on page 122.

[Van+09] Pieter Vansteenwegen et al. “Iterated local search for the team orienteering problem with time windows”. In: *Computers & Operations Research* 36.12 (2009), pp. 3281–3290
Cited on page 102.

[VDV19] Verband Deutscher Verkehrsunternehmen (VDV). *Statistik 2018*. Tech. rep. 2019
Cited on page 1.

[VG19] Pieter Vansteenwegen and Aldy Gunawan. *Orienteering problems: Models and algorithms for vehicle routing problems with profits*. Springer Nature, 2019
Cited on page 101.

[VSV11] Pieter Vansteenwegen, Wouter Souffriau, and Dirk Van Oudheusden. "The orienteering problem: A survey". In: *European Journal of Operational Research* 209.1 (2011), pp. 1–10
Cited on page 101.

[VV07] Pieter Vansteenwegen and Dirk Van Oudheusden. "The mobile tourist guide: an OR opportunity". In: *OR insight* 20.3 (2007), pp. 21–27
Cited on page 100.

[VW08] Thorsten Völkel and Gerhard Weber. "RouteCheckr: personalized multicriteria routing for mobility impaired pedestrians". In: *Proceedings of the 10th international ACM SIGACCESS conference on Computers and accessibility*. 2008, pp. 185–192
Cited on page 61.

[WHP14] Timothy N Weyrer, Hartwig H Hochmair, and Gernot Paulus. "Intermodal door-to-door routing for people with physical impairments in a web-based, open-source platform". In: *Transportation Research Record* 2469.1 (2014), pp. 108–119
Cited on page 61.

[Wit15] Sascha Witt. "Trip-based public transit routing". In: *Algorithms-ESA 2015*. Springer, 2015, pp. 1025–1036
Cited on page 5, 6, 8, 21, 43, 67, 76, 87, 96, 148, 149.

[Wit16] Sascha Witt. “Trip-Based Public Transit Routing Using Condensed Search Trees”. In: *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*. 2016, p. 1
Cited on page 6, 21.

[WZ17] Dorothea Wagner and Tobias Zündorf. “Public transit routing with unrestricted walking”. In: *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2017
Cited on page 6.

[Yu+14] Jingjin Yu et al. “Optimal tourist problem and anytime planning of trip itineraries”. In: *arXiv preprint arXiv:1409.8536* (2014)
Cited on page 101, 102.

[Zad88] Lotfi A Zadeh. “Fuzzy logic”. In: *Computer* 21.4 (1988), pp. 83–93
Cited on page 7.

Curriculum Vitae

2009 - 2012	**Bachelor of Science** Computer Science TU Darmstadt
2012 - 2015	**Master of Science** Computer Science Minor Subject: Optimization TU Darmstadt
2015 - 2020	**PhD Student / Researcher** Algorithm Group Computer Science Department TU Darmstadt